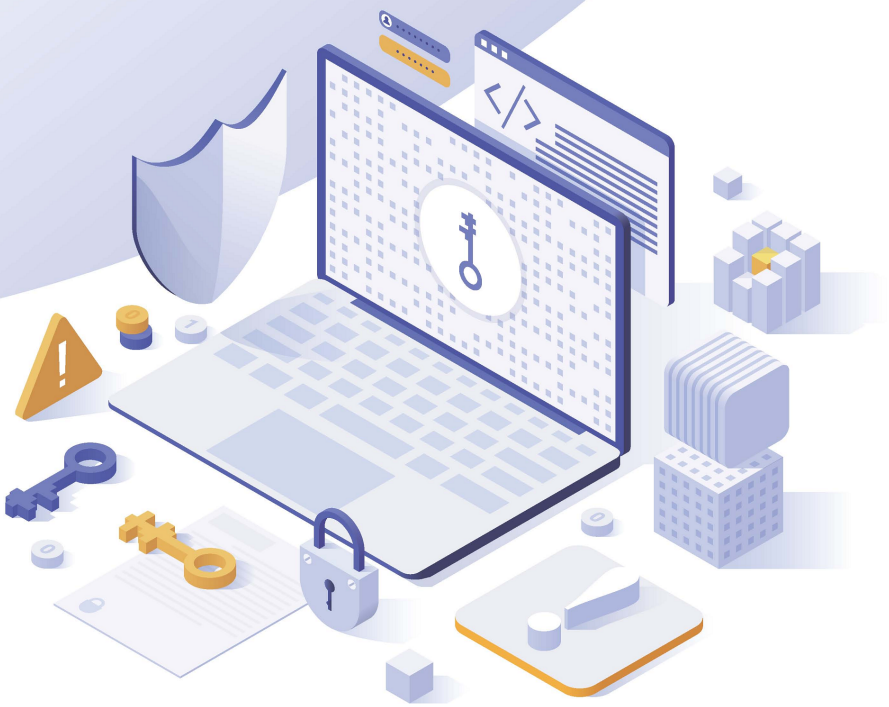
발 간 등 록 번 호
11-1311000-000395-14

소프트웨어 보안약점 진단가이드



행정안전부
KISA 한국인터넷진흥원

소프트웨어 보안약점 진단가이드

제·개정이력(Revision History)

번호	제·개정일	변경 내용	발간팀	비고
1	2012. 5.	**[제정]** • 소프트웨어 보안약점 진단가이드 제정	공공SW보호팀	
2	2013.11.	**[개정]** • 보안약점 기준 변경에 따른 별첨 가이드 추가	보안평가인증팀	
3	2019. 6.	**[개정]** • 별첨 가이드 통합, 설계단계 보안항목 진단 방법 등 추가	전자정부보호팀	
4	2021.11.	**[개정]** • 보안약점 기준 변경에 따른 가이드 개정	전자정부보호팀	

Contents

PART3 제3장 분석·설계단계 보안항목 진단

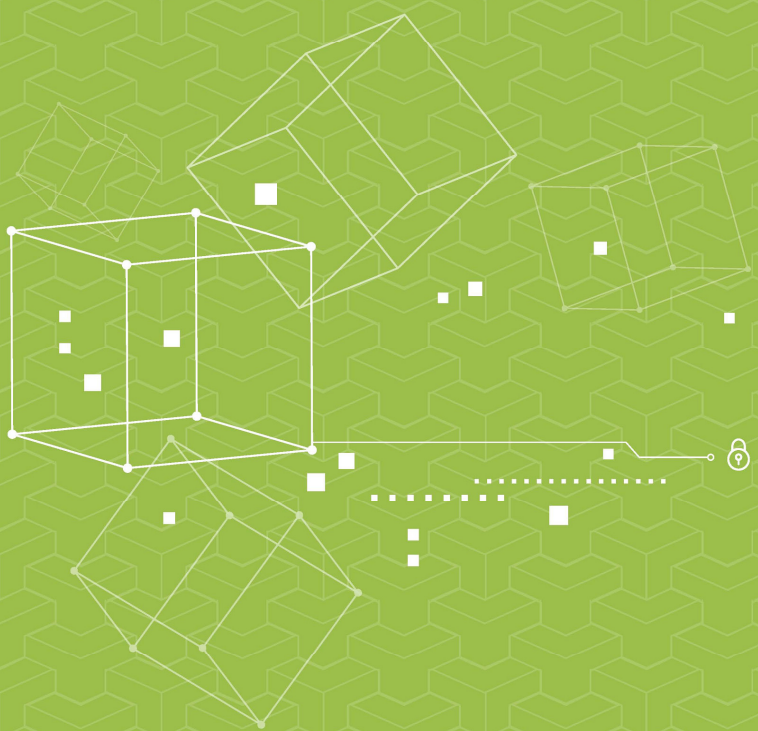
PART4 제4장 구현단계 보안약점 진단

Contents

PART4 제4장 구현단계 보안약점 진단

PART4 제4장 구현단계 보안약점 진단

PART5 제5장 부록



소프트웨어 보안약점 진단가이드

KOREA INTERNET & SECURITY AGENCY

PART 1

제1장 개요

PART 1

제1장 개요

| 제 1 절 | 배경

공격자의 초점이 지속적으로 애플리케이션 계층을 향해 이동함에 따라 소프트웨어(SW, Software) 자원보호가 중요 해졌다. 최근 발생되는 인터넷상 공격시도의 약 75%는 소프트웨어의 보안취약점을 악용하는 것으로, 특히 외부에 공개되어 불특정 다수를 대상으로 사용자 정보를 처리하는 웹 애플리케이션의 취약점으로 인해 중요정보가 유출되는 침해사고가 빈번하게 발생되고 있다. 보안강화를 위해 구축해놓은 침입 차단시스템과 같은 보안장비로는 응용프로그램 취약점에 대한 공격을 완벽히 방어하는 것은 불가능하다.

'SW개발보안'은 SW개발과정에서 개발자의 실수, 논리적 오류 등으로 인해 발생될 수 있는 보안 취약점, 보안약점들을 최소화하여 사이버 보안위협에 대응할 수 있는 안전한 SW를 개발하기 위한 일련의 보안활동을 의미한다. 즉, SW개발 생명주기(SDLC, Software Development Life Cycle)의 각 단계별로 요구되는 보안활동을 수행함으로써 안전한 소프트웨어를 만들 수 있도록 한다.

SW개발보안의 중요성을 인식한 미국의 경우 국토안보부(DHS)를 중심으로 시큐어코딩(Secure Coding)을 포함한 SW개발 전 과정(설계, 구현, 시험 등)에 대한 보안 활동 연구를 활발히 진행하고 있으며, 이는 2011년 발표한 "안전한 사이버 미래를 위한 청사진(Blueprint for a Secure Cyber Future)"에 나타나있다.

국내의 경우,2009년부터 SW개발단계에서 SW보안약점을 진단하여 제거하는 SW개발보안 관련 연구를 진행하면서, 2009년부터 2011년까지 전자정부 지원 사업을 대상으로 SW보안약점 진단 시범사업을 수행하였다. SW보안약점 제거·조치 성과에 따라 2012년 SW개발보안 제도가 도입되어 단계적으로 의무화가 이루어졌다.

| 제 2 절 | 가이드 목적 및 구성

소프트웨어 보안의 목표는 성공적인 사업을 운영하기 위한 정보자원의 기밀성, 무결성, 가용성을 유지하는 것이다. 이러한 목표를 달성하기 위해서 보안통제 기능의 구현이 요구되며, 이 가이드에서 소프트웨어의 취약점을 완화시킬 수 있는 각 개발 단계별 기술적 통제항목이 소프트웨어 개발에 올바르게 적용되었는지 진단하는 방법을 중점적으로 설명하고 있다.

소프트웨어 보안 취약점은

- 보안 요구사항이 정의되지 않았거나,
- 논리적인 오류를 가지는 설계를 수행하였거나,
- 기술취약점을 가지는 코딩 규칙을 적용하였거나,
- 소프트웨어 배치가 적절하지 않았거나,
- 발견된 취약점에 대해 적절한 관리 또는 패치를 하지 않은 경우

발견되며, 이러한 취약점으로 인해 시스템이 처리하는 중요정보가 노출되거나 정상적인 서비스가 불가능한 상황이 발생하게 된다.

이 가이드는 소프트웨어 개발 생명주기에 고려되어야 하는 보안위협을 최소화하기 위해 각 단계별 수행해야 하는 보안 활동들이 개발자들에 의해 적절하게 수행되었는지를 점검하는 SW보안약점 진단원들에게 도움이 되도록 한다.

목적	• '행정기관 및 공공기관 정보시스템 구축·운영 지침(행정안전부고시 제2021- 3호)'에 따라 정보시스템을 구축·운영함에 있어 소프트웨어 보안약점이 없도록 안전한 소프트웨어 개발을 위한 가이드 제시
대상	• 전자정부 소프트웨어(SW) 보안약점 진단원
범위	• 신규로 개발되거나 유지보수로 변경되는 소프트웨어 개발 시 설계 및 구현단계 보안약점 진단 ※ 행정기관 등이 추진하는 정보화사업 중 소프트웨어 개발 프로세스의 전체 단계(분석·설계·구현·테스트)에 초점
구성	• (1장) 가이드의 목적과 구성 • (2장) SW개발보안 의무화 대상, 범위, 기준과 정보화사업 단계별, 주체별 개발보안활동 소개 • (3장) 분석·설계단계 보안활동 이해 및 진단방법 • (4장) 구현단계 SW보안약점 이해 및 진단방법 • (부록) 설계단계 보안설계 기준, 구현단계 보안약점 제거 기준, 설계단계 보안설계 적용계획서 및 산출물 예시, 용어정리
활용	• (발주자) 안전한 소프트웨어를 구축하기 위해 분석·설계·구현·테스트의 각 단계별로 보안 강화 정책이 적용될 수 있도록 요구사항 도출시 활용 • (사업자) SW개발보안을 적용하여 소프트웨어 개발시 활용 • (감리법인) 설계단계의 보안설계 적용 확인, 구현단계의 소스코드에 대한 보안약점 진단 및 제거, 테스트단계의 SW취약점 진단 및 조치 확인 시 활용

소프트웨어 보안약점 진단가이드

KOREA INTERNET & SECURITY AGENCY

PART 2

제2장 소프트웨어 개발보안

제1절 개요
제2절 소프트웨어 개발보안 체계
제3절 소프트웨어 보안약점 진단

PART 2

제2장 소프트웨어 개발보안

| 제 1 절 | 개요

SW개발보안은 해킹 등 사이버공격의 원인인 보안약점을 소프트웨어(SW, Software)개발단계에서 사전에 제거하고 소프트웨어 개발 생명주기의 각 단계별로 수행하는 일련의 보안활동으로 안전한 소프트웨어를 개발·운영하기 위한 목적으로 적용하는 개발체계이다.

안전한 소프트웨어를 만들기 위해서는 프로젝트에 참여하는 각 구성원들의 역할과 책임이 명확하게 정의되어야 하며, 개발팀에게 충분한 소프트웨어 보안 교육을 제공해야 한다. 또한 소프트웨어 개발 생명주기의 각 단계에 보안활동이 수행되어야 하며, 개발보안을 위한 표준이 확립되어야 한다. 재사용 가능한 보안 라이브러리를 작성하여 비슷한 기능을 수행하는 프로젝트에 도입함으로써 일반적인 안전성을 확보할 수 있도록 해야 하며, 보안통제의 효과성 검증으로 향후 새로운 프로젝트 에서 효과적인 보안정책이 적용될 수 있도록 해야 한다.

2021년 1월에 개정·고시된 '행정기관 및 공공기관 정보시스템 구축·운영 지침(행정안전부 고시 2021-3호)'은 개발보안과 관련하여 행정기관 및 공공기관이 정보화 사업을 추진할 경우 준수해야 할 소프트웨어 개발보안 활동 및 보안약점 진단기준 정의되어 있다. 지침에 나타난 소프트웨어 개발보안 적용 대상 및 범위·기준 등을 요약하면 다음과 같다.

표 2-1 소프트웨어 개발보안 대상 및 진단기준

구분	내 용	비고
대상	• 감리대상 정보시스템 사업 ※ 감리대상外 사업도 자체적으로 소프트웨어 개발보안 적용	'전자정부법 시행령 제71조 제1항' 참조
범위	• 신규개발의 경우 : 설계단계 산출물 및 소스코드 전체 • 유지보수의 경우 : 유지보수로 인해 변경된 설계단계 산출물 및 소스코드 전체 ※ 상용 소프트웨어 제외	'행정기관 및 공공기관 정보시스템 구축·운영 지침 제50조' 참조
기준	• 설계단계 보안설계 기준(총 20개 항목) • 구현단계 보안약점 제거 기준(총 49개 항목) ※ 정보시스템 감리기준 제10조 제1항의 세부검사항목에 포함하여야 함	'행정기관 및 공공기관 정보시스템 구축·운영 지침 제52조, 제53조' 참조
기타	• 정보화사업 발주 시, 제안요청서에 "소프트웨어 개발보안 적용" 명시 ※ 소프트웨어 보안약점 진단도구 사용 여부, 개발절차와 방법의 적절성, 소프트웨어 개발보안 관련 교육계획의 적절성 등 제안서 평가 시 반영	'행정기관 및 공공기관 정보시스템 구축·운영 지침 제16조, 제51조' 참조
	• 감리법인이 소프트웨어 보안약점 진단도구 사용 시, 국가보안기술연구소장이 인증한 보안약점 진단도구 사용 ※ 정보보호시스템 평가·인증 지침	'14.1월부터 적용
	• 감리법인은 소프트웨어 보안약점 진단 시, 진단원을 우선적으로 배치 ※ 행정기관 및 공공기관 정보시스템 구축·운영 지침 '별표4'	진단원 활용

| 제 2 절 | 소프트웨어 개발보안 체계

소프트웨어 개발보안 관련 활동 주체는 행정안전부, 발주기관(행정기관 등), 한국인터넷진흥원, 사업자, 감리법인 등으로 구분할 수 있으며, 개발보안 주체별로 잘 정의된 개발보안 활동과 주체 간의 유기적인 협력이 필요하다.

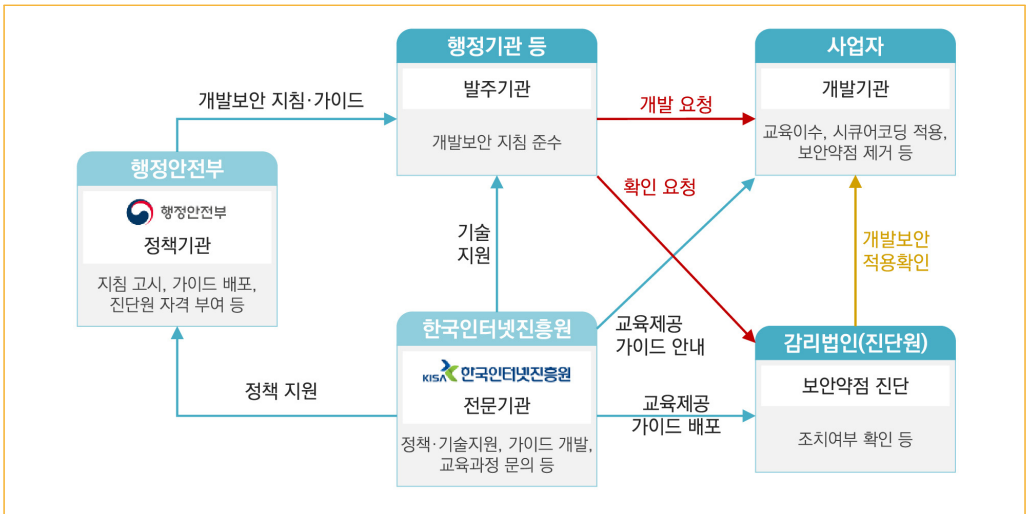


그림 2-1 | 활동 주체별 개발보안 활동

활동 주체별 개발보안의 역할은 [표2-2]과 같이 요약할 수 있다.

표 2-2 활동 주체별 개발보안 역할

구분	내 용	비고
행정안전부	• 소프트웨어 개발보안 관련 법·제도 개선, 가이드 배포 • 소프트웨어 보안약점 진단원 자격 부여	-
한국인터넷진흥원	• 소프트웨어 개발보안 정책 및 기술 지원, 가이드 개발 • 소프트웨어 개발보안 교육과정 및 자격제도 운영	-

구분	내 용	비고
발주기관	• 소프트웨어 개발보안 계획 수립 • 제안요청서에 '소프트웨어 개발보안 적용' 명시 • 소프트웨어 개발보안 역량을 갖춘 사업자 선정 • 사업자의 소프트웨어 개발보안 준수 여부 점검 • 감리법인의 소프트웨어 보안약점 제거 여부 진단 ※ CC인증 받은 진단도구 사용, 진단절차·방법의 적절성 등 평가 • 사업종료 결과물 및 증빙서류 등 최종 확인 ※ 감리법인에 의뢰(감리대상) 또는 자체적으로 수행(감리대상 外)	행정기관 및 공공기관
사업자 (개발자)	• 소프트웨어 개발보안 관련 기술 수준 및 적용 계획 명시 • 투입인력 대상 소프트웨어 개발보안 관련 교육 실시 후 투입 • 소프트웨어 개발보안 가이드를 참조하여 개발 • 자체적으로 소프트웨어 보안약점 진단 및 제거 • 소프트웨어 보안약점 관련 시정요구 이행 • 소프트웨어 보안약점이 제거된 사업 결과물 및 증빙서류 등 제출	-
감리법인	• 감리계획 수립 및 협의 • 분석단계 중요정보 및 중요기능 분류, 설계항목 정의 확인 • 설계단계 산출물에 대한 설계항목 반영 확인 • 소프트웨어 보안약점 제거 여부 진단 및 조치 결과 확인 ※ 진단도구 사용 시, CC인증(국정원)된 도구 사용 • 감리법인은 소프트웨어 보안약점 진단 시, 진단원을 우선적으로 배치 권고	진단원 활용

| 제 3 절 | 소프트웨어 보안약점 진단

1. 개요

안전한 소프트웨어를 만들기 위해서는 분석단계에서부터 보안요구항목을 식별해야 하며 발생 가능한 위협을 최소화할 수 있도록 설계, 구현하도록 노력해야 한다. 이러한 개발단계에서 적절한 보안 활동이 수행되었는지 점검하기 위한 활동을 “소프트웨어 보안약점 진단”으로 정의할 수 있다.

분석·설계단계의 진단 활동은 분석단계에 보안항목이 명확하게 식별되고, 설계단계에 이러한 보안 항목들이 적용되어 소프트웨어가 안전하게 동작되도록 설계되었는지를 점검하는 작업이다. 이 단계에서의 진단 대상은 분석과 설계 과정에서 만들어진 설계 산출물이다.

프로젝트에 적용되는 개발방법론에 따라 분석·설계단계에서 만들어지는 산출물의 종류나 내용의 범위가 다양할 수 있다. 하지만 진단가이드로 분석·설계단계의 진단 시점, 대상 산출물, 진단 방법들을 표준화하기 위한 노력을 하였으며, 이로 분석·설계단계에 보안을 고려하여 소프트웨어를 설계할 수 있도록 하고자 하였다.

분석·설계단계에서 설계항목들이 적절히 반영되었는지 여부는 일반적으로 설계가 완료되고 구현이 시작되거나 구현을 준비하는 시점에 진단하는 것이 적절하다. 보안요구항목 적용에 대한 진단은 착수 → 진단 → 보고서 작성 및 제출 → 종료 의 순서로 수행하며 진단대상 산출물은 한국정보화 진흥원에서 배포한 “CBD SW개발 표준 산출물 가이드“에서 제시하고 있는 분석·설계단계 산출물 종류와 이름을 기반으로 하였다.

구현단계에서의 진단활동은 구현된 소스코드에 대한 잠재적인 보안약점들이 소스코드 보안약점진단 도구와 진단원의 수동진단으로 진단하고, 검출된 보안약점 제거를 위한 대응방법을 제시하여 구현 단계에서 안전한 소프트웨어를 만들 수 있도록 지원한다.

2. SW보안약점 진단원의 역할

기존의 SW보안약점 진단원은 구현단계 소스코드에 대한 보안약점을 진단하고 조치방안을 제시하여 소프트웨어의 보안성을 강화하는 역할을 수행하였다.

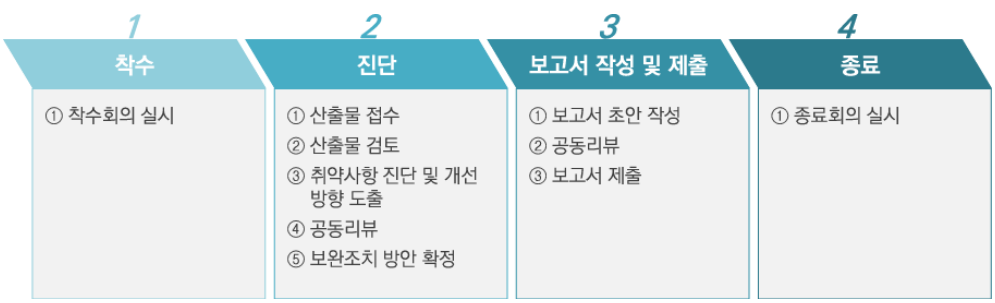
개정된 '행정기관 및 공공기관 정보시스템 구축 · 운영 지침'에서는 분석·설계단계에서 생산되는 산출물에 대해 보안설계 항목을 식별하고, 적절한 보안 대책의 수립 여부를 진단하여 안전한 설계를 할 수 있도록 지원하는 역할이 추가되었다.

표 2-3 SW보안약점 진단원 교육 과정 안내

구분	내 용	비고
교육 대상	• SW 개발경력 6년 이상 또는 취약점 진단 3년 이상	
교육 절차	• 진단원 양성과정 교육 신청 -> 경력증명 서류 제출 -> 교육 수료	
교육 시간	• 5일 40시간	
교육 내용	• SW보안약점 이론 학습 및 진단 실습	
신청 방법	• 사이버보안인재센터 홈페이지에서 신청 https://academy.kisa.or.kr/main.kisa	

3. 분석·설계단계 진단 절차

분석·설계단계 진단은 착수 → 진단 → 보고서 작성 및 제출 → 종료의 절차로 구성된다.



가. 착수단계

착수단계는 진단원, 발주자, 사업자 등 관계자가 참석하여 분석·설계단계 진단의 절차와 관련 내용을 공유하는 단계이다.

① 착수 회의 실시

착수 회의에서는 진단원, 발주자, 사업자 등이 참석하며 진단원은 분석·설계단계 진단의 목적, 적용기준과 진단항목, 일정 등 전반적인 절차와 내용에 대하여 설명하고 진단에 필요한 산출물들을 요청한다.

나. 진단단계

진단단계는 시스템 개발 사업과 관련한 산출물 접수, 산출물 검토, 보안약점 진단 및 개선방안 도출, 공동리뷰 등으로 개선방안을 확정하는 단계이다.

① 산출물 접수

개발사는 정보화사업 추진 과정 중 작성된 분석·설계 산출물을 준비하여 진단원에게 제공하며, 진단원은 제출된 산출물 목록을 확인하여 제공된 산출물에서 보안약점 진단에 필요한 자료가 모두 제출되었는지 검토한다. 산출물이 누락되었거나 추가 자료가 필요할 경우, 개발사에 요청하여 보안약점 진단에 필요한 분석·설계 산출물이 준비될 수 있도록 한다.

② 산출물 검토

진단원은 SW개발보안 가이드의 분석·설계단계 SW보안강화 활동에서 명시된 보안활동과 보안설계 기준을 기반으로 설계 산출물의 세부 내용을 검토하여 보안설계 항목 식별 및 보안 대책이 적절하게 수립되었는지 확인한다. 이 과정에서는 사업목적과 요구사항, 개발방법론 등에 대한 정확한 이해를 바탕으로 효율적이고 객관적인 개선방안을 도출하여 보안성을 향상 시킬 수 있도록 지원하는 것이 중요하다.

③ 보안약점 진단 및 개선방안 도출

진단원은 보안설계 기준을 기반으로 분석·설계 산출물의 작성 현황과 세부 내용을 검토하여 기준을 충족하고 있는지 확인한다. 분석·설계 산출물을 진단하여 개선이 필요한 사항을 발견한 경우, 이를 해결하기 위한 구체적인 개선방안을 도출한다. 보안약점으로 확인된 사항에 대해서는 SW개발보안 가이드 및 관련 법률 등 문제점에 대한 객관적 근거가 필요하며, 이를 기반으로 현실적으로 실행 가능한 수준의 개선방안을 도출하는 것이 중요하다.

④ 공동리뷰

진단원은 분석·설계 산출물을 진단하는 과정에서 발주사 및 개발사 업무 담당자와 인터뷰하여 진단현황과 진단 결과로 확인된 주요 문제점을 설명하고, 진단과정에서 파악하지 못한 사항에 대해서 의견을 청취하여 각각의 문제점을 해결하기 위한 구체적인 개선방안을 제시하며, 발주사와 개발사에서 실행 가능한지 확인한다.

⑤ 개선방안 확정

보안설계 기준 요건이 충족되지 않았거나 개선이 필요한 사항에 대해 문제 해결을 위한 구체적이고 현실적으로 실행 가능한 개선방안을 제시한다. 개선방안은 설계항목별로 일관성이 유지되어야 하며, 개선이 필요한 항목을 확인할 수 있는 증적 자료(분석·설계 산출물 등)를 기반으로 도출되어야 한다.

진단원은 진단과정에서 확인한 내용과 발주사 및 개발사 담당자와의 공동 리뷰 결과를 바탕으로 보안설계 기준에 부합하는 구체적이고 실행 가능한 개선방안을 확정한다.

다. 보고서 작성 및 제출 단계

보고서 작성 및 제출 단계에서는 진단단계에서 발견된 문제점과 이에 대한 개선방안을 보고서로 작성한 뒤, 공동리뷰를 거쳐 최종 진단 결과 보고서로 제출한다. 진단 결과 보고서에는 진단원, 진단 기간, 진단 기준, 진단 결과 등을 작성하며, 사업의 규모, 특성 등을 반영하여 목차와 내용을 조정한다.

① 보고서 초안 작성

보고서 초안 작성 단계는 진단원이 개발사가 제출한 각종 산출물에 대한 보안약점 진단, 발주사와 개발사 담당자와의 공동리뷰 등으로 확정한 보안약점과 개선방안을 보고서로 작성한다. 보고서에는 보안약점으로 판단한 사유 및 증적 자료가 명시되고 논리적으로 일관성이 유지되도록 작성하여야 한다.

보고서에는 보안설계 기준에 따른 진단 결과, 진단원, 진단 기간, 진단 대상의 범위(분석·설계 산출물 등), 개선방안 등을 명시되어야 한다.

② 공동리뷰

보고서 초안에 대해 발주사 및 개발사와 공동리뷰를 진행한다. 공동리뷰에서는 보고서에 정리된 보안약점에 대한 사실 여부 검토, 추가 증적 자료 제출 의사, 보안약점에 대한 개선 여부와 개선일정

등을 검토한다. 이 과정에서 보고서 내 존재할 수 있는 증적 자료 누락, 잘못된 보안약점 도출, 실현 가능성이 낮은 개선방안 등에 대해 수정사항을 확인하고 보고서에 반영한다.

③ 보고서 제출

공동리뷰 과정에서 수정사항이 확인된 경우 수정되어야 하는 내용에 대한 근거 및 관련 자료를 제출받아 추가로 검토하며, 확인된 수정사항을 보고서 초안에 반영하여 최종진단결과보고서를 작성하여 발주사에 제출한다.

라. 종료단계

진단결과보고서를 바탕으로 종료회의를 실시한다. 종료회의에서는 진단결과보고서에 대한 설명, 발주사 및 개발사의 개선 계획 검토 등을 수행한다.

① 종료회의 실시

종료회의에서는 진단결과보고서에 대한 최종적인 검토 결과를 설명한다. 보고서 내용에 대해 발주사와 개발사 담당자가 명확하게 이해하고 수행할 수 있도록 설명하고 질의응답으로 최대한의 공감대가 형성될 수 있도록 한다.

진단결과보고서에 개선사항이 존재할 경우 기한을 정하여 발주사와 개발사 담당자에게 수정하여 증적자료와 함께 제출하도록 요청하고, 이에 따른 개선 완료 여부를 검토 확인하여 추후 보고서에 내용을 추가한다.

※ 설계단계 진단대상 산출물

진단대상 산출물은 프로젝트의 개발방법론이나 규모 등에 따라 차이가 있겠지만 진단가이드에서는 한국정보화진흥원에서 배포한 “CBD SW개발 표준 산출물 관리가이드” 기반의 설계산출물 목록을 참조하였으며, 일반적으로 현장에서 많이 사용되는 산출물명도 포함하여 정리하였다.

표 2-4 분석단계 산출물

문서명	내 용
요구사항 정의서	• 애플리케이션 개발에 필요한 기능, 품질, 기술 등 시스템 관련 요구사항을 도출 하여 발주사와 내용을 합의하고 정리하여 체계적으로 작성한 문서를 말하며 요구사항 정의서에는 요구사항 명칭, ID를 기반으로 요구내용, 중요도, 담당자, 특기사항 등이 포함된다.
요구사항 추적표	• 도출된 요구사항을 기반으로 개발 각 단계별로 작성한 산출물들이 일관성 있게 작성되었는지 추적할 수 있도록 작성된 문서를 말하며 요구사항 정의서로부터 시작하여 분석, 설계, 구현, 시험의 각 단계별 산출물에 ID를 식별하고 전단계 산출물과 후속단계 산출물간의 연계관계가 표시되어 단계별 산출물간 연관 관계을 추적할 수 있도록 한다.
유즈케이스 명세서	• 유즈케이스 다이어그램에 표현된 각 유즈케이스를 상세하게 설명하는 문서로 액터와 유즈케이스간 상호작용과 내부업무 흐름을 설명하며, 유즈케이스의 작업이 완료되기 위한 여러 비즈니스 이벤트 의 흐름을 표현한다.
유즈케이스 다이어그램	• 액터 중심의 시스템 청사진으로서, 액터가 어떤 기능을 사용할 수 있는지 보여 주고 결과적으로 시스템이 어떤 기능을 제공하는지 알 수 있도록 작성한다.

표 2-5 설계단계 산출물

문서명	내 용
클래스 설계서	• 유즈케이스별로 시퀀스도를 작성하여 도출된 객체 및 클래스와 그 연관관계를 표현하고 클래스명, 속성 및 오퍼레이션, 각 클래스의 상세한 명세가 기술된 문서를 말한다.
클래스 다이어그램	• 시스템 요구사항에서 식별된 기능을 처리하는 클래스와 인터페이스 및 이들 의 연관관계를 표현하여 시스템의 논리적인 구조(클래스)를 이해할 수 있도록 한다.
시퀀스 다이어그램	• 여러 객체들이 다른 객체들과 어떻게 교류하는지, 프로그램이 작동할 때 어떤 메서드가 어떤 순서로 실행되는지를 표현하며 상호작용하는 오브젝트 간 메시지 시퀀스를 설명하는 UML(Unified Modeling Language) 다이어그램이다.
인터페이스 명세서	• 시스템 내·외부 연계 인터페이스를 식별하고 Source와 Target간 인터페이스 방식, 연계 주기, 관련 테이블 속성 상세 내역을 기술한다.
사용자 인터페이스 설계서 (화면설계서)	• 애플리케이션이 제공하는 사용자 화면의 전체 구조와 메뉴 형식, 화면 목록과 화면의 상세 설계 내역 이 기술된 문서이다.

문서명	내 용
아키텍처 설계서	• 개발대상 시스템에 대한 애플리케이션 구조와 시스템 환경 등 시스템의 구성 요소를 정의하고, 컴포넌트간에 상호작용하는 관계 및 가시적인 속성을 표현 한다. 설계항목에 대한 적용방안은 이 설계 문서에서 비기능 요구사항에 대한 설계로 포함될 수 있다.
엔티티관계 모형 기술서 (ERD)	• 문서로 서술된 요구사항에 대한 업무분석으로 도출된 엔티티와 엔티티간의 관계를 이해하기 쉽게 다이어그램으로 표시한 문서로서 해당 업무에서 데이터 흐름과 프로세스간의 연관성을 표현 하는 산출물을 말한다.
데이터베이스 설계서 (테이블설계서)	• 애플리케이션 요구사항을 만족시키기 위한 주요 데이터와 테이블을 정의하고 각 테이블의 용도, 컬럼 등 상세 내역을 기록한 문서를 말한다.
단위테스트 계획서	• 개발 완료된 시스템의 컴포넌트, 사용자인터페이스를 대상으로 단위테스트 대상을 선정하고, 단위 테스트 대상별 기능 수행 단위로 테스트 케이스와 테스트 절차, 테스트 데이터 등 상세 테스트 내용 을 작성한다.
통합테스트 계획서	• 개발 결과물에 대해 결함을 찾아내고 요구사항의 충족 테스트 수행을 위해 단계별 테스트 일정, 방법 및 환경 등 전반적인 계획을 수립한다.
데이터전환 및 초기데이터 설계서	• 신속하고 안정적인 데이터 이행을 위해 체계적인 절차를 수립하여 구축시 발생할 문제점을 사전에 차단할 수 있도록 데이터 전환 또는 초기화 계획을 수립한다.
개발 가이드	• 프로그램 구현시 개발자가 지켜야 할 Coding, Comment, Indentation Rule 및 보안약점을 제거하고 안전한 코드를 작성하기 위한 코딩 규칙을 정의한다. 개발팀 전체가 준수하는 코딩 규칙으로 소스 코드의 가독성, 이식성, 신뢰성 및 보안성을 향상하기 위한 것으로서 개발 및 향후 시스템 유지보수를 수월하게 하며 시스템 보안성을 확보하는 데 목적이 있다. 코딩 표준으로 보다 좋은 품질의 소스 코드를 작성할 수 있도록 한다.

표 2-6 CBD 개발방법론 표준산출물과 타 개발방법론에서 사용되는 산출물 매핑

단계	코드	CBD개발방법론 표준 산출물명	다른 방법론의 산출물명	범정부 EA
분석	R1	사용자 요구사항 정의서	상위 요구사항 정의서 과업대비표	
	R2	유즈케이스 명세서	요구사항 기술서 현행 시스템분석서	AV2응용시스템관계도
	R3	요구사항 추적표		
설계	D1	클래스 명세서	상세 클래스 설계서	
	D2	사용자 인터페이스 설계서	사용자화면정의서 화면설계서 메뉴구성도 인터페이스 상호작용 명세서	AV3응용기능분할도
	D3	컴포넌트 설계서	프로그램 명세서 컴포넌트 명세서 인터페이스 명세서	
	D4	인터페이스 설계서		
	D5	아키텍처 설계서	소프트웨어 아키텍처 정의서 시스템 구성도	TV2기반구조관계도
	D6	총괄시험 계획서		
	D7	시스템시험 시나리오		
	D8	엔티티 관계 모형 설계서	데이터모형설계서	DV4논리데이터모델
	D9	데이터베이스 설계서		DV6물리데이터모델
	D10	통합시험 시나리오		
	D11	단위시험 케이스		
	D12	데이터 전환 및 초기데이터 설계서		
구현	I1	프로그램 코드	프로그램 소스 프로그램 목록	TV3기반구조설계
	I2	단위시험 결과서		
	I3	데이터베이스 테이블		
시험	T1	통합시험 결과서		
	T2	시스템시험 결과서		
	T3	사용자 지침서		
	T4	운영자 지침서		
	T5	시스템 설치 결과서		
	T6	인수시험 시나리오		
	T7	인수시험 결과서		

4. 구현단계 진단절차

구현단계에서 개발자는 자신이 만들고 있는 프로그램의 소스코드에서 SW보안약점이 발생하지 않도록 코드를 작성해야 한다. 일부 프로젝트에서는 소스코드 보안약점 진단도구를 통합개발환경 (IDE)에 설치하여 초기 개발 단계에서부터 실시간으로 소스코드 보안약점 진단을 수행하면서 개발하기도 하지만, 대부분 소스코드 작성이 완료되는 시점에 진단을 수행하며, 이로 도출된 보안약점과 관련된 소스코드를 개선한 뒤 수정된 소스코드를 대상으로 다시 진단하여 보안약점 제거여부를 확인한다.

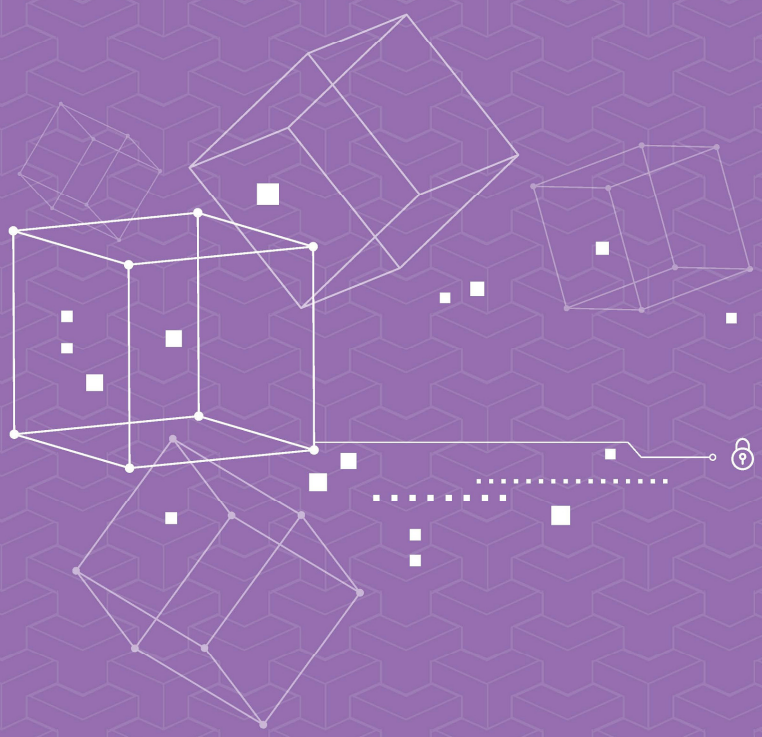
구현단계 보안약점 진단은 CC인증을 받은 SW보안약점 진단도구를 활용해서 진단을 수행할 수 있으며, 미탐될 수 있는 보안약점을 최소화하기 위해 수동진단을 병행하여 수행하는 것이 적절하다.

구현단계 진단 절차는 1차 진단 → 보안약점 제거 수행 → 2차 진단 → 종료 순서대로 진행된다.

1차 진단 수행에서 작성되는 진단보고서에는 보안약점이 검출된 파일명, 해당 라인번호, 보안약점명, 원인설명, 대응방법 설명이 포함되어야 한다.

2차 진단 수행에서 작성되는 진단보고서에는 1차 진단에서 검출되었던 보안약점에 대한 제거 여부를 추가한다.

구현단계 진단 절차의 종료단계에서는 1차와 2차에 걸쳐서 진단된 결과와 보안약점 제거 여부를 요약하여 보고함으로써 진단 절차가 종료된다.



소프트웨어 보안약점 진단가이드

KOREA INTERNET & SECURITY AGENCY

PART 3

제3장 분석·설계단계 보안항목 진단

PART 3

제3장 분석 · 설계단계 보안항목 진단

| 제 1 절 | 보안항목 식별

1. 정보에 대한 보안항목 식별

분석단계에서는 처리해야 하는 정보와 그 정보를 처리하는 기능에 대해 적용되어야 하는 보안항목들을 식별하는 작업이 우선적으로 수행되어야 한다. 정보에 대한 보안항목 식별은 권한을 가진 사용자만이 안전하게 수집, 전송, 처리, 보관, 폐기해야 하는 정보를 식별하는 것이다. 우리나라는 개인정보 보호법 등 다양한 법, 제도, 규정에 의해 보호되어야 하는 중요정보들을 정의하고 있으며, 각 기관은 내부정책자료, 외부정책자료 등을 기반으로 보안항목을 식별하고 있다. 내부정책 자료로는 기관 내 개인정보보호 규정, 정보보안 관련 규정 등이 있으며, 외부정책자료 개인정보보호법, 정보통신망법, 금융거래법 등 다양한 법, 법령, 관련지침 등으로 규정한다.

가. 외부환경 분석(법, 제도, 규정 등)으로 보안항목 식별

현재 시스템에서 처리하는 정보 중 중요정보로 분류되어 있는 정보의 대부분은 개인정보이며, 특정 IT기술과 관련된 규정도 존재한다.

표 3-1 개인정보보호 관련 법규

관련법규	주요내용
개인정보 보호법	• 개인정보의 처리 및 보호에 관한 사항을 규정
정보통신망 이용촉진 및 정보보호 등에 관한 법률	• 정보통신망으로 수집, 처리, 보관, 이용되는 개인정보의 보호에 관한 규정
신용정보의 이용 및 보호에 관한 법률	• 개인 신용정보의 취급 단계별 보호조치 및 의무사항에 관한 규정
위치정보의 보호 및 이용 등에 관한 법률	• 개인위치정보 수집, 이용, 제공 파기 및 정보주체의 권리 등 규정
표준 개인정보보호 지침 (표준지침)	• 「개인정보 보호법」 제12조제1항에 따른 개인정보의 처리에 관한 기준, 개인정보 침해의 유형 및 예방조치 등에 관한 세부적인 사항을 규정
개인정보의 안전성 확보 조치 기준 고시	• 「개인정보 보호법」 제23조제2항, 제24조제3항 및 제29조와 같은 법 시행령 제21조 및 제30조에 따라 개인정보처리자가 개인정보를 처리함에 있어서 개인정보가 분실·도난·유출·위조·변조 또는 훼손되지 아니하도록 안전성 확보에 필요한 기술적·관리적 및 물리적 안전조치에 관한 최소한의 기준을 규정
개인정보 영향평가에 관한 고시	• 「개인정보 보호법」 제33조와 같은 법 시행령 제38조에 따른 평가기관의 지정 및 영향평가의 절차 등에 관한 세부기준 규정

표 3-2 특정IT 기술관련 규정

관련 지침	주요내용
개인정보의 기술적·관리적 보호조치 기준 해설서	• 「개인정보 보호법」 제29조와 같은 법 시행령 제48조의2제3항에 따라 정보통신서비스제공자 등이 이용자의 개인정보를 처리함에 있어 안전성 확보를 위하여 필요한 보호조치의 기준에 대한 해설
자동처리 되는 개인정보 보호 가이드라인	• IoT 기기 등으로 개인정보를 자동처리 할 경우, 기획 단계부터 개인정보 침해 가능성을 충분히 고려하도록 처리 단계별 고려사항을 사례중심으로 안내
온라인 개인정보 처리 가이드라인	• 정보통신서비스제공자의 개인정보 수집, 동의, 파기, 열람, 보존 시 개인정보보호 조치 기준 ※ 2020년 8월 통합 「개인정보 보호법」 시행으로 종전 정보통신망법 상의 '정보통신서비스 제공자등'의 개인정보 처리에 대한 규정이 법 제6장의 특례로 이관됨
개인정보 위험도 분석 기준 및 해설서	• 「개인정보 보호법」 제24조제3항, 제29조와 같은 법 시행령 제30조에 따른 「개인정보의 안전성 확보조치 기준」에 따라 내부망에 고유식별정보를 저장하는 경우 암호화의 적용여부 및 적용범위 결정을 위한 위험도 분석 기준 제시
바이오정보 보호 가이드라인	• 지문, 홍채 등 생체 정보 수집, 이용 시 개인정보보호 조치 사항
위치정보의 관리적·기술적 보호조치 해설서	• 「위치정보의 보호 및 이용 등에 관한 법률」 제16조 및 같은 법 시행령 제20조에 따라 위치정보사업자 및 위치기반서비스사업자가 취하여야 하는 관리적·기술적 보호조치 기준에 대한 권고

나. 기타 중요정보 식별

시스템이 처리하는 정보 중 법적 의무사항으로 필수적인 안전조치를 해야 하는 정보 이외에도 물리적, 단 기술적 접근 허용범위와 정보유출시 예상되는 피해나 정보의 자산가치를 기준으로 중요정보를 식별하고 법적 의무사항에 준하는 보안강도를 적용하여 정보가 안전하게 처리될 수 있도록 설계되어야 한다.

2. 기능에 대한 보안항목 식별

분석단계에서는 정보처리시스템의 각 기능들을 안전하게 서비스하기 위해 필요한 설계보안사항들을 식별할 수 있어야 한다. 분석단계의 산출물인 요구사항 정의서에 다음과 같은 설계보안사항을 정의하여 설계, 구현, 테스트 단계에 적용될 수 있도록 한다.

가. 입력데이터 검증 및 표현

사용자와 프로그램의 입력 데이터에 대한 유효성검증* 체계를 갖추고, 유효하지 않은 값에 대한 처리 방법 설계

* 유효성검증(Validation) : 데이터가 특정 요구사항을 충족했다는 것을 확인하여 의도치 않는 동작 방지

번호	항목명	설명	비고
SR1-1	DBMS 조회 및 결과 검증	• DBMS 조회시 질의문(SQL) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법 설계	입·출력값 검증
SR1-2	XML조회 및 결과 검증	• XML 조회시 질의문(XPath, XQuery 등) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-3	디렉토리 서비스 조회 및 결과 검증	• 디렉토리 서비스(LDAP 등)를 조회할 때 입력값과 그 조회결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-4	시스템 자원 접근 및 명령어 수행 입력값 검증	• 시스템 자원접근 및 명령어를 수행할 때 입력값에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-5	웹 서비스 요청 및 결과 검증	• 웹 서비스(게시판 등) 요청(스크립트 게시 등)과 응답결과(스크립트를 포함한 웹 페이지)에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-6	웹 기반 중요 기능 수행 요청 유효성 검증	• 비밀번호 변경, 결제 등 사용자 권한 확인이 필요한 중요기능을 수행할 때 웹 서비스 요청에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-7	HTTP 프로토콜 유효성 검증	• 비정상적인 HTTP 헤더, 자동연결 URL 링크 등 사용자가 원하지 않은 결과를 생성하는 HTTP 헤더·응답결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-8	허용된 범위내 메모리 접근	• 해당 프로세스에 허용된 범위의 메모리 버퍼에만 접근하여 읽기 또는 쓰기 기능을 하도록 검증방법 설계 및 메모리 접근 요청이 허용범위를 벗어났을 때 처리방법 설계	
SR1-9	보안기능 입력값 검증	• 보안기능(인증, 권한부여 등) 입력 값과 함수(또는 메소드)의 외부입력 값 및 수행결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
SR1-10	업로드·다운로드 파일 검증	• 업로드·다운로드 파일의 무결성, 실행권한 등에 관한 유효성 검증방법 설계 및 부적합한 파일에 대한 처리방법 설계	파일 관리

나. 보안기능

인증, 접근통제, 권한관리, 비밀번호 등의 정책이 적절하게 반영될 수 있도록 설계

번호	항목명	설명	비고
SR2-1	인증 대상 및 방식	• 중요정보·기능의 특성에 따라 인증방식을 정의하고 정의된 인증방식을 우회하지 못하게 설계	인증 관리
SR2-2	인증 수행 제한	• 반복된 인증 시도를 제한하고 인증 실패한 이력을 추적하도록 설계	
SR2-3	비밀번호 관리	• 생성규칙, 저장방법, 변경주기 등 비밀번호 관리정책별 안 전한 적용방법 설계	
SR2-4	중요자원 접근통제	• 중요자원(프로그램 설정, 민감한 사용자 데이터 등)을 정의하고, 정의된 중요자원에 대한 신뢰할 수 있는 접근통제 방법(권한관리 포함) 설계 및 접근통제 실패 시 처리방법 설 계	권한 관리
SR2-5	암호키 관리	• 암호키 생성, 분배, 접근, 파기 등 암호키 생명주기별 암호키 관리 방법을 안전하게 설계	암호화
SR2-6	암호연산	• 국제표준 또는 검증필 암호모듈로 등재된 안전한 암호 알고리즘을 선정하고 충분한 암호키 길이, 솔트, 충분한 난수 값을 적용한 안전한 암호연산 수행방법 설계	
SR2-7	중요정보 저장	• 중요정보(비밀번호, 개인정보 등)를 저장·보관하는 방법이 안전하도록 설계	중요 정보 처리
SR2-8	중요정보 전송	• 중요정보(비밀번호, 개인정보, 쿠키 등)를 전송하는 방법이 안전하도록 설계	

다. 에러처리

에러 또는 오류상황을 처리하지 않거나 불충분하게 처리되어 중요정보 유출 등 보안약점이 발생하지 않도록 설계

번호	항목명	설명	비고
SR3-1	예외처리	오류메시지에 중요정보(개인정보, 시스템 정보, 민감 정보 등)가 노출되거나, 부적절한 에러·오류 처리로 의도치 않은 상황이 발생하지 않도록 설계	에러 처리

라. 세션통제

다른 세션간 데이터 공유 금지 등 세션을 안전하게 관리할 수 있도록 설계

번호	항목명	설명	비고
SR4-1	세션통제	다른 세션 간 데이터 공유금지, 세션ID 노출금지, (재)로그인시 기존 세션 ID 재사용금지 등 안전한 세션 관리방안 설계	세션 통제

3. 구현단계 기준과의 관계

설계단계 보안설계 기준(20개)과 구현단계 보안약점 제거 기준의 각 항목별 연관 관계는 다음과 같다.

구분	설계단계	구현단계
입력 데이터 검증 및 표현 (10개)	DBMS 조회 및 결과 검증	• SQL 삽입
	XML 조회 및 결과 검증	• XML 삽입 • 부적절한 XML 외부개체 참조
	디렉토리 서비스 조회 및 결과 검증	• LDAP 삽입
	시스템 자원 접근 및 명령어 수행 입력값 검증	• 코드 삽입 • 경로 조작 및 자원 삽입 • 서버사이드 요청 위조 • 운영체제 명령어 삽입
	웹 서비스 요청 및 결과 검증	• 크로스사이트 스크립트
	웹 기반 중요 기능 수행 요청 유효성 검증	• 크로스사이트 요청 위조
	HTTP 프로토콜 유효성 검증	• 신뢰되지 않는 URL 주소로 자동접속 연결 • HTTP 응답분할
	허용된 범위내 메모리 접근	• 포맷 스트링 삽입 • 메모리 버퍼 오버플로우
	보안기능 입력값 검증	• 보안기능 결정에 사용되는 부적절한 입력값 • 정수형 오버플로우 • Null Pointer 역참조
	업로드·다운로드 파일 검증	• 위험한 형식 파일 업로드 • 부적절한 전자서명 확인 • 무결성 검사 없는 코드 다운로드
보안 기능 (8개)	인증 대상 및 방식	• 서버사이드 요청 위조 • 적절한 인증 없는 중요기능 허용 • 부적절한 인증서 유효성 검증 • DNS lookup에 의존한 보안결정
	인증 수행 제한	• 반복된 인증시도 제한 기능 부재
	비밀번호 관리	• 하드코드된 중요정보 • 취약한 비밀번호 허용
	중요자원 접근통제	• 부적절한 인가 • 중요한 자원에 대한 잘못된 권한 설정
	암호키 관리	• 하드코드된 중요정보 • 주석문 안에 포함된 시스템 주요정보
	암호연산	• 취약한 암호화 알고리즘 사용 • 충분하지 않은 키 길이 사용 • 적절하지 않은 난수 값 사용 • 부적절한 인증서 유효성 검증 • 솔트 없이 일방향 해쉬 함수 사용
	중요정보 저장	• 암호화되지 않은 중요정보 • 사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출
	중요정보 전송	• 암호화되지 않은 중요정보
에러 처리 (1개)	예외처리	• 오류 메시지 정보노출
세션 통제 (1개)	세션통제	• 잘못된 세션에 의한 데이터 정보 노출

구현단계 49개 보안약점 제거 기준을 도표로 표시해 보면 대부분의 기존 시큐어코딩 기준들은 설계단계에서부터 고려되어야 함을 확인할 수 있으며, 일부 항목들은 개발단계에 코딩 규칙을 준수하는 것만으로도 보안 취약점들을 제거할 수 있다.

입력 데이터 검증 및 표현 | 보안기능 | 에러처리 | 세션통제

입력 데이터 검증 및 표현 (17개)	SQL 삽입	코드 삽입	경로 조작 및 자원 삽입	크로스사이트 스크립트	운영체제 명령어 삽입	위험한 형식 파일 업로드
	신뢰되지 않은 URL 주소로 자동 접속 연결	부적절한 XML 외부개체 참조	XML 삽입	LDAP 삽입	크로스사이트 요청 위조	서버사이드 요청 위조
	HTTP 응답 분할	정수형 오버플로우	보안기능 결정에 사용되는 부적절한 입력값	메모리 버퍼 오버플로우	포맷 스트링 삽입	
보안 기능 (16개)	적절한 인증 없는 중요 기능 허용	부적절한 인가	중요한 자원에 대한 잘못된 권한 설정	취약한 암호화 알고리즘 사용	암호화되지 않은 중요정보	하드코드된 중요정보
	충분하지 않은 키 길이 사용	적절하지 않은 난수값 사용	취약한 비밀번호 허용	부적절한 전자서명 확인	부적절한 인증서 인증서 유효성 검증	사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출
	주석문 안에 포함된 시스템 주요정보	솔트 없이 일방향 해쉬 함수 사용	무결성 검사 없는 코드 다운로드	반복된 인증시도 제한 기능 부재		
시간 및 상태 (2개)	경쟁조건: 검사 시점과 사용 시점(TOCTOU)	종료되지 않은 반복문 재귀함수				
에러처리 (3개)	오류메시지 정보 노출	오류 상황 대응 부재	부적절한 예외 처리			
코드오류 (5개)	Null Pointer 역참조	부적절한 자원해제	해제된 자원 사용	초기화되지 않은 변수 사용	신뢰할 수 없는 데이터의 역직렬화	
캡슐화 (4개)	잘못된 세션에 의한 데이터 정보노출	제거되지 않고 남은 디버그 코드	Public 메소드부터 반환된 Private 배열	private 배열에 public 데이터 할당		
API 오용 (2개)	DNS Lookup에 의존한 보안 결정	취약한 API 사용				

| 제 2 절 | 설계단계 보안설계 적용 방법

설계단계에서는 개발하고자 하는 소프트웨어가 가질 수 있는 보안 취약성은 무엇이며, 공격자가 이를 어떻게 이용할 수 있는가에 대해 고려하여, 이러한 취약성이 생기지 않도록 설계해야 한다.

잘 알려진 위협들을 방어하기 위해 제작 초기부터 안전하게 설계할 수 있도록 미들웨어 프레임워크를 도입하거나 공통 라이브러리를 구축하는 것도 권장하는 방법이다. 예를 들어 입력에 대한 유효성 검사 요건을 보다 쉽게 충족시킬 수 있도록 Spring과 같이 널리 사용되는 웹 프레임워크를 커스터 마이징 하여 사용하는 것이다. 하지만 안전하지 않은 미들웨어 프레임워크(또는 기타 널리 사용되는 소프트웨어)는 보안상황을 더 악화시킬 수 있으므로, 사용할 때는 반드시 보안관련 부분을 주의 깊게 검토해야 한다.

개발가이드에서는 정보처리시스템 구축시 우선적으로 고려되어야 하는 설계항목들을 정의하고 있다. 이 설계항목들을 설계단계에서 어떻게 반영할 것인가에 대한 계획수립이 요구되며, [표 3-3]의 요구사항정의서나 [표 3-5]의 아키텍처설계서와 같은 기존 분석, 설계 산출물을 활용하여 미들웨어 프레임워크 또는 공통라이브러리 구축과 같은 시스템의 구조적인 보안설계로 보안을 강화하거나 개발자들이 시큐어코딩을 적용할 수 있도록 표준코딩정의서 또는 시큐어코딩 가이드를 문서화하는 작업을 설계단계에 수행할 수 있도록 계획한다.

표 3-3 요구사항정의서

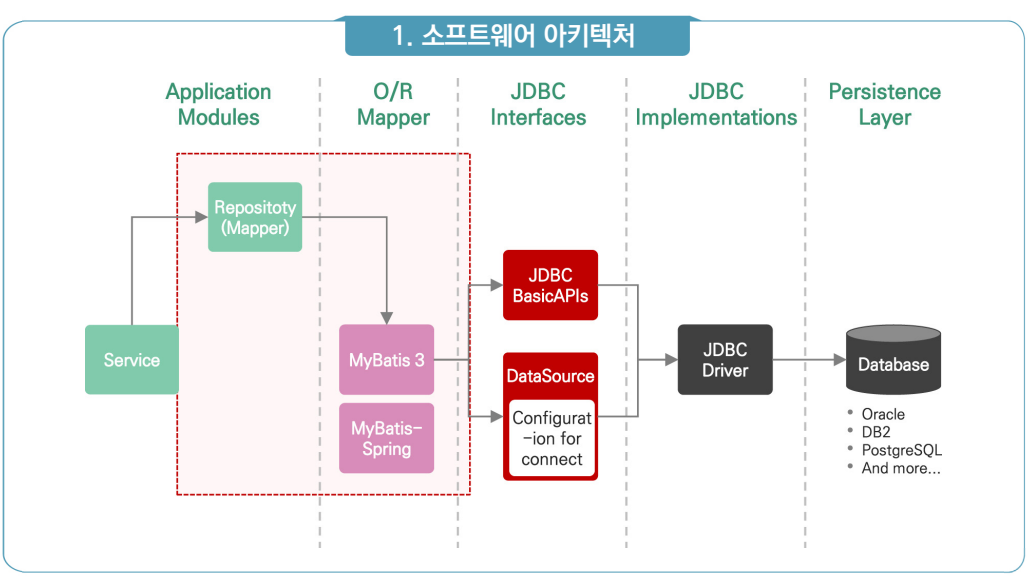
요구사항 ID	요구 사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요도	해결방안	검수기준	비고
SR-0101101	DB 사용자 계정에 최소권한 부여	비 기능	애플리케4이션에서 사용하는 DB 사용자 계정에 대해 최소권한 이 사용될 수 있도록 해야 한다.	RFP 보안요구사항 1.1	없음	상	애플리케이션에서 사용하는 DB 사용자 계정은 애플리케이션에서 사용하는 테이블, 뷰, 프로시저에 대해서만 사용 권한을 부유여하다.	애플리케이션에서 사용하는 DB 사용자 계정은 해당 애플리케이션에서 사용하는 테이블, 뷰, 프로시저에 대해서만 사용 권한이 적용 된다.	지속적인 DB 사용자 계정 관리가 필요
SR-010102	정적 SQL 사용	비 기능	모든 쿼리는 정적으로 실행되도록 한다.	RFP 보안요구사항 1.1	없음	상	외부 입력값을 Query Map에 바인딩할 경우에는 #을 이용하도록 개발 가이드에 명시하고, 개발보안 교육으로 개발자에게 전파하여 지켜질 수 있도록 한다.	동적 SQL 실행이 존재 하지 않는다.	정적분석으로 지속적인 검사가 필요

요구사항 ID	요구 사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요 도	해결방안	검수기준	비고
SR-010103	동적 SQL 사용시 입력값 검증	비 기능	동적으로 SQL 문이 생성, 실행되어야 하는 경우, 반드시 입력값 검증 후 사용되어야 한다.	RFP 보안 요구 사항 1.1	없음	상	아키텍처 정의에 따라 모든 쿼리는 MyBatis 프레임 워크의 쿼리맵으로 정의, 실행한다. 이때, 변수 바인딩은 SR 010102에 따라 # 기호를 이용한다.	해당 없음	SR-010102로 통합

표 3-4 요구사항추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		...	아키텍처정의서	개발 가이드	...	...	...
요구사항ID	요구사항 명						
SR-010101	DB 사용자 계정에 최소 권한 부여			1.1.1			
SR-010102	정적 SQL 사용		소프트웨어 아키텍처 아키텍처 요구사항 및 구현방안 SR-010102	1.1.2			
SR-010103	동적 SQL 사용시 입력값 검증		소프트웨어 아키텍처 아키텍처 요구사항 및 구현방안 SR-010102	1.1.3			

표 3-5 아키텍처설계서



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010102
요구사항 내용	
동적으로 SQL문이 생성, 실행되지 않도록 해야 한다.	
구현방안	
SQL 삽입 취약점을 방어할 수 있도록 외부 또는 사용자 입력값을 MyBatis의 쿼리맵에 바인딩하는 경우, 반드시 “#” 기호를 이용하여 정의하도록 한다. 만약. “$” 기호를 사용하는 경우에는 파라미터로 전달되는 값이 해당 애플리케이션에서 정의한 상수 또는 고정된 값인 것을 보장해야 한다.	

설계자는 구현하고자 하는 기능을 설명하기 위해 유즈케이스 다이어그램, 플로우차트, DFD(데이터 흐름 다이어그램)와 같은 그림을 그린다. 이러한 기능설명을 하는 산출물을 기준으로 보안 요구항목을 적용하여 시스템이 구현하고자하는 각각의 기능들이 안전하게 동작될 수 있도록 설계단계에서 보안이 고려되도록 한다.

| 제 3 절 | 설계단계 보안설계 기준

1. 입력데이터 검증 및 표현

1.1 DBMS 조회 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	DBMS 조회 및 결과 검증
설명	DBMS 조회시 질의문(SQL) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다. ② 외부입력값이 삽입되는 SQL질의문을 동적으로 생성해서 실행하지 않도록 해야 한다. ③ 외부입력값을 이용해 동적으로 SQL질의문을 생성해야 하는 경우, 입력값에 대한 검증을 수행한 뒤 사용해야 한다.

가. 취약점 개요

데이터베이스(DB)와 연동된 응용프로그램에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력데이터에 SQL 질의문을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안 취약점을 말한다.

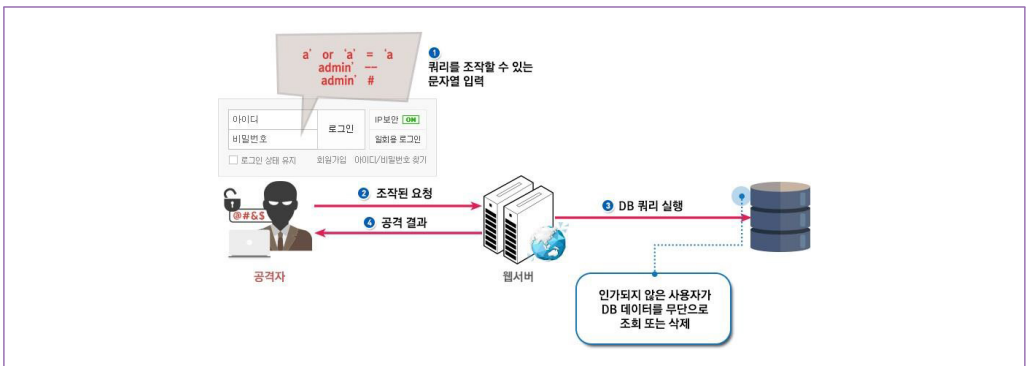


그림 3-1 | SQL 삽입 취약점

[그림 3-1]에서 처럼 공격자는 쿼리를 조작할 수 있는 문자열을 입력하여 조작된 요청을 보낸다. 서버가 입력값을 검증하지 않고 DB 쿼리 실행에 사용하는 경우 인가되지 않는 사용자가 DB 데이터를 무단으로 조회 또는 삭제할 수 있다.

나. 설계시 고려사항

① 애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다.

취약한 애플리케이션으로 인해 침해사고가 발생하더라도 나머지 부분에 대해 공격자가 액세스 권한을 가지지 않도록 애플리케이션에서 사용하는 DB연결 계정은 해당 애플리케이션이 사용하는 데이터에 대한 읽기, 쓰기, 삭제, 업데이트 권한만 설정한다.

② 외부 입력값이 삽입되는 SQL질의문을 동적으로 생성해서 실행하지 않도록 해야 한다.

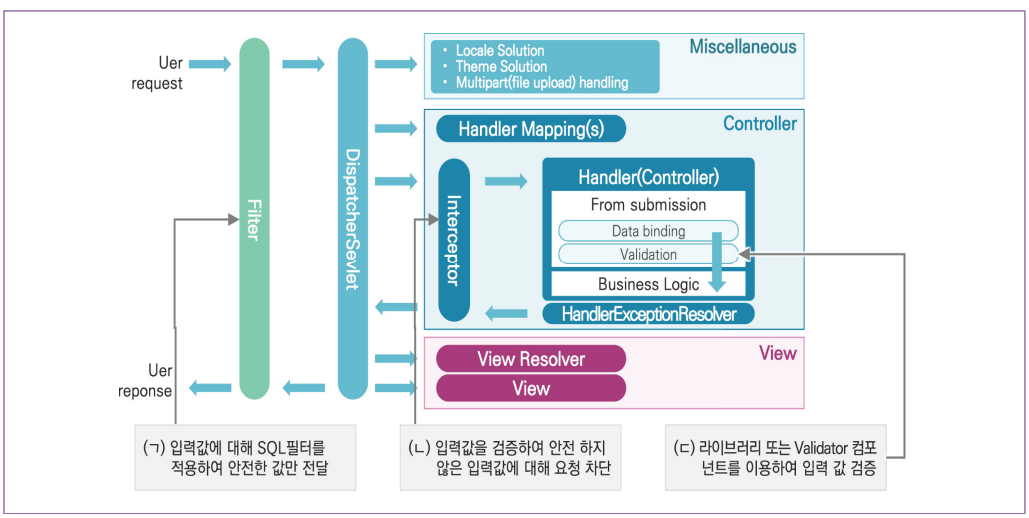
SQL 질의문의 구조가 외부 입력값에 의해 변경되지 않는 API를 사용하도록 시큐어코딩 규칙을 지정 한다. ORM프레임워크를 사용하여 안전한 정적쿼리구조로 SQL문을 수행할 수 있도록 개발환경을 설정하고, ORM프레임워크에서 제공하는 함수를 활용하여 외부 입력값에 의해 SQL 질의문의 구조가 변경되지 않도록 한다.

③ 외부 입력값을 이용해 동적으로 SQL질의문을 생성해야 하는 경우, 입력값에 대한 검증을 수행한 뒤 사용해야 한다.

클라이언트와 서버 양측에서 입력값에 대해 안전한 값만 사용될 수 있도록 검증작업을 수행한다.

(ㄱ) 필터를 이용한 입력값 검증

외부입력값에서 SQL삽입이 가능한 문자열들을 필터링하여 안전한 값으로 치환하도록 하는 Filter 컴포넌트를 생성하고, DB에서 관리하는 데이터를 처리하는 모든 애플리케이션에 일괄 적용한다.



(ㄴ) 인터셉트를 이용한 입력값 검증

MVC프레임워크를 사용하는 경우 Interceptor 컴포넌트를 사용하여 입력값에 대한 검증 작업을 수행한 뒤 요청을 차단하거나 허용하는 정책을 애플리케이션에 일괄 적용 한다.

(ㄷ) 라이브러리 또는 Validator 컴포넌트를 이용한 입력값 검증

입력값을 검증하는 Validator 컴포넌트를 공통코드로 생성하고, 모든 개발자가 SQL질의문에 삽입되는 입력값에 대해 검증작업을 해당 컴포넌트에서 수행하도록 시큐어코딩 규칙을 정의한다.

SQL삽입 취약점을 최소화하기 위해 SQL문을 안전하게 처리할 수 있도록 [표 3-6]와 같은 프레임워크나 라이브러리의 사용을 고려할 수 있다.

표 3-6 SQL삽입 취약점 대응 프레임워크 및 라이브러리

개발환경	활용 가능한 프레임워크 또는 라이브러리
Java	Hibernate : http://hibernate.org/ 자바언어를 위한 객체관계매핑(ORM) 프레임워크. 객체지향도메인 모델을 관계형 데이터베이스로 매칭하기 위한 프레임워크로 GNU LGPL[1] v2.1로 배포되는 자유소프트웨어이다.
	MyBatis : http://blog.mybatis.org/ 객체지향언어인 자바의 관계형 데이터베이스 프로그래밍을 좀 더 쉽게 도와주는 개발 프레임워크이다. Apache Software License v2.0[2] 정책에 따라 자유롭게 사용가능하다.
	JPA(Java PersistenceAPI) : https://jcp.org/aboutJava/communityprocess/final/jsr220/index.html 관계형 데이터베이스에 접근하기 위한 표준 ORM 기술을 제공하며, 자바플랫폼, 표준 에디션 및 자바플랫폼, 엔터프라이즈 에디션을 사용하는 애플리케이션에서의 관계형데이터 관리를 목적으로 하는 프로그래밍 인터페이스다.
ASP.NET (확장가능)	AntiSQLi 라이브러리: http://ironbox.github.io/AntiSQLi/ SQL인젝션의 위협이 있는 데이터를 자동으로 파라미터화하는 오픈소스 라이브러리이다. BSD License[3]에 따라 사용할 수 있다.
PHP	MeekroDB 라이브러리: http://meekro.com/ SQL인젝션 방어를 위한 PHP-MySQL 오픈소스 라이브러리이며, LGPLv3 License에 따라 사용가능하다.
	HTML Purifier 라이브러리 : http://htmlpurifier.org/ XSS, SQL인젝션 방어를 제공하는 화이트리스트 구현방식의 HTML 필터 라이브러리이다. LGPL v2.1+에 따라 사용가능하다.

1) LGPL(Lesser General Public License)은 LGPL로 작성된 소스코드를 라이브러리(정적, 동적)로만 사용하는 경우에는 소스코드를 공개하지 않아도 된다. 그 외의 경우에는 수정한 소스코드 또는 GPL 소스코드를 활용한 소프트웨어 모두를 GPL로 공개해야 한다. 상업적 목적으로 사용가능하며 배포, 수정이 가능하고 이 때 라이선스 및 저작권, 변경사항을 명시해야 한다.

2) Apache Software License 2.0은 누구든 자유롭게 아파치 소프트웨어를 다운 받아 부분 혹은 전체를 개인적 혹은 상업적

다. 진단 세부사항

애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다.

애플리케이션에서 사용할 계정이 최소권한을 가지도록 명시하고 있는지 여부, 진단 시점에 개발서버가 구축되고 개발서버에서 사용할 DB가 생성되어 있는 경우에는 실제 DB 접속 계정의 권한을 확인한다.

요구사항 ①

애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다.

애플리케이션에서 사용할 계정이 최소권한을 가지도록 명시하고 있는지 여부, 진단 시점에 개발서버가 구축되고 개발서버에서 사용할 DB가 생성되어 있는 경우에는 실제 DB 접속 계정의 권한을 확인한다.

진단기준

1. 애플리케이션별 DB접속 계정이 할당되고, 각 계정의 권한이 최소권한으로 설정되어 있는가?
2. 설정된 사용자 권한 외의 요청에 대해 차단되는지에 대한 테스트 계획이 수립되어 있는가?

진단방법

1. 애플리케이션별 DB접속 계정이 할당되고, 각 계정의 권한이 최소권한으로 설정되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처설계서 검토
1-2 애플리케이션별 DB연결 계정할당과 해당계정에 대한 최소 권한 할당 여부 확인	아키텍처설계서, 초기데이터설계서 검토

2. 설정된 사용자 권한 외의 요청에 대해 차단되는지에 대한 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 DB연결 계정의 권한 외의 요청이 차단되는지를 점검할 수 있는 테스트 계획 수립이 수립되어 있는지 확인	단위테스트케이스 검토

목적으로 이용할 수 있으며 재배포시에는 원본소스코드 또는 수정한 소스 코드를 반드시 포함시키지 않아도 되지만 아파치 라이센스 버전 2.0을 포함시켜야 하며 아파치 소프트웨어 재단에 개발된 소프트웨어라는 것을 명확하게 밝혀야 한다.

3) BSD License는 자유소프트웨어 저작권의 일종으로써, 해당 소프트웨어는 누구나 부분 혹은 전체적으로 수정가능하고 제한 없이 재배포할 수 있다. 수정한 소스코드는 의무적으로 재배포하지 않아도 되므로 상용 소프트웨어에도 사용할 수 있다.

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처 설계서, 초기데이터 설계서, 개발가이드, 단위테스트 케이스, 단위테스트 시나리오 등

요구사항 ②

외부입력값이 삽입되는 SQL문을 동적으로 생성해서 실행하지 않도록 해야 한다. 사용하는 플랫폼(JAVA, ASP.Net 등), 프레임워크에 맞는 정적 SQL을 작성하도록 명시하고 있는지 확인한다.

진단기준

1. DB 데이터 처리기능 구현시, 외부 입력값이 쿼리의 구조에 영향을 미치지 않도록 보안설계가 적용되어 있는가?
2. 입력값이 DB 쿼리의 구조를 변경시키는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법

1. DB 데이터 처리기능 구현시, 외부 입력값이 쿼리의 구조에 영향을 미치지 않도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 안전한 쿼리 실행환경을 제공할 수 있는 ORM 프레임워크와 같은 프레임 워크 사용여부확인	아키텍처설계서의 보안요구항목 적용 계획을 검토
1-3 개발가이드로 안전한 정적 쿼리 코딩 방법을 정의하고 있는 확인 - ibatis, mybatis 사용시 #{변수} 사용 정의 - PreparedStatement와 같은 정적쿼리를 수행하는 API 사용정의	개발가이드 검토

2. 입력값이 DB 쿼리의 구조를 변경시키는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 쿼리문의 구조를 변경할 수 있는 입력값이 쿼리 구조 변경에 영향을 주는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인 테스트 입력값 : 특수문자(‘ , “ , = , & , | , ! , (,) , { , } , $, % , @ 등) 예약어 (UNION, SELECT, THEN, IF, INSTANCE, END, COLUMN 등) 함수 명(DATABASE(), CONCAT(), COUNT(), LOWER() 등)	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트 케이스, 단위테스트 시나리오 등

요구사항 ③

외부입력값을 이용해 동적으로 SQL쿼리문을 생성해야 하는 경우, 입력값에 대한 검증을 수행한 뒤 사용해야 한다.

불가피하게 동적으로 쿼리문을 생성하고 외부입력값을 사용해야 하는 경우 개발자가 입력값 검증을 구현할 수 있도록 세부사항을 명시하고 있는지 확인한다.

진단기준

1. 외부 입력값을 이용한 동적 쿼리를 수행하는 기능 구현시, 입력값을 필터링하는 기능이 구현되어 있는가?
2. 입력값이 DB 쿼리의 구조를 변경시키는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법

1. 외부 입력값을 이용한 동적 쿼리를 수행하는 기능 구현시, 입력값을 필터링하는 기능이 구현되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 SQL필터링 기능이 설계되어 있거나 외부라이브러리를 사용하는지 확인	프로그램명세서, 클래스설계서에서 필터링기능 검토, 외부라이브러리 사용 진단을 위해 아키텍처설계서 검토
1-3 SQL필터링 적용방법이 공통 적용으로 정의되어 있는 경우, DB 접근을 수행하는 모든 기능에 적용되도록 설계되어 있는지 확인	아키텍처설계서의 보안요구항목 적용 계획을 검토
1-4 각각의 기능에서 SQL필터링을 적용하는 경우 SQL필터링을 적용하기 위한 코딩규칙이 개발가이드에 정의되어 있는지 확인	개발가이드의 코딩 규칙 검토

2. 입력값이 DB 쿼리의 구조를 변경시키는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 쿼리문의 구조를 변경할 수 있는 입력값이 쿼리 구조 변경에 영향을 주는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인 테스트 입력값 : 특수문자(‘ , “ , = , & , | , ! , (,) , { , } , $, % , @ 등) 예약어 (UNION, SELECT, THEN, IF, INSTANCE, END, COLUMN 등) 함수 명(DATABASE(), CONCAT(), COUNT(), LOWER() 등)	

• 관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 프로그램명세서, 클래스설계서, 개발가이드, 단위 테스트케이스, 단위테스트시나리오 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	SQL 삽입

마. 참고자료

① SDL Quick security references on SQL injection, Bala Neerumalla, http://go.microsoft.com/?linkid=9707610

② SQL Injection Prevention Cheat Sheet, OWASP, www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

③ CAPEC-66 SQL Injection, CAPEC, http://capec.mitre.org/data/definitions/66.html

④ CWE-89 SQL Injection, MITRE, http://cwe.mitre.org/data/definitions/89.html

⑤ 2016 OWASP Application Security Verification Standard, OWASP, Malicious Input Handling Verification Re- quirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Stan- dard_Project

1.2 XML 조회 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	XML 조회 및 결과 검증
설명	XML 조회시 질의문(XPath, XQuery 등) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등)과 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① XML문서를 조회하는 기능을 구현해야 하는 경우 XML질의문에 사용되는 파라미터는 반드시 XML쿼리를 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 질의문에 입력값을 자료형에 따라 바인딩해서 사용해야 한다.

가. 취약점 개요

XML 문서를 조회할 경우 입력값 조작으로 XQuery나 XPath와 같은 XML 질의문의 구조를 임의로 변경하여 허가되지 않은 데이터를 조회하거나 인증절차를 우회할 수 있다.

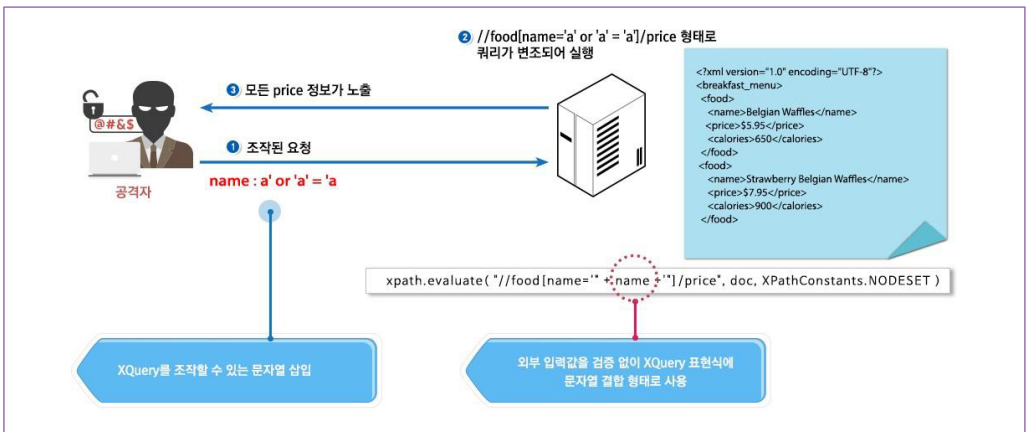


그림 3-2 | XML 삽입

나. 설계시 고려사항

① XML문서를 조회하는 기능을 구현해야 하는 경우 XML질의문에 사용되는 파라미터는 반드시 XML질의문을 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 질의문에 입력값을 자료형에 따라 바인딩해서 사용해야 한다.

(ㄱ) 공통 검증 컴포넌트를 이용한 입력값 필터링

외부입력값에서 XML삽입 공격이 가능한 문자열들을 필터링하는 Validator 컴포넌트를 개발하여 XML조회를 수행하는 애플리케이션 작성시 입력값에 대한 검증 작업이 일괄 적용되도록 설계한다.

(ㄴ) 필터 컴포넌트를 이용한 입력값 필터링

Filter컴포넌트에서 XML 삽입 공격에 활용될 수 있는 입력값(‘, [,], /, =, @)을 필터링하도록 작성하여 전체 요청 또는 XML필터링이 요구되는 요청에 대해 프레임워크에서 일괄 적용하도록 설계한다.

(ㄷ) 개별 코드에서 입력값 필터링하도록 시큐어코딩 규칙 정의

각각의 컴포넌트에서 입력값에 대해 XML삽입을 발생시킬 수 있는 문자열(‘, [,], /, =, @ 등)을 제거 또는 안전하게 치환하여 사용할 수 있도록 시큐어코딩 규칙을 정의한다.

(ㄹ) 안전한 API를 사용하도록 시큐어코딩 규칙 정의

XML 조회를 수행하는 질의문 작성 시 외부입력값이 질의문의 구조를 바꿀 수 없는 API(예. Java API- XQuery) 를 사용하도록 시큐어코딩 규칙을 정의한다.

다. 진단 세부사항

요구사항 ①

XML문서를 조회하는 기능을 구현해야 하는 경우 XML질의문에 사용되는 파라미터는 반드시 XML질의문을 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 질의문에 입력값을 자료형에 따라 바인딩해서 사용해야 한다.

XML 조회를 위한 질의문(XPath, XQuery 등) 생성 시 사용되는 입력값과 조회결과에 대한 검증방법(필터링 등)을 설계하고 유효하지 않은 값에 대한 처리방법이 명시되어 있는지 확인한다.

진단기준

1. 외부 입력값이 XML 데이터 조회에 사용되는 경우, 입력값이 조회 구문을 변경하지 않도록 보안설계가 적용되어 있는가?
2. XML삽입 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법

1. 외부 입력값이 XML 데이터 조회에 사용되는 경우, 입력값이 조회 구문을 변경하지 않도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 요구사항 추적표로 요구사항 추적여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 XML조회에 사용되는 외부 입력값을 안전하게 필터링 하는 기능이 설계되어 있거나 안전한 외부라이브러리를 사용하도록 설계되어 있는지 확인	프로그램명세서, 컴포넌트설계서, 유즈케이스설계서, 클래스설계서 등에서 필터링 기능 검토
1-3 XML데이터를 조회하는 기능 구현시, XML필터링을 적용하기 위한 코딩규칙 이나 안전하게 사용할 수 있는 API에 대한 설명이 개발가이드에 정의되어 있는지 확인	개발가이드에서 코딩규칙 검토

2. XML삽입 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 XML조회 구문을 변경할 수있는 입력값을 사용하여 XML조회 구문이 변경 되는지를 점검하는 테스트계획이 수립되어 있는지 확인 - 테스트 입력값: 쿼리예약어, ‘, [,], /, =, @ 등	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 프로그램명세서, 컴포넌트설계서, 유즈케이스설계서, 클래스설계서, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	XML 삽입
입력데이터 검증 및 표현	부적절한 XML 외부개체 참조

마. 참고자료

① “XML Path Language (XPath) Version 1.0” W3C Recommendation, 16 Nov 1999
http://www.w3.org/TR/xpath

② XQuery Injection, Common Attack Pattern Enumeration and Classification (CAPEC)
http://capec.mitre.org/data/definitions/84.html

③ “Blind XPath Injection”, Amit Klein
http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf

④ Failure to Sanitize Data within XPath Expressions(‘XPath injection’)
http://cwe.mitre.org/data/definitions/643.html

⑤ CWE-652 XQuery Injection, MITRE,
http://cwe.mitre.org/data/definitions/652.html

⑥ CWE-643 XPath Injection, MITRE,
http://cwe.mitre.org/data/definitions/643.html

⑦ 2016 OWASP Application Security Verification Standard, OWASP, Malicious Input Handling Verification Requirements,
http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

1.3 디렉토리 서비스 조회 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	디렉토리 서비스 조회 및 결과 검증
설명	디렉토리 서비스(LDAP 등)를 조회할 때 입력값과 그 조회결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계한다.
보안대책	① LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용되는 외부입력값은 LDAP 삽입 취약점을 가지지 않도록 필터링해서 사용해야 한다.

가. 취약점 개요

외부입력값이 LDAP[4] 조회를 수행하기 위한 필터 생성에 사용되는 경우 필터규칙을 변경할 수 있는 입력값에 대한 검증 작업을 수행하지 않게 되면 공격자가 의도하는 LDAP 조회가 수행될 수 있는 취약점이다.

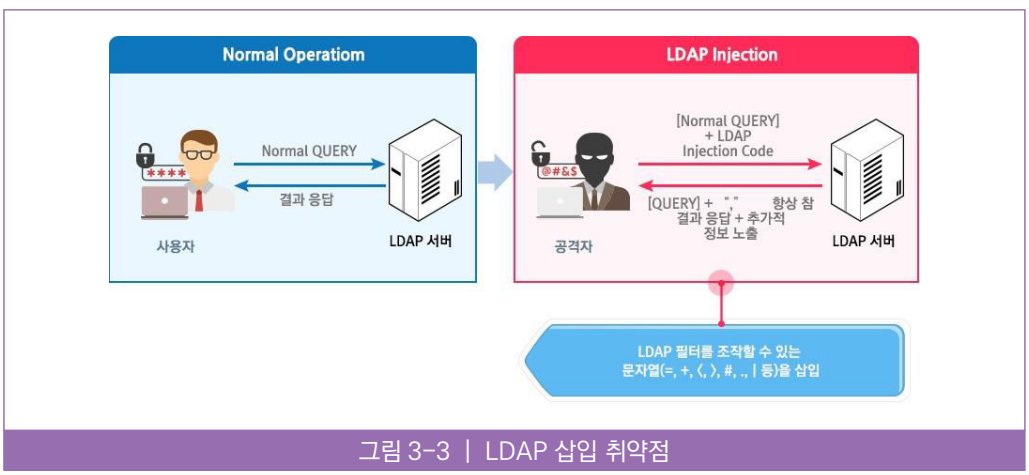


그림 3-3 | LDAP 삽입 취약점

4) LDAP(Lightweight Directory Access Protocol)은 TCP/IP 위에서 디렉토리 서비스를 조회하고 수정하는 응용 프로토콜이다.

나. 설계시 고려사항

① LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용되는 외부입력값은 LDAP 삽입 취약점을 가지지 않도록 필터링해서 사용해야 한다.

LDAP 인증이 포함되는 기능 설계시, 외부입력값이 LDAP 조회를 위한 검색 필터 생성에 삽입되어 사용되는 경우, 필터 규칙으로 인식 가능한 특수문자(=, +, 〈, 〉, #, ;, \ 등)들을 제거하고 사용할 수 있도록 시큐어코딩 규칙을 정의해야 한다.

표 3-7 아키텍처설계서

개발환경	활용 가능한 프레임워크 또는 라이브러리
Java, PHP, ASP	LDAP Syntax Filters: http://social.technet.microsoft.com/wiki/contents/articles/5392.active-directory-ldap-syntax-filters.aspx LDAP 검색필터를 액티브 디렉토리 조회시의 검색기준을 정의하고 효율적인 검색을 수행할 수 있다. 위 MS 웹문서의 내용과 참조목록으로 필터 작성에 필요한 문법을 참고하여 특수문자를 필터링한다.

다. 진단 세부사항

요구사항 ①

LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용되는 외부입력값은 LDAP 삽입 취약점을 가지지 않도록 필터링해서 사용해야 한다.

LDAP 질의문 생성 시 사용되는 입력값과 조회결과에 대한 검증방법(필터링 등)을 설계하고 유효하지 않은 값에 대한 처리방법이 명시되어 있는지 확인한다.

진단기준

1. 외부입력값이 LDAP조회를 위한 필터생성에 삽입되어 사용되는 경우, 안전하게 사용될 수 있도록 보안설계가 적용되어 있는가?
2. LDAP 삽입 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법

1. 외부입력값이 LDAP조회를 위한 필터생성에 삽입되어 사용되는 경우, 안전하게 사용될 수 있도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항 추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 LDAP 조회필터 생성에 사용되는 입력값을 필터링하는 기능이 설계되어 있거나, 안전한 외부라이브러리를 사용하도록 설계되어 있는지 확인	프로그램명세서, 컴포넌트설계서, 유즈케이스설계서, 클래스설계서 등에서 필터링 기능 검토
1-3 LDAP조회기능 구현시, LDAP필터링을 적용하기 위한 코딩규칙이 개발가이드에 정의되어 있는지 확인	개발가이드에서 코딩규칙 검토

2. LDAP 삽입 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 LDAP 필터구문을 변경할 수 있는 입력값을 사용하여 LDAP필터의 규칙이 변경되는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인 - 테스트 입력값: = , + , < , > , # , ; , \ 등	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램설계서, 유즈케이스설계서, 클래스설계서, 컴포넌트설계서, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	LDAP 삽입

마. 참고자료

① CWE-90 LDAP Injection, MITRE, http://cwe.mitre.org/data/definitions/90.html

② 2016 OWASP Application Security Verification Standard, OWASP, Malicious Input Handling Verification Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

③ "LDAP Resources", http://ldapman.org/

④ "LDAP Injection & Blind LDAP Injection" http://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf

⑤ "A String Representation of LDAP Search Filters", http://www.ietf.org/rfc/rfc1960.txt

⑥ Failure to Sanitize Data into LDAP Queries('LDAP Injection'), http://cwe.mitre.org/data/definitions/90.html

1.4 시스템 자원 접근 및 명령어 수행 입력값 검증

유형	입력데이터 검증 및 표현
설계항목	시스템 자원 접근 및 명령어 수행 입력값 검증
설명	시스템 자원접근 및 명령어를 수행할 때 입력값에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 외부입력값을 이용하여 시스템자원(IP, PORT번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다. ② 서버프로그램 안에서 셸을 생성하여 명령어를 실행해야 하는 경우 외부입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

가. 취약점 개요

사례1 : 경로 조작 및 자원 삽입

공격자가 입력값 조작으로 시스템이 보호하는 자원에 임의로 접근하여 자원의 수정·삭제, 시스템 정보누출, 시스템 자원 간 충돌로 인한 서비스 장애 등을 유발시킬 수 있는 취약점이다. 즉, 경로 조작 및 자원 삽입으로 공격자가 허용되지 않은 권한을 획득하여, 설정에 관계된 파일을 변경하거나 실행시킬 수 있다.

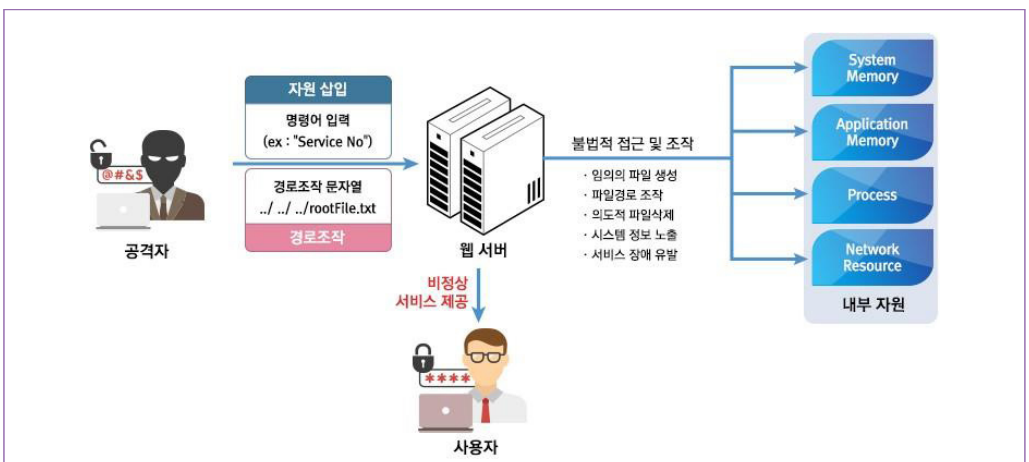


그림 3-4 | 경로 조작 및 자원 삽입

사례2 : 입력값을 조작하여 허가되지 않은 명령 실행

적절한 검증절차를 거치지 않은 사용자 입력값에 의해 의도하지 않은 시스템 명령어가 실행되어 부적절하게 사용자 권한이 변경되거나 시스템 동작 및 운영에 악영향을 미칠 수 있다.

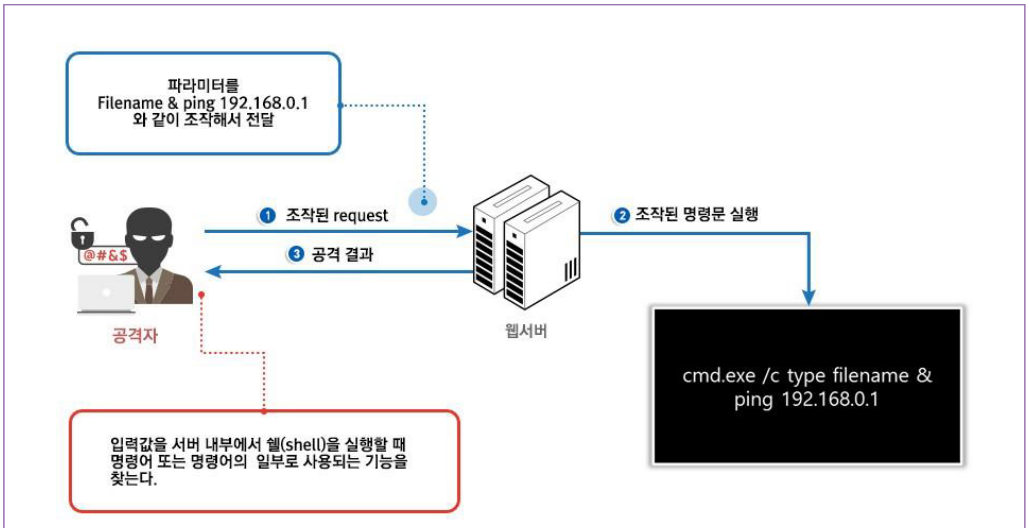


그림 3-5 | 운영체제 명령어 삽입

나. 설계시 고려사항

① 외부입력값을 이용하여 시스템자원(IP, PORT번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다.

외부입력값이 프로그램 내부에서 사용하는 리소스를 결정하는데 직접적으로 사용되지 않도록 설계한다. 즉, 기능 설계시 사용해야 하는 리소스 목록을 정의하여 지정된 범위 안에서 리소스를 선택하여 사용하도록 해야하며, 리소스 목록은 프로퍼티파일이나 XML파일로 정의하여 리소스 정보를 변경하는 경우 프로그램 수정을 최소화 할 수 있도록 관리한다.

사용자의 요청 리소스가 특정 디렉토리내의 모든 파일인 경우에는 모든 파일명을 목록화하는 것은 어렵다. 이런 경우는 입력값 중 경로조작을 일으킬 수 있는 문자(.. / ₩)를 제거하고 사용하여 지정된 경로내의 파일만 접근 가능하도록 시큐어코딩 규칙을 정의한다.

② 서버프로그램 안에서 셸을 생성하여 명령어를 실행해야 하는 경우 외부입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

먼저, 서버프로그램 안에서 셸을 생성해서 명령어가 실행되는 구조를 가지지 않도록 설계하는 것이 우선이다. 하지만 경우에 따라 이러한 기능이 꼭 필요한 경우에는 외부입력값이 직접적으로 명령어의 일부로 사용되지 않도록 해야 한다.

명령어의 일부로 사용되어야 하는 값들을 목록화하여 목록 내에 있는 값들로만 명령어가 조립되어 실행될 수 있도록 해야 하며, 목록화 되어 있는 값들이 경우에 따라 변경되어야 한다면, 이로 인해 프로그램이 수정되지 않도록 프로퍼티파일이나 XML파일을 사용하여 허용목록을 작성한다.

이 때 외부입력 값은 목록화된 정보를 검색하는 인덱스값으로 사용하여 시스템 명령어 사용을 최소화한다.

다. 진단 세부사항

요구사항 ①

외부 입력값을 이용하여 시스템자원(IP, PORT번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다.

IP, PORT, 파일 등 시스템자원을 식별하기 위해 사용자 입력값을 사용하는 경우 입력값을 검증하도록 설계하고 있는지 확인한다.

• 진단기준

1. 외부입력값이 시스템자원을 식별하는 값으로 사용되는 경우, 허가된 자원에만 접근하도록 보안설계가 적용되어 있는가?
2. 허가되지 않은 자원에 대해 접근이 가능한지를 점검하는 테스트 계획이 수립되어 있는가?

• 진단방법

1. 외부입력값이 시스템자원을 식별하는 값으로 사용되는 경우, 허가된 자원에만 접근하도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토

진단방법	관련산출물 검토예시
1-2 접근이 허가된 시스템자원이 식별되고, 사용가능한 기능과 매핑되어 있는지 아키텍처설계서 등에서 확인	아키텍처설계서의 보안요구항목 적용 계획 검토, 자원과 기능 매핑을 체크하기 위해 DB설계서 검토
1-3 시스템자원 접근 이력이 로그로 기록되도록 설계되어 있는지 아키텍처설계서, 클래스설계서 등에서 확인 - 로그기록내용에는 시간, 사용자ID, 사용자IP 및 입력값이 포함 되도록 설계 되어야 하며, 권한없는 접근의 경우도 기록하도록 설계되어 있는지 확인	아키텍처설계서, 유즈케이스설계서, 클래스설계서 검토

2. 허가되지 않은 자원에 대해 접근이 가능한지를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 조작된 입력으로 허가되지 않은 자원(파일이나, 서비스, 프로세스, 데이터 등)의 접근가능 여부를 점검할 수 있는 테스트 계획이 수립되어 있는 확인 테스트 입력값: 경로 조작에 사용되는 .., /, \\ 문자 등)	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, DB설계서, 개발가이드, 단위테스트케이스, 단위테스트 시나리오등

요구사항 ②

서버 프로그램 안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

시스템 명령어를 사용할 때 외부 입력값을 사용하는 경우 입력값을 검증하도록 설계하고 있는지 확인한다.

진단기준

1. 외부입력값이 운영체제 명령어 실행시 명령어 또는 파라메터로 사용되는 경우, 허가된 명령만 실행되도록 보안 설계가 적용되어 있는가?
2. 허가되지 않은 명령어 실행가능여부를 점검하는 계획이 수립되어 있는가?

진단방법

1. 외부입력 값이 운영체제 명령어 실행시 명령어 또는 파라메터로 사용되는 경우, 허가된 명령만 실행되도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 프로그램내에서 명령어를 실행하는 기능 포함여부를 식별하고, 해당 기능에서 실행 가능한 명령어 또는 파라미터가 제한되도록 설계되어 있는지 확인	아키텍처설계서, 클래스 설계서, 유즈케이스설계서 검토
1-3 운영체제 명령어 실행 이력이 로그로 기록되도록 설계되어 있는지 아키텍처 설계서, DB설계서 등에서 확인 - 로그기록내용에는 시간, 사용자ID, 사용자IP 및 입력값이 포함 되도록 설계 되어야 하며, 실행한 명령어와 파라미터가 기록되도록 설계 되었는지 확인	아키텍처설계서의 보안 요구항목 적용 계획 검토, 로그기록내용과 기능 설계시 로깅이 수행되도록 설계되어 있는지 진단을 위해 클래스설계서 검토, 로깅 코딩 규칙 진단을 위해 개발 가이드 검토

2. 허가되지 않은 명령어 실행가능여부를 점검하는 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 조작된 입력으로 허가되지 않은 명령어가 실행가능한지 여부를 테스트하는 계획을 설계문서(ex. 단위테스트케이스, 단위테스트시나리오 등)에서 확인 - 테스트 계획에는 검증해야 할 문자열(|, &, ; 등)을 정의하고 검증 및 예상 결과가 포함되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스 설계서, 클래스설계서, 개발가이드, 단위테스트케스, 단위테스트시나리오 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	코드 삽입
입력데이터 검증 및 표현	경로 조작 및 자원 삽입
입력데이터 검증 및 표현	서버사이드 요청 위조
입력데이터 검증 및 표현	운영체제 명령어 삽입

마. 참고자료

① CWE-99 Resource Injection, MITRE,
http://cwe.mitre.org/data/definitions/99.html

② CWE-78 OS Command Injection, MITRE,
http://cwe.mitre.org/data/definitions/78.html

③ CWE-22 Path Traversal, MITRE, http://cwe.mitre.org/data/definitions/22.html

④ WASC Threat Classification v2.0 OS Commanding, WASC,
http://projects.webappsec.org/w/ page/13246950/OS%20Commanding

⑤ WASC Threat Classification v2.0 Path Traversal, WASC,
http://projects.webappsec.org/w/page/13246952/ Path%20Traversal

⑥ Command Injection, OWASP,
http://www.owasp.org/index.php/Command_Injection

⑦ File System Path Traversal, OWASP,
http://www.owasp.org/index.php/File_System#Path_traversal

⑧ Directory traversal attack, Wikipedia,
http://en.wikipedia.org/wiki/Directory_traversal_attack

1.5 웹 서비스 요청 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	웹 서비스 요청 및 결과 검증
설명	웹 서비스(게시판 등) 요청(스크립트 게시 등)과 응답결과(스크립트를 포함한 웹 페이지)에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 사용자로부터 입력받은 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스 사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다. ② DB조회결과를 동적으로 생성되는 응답페이지에 사용하는 경우 HTML인코딩 또는 크로스 사이트스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

가. 취약점 개요

사례1 : 외부 입력값을 검증 없이 응답페이지 생성에 사용하는 경우

웹 페이지에 악의적인 스크립트가 포함될 수 있으며, 해당 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 실행되어 정보 유출 등의 공격을 유발할 수 있다.

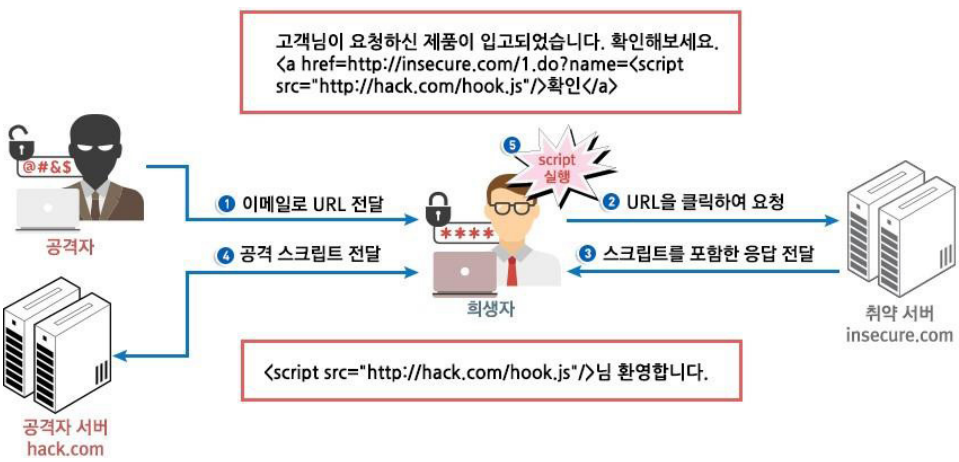


그림 3-6 | Reflective XSS

사례2 : DB에 저장된 값을 검증 없이 응답페이지 생성에 사용하는 경우

공격자가 미리 취약한 서버에 악의적인 스크립트가 포함된 정보를 저장해서 일반 사용자들이 해당 정보를 조회하는 경우 접속자의 권한으로 부적절한 스크립트가 수행되어 정보 유출 등의 공격을 유발할 수 있다.

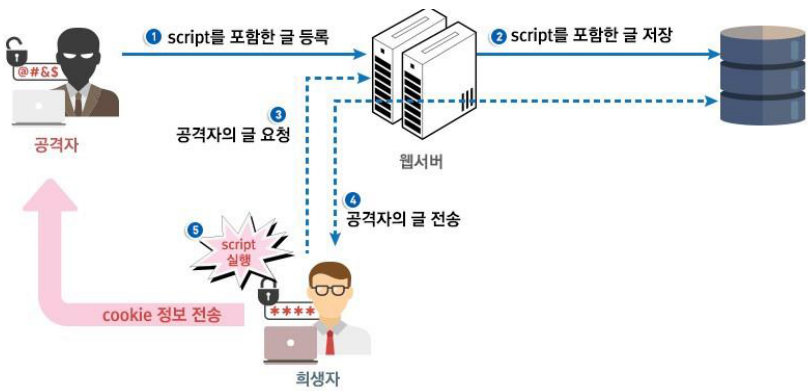


그림 3-7 | Stored XSS

나. 설계시 고려사항

① 사용자로부터 입력받은 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

입력값에 대해 필터링 또는 인코딩 정책을 적용하는 공통코드를 작성하여 웹 컨테이너, 또는 MVC 프레임워크에 적용한다.

(ㄱ) 필터를 이용한 입력값 검증

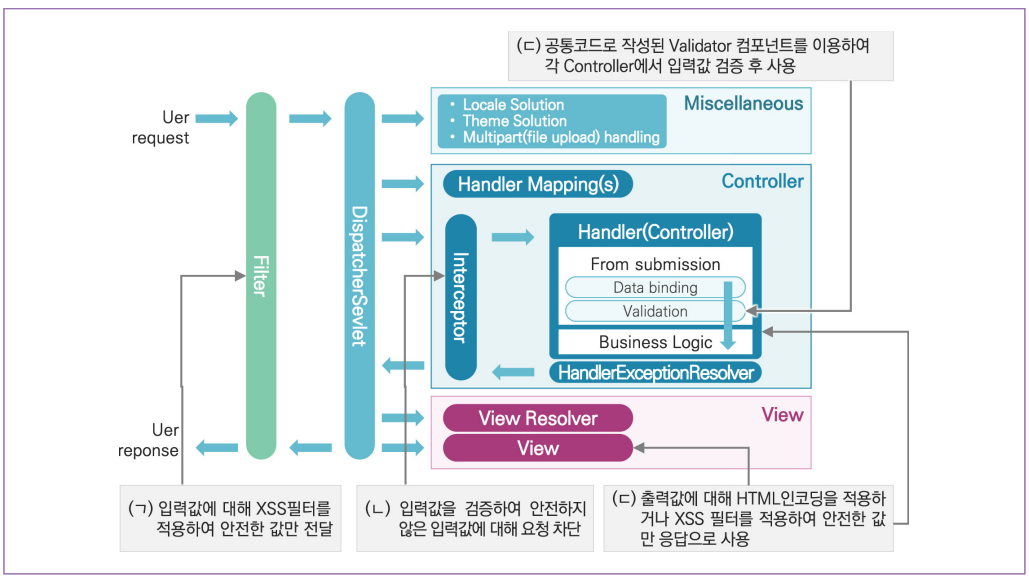
웹 컴포넌트인 Filter를 사용하여 사용자의 입력값에 대해 XSS 필터나 HTML인코딩을 적용하여 안전한 값으로 치환한 뒤 사용할 수 있도록 모든 애플리케이션에 일괄 적용한다.

(ㄴ) 인터셉트를 이용한 입력값 검증

MVC프레임워크를 사용하는 경우 Interceptor 컴포넌트를 사용하여 사용자의 입력값에 대해 XSS 공격 패턴의 문자열이 포함되었는지를 검사하여 요청을 차단 또는 허용하는 정책을 모든 애플리케이션에 일괄 적용한다.

(ㄷ) 라이브러리 또는 Validator 컴포넌트를 이용한 입력값 검증

공통코드로 입력값을 검증하는 Validator 컴포넌트를 작성하여 XSS 공격패턴의 사용자 입력값을 필터링 할 수 있도록 설계한다.



② DB조회결과를 동적으로 생성되는 응답페이지에 사용하는 경우 HTML인코딩 또는 크로스사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

각각의 컴포넌트에서 출력값에 대해 XSS필터 또는 HTML인코딩을 적용하여 안전한 값만 응답에 사용한다.

(ㄱ) View 컴포넌트에서 출력값에 대해 HTML인코딩 적용

View 컴포넌트에서 사용자 입력값을 동적으로 생성되는 응답페이지에 사용하는 경우 XSS필터 또는 HTML인코딩을 적용하여 코드를 작성하도록 시큐어코딩 규칙을 정의한다.

(ㄴ) DB조회 결과값에 대한 XSS 필터 적용

DB조회 결과값으로 응답페이지를 생성하는 경우 XSS 필터를 적용하여 사용해야 한다.

DB에서 읽어오는 데이터도 외부 입력값의 범위에 포함시켜, 응답페이지에 출력하기 전에 반드시 검증작업을 수행해야 한다.

DB에서 읽어온 값에 대한 검증작업을 프레임워크의 컴포넌트로 일괄 필터링하는 것이 쉽지 않다. 이 경우 각 개발자들은 출력값에 대해 검증 작업을 수행하도록 시큐어코딩 규칙을 정의한다.

HTML, URL 등의 문자를 인코딩 및 필터링하여 XSS 취약점을 최소화할 수 있도록 [표 3-8]과 같은 라이브러리의 사용을 고려할 수 있다.

표 3-8 XSS 방어를 위한 라이브러리 및 클래스

개발환경	활용 가능한 프레임워크 또는 라이브러리
ASP.NET	MS Anti-XSS Library: http://wpl.codeplex.com/ ASP .NET 웹기반 애플리케이션 개발시, XSS공격으로부터 안전할 수 있도록 도와주는 인코딩 라이브러리이다.
	MS AntiXSSEncoder 클래스: http://social.msdn.microsoft.com/Search/en-US?query=antixssencoder&pgArea=header&emptyWater- mark=true&ac=4 HTML, XML, CSS, URL 문자열을 인코딩하여 XSS공격의 위협에 대응할 수 있는 클래스이다. HttpEncoder클래스의 HttpUtility, HttpServerUtility, HttpResponseH-eader 메서드를 오버라이드하여 사용가능하다.
	HTML Sanitizer 라이브러리: https://github.com/mganss/HtmlSanitizer HTML/CSS문서를 파싱하고 조작하여 XSS공격으로 이어질 수 있는 구조를 정리해주는 XSS 방어 라이브러리이다. HTML 파서를 기반으로 하기 때문에 HTML태그의 변조도 방 지가능하다. MIT License에 따라 사용가능하다.
Java	LUCY XSS Filter: http://github.com/naver/lucy-xss-filter 웹애플리케이션을 XSS공격으로부터 보호하기 위한 화이트리스트 설정방식의 자바 기반 라이브러리이다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
	OWASPESAPIXSSFilter:https://www.owasp.org/index.php/Catego-ry:OWASP_Enterprise_Security_API ESAPI(Enterprise Security API)는 웹 애플리케이션 개발 과정에서 발생하는 다양한 보안 침해사고를 해결하기 위해 OWASP에서 OWASP TOP10과 함께 해당 문제점을 개선하기 위해 제작, 배포되는 보안 라이브러리이다.
PHP	HTML Purifier 라이브러리: http://htmlpurifier.org/ XSS, SQL인젝션 방어를 제공하는 화이트리스트 구현방식의 HTML 필터 라이브러리이다. LGPL v2.1+에 따라 사용가능하다.

다. 진단 세부사항

요구사항 ①

사용자로부터 입력된 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

동적으로 생성되는 응답페이지에 외부입력값이 사용되는 경우 사용자 입력값에 스크립트가 포함되어 있는지 검증하도록 설계하고 있는지 확인한다.

진단기준

1. 외부입력값에 포함된 XSS 공격코드를 안전하게 필터링할 수 있도록 보안설계가 적용되어 있는가?
2. 입력값에 대한 XSS필터링이 안전하게 수행되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 외부입력값에 포함된 XSS 공격코드를 안전하게 필터링할 수 있도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항 추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 XSS 필터 컴포넌트 설계 확인 - 외부라이브러리를 사용하는 경우 안전한 필터링 가능 여부 확인 - XSS필터 생성 시 필터링 규칙이 안전한 화이트 리스트 정책이 적용되도록 설계되었는지 확인	프로그램명세서, 클래스설계서, 유즈케이스설계서 등 검토, 아키텍처설계 서의 보안요구항목 적용계획 검토
1-3 외부입력값에 대한 XSS 필터링 방법이 정의되어 있는지 아키텍처설계서 등에서 확인 - FIlter를 사용하는 경우 필터링 적용대상이 웹서버 설정파일에 정확하게 정의되어 있는지 확인 - 개별 프로그램서 입력값 필터링을 수행하는 경우, 필터링 코딩 규칙이 개발 가이드에 정의 되어 있는지 확인 - 개별 프로그램(Controller 또는 View 컴포넌트)에서 출력값에 대해 필터링을 수행하는 경우, 필터링 코딩규칙이 개발가이드에 정의되어 있는지 확인	키텍처설계서 검토, 코딩규칙은 개발 가이드 검토

2. 입력값에 대한 XSS필터링이 안전하게 수행되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 XSS 발생시킬 수 있는 입력값에 대해 필터링이 적용되어 안전한 값만 응답에 사용되는지를 점검하기 위한 계획이 수립되어 있는지 확인 테스트입력 값 : 필터를 우회할수 있는 입력값 사용 〈IMG SRC="javascript:alert('XSS');"〉 〈IMG ""〉〈SCRIPT〉alert("XSS")〈/SCRIPT〉"〉 〈IMG SRC="jav
ascript:alert('XSS');"〉 등 https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet 문서참조	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 프로그램명세서, 클래스설계서, 유즈케이스설계서, 개발가이드, 단위테스트케이스 등

요구사항 ②

DB조회결과를 동적으로 생성되는 응답페이지에 사용하는 경우 HTML인코딩 또는 크로스 사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

설계 산출물을 검토하여 동적으로 생성되는 응답페이지에 외부 입력값이 사용되는 경우 사용자 입력값에 스크립트가 포함되어 있는지 검증하도록 설계하고 있는지 확인한다.

진단기준

1. DB조회 결과에 포함된 XSS 공격코드를 안전하게 필터링할 수 있도록 보안설계가 적용되어 있는가?
2. DB조회 결과에 대한 XSS필터링이 안전하게 수행되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. DB조회 결과에 포함된 XSS 공격코드를 안전하게 필터링할 수 있도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항 추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 XSS 필터 컴포넌트 설계 확인 - 외부라이브러리를 사용하는 경우 안전한 필터링 가능 여부 확인 - XSS필터 생성 시 필터링 규칙이 안전한 화이트 리스트 정책이 적용되도록 설계 되었는지 확인	아키텍처설계서의 보안요구항목 적용 계획 검토, 프로그램명세서, 클래스설계서, 유즈케이스설계서 등 검토, 아키텍처설계서의 보안요구항목 적용 계획 검토
1-3 DB조회결과를 응답페이지 생성에 사용하는 경우 XSS필터 또는 HTML인 코딩을 적용하여 응답페이지를 만들도록 코딩 규칙이 개발 가이드에 정의되어 있는지 확인	아키텍처설계서 검토, 코딩규칙은 개발 가이드 검토

2. DB조회 결과에 대한 XSS필터링이 안전하게 수행되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 DB조회의 결과값에 대해 XSS필터링이 적용되어 안전한 값만 응답에 사용되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	크로스사이트 스크립트

마. 참고자료

① 2013 OWASP Top 10 - A3 Cross-Site Scripting(XSS), OWASP, http://www.owasp.org/index.php/Top_10_2013

② “Feed Injection In Web 2.0: Hacking RSS and Atom Feed Implementations” By Robert Auger http://www.cgisecurity.com/papers/HackingFeeds.pdf

③ 2010 OWASP Secure Coding Practices, OWASP, Input Validation, http://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

④ “CERT” Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests” http://www.cert.org/advisories/CA-2000-02.html

⑤ “Cross-site Scripting Explained”, By Amit Klein http://crypto.stanford.edu/cs155/papers/CSS.pdf

⑥ “DOM Based Cross Site Scripting or XSS of the Third Kind” By Amit Klein (WASC article) http://www.webappsec.org/projects/articles/071105.shtml

⑦ “Feed Injection In Web 2.0: Hacking RSS and Atom Feed Implementations” By Robert Auger http://www.cgisecurity.com/papers/HackingFeeds.pdf

1.6 웹 기반 중요 기능 수행 요청 유효성 검증

유형	입력데이터 검증 및 표현
설계항목	웹 기반 중요 기능 수행 요청 유효성 검증
설명	비밀번호 변경, 결제 등 사용자 권한 확인이 필요한 중요기능을 수행할 때 웹 서비스 요청에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별할 수 있도록 해야 한다.

가. 취약점 개요

공격자는 세션탈취, XSS 등으로 자신이 의도한 행위(수정, 삭제, 등록 등)를 사이트가 신뢰하는 인증된 사용자의 권한으로 실행되게 할 수 있다.

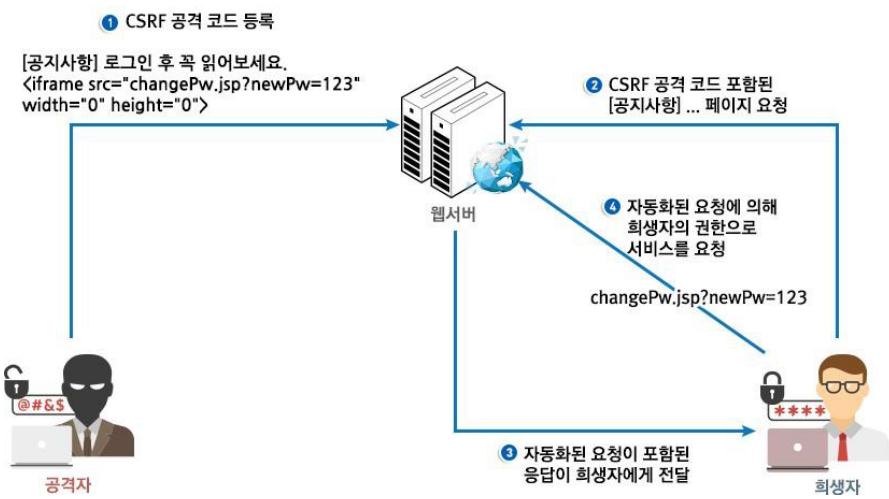


그림 3-8 | 크로스 사이트 요청 위조(CSRF)

나. 설계시 고려사항

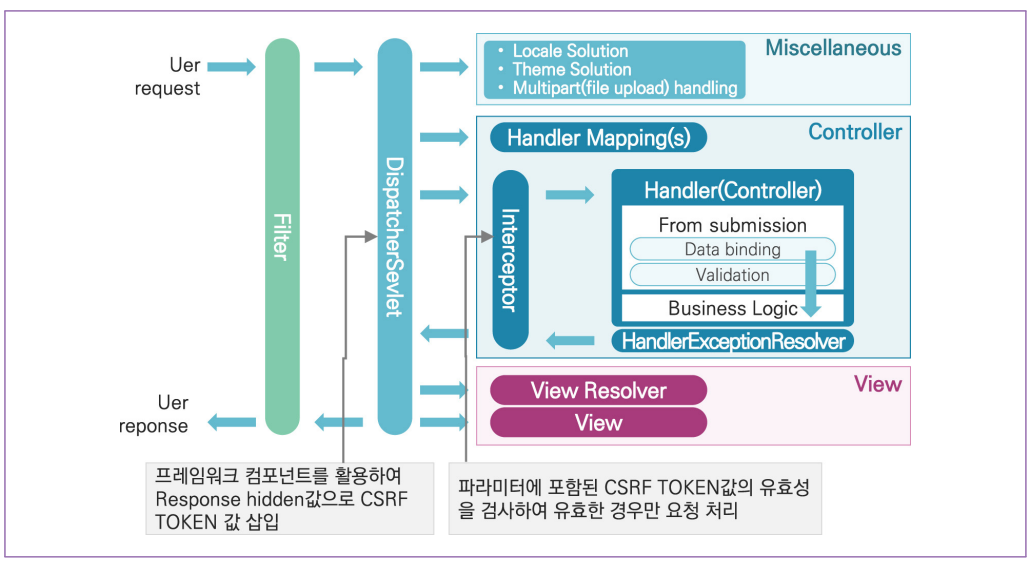
① 시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별할 수 있도록 해야 한다.

(ㄱ) CSRF 토큰 사용

웹은 URL기반으로 요청을 처리하는 구조이다. 해당 요청이 특정 사용자의 정상적인 요청인지를 구분하기 위한 정책이 적용되지 않는 경우, 스크립트나 자동화된 도구에 의해 보내지는 요청이 검증절차 없이 처리될 수 있다.

그래서 해당 요청이 정상적인 사용자의 정상적인 절차에 의한 요청인지를 구분하기 위해 세션별로 CSRF 토큰을 생성하여 세션에 저장하고, 사용자가 작업페이지를 요청할 때마다 hidden값으로 클라이언트에게 토큰을 전달한 뒤, 해당 클라이언트의 데이터처리 요청 시 전달되는 CSRF 토큰값과 세션에 저장된 토큰값을 비교하여 유효성을 검사하도록 설계한다.

CSRF 토큰값에 대한 검사 방법은 MVC프레임워크의 컴포넌트를 이용하여 데이터 처리 요청 수신 시 자동으로 검사될 수 있도록 설계한다. Spring이나 Struts와 같은 MVC프레임워크의 경우 사용자의 요청을 중간 에 가로채서 값의 유효성을 검사할 수 있는 Interceptor 컴포넌트를 이용하여 파라미터로 전달된 CSRF 토큰값이 세션에 저장된 토큰값과 동일한지를 검사하여 동일한 경우만 요청이 처리될 수 있도록 설계한다.



사용자의 요청을 검증하여 CSRF를 방어할 수 있도록 [표 3-9]과 같은 프레임워크나 라이브러리의 사용을 고려할 수 있다.

표 3-9 CSRF 방어를 위한 프레임워크 및 라이브러리

개발환경	활용 가능한 프레임워크 또는 라이브러리
Java	Spring Security: http://spring.io/ 인증, 인가 등 기업 애플리케이션의 보안기능을 제공하는 Java/Java EE 프레임워크이며, 3.2 이상의 버전에서는 CSRF공격에 대한 방어기능을 제공한다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
	Apache Struts: http://struts.apache.org/ Java EE 웹애플리케이션 개발을 위한 오픈소스 프레임워크이다. MVC아키텍처를 적용하는 개발자를 지원하기 위하여 자바 서블릿 API를 사용 및 확장하였다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
	OWASP CSRFGuard: https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project HTML에서 세션 또는 요청단위의 CSRF 토큰을 사용할 수 있도록 하는 자바EE 필터를 제공하는 라이브러리이다. BSD License에 따라 사용할 수 있다.
Python PHP	Django: https://www.djangoproject.com/ 파이썬으로 작성된 오픈소스 웹 애플리케이션 프레임워크로써 CSRF공격에 대한 방어기능을 제공한다. 3-clause BSD License 정책에 따라 사용가능하다.
PHP	CSRF Protector : http://github.com/mebjas/CSRF-Protector-PHP PHP 웹 개발 시 사용할 수 있는 CSRF 방어 라이브러리이다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.

(ㄴ) 사용자와 상호 처리 기능 적용

CSRF 토큰 방식도 XSS 취약점이 있는 사이트를 공격하게 되면 무력화될 수 있으므로 CAPTCHA[5]와 같은 사용자와 상호 처리 가능한 기법을 적용하여 위조된 요청이 차단될 수 있도록 설계한다.

(ㄷ) 재인증 요구

중요기능의 경우 재인증으로 안전하게 실제 요청 여부를 확인하도록 설계한다.

5) CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart, 완전 자동화된 사람과 컴퓨터 판별, 캡 차)는 HIP(Human Interaction Proof) 기술의 일종으로, 어떠한 사용자가 실제 사람인지 컴퓨터 프로그램인지를 구별하기 위해 사용되는 방법이다.

다. 진단 세부사항

요구사항 ①

시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 판별 할 수 있도록 해야 한다.

모든 사용자 요청(Request)에 대해서 정상적인 사용자의 요청(Request)임을 확인하는 절차가 설계되어 있는지 확인한다.

진단기준

1. 중요기능 요청에 대한 데이터 처리 요청에 대해 요청 유효성을 판단하여 처리할 수 있도록 설계 되어 있는가?
2. 요청에 대한 유효성 검증 기능을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 중요기능 요청에 대한 데이터 처리 요청에 대해 요청 유효성을 판단하여 처리할 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처설계서 검토
1-2 데이터 처리 요청(request)에 대한 정상적인 절차에 따른 요청인지를 검증 하는 기능이 안전하게 설계되어 있는지 확인 - CSRF 토큰을 사용하여 요청 유효성 검증 - CAPTCHA를 이용하여 요청 유효성 검증	프로그램명세서, 클래스설계서, 아키텍처설계서 등 검토
1-3 요청유효성 점검 기능이 안전하게 적용되도록 설계되어 있는지 확인 - 모든 데이터 처리 요청에 적용 - 각 데이터를 처리하는 기능에서 적용	아키텍처설계서의 보안요구항목 적용 계획을 검토, 개별 적용의 경우 클래스 설계서, 개발가이드 추가 검토
1-4 주요기능에 대해 재인증/재인가 처리할 방법과 모듈이 설계되어 있는지 확인	클래스설계서 검토

2. 요청에 대한 유효성 검증 기능을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 데이터 처리를 요청하는 주요기능이 자동화된 스크립트나 프로그램을 이용한 자동 요청을 식별하여 요청이 차단되는지를 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램 명세서, 클래스설계서, 단위 테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	크로스사이트 요청 위조

마. 참고자료

① "Cross-Site Request Forgery", Wikipedia
http://en.wikipedia.org/wiki/Cross-site_request_forgery

② 2013 OWASP Top 10 - A8 Cross-Site Request Forgery(CSRF), OWASP,
https://www.owasp.org/index.php/ Top_10_2013

③ CSRF Prevention Cheat Sheet, OWASP,
http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

④ Spring MVC support for CSRF prevention, eyallupu,
http://blog.eyallupu.com/2012/04/csrf-defense-in-spring-mvc-31.html

⑤ Cross-Site Request Forgery: Demystified, ynori7,
http://halls-of-valhalla.org/beta/articles/cross-site-re-quest-forgery-demystified,47/

⑥ Cross-Site Request Forgeries: Exploitation and Prevention, William Zeller and Edward W. Felten,
http://people.eecs.berkeley.edu/~daw/teaching/cs261-f11/reading/csrf.pdf

⑦ Cross Site Request Forgery Protection, Django,
http://docs.djangoproject.com/en/1.8/ref/csrf/

1.7 HTTP 프로토콜 유효성 검증

유형	입력데이터 검증 및 표현
설계항목	HTTP 프로토콜 유효성 검증
설명	비정상적인 HTTP 헤더, 자동연결 URL 링크 등 사용자가 원하지 않은 결과를 생성하는 HTTP 헤더·응답결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 외부입력값을 쿠키 및 HTTP 헤더정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다. ② 외부입력 값이 페이지 이동(리다이렉트 또는 포워드)을 위한 URL으로 사용되어야 하는 경우, 해당 값은 시스템에서 허용된 URL목록의 선택자로 사용되도록 해야 한다.

가. 취약점 개요

사례1 : HTTP 응답분할

공격자가 HTTP 요청에 삽입한 인자 값이 HTTP 응답헤더에 포함되어 사용자에게 다시 전달될 때 개행문자를 이용하여 첫 번째 응답을 종료시키고 두 번째 응답에 악의적인 코드가 주입되어 XSS공격 등이 가능해진다.

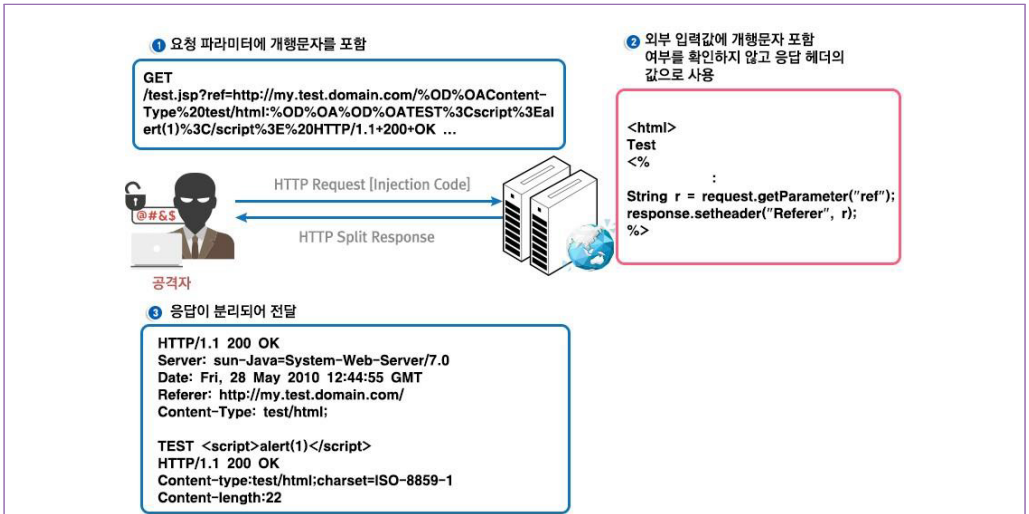


그림 3-9 | HTTP 응답분할

사례2 : 신뢰되지 않은 URL로 자동 접속연결

사용자의 입력값을 외부 사이트의 주소로 사용하여 자동으로 연결하는 서버 프로그램에서는 공격자가 사용자를 피싱(Phishing)사이트 등 위험한 URL으로 접속하도록 유도할 수 있게 된다.

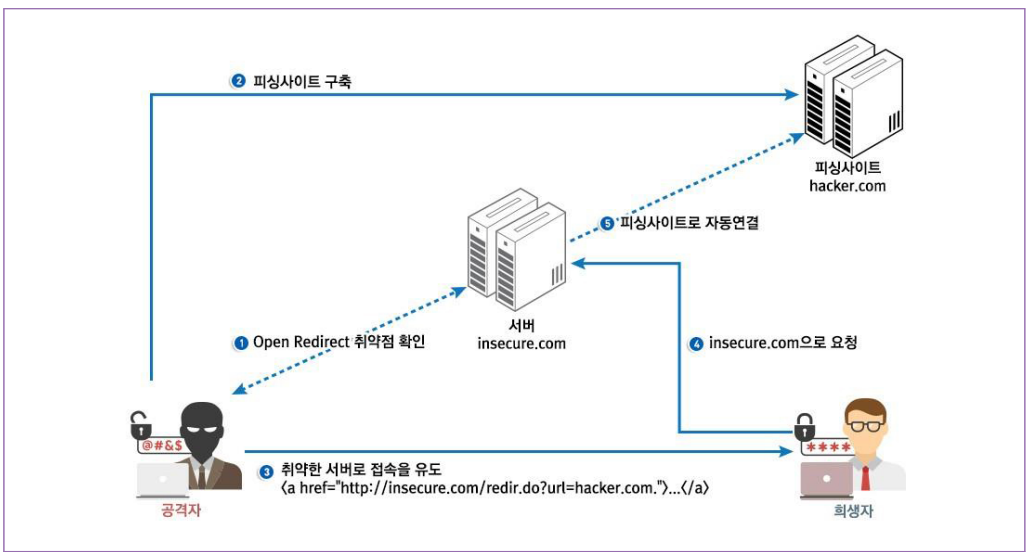


그림 3-10 | 신뢰되지 않는 URL 주소로 자동접속 연결

나. 설계시 고려사항

① 외부 입력값을 쿠키 및 HTTP 헤더정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다.

₩r₩n 문자는 HTTP응답에서 헤더와 바디를 구분하는 구분자로 사용되기 때문에 HTTP응답 헤더에 삽입되는 외부입력값은 반드시 ₩r₩n 문자를 제거하여 사용할 수 있도록 시큐어코딩 규칙을 정의한다.

프로그램에서 Cookie 설정, 응답 헤더 설정, 페이지 리다이렉트를 위한 Location정보 삽입 등 응답헤더에 외부 입력값이 삽입되는 경우 HTTP 응답 분할을 일으킬 수 있는 문자(₩r₩n)을 필터링 하도록 검증절차를 적용한다.

② 외부입력값이 페이지 이동(리다이렉트 또는 포워드)을 위한 URL로 사용되어야 하는 경우, 해당 값은 시스템에서 허용된 URL목록의 선택자로 사용되도록 해야 한다.

페이지 이동을 허용하는 URL목록을 소스코드에 하드코딩 하거나, 설정파일(XML, properties)에 저장하여 허용된 URL로만 이동할 수 있도록 설계한다.

다. 진단 세부사항

요구사항 ①

외부입력값을 쿠키 및 HTTP 헤더정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다.

외부 입력값이 쿠키 또는 HTTP 헤더에 포함되는 경우에는 사용자 입력값을 필터링하도록 설계하고 있는지 확인한다.

진단기준

1. HTTP 응답분할이 발생하지 않도록 입력값에 대한 검증을 수행되도록 설계되어 있는가?
2. HTTP 응답분할 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법

1. HTTP 응답분할이 발생하지 않도록 입력값에 대한 검증을 수행되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항 추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처설계서 검토
1-2 응답헤더에 값을 쓰는 기능이 식별되고, 입력값에 포함된 개행문자를 필터링 하는 기능이 설계되어 있는지 확인. - 응답헤더에 값을 쓰는 함수예: setHeader, addCookie, sen-dRedirect, .. 등 - 개행문자: CR(\r) LF(\n)	프로그램명세서, 클래스설계서, 아키텍처설계서 등 검토
1-3 개행문자에 대한 필터링 적용방법에 따라 안전하게 적용되었는지 확인 - 필터컴포넌트를 이용하여 공통적용하는 경우: 입력값이 헤더에 포함되는 모든 기능에 안전하게 적용되는지 확인 - 각 기능에서 필터기능을 사용하는 경우: 코딩규칙을 개발 가이드에 정의하고 있는지 확인	적용규칙 진단을 위해 아키텍처 설계서 검토, 코딩규칙 진단을 위해 개발 가이드 검토

2. HTTP 응답분할 취약점을 점검할 수 있는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 응답분할이 발생 가능한 입력값을 사용하여 응답분할이 발생되는지를 점검 할 수 있는 테스트 계획이 수립되어 있는지 확인 테스트 입력값 예 : CR(\r) LF(\n) 이 포함된 입력값 사용	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트케이스 등

요구사항 ②

외부입력값이 페이지 이동(리다이렉트 또는 포워드)을 위한 URL으로 사용되어야 하는 경우, 해당값은 시스템에서 허용된 URL목록의 선택자로 사용되도록 해야 한다.

페이지가 이동(리다이렉트 또는 포워드)할 URL을 사전에 정의하고 있는지 확인한다.

진단기준

1. 입력값을 페이지 이동을 위한 URL로 직접 사용하는 프로그램 설계시, 오픈 리다이렉트 취약점이 발생하지 않도록 설계되어 있는가?
2. 오픈 리다이렉트 취약점을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 입력값을 페이지 이동을 위한 URL로 직접 사용하는 프로그램 설계시, 오픈 리다이렉트 취약점이 발생하지 않도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 아키텍처설계서, 요구사항추적표 검토
1-2 시스템에서 이동이 허용된 도메인 또는 주소 목록을 정의하고 있는지 확인	클래스설계서 검토
1-3 페이지 이동(리다이렉트 또는 포워드) 기능 설계에, 미리 정의한 허용 목록을 이용한 입력값을 검증하는 방법과 모듈이 설계되어 있는지 확인	클래스설계서, 코딩규칙 진단을 위해 개발가이드 검토

2. 오픈 리다이렉트 취약점을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
1-1 외부입력값을 조작하여 의도된 페이지로 리다이렉트 되는지를 점검할 수 있는 테스트계획이 수립되어 있는지 확인 테스트 입력값 예 : 임의의 URL주소	단위테스트케이스 검토

• **관련산출물**

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	HTTP 응답분할
입력데이터 검증 및 표현	신뢰되지 않은 URL주소로 자동접속 연결

마. 참고자료

① CWE-113 Failure to Sanitize CRLF Sequences in HTTP Headers('Http Response Splitting'), MITRE, http://cwe.mitre.org/data/definitions/113.html

② Unvalidated Redirects and Forwords Cheat Sheet, OWASP, www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

③ WASC Threat Classsification v2.0 HTTP Response Splitting, WASC, http://projects.webappsec.org/w/page/13246931/HTTP%20Response%20Splitting

④ WASC Threat Classification v2.0 URL Redirector Abuse, WASC, http://projects.webappsec.org/w/page/13246981/URL%20Redirector%20Abuse

⑤ Google blog article on the dangers of open redirects, Jason Morrison, http://googlewebmastercentral.blog-spot.com/2009/01/open-redirect-urls-is your-site-being.html

⑥ Preventing Open Redirection Attacks (C#), Hon Calloway, http://www.asp.net/mvc/overview/security/pre-venting-open-redirection-attacks

1.8 허용된 범위내 메모리 접근

유형	입력데이터 검증 및 표현
설계항목	허용된 범위내 메모리 접근
설명	해당 프로세스에 허용된 범위의 메모리 버퍼에만 접근하여 읽기 또는 쓰기 기능을 하도록 검증 방법 설계 및 메모리 접근요청이 허용범위를 벗어났을 때 처리방법을 설계해야 한다.
보안대책	① C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계값을 넘어서 메모리를 읽거나 저장하지 않도록 경계설정 또는 검사를 반드시 수행해야 한다. ② 개발시, 메모리 버퍼오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 해야 한다.

가. 취약점 개요

사례1 : 버퍼 오버플로우

스택(Stack)이나 힙(Heap)에 할당되는 메모리에 문자열 등이 저장될 때 최초 정의된 메모리의 크기를 초과하여 문자열을 저장하는 경우 버퍼 오버플로우가 발생한다.

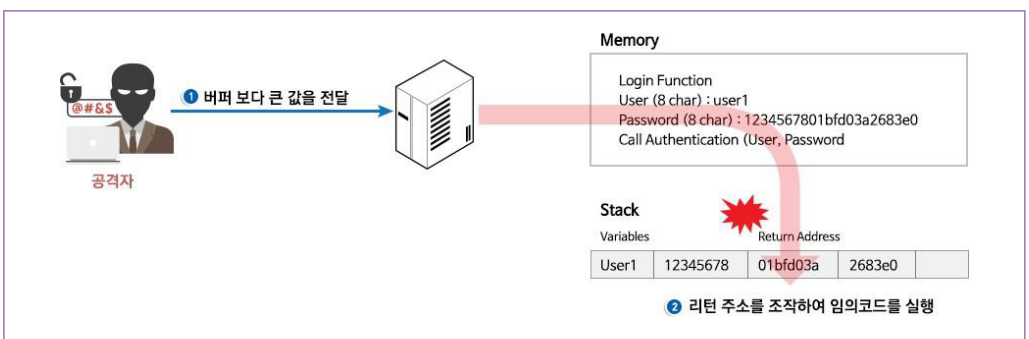


그림 3-11 | 버퍼오버플로우

사례2 : 포맷 스트링 삽입

공격자는 외부입력값에 포맷 문자열을 삽입하여 취약한 프로세스를 공격하거나 메모리 내용을 읽거나 쓸 수 있다. 그 결과, 공격자는 취약한 프로세스의 권한을 취득하여 임의의 코드를 실행할 수 있다.

나. 설계 시 고려사항

① C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계값을 넘어서 메모리를 읽거나 저장하지 않도록 경계설정 또는 검사를 반드시 수행해야 한다.

실행되는 시스템 Non-executable Stack, 랜덤스택(ASLR), 스택가드(StackGuard)와 같은 메모리 보호 정책이 적용되도록 해야 하며, 구현단계에 안전한 함수사용법에 대한 설명과 메모리 사용시 경계값을 검사하고 사용할 수 있도록 시큐어코딩 규칙을 정의하여 개발자들이 준수할 수 있도록 한다.

② 개발시, 메모리 버퍼 오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 해야 한다.

메모리 버퍼오버플로우 취약성을 가지고 있는 함수들을 식별하여 개발자들이 해당 함수를 사용하지 않고 안전한 함수를 사용할 수 있도록 시큐어코딩 규칙을 정의하고 개발자가 준수할 수 있도록 한다.

다. 진단 세부사항

요구사항 ①

C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계값을 넘어서 메모리를 읽거나 저장하지 않도록 경계 설정 또는 검사를 반드시 수행해야 한다.

설계산출물을 검토하여 배열의 값을 다른 배열로 복사하여 넣는 경우 길이 검사 수행 방식이나 공통 함수가 설계되어 있는지 확인한다.

진단기준

1. 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우, 메모리 버퍼오버플로우가 발생하지 않도록 보안 설계가 적용되어 있는가?
2. 메모리 버퍼오버플로우를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우, 메모리 버퍼오버플로우가 발생하지 않도록 보안설계가 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 리눅스 환경인 경우, ASLR(Address Space Layout Randomiza-tion), StackGuard와 같은 메모리 보호를 위한 설정을 사용하도록 설계되어 있는지 확인	아키텍처설계서의 보안요구항목 적용 계획 검토
1-3 개발가이드에서 배열의 값을 다른 배열로 복사하여 넣는 경우, 처리되는 데이터의 길이를 검사하고 사용하도록 코딩규칙을 정의하고 있는지 확인	개발가이드에서 코딩 규칙 검토

2. 메모리 버퍼오버플로우를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 할당된 메모리보다 더 큰입력값을 사용하여 버퍼오버플로우가 발생되는지를 점검할 수 있는 테스트계획이 수립되어 있는지 확인 테스트 입력값 예 : 충분히 긴 문자열	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트케이스 등

요구사항 ②

개발시, 메모리 버퍼오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 통제해야 한다.

취약한 API를 정의하고 있는지 확인한다.

진단기준

1. 메모리 버퍼오버플로우 방지를 위한 안전한 API 목록이 정의 되어있는가?
2. 메모리 버퍼오버플로우를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 메모리 버퍼 오버플로우 방지를 위한 안전한 API 목록이 정의 되어있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 메모리 버퍼오버플로우를 발생시킬 수 있는 API를 정의하고, 사용하지 않도록 하는 코딩규칙이 개발가이드에 정의되어 있는지 확인	개발가이드에서 코딩규칙 검토

2. 메모리 버퍼오버플로우를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 할당된 메모리보다 더 큰 입력값을 사용하여 버퍼 오버플로우가 발생하는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인 테스트 입력값 예 : 충분히 긴 문자열	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	메모리 버퍼 오버플로우
입력데이터 검증 및 표현	포맷 스트링 삽입

마. 참고자료

① CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer, MITRE, http://cwe.mitre.org/data/definitions/119.html

② Buffer Overflow, Wikipedia, http://en.wikipedia.org/wiki/Buffer_overflow

③ WASC Threat Classification v2.0 Buffer Overflow, WASC, http://projects.webappsec.org/w/page/13246916/ Buffer%20Overflow

④ Smashing The Stack For Fun And Profit, Aleph One, http://insecure.org/stf/smashstack.html

⑤ Stack BufferOverflow(스택 버퍼오버플로우), Inhack, http://inhack.org/wordpress/?p=2932

⑥ Stack-based Overflow Exploit: Introduction to Classical and Advanced Overflow Technique, wowhacker, http://web.archive.org/web/20070818115455/http://www.neworder.box.sk/newsread.php?newsid=12476

⑦ CWE-134 Uncontrolled Format String, MITRE, http://cwe.mitre.org/data/definitions/134.html

⑧ WASC Threat Classification Format String, WASC, http://projects.webappsec.org/w/page/13246926/For- mat%20String

1.9 보안기능 입력값 검증

유형	입력데이터 검증 및 표현
설계항목	보안기능 입력값 검증
설명	보안기능(인증, 권한부여 등) 입력 값과 함수(또는 메소드)의 외부입력값 및 수행 결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다. ② 쿠키값, 환경변수, 파라미터 등 외부입력값이 보안기능을 수행하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다. ③ 중요상태정보나 인증, 권한 결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송하는 경우에는 해당 정보를 암호화해서 전송해야 한다.

가. 취약점 개요

사례1 : 보안기능 결정에 사용되는 부적절한 입력값

서버는 사용자가 전달하는 쿠키, 환경변수, 파라미터 등을 충분히 검증하지 않고 사용할 경우 공격자는 이에 포함된 사용자 권한, 역할 등을 나타내는 변수를 조작한 뒤 서버로 요청하여 상승된 권한으로 작업을 수행한다.

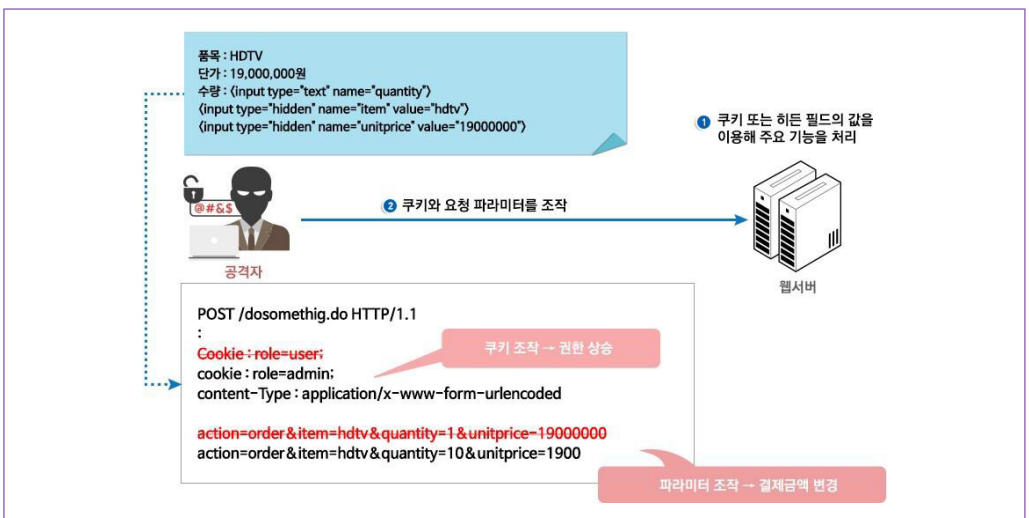


그림 3-12 | 보안기능 결정에 사용되는 부적절한 입력값

사례2 : 정수형 오버플로우

정수형 변수의 오버플로우는 정수 값이 증가하면서, 허용된 가장 큰 값보다 더 커져서 실제 저장되는 값이 의도하지 않게 아주 작은 수이거나 음수가 되어 발생한다. 특히 반복문 제어, 메모리 할당, 메모리 복사 등을 위한 조건으로 사용하는 외부입력값이 오버플로우 되는 경우 보안상 문제를 유발할 수 있다.

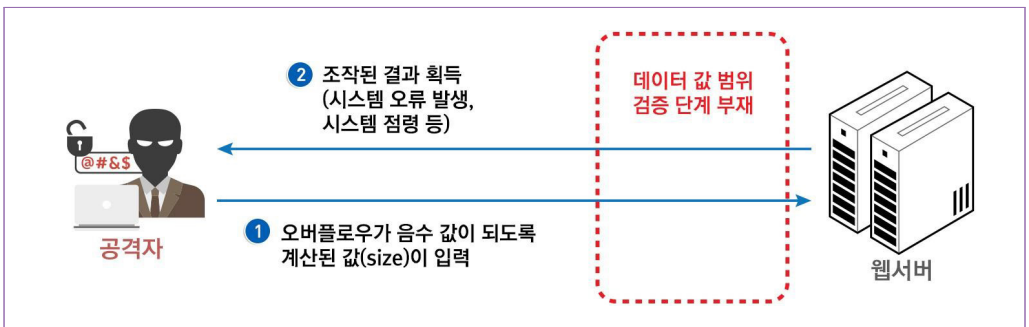


그림 3-13 | 정수형 오버플로우

사례3 : Null Pointer 역참조

일반적으로, 그 객체가 '널(Null)이 될 수 없다'라고 하는 가정을 위반했을 때 발생한다. 공격자가 의도적으로 널 포인터 역참조를 발생시키는 경우, 그 결과 발생하는 예외 상황을 이용하여 추후의 공격을 계획하는 데 사용될 수 있다.

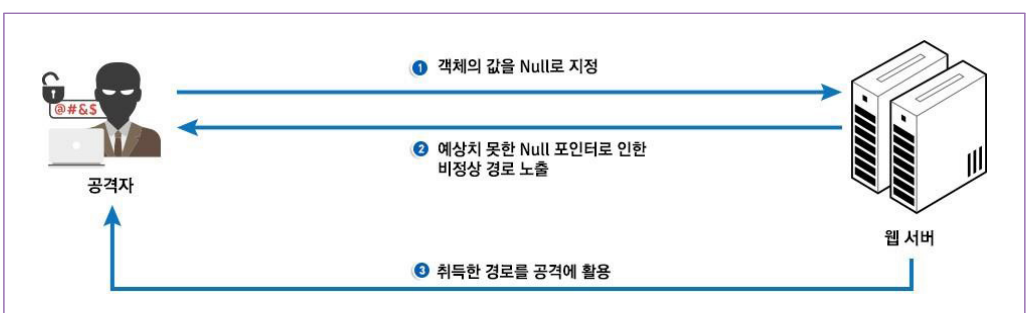


그림 3-14 | NULL Pointer 역참조

나. 설계 시 고려사항

① 사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다.

상태정보나 인증, 인가, 권한에 관련된 중요정보는 서버 측의 세션이나 DB에 저장해서 사용하도록 설계한다.

② 쿠키값, 환경변수, 파라미터 등 외부입력값이 보안기능을 수행하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다.

보안기능을 수행하는 함수 설계 시, 외부의 입력값에 의존할 필요가 없는 구조를 가지도록 설계해야 하며, 만약 외부입력값(쿠키, 환경변수, 파라미터 등)을 받아서 수행해야 하는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다.

특히 입력값에 대한 검증작업은 클라이언트측에서 수행하는 검증방식과 서버에서 수행하는 검증방식이 동일해야 한다.

또한, 외부에서 입력된 값에 대해서는 사용 전에 NULL 여부를 체크한 뒤 사용해야 한다.

③ 중요상태정보나 인증, 권한 결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송하는 경우에는 암호화해서 전송해야 한다.

중요정보가 포함되는 쿠키는 암호화하여 전송하거나 HMAC(Hash-based Message Authentication Code)과 같은 메시지인증코드를 이용하여 서버측에서 무결성을 검증한다.

다. 진단 세부사항

요구사항 ①

사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다.

사용자 및 권한 확인 절차를 확인하여 서버에서 관리하는 세션정보를 사용하도록 설계하고 있는지 확인한다.

진단기준

1. 사용자의 역할 및 권한 결정을 서버에서 관리하는 정보를 이용하여 서버에서 처리하도록 설계되어 있는가?
2. 인증우회 및 권한상승이 가능한지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 사용자의 역할 및 권한 결정을 서버에서 관리하는 정보를 이용하여 서버에서 처리하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 사용자에 대한 역할을 구분하고 있는지 확인 - 일반사용자, 관리자, 최고관리자 등과 같이 역할구분 확인	사용자의 역할이 구분되어 있는지 클래스 설계서, DB설계서 등 검토
1-3 각 역할에 대한 권한을 구분하고 있는지 확인	역할에 대한 권한이 할당되어 있는지 클래스설계서, DB설계서 검토
1-4 사용자의 역할과 권한을 정의하여 저장할 수 있도록 데이터베이스가 설계되어 있는지 확인	DB설계서 검토
1-5 사용자의 역할과 권한을 검증하는 모듈이 설계되어 있거나, 외부 라이브러리를 사용하는 경우 사용자의 권한이나 역할 검증을 위한 정보가 DB 또는 세션 에 저장되도록 설계되어 있는지 확인	권한이나 역할검증을 위한 정보를 DB 또는 세션에 저장하고 사용하는지 클래스 설계서, 유즈케이스설계서 검토

2. 인증우회 및 권한상승이 가능한지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 조작된 입력 파라미터, 쿠키 등을 전달하여 사용자의 역할이나 권한 체크가 우회되는지 점검하기 위한 테스트계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, DB설계서, 유즈케이스설계서, 클래스설계서, 단위테스트시나리오 등

요구사항 ②

쿠키값, 환경변수, 파라미터 등 외부입력값이 보안기능을 수행하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다.

외부입력값을 이용하여 사용자 및 권한 확인 등 보안기능에 사용하는 경우 입력값을 검증하도록 명시하고 있는지 확인한다.

진단기준

1. 보안기능 수행에 사용되는 입력값이 안전하게 사용되도록 설계되어 있는가?
2. 외부 입력값 조작으로 보안기능 우회 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 보안기능 수행에 사용되는 입력값이 안전하게 사용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 보안기능 수행에 사용되는 값의 종류, 출처, 의미 등을 정의하고 있는지 확인	보안기능에 대한 유즈케이스설계서, 클래스설계서 검토
1-3 보안기능 수행에 외부 입력값을 사용해야 하는 경우, 타당한 근거/사유가 명시되어 있는지 확인	보안기능에 대한 유즈케이스설계서, 클래스설계서 검토
1-4 보안기능 수행에 외부 입력값이 사용되는 경우, 외부 입력값에 대한 검증 방법이 정의되어 있는지 확인 - 외부입력값에 대해 NULL 검사를 수행하고 사용하도록 개발보안가이드에 정의 - 외부입력값에 대한 정수오버플로우와 같이 값의 범위를 넘어서는 값이 사용되지 않도록 점검 후 사용하도록 개발보안가이드에 정의	외부입력값 검증을 위한 코딩 규칙은 개발가이드 검토

2. 외부 입력값 조작으로 보안기능 우회 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 외부입력값을 조작하여 보안 기능의 무력화 또는 오작동 여부를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스설계서, 개발보안가이드, 프로그램명세서, 단위테스트시나리오 등

요구사항 ③

중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송하는 경우에는 해당 정보를 암호화해서 전송해야 한다.

생성되는 응답페이지에 외부입력값이 사용되는 경우 사용자 입력값에 스크립트가 포함되어 있는지 검증하도록 설계하고 있는지 확인한다.

진단기준

1. 쿠키로 전달되는 정보가 분류되어 있으며, 그 중 불가피하게 전송되어야 하는 중요정보의 경우 안전하게 전달될 수 있도록 설계되어 있는가?
2. 쿠키를 조작하여 중요정보 탈취 및 변조 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 쿠키로 전달되는 정보가 분류되어 있으며, 그 중 불가피하게 전송되어야 하는 중요정보의 경우 안전하게 전달될 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 쿠키에 포함될 정보의 이름, 값, 의미, 유효기간 등을 정의하고 있는지 확인 - 쿠키에 중요정보가 포함되어야 할 경우, 타당한 근거/사유 명시	쿠키값 정의 확인을 위해 유즈케이스 명세서, 클래스설계서 검토
1-3 중요정보를 쿠키로 전달해야 하는 경우, 중요정보에 대한 암호화 처리 방식이 설계되어 있는지 확인 - 쿠키설정시 암호화 통신채널로만 쿠키가 전송되도록 쿠기속성 설정 - 쿠키를 안전한 암호알고리즘을 사용하여 암호화하여 전송 하도록 설정	안전하게 암호화되어 전송되는 쿠키 값을 확인하기 위해 유즈케이스명세서, 클래스설계서 검토
1-4 쿠키로 전달된 중요정보의 무결성 검사 방법과 모듈이 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서 검토

2. 쿠키를 조작하여 중요정보 탈취 및 변조 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 쿠키에 포함된 값을 복호화하거나, 변조된 쿠키를 전달하여 정보 조작이 가능 한지를 점검하기 위한 테스트계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스명세서, 클래스설계서, 단위테스트 계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	보안기능 결정에 사용되는 부적절한 입력값
입력데이터 검증 및 표현	정수형 오버플로우
코드오류	Null Pointer 역참조

마. 참고자료

① CWE-807 Reliance on Untrusted Inputs in a Security Decision, MITRE, http://cwe.mitre.org/data/defini- tions/807.html

② CWE-476 NULL Pointer Dereference, MITRE, http://cwe.mitre.org/data/definitions/476.html

③ CWE-190 Integer Overflow, MITRE, http://cwe.mitre.org/data/definitions/190.html

④ Null Pointer Dereference란?, Wisedog, http://story.wisedog.net/null-pointerdereference-%EB%9E%80/

⑤ “Understanding ASP.NET View State”, Microsoft, http://cwe.mitre.org/data/definitions/807.html

⑥ Hash-based message authentication code, Wikipedia, http://en.wikipedia.org/wiki/Hash-based_message_authentication_code#External_links

1.10 업로드·다운로드 파일 검증

유형	입력데이터 검증 및 표현
설계항목	업로드·다운로드 파일 검증
설명	업로드·다운로드 파일의 무결성, 실행권한 등에 관한 유효성 검증방법 설계 및 부적합한 파일에 대한 처리방법을 설계한다.
보안대책	① 업로드되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다. ② 업로드되어 저장되는 파일은 외부에서 식별되지 않아야 한다. ③ 파일 다운로드 요청 시, 요청파일명에 대한 검증작업을 수행해야 한다. ④ 다운로드 받은 소스코드나 실행파일은 무결성 검사를 실행해야 한다.

가. 취약점 개요

사례1 : 위험한 형식 파일 업로드

서버 측에서 실행될 수 있는 스크립트 파일(asp, jsp, php 파일 등)을 업로드 가능하고, 이 파일을 공격자가 웹으로 직접 실행시킬 수 있는 경우 시스템 내부 명령어를 실행하거나 외부와 연결하여 시스템을 제어할 수 있게 된다.

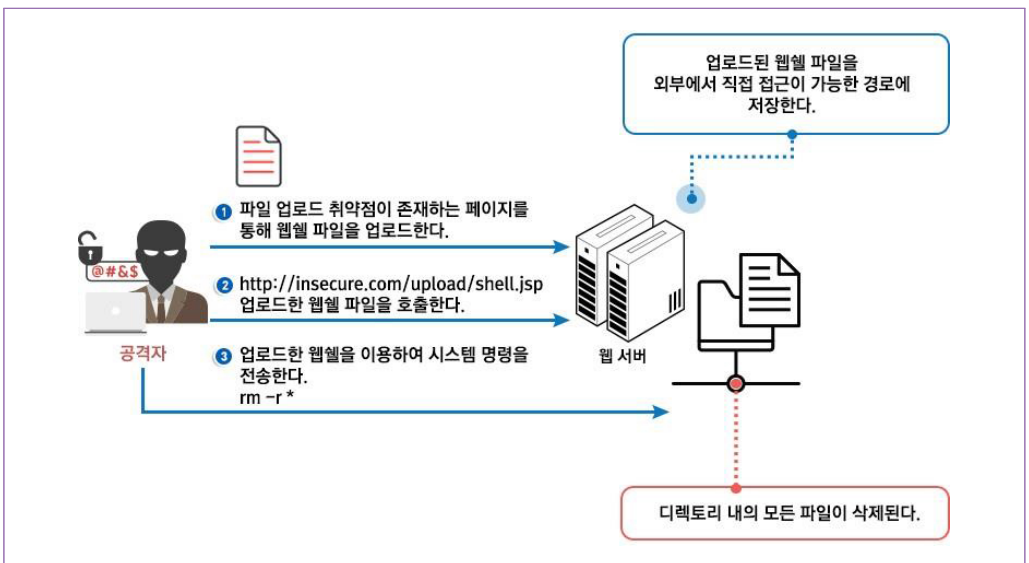


그림 3-15 | 위험한 형식 파일 업로드

사례2 : 경로조작 문자를 이용한 파일 다운로드 취약점

외부입력값에 대해 경로 조작에 사용될 수 있는 문자를 필터링하지 않으면, 예상 밖의 접근제한 영역에 대한 경로 문자열 구성이 가능해져 시스템 정보누출, 서비스 장애 등을 유발할 수 있는 취약점이다.

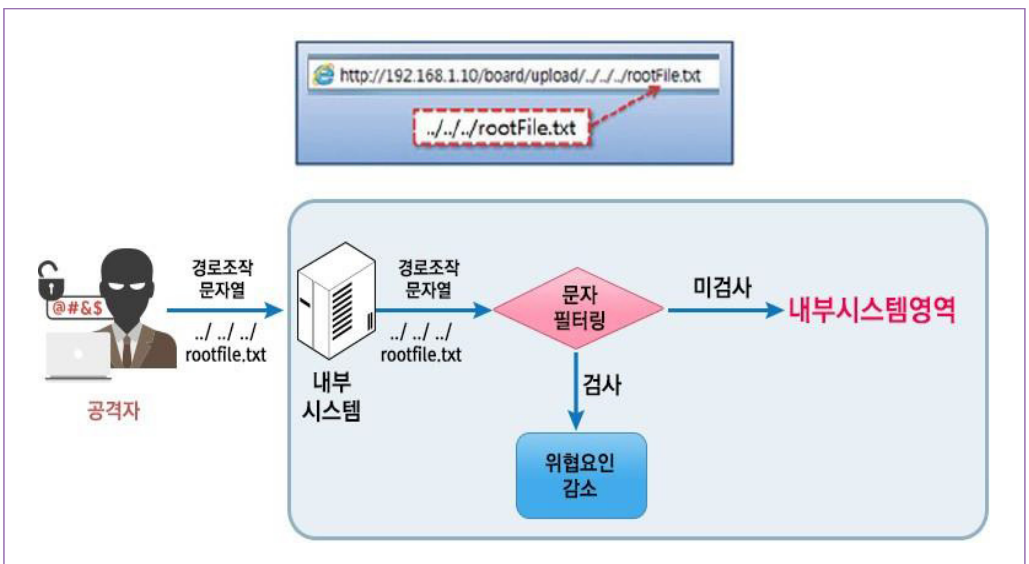


그림 3-16 | 파일 다운로드 취약점

사례3 : 무결성 검사 없는 코드 다운로드

원격으로부터 소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받고 이를 실행할 경우, 호스트 서버의 변조, DNS 스푸핑 (Spoofing) 또는 전송 시의 코드 변조 등의 방법을 이용하여 공격자가 악의적인 코드를 실행할 수 있게 된다.

나. 설계시 고려사항

① 업로드되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다.

(ㄱ) 업로드 파일의 크기 제한

업로드되는 파일의 크기 제한은 프레임워크에서 설정할 수도 있고, 프로그램에서 설정할 수도 있다. Spring 프레임워크를 사용하는 경우 MultipartResolver 컴포넌트의 속성 값을 설정하여 업로드 파일의 크기를 제한할 수 있다.

```
〈bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipart Resolver"〉
〈!– max upload size in bytes –〉
〈property name="maxUploadSize" value="20971520" /〉 〈!– 20MB –〉
〈!– max size of file in memory (in bytes) –〉
〈property name="maxInMemorySize" value="1048576" /〉 〈!– 1MB –〉
〈/bean〉
```

(ㄴ) 파일의 타입 제한

업로드되는 파일유형의 통제는 서버와 클라이언트가 동일한 정책을 적용하도록 설계해야 하며, 파일의 MIME-TYPE과 확장자가 동시에 검사될 수 있도록 업로드 파일에 대한 필터링 기능을 구현한 공통모듈을 설계하여 모든 파일 업로드 기능에 적용하거나 업로드 기능을 가진 컴포넌트에서 직접 필터링하도록 시큐어 코딩 규칙을 정의한다.

(ㄷ) 업로드되어 저장되는 파일의 실행권한 제거

업로드된 파일은 반드시 실행권한을 가지지 않도록 시스템이 설정되어 있어야 한다. 업로드된 파일이 저장 되는 디렉터리에 대한 디폴트 파일 또는 디렉터리 생성시 권한을 확인하고, 업로드된 파일에 실행권한이 주어지지 않도록 시스템설정을 확인한다. 파일이 읽기 권한을 가지고 있기만 해도 서버에서 해당 파일을 읽어서 실행할 수 있으므로 서버 설정으로 업로드 경로의 파일이 실행되지 않도록 설정해야한다.

② 업로드되어 저장되는 파일은 외부에서 식별되지 않아야 한다.

(ㄱ) 업로드되는 파일의 저장경로는 외부에서 직접 접근이 불가능한 경로사용

파일이 저장되는 경로는 URL로 직접적으로 접근 가능하지 않은 디렉터리를 사용한다. 이 경우 외부에서 URL을 조작하여 업로드된 파일에 접근할 수 없으므로, 업로드로 저장된 파일에 대한 통제가 가능하다. 업로드된 파일을 다운로드하기 위해서는 다운로드 처리를 수행하는 컨트롤러를 구현해야 하며, 컨트롤러 구현 시 허가된 파일에 대한 허가된 사용자만 다운로드 기능을 사용할 수 있도록 통제한다.

(ㄴ) 업로드되어 저장되는 파일의 파일명은 랜덤하게 생성하여 사용

저장된 파일을 공격자가 찾을 수 없도록 랜덤하게 생성된 파일명을 이용하여 디렉토리에 저장한다. 이 경우 업로드한 파일명과 저장된 파일명이 매핑되어 관리될 수 있도록, DB 테이블의 칼럼을 설계할 때 업로드 파일명, 저장된 파일명이 함께 저장되도록 설계한다.

③ 파일 다운로드 요청시, 요청파일명에 대한 검증작업을 수행해야 한다.

(ㄱ) 파일 다운로드를 위해 요청되는 파일명에 경로를 조작하는 문자가 포함되어 있는지 점검

파일명에 경로조작이 가능한 '..', '/', '₩' 문자는 제거하고 사용할 수 있도록 필터링한다.

(ㄴ) 허가된 사용자의 허가된 안전한 파일에 대한 다운로드 요청인지 확인

파일 다운로드 요청 시 허가된 사용자와 파일인지를 확인하는 기능이 구현되도록 파일 다운로드 컴포넌트가 설계되어야 하며, 파일을 다운로드하는 사용자의 안전을 위해 파일에 대한 악성코드 또는 바이러스 검사를 수행한 뒤 파일이 다운로드 되도록 기능을 설계한다.

(ㄷ) 사용자의 요청에 의해 서버에 존재하는 파일이 참조되는 경우 화이트리스트 정책으로 접근 통제

시스템에 저장되어 있는 파일을 다운로드 서비스하는 경우, 허용 파일에 대한 목록을 작성하고 목록의 범위 안의 파일에 대해서만 제한적으로 접근 또는 다운로드할 수 있도록 설계한다.

④ 다운로드 받은 소스코드나 실행파일은 무결성 검사를 실행해야 한다.

원격지에서 다운로드 받은 파일을 사용하거나 실행하는 기능을 구현해야 하는 경우 해당 파일과 파일의 체크섬 값(해시 값 등)을 같이 다운로드 받아 파일에 대한 무결성 검사를 수행한 뒤 사용 할 수 있도록 설계한다.

파일 업로드 기능 구현시 [표 3-10]와 같은 프레임워크의 사용을 고려하여 기능의 목적에 부합하는 형식의 파일만 업로드하도록 허용한다.

표 3-10 위험한 형식의 파일 업로드 대응 프레임워크

개발환경	활용 가능한 프레임워크 또는 라이브러리
Java	Spring: http://spring.io/ 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크로써, 업로드 파일에 대해 허용 가능한 확장자를 제한할 수 있다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
PHP	CodeIgniter: https://codeigniter.com/ 비교적 빠르고 간편한 웹 개발 PHP 프레임워크로써, 파일 업로드 기능 구현시 허용할 파일유형과 파일크기제한을 정의할 수 있다. 2.x버전은 OSLv3.0에 따라, 3.x버전은 MIT License에 따라 사용 가능하다.
ASP.NET	.NET Framework: https://www.microsoft.com/net 동적 웹페이지 구축을 돕는 웹 애플리케이션 프레임워크로써, 사용자 인증 및 권한 부여를 위한 확장 가능한 프레임워크를 제공한다. Content-Type 속성 파일의 MIME-Type 형식을 가져와 허용할 파일 확장명을 지정할 수 있다.

다. 진단 세부사항

요구사항 ①

업로드되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다. 업로드되는 파일의 타입, 크기, 개수 및 실행권한을 제한하도록 설계하고 있는지 확인한다.

진단기준

1. 업로드 파일의 제한 정책과 방법이 설계되어 있는가?
2. 업로드 파일의 제한 정책을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 업로드 파일의 제한 정책과 방법이 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토

진단방법	관련산출물 검토예시
1-2 업로드 가능한 파일의 타입, 크기, 개수와 저장시 파일의 퍼미션을 정의하고 있는지 확인 - 파일 업로드 기능을 지원하는 프레임워크 또는 라이브러리 사용 여부를 확인하여 업로드되는 파일의 속성을 제한하는 설정이 계획되어 있는지 확인 - 파일 업로드를 수행하는 프로그램 설계시 업로드되는 파일의 속성을 제한하도록 설계되었는지 확인	아키텍처설계서, 파일업로드 기능의 유즈케이스설계서, 클래스설계서 검토

2. 업로드 파일의 제한 정책을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 허용되지 않는 파일의 타입, 크기, 개수의 업로드가 제한되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 업로드 후 서버에 저장된 파일의 실행이 제한되는지 검검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스설계서, 단위테스트케이스 등

요구사항 ②

업로드되어 저장되는 파일은 외부에서 식별되지 않아야 한다.

업로드된 파일을 외부에서 식별되거나 직접 접근할 수 없도록 설계하고 있는지 확인한다.

진단기준

1. 업로드 파일이 외부에서 식별되지 않도록 안전한 저장위치에 저장되도록 설계되어 있는가?
2. 업로드 되어 저장된 파일이 외부에서 식별가능한지 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 업로드 파일이 외부에서 식별되지 않도록 안전한 저장 위치에 저장되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 저장 경로와 파일명을 외부에서 알 수 없도록 정의하고 있는지 확인 - 안전한 난수를 이용하여 파일명을 생성 - 저장되는 파일명이 중복되지 않도록 안전한 알고리즘 적용	유즈케이스명세서, 클래스설계서 검토
1-3 원본 파일명과 저장경로/파일명의 매핑정보를 관리하도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토

2. 업로드되어 저장된 파일이 외부에서 식별 가능한지 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 업로드되어 저장되는 파일이 외부에서 식별 가능한지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인 - 저장된 경로 - 저장된 파일명	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 단위테스트케이스 등

요구사항 ③

파일 다운로드 요청 시, 요청파일명에 대한 검증작업을 수행해야 한다.

파일 다운로드 요청 시 파일에 대한 검증작업이 이루어지도록 설계하고 있는지 확인한다.

진단기준

1. 다운로드 요청 파일명에 대한 검증이 수행될 수 있도록 설계되어 있는가?
2. 파일 다운로드 정책을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 다운로드 요청 파일명에 대한 검증이 수행될 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 파일 다운로드 요청방식이 안전하게 설계되어 있는지 확인 - 다운로드 요청 파일명에 대한 경로 조작 문자 포함 여부 - DB정보를 이용하여 요청파일의 유효성 점검 여부	유즈케이스명세서, 클래스설계서, DB 설계서 검토

2. 파일 다운로드 정책을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 조작된 파일명을 이용한 파일 다운로드 요청이 차단되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인 - 조작된 파일명 입력값 예: 경로 조작 문자(../ ..\\)가 포함된 파일명 - 파라미터를 조작하여 기능에서 제공하지 않은 다른 파일 요청에 대한 차단 여부 점검	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스명세서, 클래스설계서, 단위테스트케이스 등

요구사항 ④

다운로드받은 소스코드나 실행파일은 무결성 검사를 실행해야 한다.

다운로드받은 소스코드와 실행파일에 대해 무결성을 검증하도록 설계하고 있는지 확인한다.

진단기준

1. 파일 다운로드 시 파일의 무결성을 확인할 수 있도록 설계되어 있는가?
2. 다운로드한 파일(소스코드 또는 실행파일)에 대한 무결성 점검을 위한 테스트 계획이 수립되어 있는가?

진단방법

1. 파일 다운로드시 파일의 무결성을 확인할 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 파일의 무결성 검사를 위한 값(예: 해시)이 제공되도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토
1-3 무결성 검사 수행 후 파일이 다운로드 될 수 있도록 파일 다운로드 기능이 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서 검토

2. 다운로드한 파일(소스코드 또는 실행파일)에 대한 무결성 점검을 위한 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 변조된 파일에 대한 다운로드 처리가 차단되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스명세서, 클래스설계서, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	위험한 형식 파일 업로드
보안기능	부적절한 전자서명 확인
입력데이터 검증 및 표현	무결성 검사 없는 코드 다운로드

마. 참고자료

① CWE-434 Unrestricted Upload of File with Dangerous Type, MITRE, http://cwe.mitre.org/data/definitions/807.html

② CWE-494 Download of Code Without Integrity Check, MITRE, http://cwe.mitre.org/data/definitions/494.html

③ 2010 OWASP Secure Coding Practices, OWASP, File Management, http://www.owasp.org/index.php/OWASP_Se-cure_Coding_Practices_-_Quick_Reference_Guide

④ 2016 OWASP Application Security Verification Standard, OWASP, Files and Resources Verification Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

⑤ Unrestricted File Upload, OWASP, http://www.owasp.org/index.php/Unrestricted_File_Upload

⑥ Improve File Uploaders' Protections, Sorouch Dalili, http://soroush.secproject.com/downloadable/Improve%20File%20Uploaders%E2%80%99%20Protections.pdf

⑦ 8 Basic Rules to Implement Secure File Uploads, SANS, http://software-security.sans.org/blog/2009/12/28/8-basic-rules-to-implementsecure-file-uploads

⑧ Top 25 Series-Rank 20-Download of Code Without Integrity Check, SANS, http://software-security.sans.org/blog/2010/04/05/top-25-series-rank-20-download-code-integrity-check/

⑨ "Introduction to Code Signing", Microsoft, http://msdn.microsoft.com/en-us/library /ms537361(VS.85).aspx

⑩ Authenticode, Microsoft, http://msdn.microsoft.com/en-us/library/ms537359(v=VS.85).aspx

2. 보안기능

2.1 인증 대상 및 방식

유형	보안기능
설계항목	인증 대상 및 방식
설명	중요정보·기능의 특성에 따라 인증방식을 정의하고 정의된 인증방식을 우회하지 못하게 설계해야 한다.
보안대책	① 중요기능이나 리소스에 대해서는 인증 후 사용 정책이 적용되어야 한다. ② 안전한 인증방식을 사용하여 인증우회나 권한 상승이 발생하지 않도록 해야 한다. ③ 중요기능에 대해 2단계(2-factor)인증을 고려해야 한다.

가. 취약점 개요

중요기능이나 리소스를 요청하는 경우 인증이 되었는지를 먼저 확인하지 않고 요청을 처리하는 경우 중요 정보나 리소스가 노출될 수 있다.

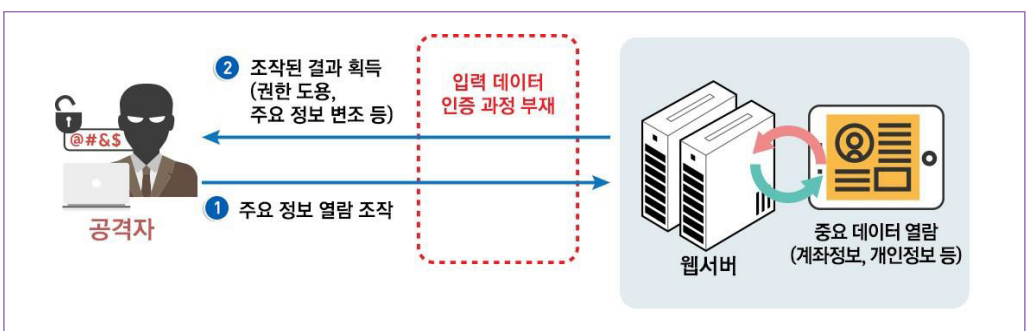


그림 3-17 | 적절한 인증 없는 중요기능 허용

나. 설계시 고려사항

① 중요기능이나 리소스에 대해서는 인증 후 사용 정책이 적용되어야 한다.

분석단계에 분류된 중요기능에 대해 “인증 후 사용” 정책이 반드시 적용될 수 있도록 한다.

각각의 중요기능에서 인증을 요청하도록 구현하는 것은 쉽지 않다. 시스템 설계시 중요기능을 분류 하고 식별된 중요기능에 대해 일괄적으로 인증을 요구하도록 시스템을 설계한다. 이 경우 직접적으로 기능과 인증을 매핑시켜 처리하는 컴포넌트를 개발하거나, 인증기능을 제공하는 프레임워크 또는 라이브러리를 활용하여 중앙집중식 인증이 적용되도록 설계한다.

② 안전한 인증방식을 사용하여 인증우회나 권한 상승이 발생하지 않도록 해야 한다.

인증정보는 서버 측에 저장하여 인증이 필요한 기능이나 리소스에 접근을 시도할 때 서버에서 인증 여부를 확인할 수 있도록 해야 한다.

(ㄱ) 일회용 비밀번호 사용시

일회용 비밀번호를 적용하는 경우 타 서비스나 시스템과의 연동이 보장되도록 설계해야 한다. 일회용 비밀번호를 도입하는 경우 다음과 같은 규칙을 적용하여 설계한다.

A. 일회용 비밀번호는 시각정보, 이벤트정보, 질의응답 방식으로 취득한 정보를 이용해 생성할 수 있다.
B. 시각정보 기반의 연계정보는 특정시간 동안만 유효하여야 하며, 이벤트/질의-응답 방식을 사용할 경우에는 연계정보를 추측할 수 없도록 보호방안을 제공할 수 있어야 한다.
C. 일회용 비밀번호에는 시간적 제한을 설정해야 한다. (금융영역에서는 30~60초)
D. 일회용 비밀번호는 중복 및 유추가 불가능하도록 6자리 이상의 숫자 및 문자로 구성한다.
E. OTP발생기와 인증서버에서 동일한 정보를 생성해야 한다.

③ 중요기능에 대해 2단계(2-factor)인증을 고려해야 한다.

중요기능에 대해 멀티 디바이스(SMS, ARS 등)를 이용한 추가인증이나, 공인인증서, 바이오정보(지문, 홍채 등)를 이용한 2단계 인증을 고려해야 한다.

2단계인증은 Type1(비밀번호/PIN 등 지식기반인증), Type2(토큰/스마트카드 등 소유기반인증), Type3(지문/홍채 등 생체기반인증) 중 2개 이상의 인증기법을 사용하도록 설계한다.

중앙 집중화된 형식의 인증 메커니즘을 제공하기 위해 아래와 같은 프레임워크를 활용할 수 있다.

표 3-11 인증기능을 제공하는 프레임워크

개발환경	활용 가능한 프레임워크 또는 라이브러리
Java	Spring Security: http://spring.io/ 인증, 인가 등 기업애플리케이션의 보안기능을 제공하는 Java/Java EE 프레임워크이다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.

개발환경	활용 가능한 프레임워크 또는 라이브러리
ASP.NET	.NET Framework: https://www.microsoft.com/net 동적 웹페이지 구축을 돕는 웹애플리케이션 프레임워크로써, 사용자 인증 및 권한 부여를 위한 확장 가능한 프레임워크를 제공한다.
PHP	Zend Framework : http://framework.zend.com/객체지향 웹애플리케이션을 위한 프레임워크로써 사용자 인증모듈을 제공한다. New BSD License에 따라 사용할 수 있다.
	Symfony: http://symfony.com/legacy 재사용 가능한 PHP 컴포넌트/라이브러리를 포함한 웹 애플리케이션 프레임워크이다. MIT License에 따라 사용할 수 있다.

다. 진단 세부사항

요구사항 ①

중요기능이나 리소스에 대해서는 인증 후 사용 정책이 적용되어야 한다.

중요기능은 안전한 인증방식을 사용하여 인증 후 사용하도록 설계하고 인증이 수행되도록 설계되어 있는지 확인한다.

진단기준

1. 중요기능과 중요 리소스가 분류되어 있는가?
2. 중요기능이나 리소스에 접근하기 위한 인증 정책이 안전하게 적용되도록 설계되어 있는가?
3. 중요기능과 리소스 접근 우회 취약점을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 중요기능과 중요 리소스가 분류되어 있는가?

진단방법	관련산출물 검토예시
1-1 중요기능과 중요 리소스(정보)가 분류되어 있는지 확인	요구사항정의서 검토

2. 중요기능이나 리소스에 접근하기 위한 인증 정책이 안전하게 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
2-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토

진단방법	관련산출물 검토예시
2-2 중요기능과 중요 리소스(정보)에 대한 접근 권한이 분류되어 있는지 확인	요구사항정의서, 아키텍처서설계서 검토
2-3 접근권한을 기반으로 인증이 요구되는 중요 기능이나 리소스가 분류되어 있는지 확인	아키텍처설계서, DB설계서 검토
2-4 인증 누락이 발생하지 않도록 인증적용 방법이 안전하게 설계했는지 확인 - 프레임워크의 컴포넌트를 활용하여 인증 적용하여 인증누락이 발생하지 않도록 설정되었는지 확인	아키텍처설계서, 유즈케이스설계서, 클래스설계서 검토

3. 중요기능과 리소스 접근 우회 취약점을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
3-1 중요기능이나 리소스에 할당된 접근권한에 의해 접근통제가 수행되고 있는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	요구사항정의서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, DB설계서,유즈케이스다이어그램, 클래스설계서, 개발가이드, 단위테스트 케이스, 단위테스트 시나리오 등

요구사항 ②

안전한 인증방식을 사용하여 인증우회나 권한 상승이 발생하지 않도록 해야 한다. 안전한 인증방식을 사용하여 인증 후 사용하도록 설계되어 있는지 확인한다.

진단기준

1. 안전한 인증방식이 적용되도록 설계되어 있는가?
2. 인증우회 또는 권한상승 발생 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 안전한 인증방식이 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 인증기능 설계시 안전한 인증방식을 사용하도록 설계되어 있는지 확인 - 비밀번호/PIN 등 지식기반 인증 - 토큰/스마트카드 등 소유기반 인증 - 지문/홍채 등 생체기반 인증	아키텍처 설계서, 클래스설계서, 유즈케이스명세서 검토
1.3 인증정보의 저장방식이 안전하게 설계되어 있는지 확인 - 인증에 사용되는 값은 안전하게 암호화되어 서버에 저장되어야 함.	아키텍처설계서, DB설계서, 클래스 설계서, 유즈케이스명세서 검토
1-4 인증에 대한 인증횟수 제한 및 오류처리기능이 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토
1-5 인증오류 또는 인증실패에 대한 로깅이 설계되어 있는지 확인 - 인증오류 또는 실패 로깅시 인증시도를 추적할 수 있는 인증시도시간, IP, ID 정보 등의 포함 여부 확인	유즈케이스명세서, 클래스설계서 검토

2. 인증우회 또는 권한상승 발생 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 인증오류 또는 실패 시 정상적으로 인증실패 로그가 기록되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 인증 없이 중요기능에 접근시도 등 적절한 인증 및 권한 확인 절차가 실행되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 클래스설계서, 유즈케이스다이어그램, 유즈케이스명세서, 시퀀스다이어그램, 단위테스트계획서 등

요구사항 ③

중요기능에 대해 2단계(2 factor)인증을 고려해야 한다.

중요기능은 안전한 인증방식을 사용하여 인증 후 사용하도록 설계하고 2단계 인증 등 보안을 강화하는 방법을 고려하여야 한다.

진단기준

1. 추가적인 인증이 필요한 중요기능의 경우, 안전한 인증방식을 사용하도록 설계되어 있는가?
2. 중요기능에 대한 추가적인 인증요구 수행을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 추가적인 인증이 필요한 중요기능의 경우, 안전한 인증방식을 사용하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 개인정보변경, 비밀번호 재설정, 권한 관리, 관리자 기능과 같은 중요기능의 경우 추가인증을 요청하도록 설계되어 있는지 확인	아키텍처 설계서, 유즈케이스명세서, 클래스설계서 검토
1-3 추가 인증기능 설계시 안전한 인증방식이 사용되도록 설계되어 있는지 확인 - ID/비밀번호 또는 일회용 비밀번호(OTP) - 멀티디바이스(SMS, ARS 등)를 이용한 추가 인증 - (공인 및 사설)인증서 - 바이오정보(지문, 홍채 등)	아키텍처설계서, 유즈케이스명세서, 클래스설계서 검토
1-4 추가인증이 요구되는 중요기능을 사용하거나 해당 인증을 실패하는 경우 사용시간, IP주소, ID정보, 사용기능 등이 로깅 되도록 설계되었는지 확인	유즈케이스명세서, 클래스설계서 검토

2. 중요기능에 대한 추가적인 인증요구 수행을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 중요기능 수행 시 추가인증이 요구되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 인증오류 또는 인증 실패 시 로깅이 정상적으로 수행되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 클래스설계서, 유즈케이스다이어그램, 유즈케이스명세서, 시퀀스다이어그램, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	서버사이드 요청 위조
보안기능	적절한 인증 없는 중요기능 허용
보안기능	부적절한 인증서 유효성 검증
API 오용	DNS lookup에 의존한 보안 결정

마. 참고자료

① CWE-306 Missing Authentication for Critical Function, MITRE, http://cwe.mitre.org/data/definitions/306.html

② 2013 OWASP Top 10 - A7 Missing Function Level Access Control, OWASP, https://www.owasp.org/index.php/ Top_10_2013

③ Authentication Cheat Sheet, OWASP, http://www.owasp.org/index.php/Authentication_Cheat_Sheet

④ 2016 OWASP Application Security Verification Standard, OWASP, Authentication Verification Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

⑤ Digital Authentication – The Basics, Dawn M. Turnet, http://www.cryptomathic.com/news-events/blog/digi-tal-authentication-the-basics

⑥ 일회용 비밀번호(OTP) 알고리즘 프로파일, 한국정보통신기술협회, http://www.tta.or.kr/data/ttas_view.jsp?rn=1&rn1=Y&rn2=&rn3=&pk_num=TTAK.KO-12.0193&nowSu=4&standard_no=&kor_standard=otp&-publish_date=&section_code=&acode1=&acode2=&scode1=&scode2=&order=publish_date&by=desc&to-talSu=14

⑦ 홍채 등 생채인식 기반 간편 공인인증 가이드라인, 한국인터넷진흥원, http://www.kisa.or.kr/notice/press_View.jsp?mode=view&p_No=8&b_No=8&d_No=1484

⑧ Spring Security – Authentication, eGovFrame 표준프레임워크 포털, http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte:fdl:server_security:authentication

⑨ ASP.NET Authentication, Microsoft, http://msdn.microsoft.com/en-us/library/eeyk640h.aspx

2.2 인증 수행 제한

유형	보안기능
설계항목	인증 수행 제한
설명	반복된 인증 시도를 제한하고 인증 실패한 이력을 추적하도록 설계한다.
보안대책	① 로그인 기능 구현 시, 인증시도 횟수를 제한하고, 초과된 인증시도에 대해 인증제한 정책을 적용해야 한다. ② 실패한 인증시도에 대한 정보를 로깅하여 인증시도 실패가 추적될 수 있게 해야 한다.

가. 취약점 개요

로그인 시도에 대한 횟수를 검사하지 않으면 로그인 시도 횟수와 상관없이 지속적으로 로그인 시도가 이루어지는 비밀번호 무차별 대입 공격이 시도되어 계정정보가 노출될 수 있다.

그림 3-18 | 반복된 인증시도 제한 기능 부재

나. 설계시 고려사항

① 로그인 기능 구현 시, 인증시도 횟수를 제한하고, 초과된 인증시도에 대해 인증제한 정책을 적용해야 한다.

로그인 시도횟수를 3~5번 이내로 제한하고 이를 초과하여 로그인에 실패하는 경우 추가 입력값을 요구하거나 계정 잠금을 수행하도록 다음과 같은 메커니즘으로 설계한다.

사용자ID별, 세션ID별 로그인 횟수를 추적하기 위해 사용자 DB테이블에 로그인 실패 횟수, 계정 잠금 여부, 마지막으로 성공·실패한 로그인 시간, 로그아웃한 시간 등을 저장할 수 있도록 설계하고 일정 횟수 이상 연속적으로 로그인 실패 시 사용자 ID와 비밀번호 외의 추가적인 정보를 확인하도록 한다.

계정정보 입력 시 자동입력방지 문자와 같은 장치를 마련하도록 설계한다.

보안문자 이미지 생성 및 입력값과 보안문자를 비교하기 위해 다음과 같은 서비스나 솔루션의 사용을 고려할 수 있다.

표 3-12 CAPTCHA 기능을 제공하는 서비스 및 솔루션

개발환경	활용 가능한 프레임워크 또는 라이브러리
서비스	reCAPTCHA: https://developers.google.com/recaptcha/ CAPTCHA 시스템의 일종으로 OCR소프트웨어가 판독할 수 없는 글자이미지를 생성하여 사용자에게 해당 문자의 입력을 요구한다.
솔루션	BotDetect CAPTCHA Generator: https://captcha.com/ 웹페이지의 자동제출을 방지하기 위해 CAPTCHA기능을 제공하는 보안솔루션이다. 보안 문자를 이미지 및 음성으로 제공하고 다중언어를 지원한다.

② 실패한 인증시도에 대한 정보를 로깅하여 인증시도 실패가 추적될 수 있게 해야 한다.

반복된 로그인 실패에 대한 로깅 정책을 설정하고 로그 기록으로 허용되지 않은 로그인 시도를 분석할 수 있도록 설계한다.

다. 진단 세부사항

요구사항 ①

로그인 기능 구현 시, 인증시도 횟수를 제한하고 초과된 인증시도에 대해 인증제한 정책을 적용해야 한다.

로그인 기능에 인증시도 횟수를 제한하도록 설계되어 있는지 확인한다.

진단기준

1. 로그인 기능 설계 시 인증시도 횟수가 제한되도록 설계되어 있는가?
2. 초과된 인증시도에 대한 인증제한 정책이 적용되도록 설계되어 있는가?
3. 인증시도 횟수 제한을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 로그인 기능 설계시 인증시도 횟수가 제한되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 로그인 시도 횟수(5회 이내) 제한 정책이 설계 시 반영되었는지 확인	유즈케이스명세서, 클래스설계서 검토
1-3 인증 실패 횟수 정보를 관리하기 위해 영속성이 있는 저장장치(데이터베이스 또는 파일 등)에 인증시도 정보가 저장되도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토

2. 초과된 인증시도에 대한 인증제한 정책이 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
2-1. 인증 실패 제한 횟수 초과 시 계정 잠금, 추가정보 요구 등 제한 정책이 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서 검토

3. 인증시도 횟수 제한을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
3-1 제한된 횟수를 초과한 로그인 시도가 제한되는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인 - 로그인 실패를 발생시킬 수 있는 입력값을 사용 - 실패 시 제한 정책이 적용되는지 확인 - 실패 시 영속성 있는 저장장치에 기록되는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스명세서, 클래스설계서, 단위테스트케이스 등

요구사항 ②

실패한 인증시도에 대한 정보를 로깅하여 인증시도 실패가 추적될 수 있게 해야 한다.

로그인 기능에 인증시도 실패 로그를 기록하도록 설계되어 있는지 확인한다.

진단기준

1. 초과된 인증시도에 대한 로깅 정책이 적용되어 있는가?
2. 초과된 인증시도시 로깅이 정상적으로 실행되는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법

1. 초과된 인증시도에 대한 로깅 정책이 적용되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 사용자ID, 로그인 실패횟수, 로그인 시도 시간, IP주소, 계정 상태정보 등이 로깅정보에 포함되도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서 검토

2. 초과된 인증시도시 로깅이 정상적으로 실행되는지 점검하기 위한 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 초과된 인증시도에 대한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 초과된 인증시도에 대해 로깅이 정상적으로 수행되는지 점검을 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 유즈케이스명세서, 프로그램명세서, 클래스설계서, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	반복된 인증시도 제한 기능 부재

마. 참고자료

① CWE-307, Improper Restriction of Excessive Authentication Attempts, MITRE, http://cwe.mitre.org/data/ definitions/307.html

② 2013 OWASP Top 10 - A2 Broken Authentication and Session Management, OWASP, https://www.owasp. org/index.php/Top_10_2013

③ 2016 OWASP Application Security Verification Standard, OWASP, Authentication Verification Require-ments, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_ Project

④ Authentication Cheat Sheet Prevent Brute Force Attacks, OWASP, http://www.owasp.org/index.php/Au-thentication_Cheat_Sheet#Prevent_Brute-Force_Attacks

⑤ WASC Insufficient Anti-automation, WASC, http://projects.webappsec.org/w/page/13246938/Insuffi-cient%20Anti-automation

2.3 비밀번호 관리

유형	보안기능
설계항목	비밀번호 관리
설명	생성규칙, 저장방법, 변경주기 등 비밀번호 관리정책별 안전한 적용방법을 설계한다.
보안대책	① 비밀번호 설정 시 한국인터넷진흥원『비밀번호 선택 및 이용 안내서』의 비밀번호 보안 지침을 적용 한다. ② 네트워크로 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화하거나 암호화된 통신 채널을 이용해야 한다. ③ 비밀번호 저장 시, 솔트가 적용된 안전한 해시함수를 사용해야 하며 비밀번호에 대한 해시는 서버에서 실행되도록 해야 한다. ④ 비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다. ⑤ 비밀번호 관리 규칙을 정의해서 적용해야 한다.

가. 취약점 개요

사례1 : 취약한 비밀번호 사용

회원가입 시 안전한 비밀번호 생성규칙이 적용되지 않아서 취약한 비밀번호로 회원가입이 가능할 경우 무차별 대입 공격으로 비밀번호가 누출될 수 있다.

그림 3-19 | 취약한 비밀번호 허용

사례2 : 취약한 비밀번호 복구

비밀번호 복구 메커니즘(아이디/비밀번호 찾기 등)이 취약한 경우 공격자가 불법적으로 다른 사용자의 비밀번호를 획득, 변경, 복구할 수 있다.

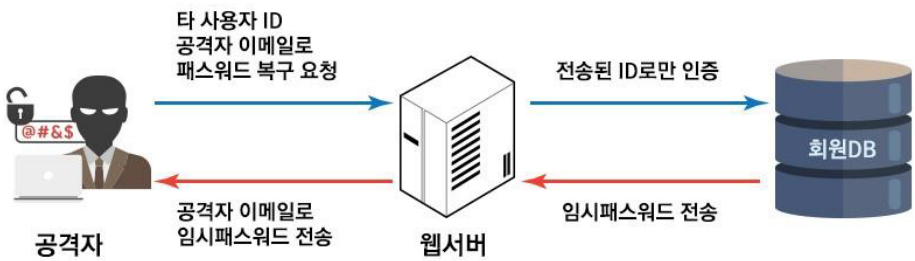


그림 3-20 | 취약한 비밀번호 복구

사례3: 하드코드된 중요정보

프로그램 코드 내부에 비밀번호를 하드 코딩하여 내부 인증에 사용하거나 외부 컴포넌트와 통신을 하는 경우, 관리자용 계정 정보가 노출될 수 있어 위험하다. 또한, 코드 내부에 하드코드된 비밀번호가 인증실패를 발생시키는 경우, 시스템 관리자가 그 실패의 원인을 파악하기 쉽지 않다.

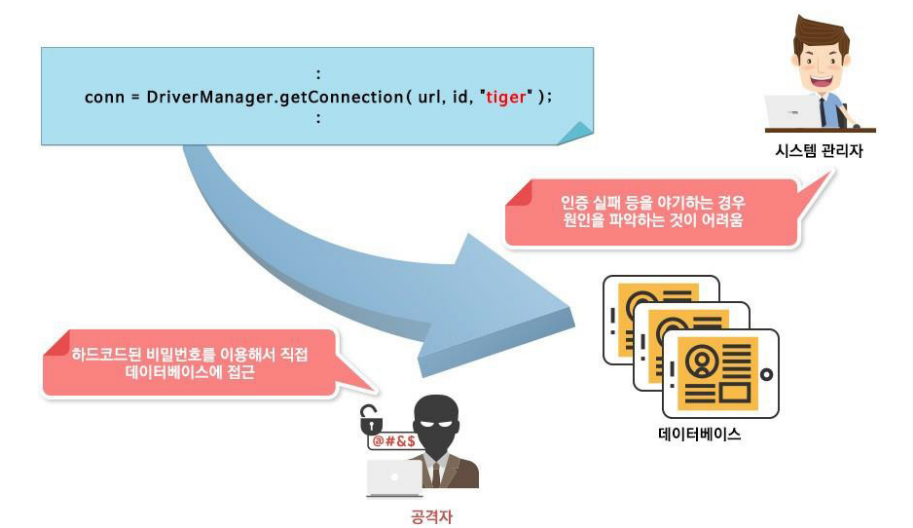


그림 3-21 | 하드코드된 중요정보

나. 설계시 고려사항

① 비밀번호 설정 시 한국인터넷진흥원의 『패스워드 선택 및 이용 안내서』의 안전한 비밀번호 생성규칙을 적용한다.

표 3-13 비밀번호 설정규칙

구분	안전한 비밀번호	안전하지 않은 비밀번호
설명	• 두 종류 이상의 문자 구성과 8자리 이상의 길이로 구성된 문자열 ※ 문자 종류는 알파벳 대·소문자, 특수문자, 숫자 등 4가지 • 10자리 이상의 길이로 구성된 문자열 ※ 숫자로만 구성할 경우 취약할 수 있음	• 특정 패턴을 갖는 비밀번호 ※ 동일한 문자의 반복(예, '123123') ※ 키보드 상 연속한 위치에 존재하는 문자열(예, 'qwerty') ※ 숫자가 제일 앞이나 뒤에 오는 구성(예, '1security') • 제3자가 알 수 있는 개인정보를 바탕으로 구성 ※ 가족, 이름, 생일, 주소, 휴대전화 번호를 포함하는 비밀번호 • 이용자 ID를 이용 ※ 이용자 ID가 'kisa'일 경우 비밀번호를 'kisa1' 등으로 설정 • 한글, 영어 등 사전적 단어로 구성 (예, '바다나라', 'love12' 등) • 특정 인물이나 널리 알려진 단어를 포함하는 비밀번호 ※ 사이트, 기업명, 연예인 이름 등의 특정 명칭을 의미 • 숫자와 영문자를 비슷한 문자로 치환한 형태로 구성 ※ 영문자 'O'을 숫자'0', 영문자 'l'를 숫자 '1'로 치환 등 • 기타 ※ 시스템에서 초기 설정되거나 예제로 제시된 비밀번호 ※ 한글의 발음을 영문으로, 영문단어의 발음을 한글로 변경한 형태의 비밀번호

[표 3-14]와 같은 솔루션 사용을 고려하여 사용자가 안전한 비밀번호를 사용하도록 유도한다.

표 3-14 비밀번호 안전성 검증 솔루션

개발환경	활용 가능한 프레임워크 또는 라이브러리
PHP, JSP, ASP	비밀번호 안전성 검증 S/W : http://seed.kisa.or.kr/iwt/ko/sup/EgovSafePwdLb. do 웹 사이트의 회원가입, 비밀번호 변경 및 로그인 메뉴에서 사용자가 입력한 비밀번호에 대한 안전성을 검사하여 그 결과를 알려주는 무료 S/W이다. 기본 알고리즘은 C언어로 개발되었으나 PHP 확장기능, JNI기능을 활용하여 다양한 서버측 언어로 이용가능하다.

② 네트워크로 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화하거나 암호화된 통신 채널을 이용해야 한다.

웹브라우저와 같은 클라이언트와 웹서버 간의 통신, 서버와 서버 간의 통신 등 인터넷과 같은 공중망 환경에서는 비밀번호와 같은 중요정보를 송·수신하는 경우 보호대책이 필요하다. 이러한 보호대책으로 TLS, VPN 등과 같은 다양한 통신 암호기술을 적용할 수 있다.

시스템관리자 및 보안관리자는 TLS를 적용하거나 관련 솔루션을 도입할 때 제품이 표준에 부합하는지, 상호호환성을 보장하는지, 검증된 제품인지, 오픈소스를 이용하는지 등을 확인해야 한다.

[표 3-15]는 NIST에서 제정한 TLS버전별로 안전하게 이용할 수 있는 암호화알고리즘 구성이다.

표 3-15 TLS 버전별 안전한 암호화 알고리즘

암호화방식		
구분	알고리즘 (TLS_키교환알고리즘_WITH_암호알고리즘_메시지인증알고리즘)	서명
TLS 1.2 서버	TLS_RSA_WITH_AES_256_GCM_SHA384	DSA, ECDSA
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	
TLS 서버	TLS_RSA_WITH_AES_256_CBC_SHA	DSA, ECDSA
	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	
	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	

③ 비밀번호 저장 시, 솔트가 적용된 안전한 해시함수를 사용해야 하며, 해시는 서버에서 실행되도록 해야 한다.

비밀번호는 복호화 되지 않는 일방향 해시함수를 사용해서 암호화하여 저장해야 한다.

[참고 3] 일방향 해시함수

> 일방향 해시함수는 수학적인 연산으로 원본 메시지를 변환하여 암호화된 메시지인 다이제스트를 생성한다. 원본 메시지를 알면 암호화된 메시지를 구하기는 쉽지만 암호화된 메시지로는 원본 메시지를 구할 수 없어야 하며 이를 '일방향성'이라고 한다.

[참고 4] 일방향 해시함수의 문제점

일부에서는 SHA-256과 같은 해시 함수를 사용해 비밀번호를 암호화하여 저장하고 인증 요청 시, 저장된 값과 비교하는 것만으로 충분한 암호화 메커니즘을 적용했다고 생각하지만, 해시 함수는 동일한 메시지는 동일한 다이제스트로 생성되는 구조이므로 무차별 대입으로 원본 문자열의 인식 가능성(recognizability)의 문제가 존재하며, 해시함수의 빠른 처리 속도 덕분에 공격자도 매우 빠른 속도로 임의의 문자열을 다이제스트 할 수 있는 문제점을 가질 수 있다.

이러한 문제점을 해결하기 위해 솔트(salt)값을 추가하여 해시함수를 실행한다. 솔트(salt)는 일방향 해시 함수에서 다이제스트를 생성할 때 추가되는 바이트 단위의 임의의 문자열이다. 예를 들어 비밀번호가 "!t0Et67d3" 이라면 여기에 랜덤하게 생성된 솔트인 “rG7f32-1dYjfgfd-9F3fgd-l4fGdg-f4Tmlf”를 추가해 다이제스트를 생성하면, 공격자가 “!t0Et67d3”의 다이제스트를 알아내더라도 솔트 값이 추가된 다이제스트를 대상으로 비밀번호 일치 여부를 확인하는 것이 어렵게 된다.

솔트와 비밀번호의 다이제스트를 데이터베이스에 저장하고, 사용자가 로그인할 때 입력한 비밀번호를 해시하여 일치 여부를 확인하므로, 즉 모든 비밀번호가 고유의 솔트를 갖고 솔트의 길이가 32바이트 이상이면 솔트 값이 노출되지 않는 이상 다이제스트를 추측하기 어렵다.

④ 비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다.

비밀번호 변경은 주기적인 변경과 분실 시 재설정으로 나누어 볼 수 있다.

(ㄱ) 비밀번호 변경

사용자 및 관리자는 안전한 비밀번호 관리를 위해 주기적으로 비밀번호를 변경하여 비밀번호의 노출위협을 최소화하여야 한다. 사용자는 자신의 비밀번호가 제3자에게 노출되었을 경우 즉시 새로운 비밀번호로 변경해야 한다. 비밀번호 변경 시 이전에 사용하지 않은 새로운 비밀번호로 변경해야 하며, 이전의 비밀번호와 연관성이 없어야 한다.

(ㄴ) 비밀번호 재설정

비밀번호를 잊어버렸거나 분실하는 경우 비밀번호 재설정이 필요하다. “비밀번호 찾기” 기능 구현 시 I-pin 인증, 휴대전화 인증, 질의답변 검증 등으로 비밀번호 재설정 권한을 확인하고 회원가입 시 등록된 이메일 주소를 이용하여 비밀번호를 재설정할 수 있는 링크를 전송한다. 사용자는 해당 링크를 클릭하여 자신의 비밀번호를 재설정할 수 있도록 설계한다.

⑤ 비밀번호 관리 규칙을 정의해서 적용해야 한다.

안전한 비밀번호 관리를 위해 다음과 같은 항목을 고려한다.

(ㄱ) 변경주기

비밀번호는 3개월(또는 6개월)마다 주기적으로 변경하도록 한다.

(ㄴ) 만료기간 설정

일정기간 시스템 사용자에 대해서는 비밀번호 만료기간을 설정한다.

사용자 정보를 저장하는 DB 테이블에 개인정보 변경주기를 추가한 뒤 일단위로 해당 필드가 업데이트 되도록 한다. 비밀번호 기간이 만료되면 로그인시 사용자에게 비밀번호 변경을 요청하고, 비밀번호 변경시 개인정보 변경주기를 초기화하도록 설계한다.

(ㄷ) 성공한 로그인 시간 관리

마지막으로 성공한 로그인 시간 정보를 관리해야 한다.

사용자 테이블에 마지막으로 로그인한 시간정보를 저장하고 사용자에게 알림으로써 계정도용 여부를 점검 할 수 있도록 개발가이드 구현단계를 작성한다.

[참고 5] 비밀번호 관리 주기

A. **비밀번호 생성**
개인 비밀번호는 사용자가 직접 생성하고 그룹 비밀번호는 그룹의 장이 생성하여 구성원들에게 안전한 방법으로 전달한다.

B. **비밀번호 사용**
비밀번호는 제 3자에게 노출되지 않도록 해야 하며, 자신의 비밀번호와 관련된 정보 및 힌트를 제공하지 않아야 한다. 비밀번호 변경주기는 3개월(또는 6개월)이다. 시스템 및 소프트웨어의 초기 비밀번호는 설치시 즉시 변경해야 한다.

C. **비밀번호 폐기**
비밀번호는 사용용도가 끝나거나 사용주기가 지난 경우 폐기한다. 인증 비밀번호는 시스템 담당자가 사용자 계정의 삭제와 함께 폐기하고, 암호화 비밀번호는 사용자가 직접 폐기한다.

다. 진단 세부사항

요구사항 ①

비밀번호를 설정할 때 한국인터넷진흥원『패스워드 선택 및 이용 안내서』의 비밀번호 설정규칙을 적용해야 한다. 조합규칙, 길이 등 비밀번호가 안전하게 설정되도록 설계되어 있는지 확인한다.

진단기준

1. 비밀번호 설정 규칙이 안전하게 정의되어 있는가?
2. 비밀번호 설정 규칙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 비밀번호 설정 규칙이 안전하게 정의되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 비밀번호 설정 규칙 정의 시 "문자구성 및 길이조건", "특정 정보 이용 및 패턴 조건", "기타조건"과 같은 안전한 비밀번호 설정규칙이 설계에 적용되었는지 확인 - 길이 : 조합 규칙의 복잡도에 따라 최소 8자리 이상 - 조합 규칙 : 숫자, 영 대·소 문자, 특수문자 포함 - 제한 문자열 : 전화번호, 동일 문자열 반복 제한 등 ※ 상세 내용은 패스워드 선택 및 이용 안내서(한국인터넷진흥원) 참조	요구사항정의서, 유즈케이스설계서, 프로그램명세서, 클래스설계서 검토

2. 비밀번호 설정규칙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 비밀번호 설정규칙을 위배하는 입력값과 그 예상 결과를 포함하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 단위테스트계획서 등

요구사항 ②

네트워크로 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화하거나 암호화된 통신 채널을 이용해야 한다.

네트워크로 비밀번호 전송 시 암호화 또는 암호화 채널을 사용하도록 설계되어 있는지 확인한다.

진단기준

1. 네트워크로 전송되는 비밀번호가 안전한 암호 알고리즘으로 암호화되거나, 안전한 암호화된 통신 채널을 사용하도록 설계되어 있는가?
2. 네트워크로 전송되는 비밀번호의 암호화를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 네트워크로 전송되는 비밀번호가 안전한 암호 알고리즘으로 암호화되거나, 안전한 암호화된 통신 채널을 사용하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 회원가입, 로그인 등 네트워크로 비밀번호가 전송되는 기능이 분류되어 있는지 확인	요구사항정의서, 유즈케이스설계서, 프로그램명세서, 클래스설계서 검토
1-3 비밀번호를 암호화해서 전송하는 경우, 안전한 암호 알고리즘을 사용하여 암호화하도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서 검토
1-4 비밀번호를 암호화된 통신채널을 이용하여 전송하는 경우, TLS, VPN 등 안전한 통신 채널을 사용하도록 설계되어 있는지 확인	아키텍처설계서 검토

2. 네트워크로 전송되는 비밀번호의 암호화를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 네트워크 구간의 패킷 스니핑 등으로 비밀번호의 암호화 전송을 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스명세서, 개발가이드, 프로그램명세서, 단위테스트케이스 등

요구사항 ③

비밀번호 저장 시, 솔트가 적용된 안전한 해시함수를 사용해야 하며, 해시함수 실행은 서버에서 해야 한다.

비밀번호 저장 시 해시함수를 사용하도록 설계하고, 사용할 해시함수와 솔트 값을 포함한 해시함수 사용 방법이 명시되어 있는지 확인한다.

진단기준

1. 비밀번호 저장 또는 변경시 솔트가 적용된 안전한 해시함수를 이용하여 암호화 하도록 설계되어 있는가?
2. 해시함수를 이용한 비밀번호 암호화가 서버에서 실행되도록 설계되어 있는가?
3. 저장된 비밀번호의 안전성을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 비밀번호 저장 또는 변경시 솔트가 적용된 안전한 해시함수를 이용하여 암호화하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 비밀번호를 암호화하기위해 SHA-2 계열(SHA-224, SHA-256, SHA 384, SHA-512)의 해시함수를 사용하는지 확인	요구사항정의서, 아키텍처설계서, 유즈케이스설계서, 클래스설계서 검토
1-3 해시함수 사용시 솔트값(랜덤하게 생성하여 사용)을 사용하도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토
1-4 솔트값은 데이터 암·복호화에 사용되는 암호키가 저장되는 장소에 저장되도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토

2. 해시함수를 이용한 비밀번호 암호화가 서버에서 실행되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
2-1 해시함수를 이용해서 일방향 암호를 서버에서 수행하는 프로그램이 설계되어 있는지 확인	유즈케이스설계서, 프로그램명세서, 클래스설계 검토

3. 저장된 비밀번호의 안전성을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
3-1 비밀번호 크랙도구를 이용하여 DB에 저장된 비밀번호의 안전을 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스명세서, 프로그램명세서, 클래스 설계서, 단위테스트케이스 등

요구사항 ④

비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다.

비밀번호를 주기적으로 변경하고, 비밀번호 변경 및 비밀번호 분실로 인한 재설정 시 안전하게 변경 절차가 이루어지도록 설계되어 있는지 확인한다.

진단기준

1. 비밀번호를 안전하게 변경할 수 있도록 설계되어 있는가?
2. 비밀번호를 안전하게 재설정 할 수 있도록 설계되어 있는가?
3. 비밀번호 변경 또는 재설정이 안전하게 수행되는지를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 비밀번호를 안전하게 변경할 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 비밀번호 변경 시 기존 비밀번호를 확인하도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서, 유즈 케이스설계서, 클래스설계서 검토

진단방법	관련산출물 검토예시
1-3 안전한 비밀번호 설정규칙이 적용되도록 설계되어 있는지 확인	유즈케이스명세서, 클래스설계서, DB 설계서 검토

2. 비밀번호를 안전하게 재설정 할 수 있도록 설계되어 있는가?

진단방법	관련산출물 검토예시
2-1 비밀번호 재설정 요청 시, 안전한 본인인증 절차를 거치도록 설계되어 있는지 확인 - 본인 휴대전화인증 - I-pin인증 - 공인인증서인증 - 신용카드인증 등	유즈케이스설계서, 프로그램명세서, 클래스설계서 검토
2-2 본인 인증 후 비밀번호 재설정 페이지 연결방식이 안전하게 설계되어 있는지 확인 - 회원가입 시 등록한 이메일을 이용한 재설정페이지 전송 등	요구사항정의서, 아키텍처설계서, 유즈케이스설계서, 클래스설계서 검토

3. 비밀번호 변경 또는 재설정이 안전하게 수행되는지를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
3-1 비밀번호 변경 시 권한이 있는 사용자만 비밀번호 변경이 가능한지를 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
3-2 비밀번호 재설정 시 본인인증 절차를 우회하거나 비밀번호 재설정 경로로 직접 접근하여 비밀번호 재설정이 가능한지를 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

• 관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스설계서,DB설계서, 클래스설계서, 개발가이드, 프로그램명세서, 단위테스트계획서 등

요구사항 ⑤

비밀번호 관리 규칙을 정의해서 적용해야 한다.

비밀번호 변경주기, 만료기간 설정, 성공한 로그인 시간 관리를 포함하여 비밀번호 관리규칙이 설계되어 있는지 확인한다.

진단기준

1. 비밀번호 관리 규칙이 설계되어 있는가?
2. 비밀번호 관리 규칙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 비밀번호 관리 규칙이 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 비밀번호 만료기간이나 변경 주기에 대한 정책이 정의되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-3 비밀번호 만료기간이나 변경주기 정책을 관리할 수 있도록 데이터베이스와 프로그램 기능이 설계되어 있는지 확인 - 비밀번호 최종변경일, 비밀번호 만료기간, 변경주기 등의 정보 관리	프로그램명세서, 유즈케이스설계서, 클래스설계서, DB설계서 검토

2. 비밀번호 관리 규칙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 비밀번호 만료기간이나 변경주기 정책이 수립되어 있는 경우, 해당 정책이 안전하게 적용되었는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 유즈케이스설계서, 프로그램명세서, 클래스설계서, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	하드코드된 중요정보
보안기능	취약한 비밀번호 허용

마. 참고자료

① CWE-521 Weak Password Requirements, MITRE,
http://cwe.mitre.org/data/definitions/521.html

② 2013 OWASP Top 10-A2 Broken Authentication and Session Management, OWASP,
https://www.owasp.org/ index.php/Top_10_2013

③ 2016 OWASP Application Security Verification Standard, OWASP, Authentication Verification Requirements,
http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

④ WASC Threat Classification Abuse of Functionality, WASC,
http://projects.webappsec.org/w/page/13246913/Abuse%20of%20Functionality

⑤ WASC Threat Classification Insufficient Password Recovery, WASC,
http://projects.webappsec.org/w/page/13246942/Insufficient%20Password%20Recovery

⑥ 암호이용 안내서, 한국인터넷진흥원,
http://www.kisa.or.kr/jsp/common/downloadAction.jsp?bno=259&dno=31&fseq=1

⑦ Forgot Password Cheat Sheet, OWASP,
http://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

⑧ Password Storage Cheat Sheet, OWASP,
http://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

2.4 중요자원 접근통제

유형	보안기능
설계항목	중요자원 접근통제
설명	중요자원(프로그램 설정, 민감한 사용자 데이터 등)을 정의하고, 정의된 중요자원에 대한 신뢰할 수 있는 접근통제 방법(권한관리 포함) 설계 및 접근통제 실패 시 처리방법을 설계한다.
보안대책	① 중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다. ② 중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다. ③ 관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.

가. 취약점 개요

사례1 : 관리자 페이지 노출

관리자페이지가 인터넷으로 접근 가능할 경우, 해당 페이지는 공격자의 주 공격대상으로 SQL삽입, 무차별 대입 공격 등 다양한 형태의 공격 빌미를 제공하게 되는 취약점이다.

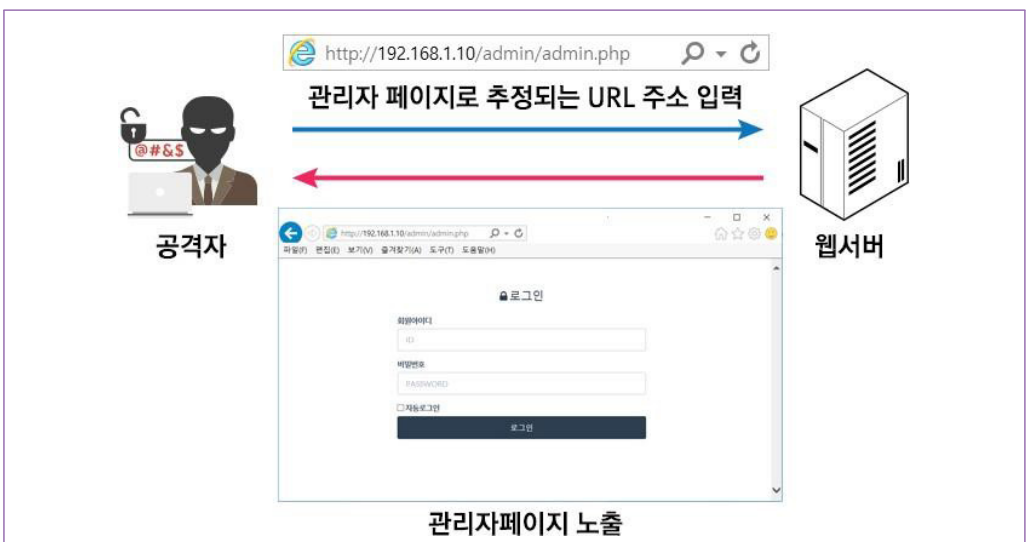


그림 3-22 | 관리자페이지 노출

사례2 : SSI 삽입

SSI(Server-side Includes)는 HTML 문서 내 변수 값으로 입력된 후, 이를 서버가 처리하게 되는데, 이 때 SSI 삽입 명령문이 수행되어 서버 데이터 정보가 누출되는 취약점이다.

그림 3-23 | SSI 인젝션

사례3 : 부적절한 인가

프로그램이 모든 가능한 실행경로에 대해서 접근제어를 검사하지 않거나 불완전하게 검사하는 경우, 공격자는 접근 가능한 실행경로로 정보를 유출할 수 있는 취약점이다.

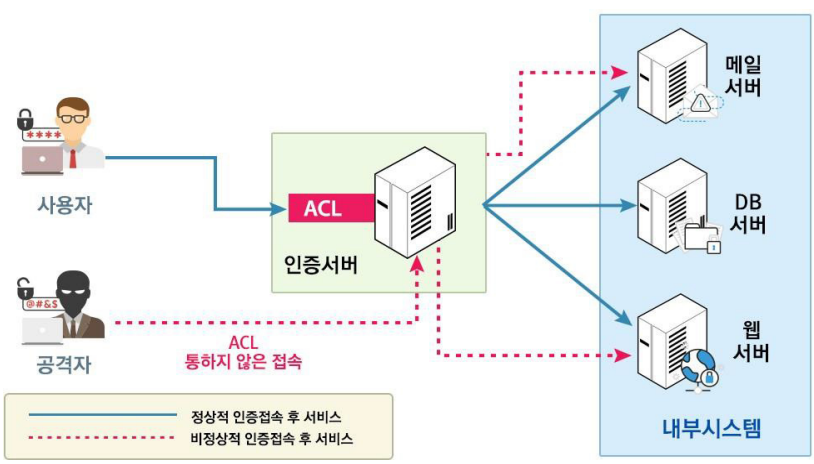


그림 3-24 | 부적절한 인가

사례4 : 중요자원에 대한 잘못된 권한 설정

소프트웨어가 중요한 자원에 대하여 읽기, 수정 등의 권한을 의도하지 않게 허가할 경우 권한을 갖지 않은 사용자가 해당 자원을 사용하게 될 수 있는 취약점이다.

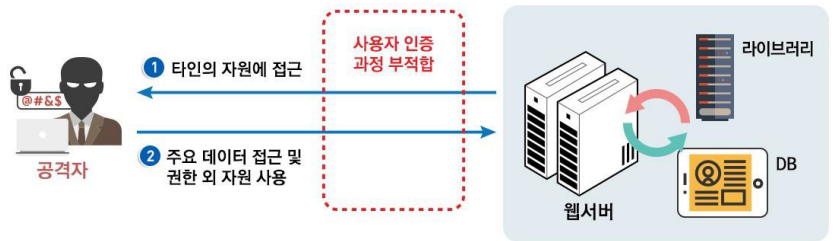
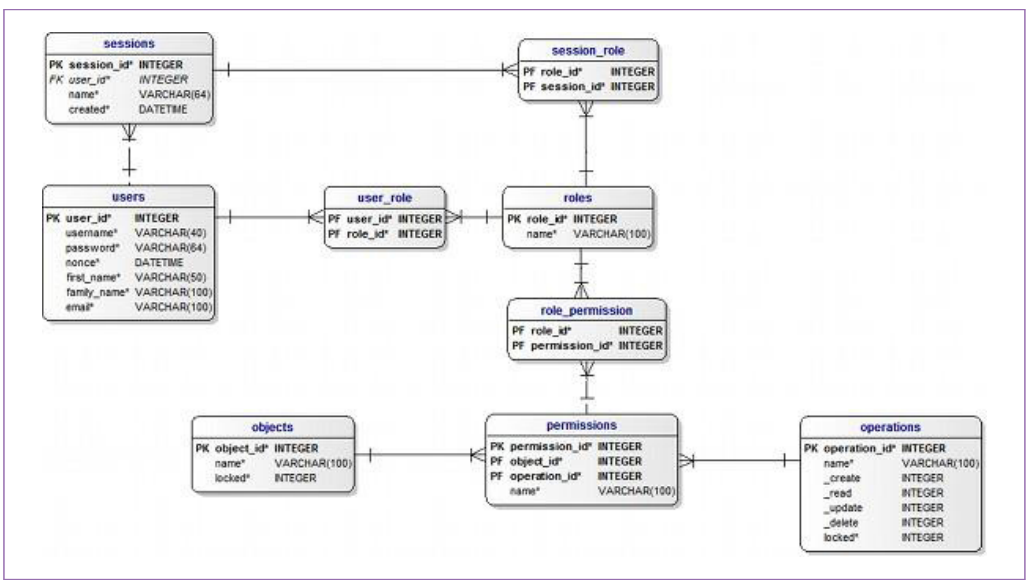


그림 3-25 | 중요한 자원에 대한 잘못된 권한 설정

나. 설계시 고려사항

RBAC(Role Based Access Controll : 역할기반 접근제어) 모델을 사용하여 기업, 정부 등 다수의 사용자와 정보객체들로 구성된 조직체계에서 사용자에게 할당된 역할을 기반으로 권한을 부여하도록 설계한다.

[예시1] RBAC 데이터모델



접근제어목록(Access Control List)을 구성하여 자원에 대한 접근 권한을 설정한다.

[예시2] 접근제어목록

파일	접근제어목록
파일1	(대표, 임원1, 임원2, rw-)
파일2	(대표, 임원1, 사원1, 사원2, rwx)
파일3	(사원3, r-x)(사원4, r–)
파일4	(*, r–)

예를들면, Spring Security 프레임워크 사용시 ACL 모듈을 추가할 수 있다. 다음과 같은 세 가지 ACL 관련기능을 애플리케이션에 적용하여 객체에 대한 접근제어를 구현할 수 있다.

1. 모든 도메인 객체에 대한 ACL엔트리를 효과적으로 검색하고 수정한다.

2. 메서드 호출에 앞서, 각 사용자가 객체에 대해 특정 작업을 수행할 권한이 있는지 검증한다.

3. 메서드 호출이 끝난 후, 각 사용자가 객체(또는 반환되는 객체)에 대해 특정 작업을 수행할 권한이 있는지 검증한다.

① 중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다.

- 중요자원에 대한 접근권한을 최소권한으로 설정한다.
- 중요자원에 대한 접근통제 정책을 설정하고, 사용자별 또는 그룹별 접근을 체크한다.

중요자원(파일, 프로세스, 메모리, 데이터베이스와 같은)에 대한 접근을 통제하기 위해 ACL이나 RBAC을 적용하도록 설계한다. 접근통제 정책을 수립할 때는 최소권한[6]의 원칙과 권한 분리[7] 정책에 따라 자원에 대한 권한을 할당하고, 자원에 대한 접근은 요구조건을 충족할 때만 허가하도록 설계해야 한다.

② 중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다.

중요기능에 대한 접근통제는 소프트웨어를 익명·일반·특권사용자와 관리자 영역으로 구분하여 역할기반 접근통제(RBAC) 정책 및 비즈니스 로직에 따라 접근통제가 실시되도록 다음과 같은 조건에 따라 설계한다.

6) 최소권한 : 해당 작업을 수행하기 위한 최소한의 권한부여
7) 권한분리: 작업수행에 대한 권한을 분리하여 권한의 독점을 방지하는 것 ex. 암호키관리/암호키변경, 개발/운영

- 중요기능에 대한 접근 권한은 최소권한으로 설정한다.
- 중요기능에 대한 접근통제 정책을 설정하고, 사용자별 또는 그룹별 접근을 체크한다.
- 프로그램에서 사용자 또는 자원에 대한 권한의 할당, 수정, 확인, 검사를 수행하여 의도치 않은 범위의 권한을 획득하지 않도록 한다.
- 파라미터 변조로 인증이 올바르게 수행되지 않을 수 있으므로, 파라미터가 변조되지 않았는지 확인하는 절차를 구현한다.
- 상위권한을 사용해 수행되는 기능을 구현해야 하는 경우, 권한상승은 가능한 가장 마지막에 수행하고 수행종료 즉시 원상 복귀한다.

③ 관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.

- 관리자 페이지의 URL은 쉽게 추측할 수 없도록 설정한다.
- 관리자 페이지 접속 시 암호화 통신 채널(TLS 등)을 사용해야 한다.
- 관리자페이지에 접속 가능한 IP를 설정하고 80번이 아닌 별도의 포트를 사용하도록 한다.
- 관리자페이지 접속 시 추가인증을 요구하도록 해야 한다.

중앙 집중화된 접근제어를 제공하는 라이브러리나 프레임워크를 사용하여 각 종류의 자원에 대한 접근을 보호할 수 있다.

표 3-16 접근제어를 제공하는 프레임워크 및 라이브러리

개발환경	활용 가능한 프레임워크 또는 라이브러리
1Java	Spring Security: http://spring.io/ 인증, 인가 등 기업애플리케이션의 보안기능을 제공하는 Java/Java EE 프레임워크이며, 접근제어 리스트(ACL) 및 역할기반접근제어(RBAC)로 접근통제를 제공한다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
ASP.NET	.NET Framework: https://www.microsoft.com/net 윈도우 프로그램 개발 및 실행환경을 제공하는 소프트웨어 프레임워크로써 접근제어리스트 및 역할기반접근제어 기능을 제공한다.
PHP	PHP-RBAC: http://phprbac.net/ 역할기반접근제어를 구현할 수 있도록 돕는 인가기능 라이브러리이다. Apache Soft-ware License v2.0 정책에 따라 자유롭게 사용가능하다.

다. 진단 세부사항

요구사항 ①

중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다.

중요자원에 대한 접근통제가 이루어지도록 설계되어 있는지 확인한다.

진단기준

1. 중요자원에 대한 접근통제 정책이 수립되고 적용되도록 설계되어 있는가?
2. 중요자원에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 중요자원에 대한 접근통제 정책이 수립되고 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 중요자원이 식별되어 있는지 확인 - 파일, 프로세스, 메모리, 데이터베이스, 정보 등으로 분류되어 식별	요구사항정의서, 아키텍처설계서 검토
1-3 중요자원에 접근하는 사용자 또는 프로그램이 분류되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 중요자원에 접근하는 사용자 또는 프로그램의 접근권한이 최소권한으로 할당 되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-5 중요자원에 접근하는 사용자 또는 프로그램의 접근권한이 자원별로 분리되어 할당되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-6 중요자원에 대한 접근통제 정책(ACL 또는 RBAC 등)이 설정되고, 접근통제 기능을 구현하는 프로그램이 설계되어 있거나 접근통제 기능을 제공하는 라이브러리를 활용하도록 설계되어 있는지 확인	아키텍처설계서, 프로그램명세서, 유즈케이스설계서, 클래스설계서 검토
1-7 접근통제 기능의 적용방식이 프레임워크의 컴포넌트를 활용하여 적용하는 경우, 중요자원을 사용하는 모든 요청에 적용될 수 있도록 설계되어 있는지 확인	아키텍처설계서 검토
1-8 접근통제 기능의 적용방식이 개별 프로그램에서 적용하는 경우, 대상 프로그램과 통제 기능 호출방식에 대한 코딩규칙이 개발가이드에 반영되어 있는지 확인	개발가이드 검토

2. 중요자원에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 중요자원별로 할당된 접근 권한 외의 작업을 요청하여 접근권한 통제가 안전하게 수행되는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 클래스설계서, 유즈케이스설계서, 단위테스트계획서 등

요구사항 ②

중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다.

중요기능에 대한 접근통제가 설계되어 있는지 확인한다.

진단기준

1. 중요기능에 대한 접근통제 정책이 수립되고 적용되도록 설계되어 있는가?
2. 중요기능에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 중요기능에 대한 접근통제 정책이 수립되고 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 중요기능(ex. 개인정보조회, 변경, 관리자등록, 사용자등급 등록 등)이 분석 단계 산출물에서 식별되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-3 중요기능에 접근하는 사용자 또는 프로그램이 분류되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 중요자원에 접근하는 사용자 또는 프로그램의 접근권한이 최소권한으로 할당 되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토

진단방법	관련산출물 검토예시
1-5 중요자원에 접근하는 사용자 또는 프로그램의 접근권한이 자원별로 분리되어 할당되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-6 중요자원에 대한 접근통제 정책(ACL 또는 RBAC 등)이 설정되고, 접근 통제 기능을 구현하는 프로그램이 설계되어 있거나 접근통제 기능을 제공하는 라이브러리를 활용하도록 설계되어 있는지 확인	아키텍처설계서, 프로그램명세서, 유즈케이스설계서, 클래스설계서 검토
1-7 접근통제 기능의 적용방식이 프레임워크의 컴포넌트를 활용하여 적용하는 경우, 중요자원을 사용하는 모든 요청에 적용될 수 있도록 설계되어 있는지 확인	아키텍처설계서 검토
1-8 접근통제 기능의 적용방식이 개별 프로그램에서 적용하는 경우, 대상 프로그램과 통제 기능 호출 방식에 대한 코딩규칙이 개발가이드에 반영 되어 있는지 확인	개발가이드 검토

2. 중요기능에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 중요기능별로 할당된 접근 권한 외의 작업을 요청하여 접근권한 통제가 안전하게 수행되는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스설계서, 클래스설계서, 개발가이드, 프로그램명세서, 단위테스트계획서 등

요구사항 ③

관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.

관리자 페이지 접근통제 정책이 설계되어 있는지 확인한다.

진단기준

1. 관리자 페이지에 대한 접근통제정책이 적용되도록 설계되어 있는가?
2. 관리자 페이지에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 관리자 페이지에 대한 접근통제정책이 적용되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 관리자페이지를 사용하는 포트가 일반 사용자 페이지와 다른 포트를 사용하고 있는지 설계산출물(ex. 아키텍처설계서)에서 확인	요구사항정의서, 아키텍처설계서 검토
1-3 외부 네트워크로부터 관리자 페이지의 접근이 차단되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 관리자 페이지에 접근 가능한 IP주소가 정의되어 있으며 설계에 반영되었는지 확인	요구사항정의서, 아키텍처설계서 검토
1-5 관리자 페이지 접속시 반드시 암호화된 통신을 사용하도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-6 관리자 페이지에 접속하는 경우 접속자나 접속페이지의 정보가 로깅되도록 개발가이드가 작성되고, 프로그램이 설계되었는지 확인	아키텍처설계서, 프로그램명세서, 유즈케이스설계서, 클래스설계서 검토

2. 관리자 페이지에 대한 접근통제를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 관리자 페이지 접근통제 확인을 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 외부네트워크에서 접근하는 경우 접속 IP주소, 포트번호 확인 등을 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-3 네트워크 스니핑으로 관리자페이지 요청이 암호화되어서 전달되는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 유즈케이스설계서, 프로그램명세서, 클래스설계서, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	부적절한 인가
보안기능	중요한 자원에 대한 잘못된 권한 설정

마. 참고자료

① CWE-285 Improper Authorization, MITRE,
http://cwe.mitre.org/data/definitions/285.html

② 2013 OWASP Top 10 - A7 Missing Function Level Access Control, OWASP,
https://www.owasp.org/index. php/Top_10_2013

③ CWE-732 Incorrect Permission Assignment for Critical Resource, MITRE,
http://cwe.mitre.org/data/defini- tions/732.html

④ 2016 OWASP Application Security Verification Standard, OWASP, Access Control Verification Requirements,
http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

⑤ Access Control Cheat Sheet, OWASP,
http://www.owasp.org/index.php/Access_Control_Cheat_Sheet

⑥ WASC Threat Classification Directory Indexing, WASC,
http://projects.webappsec.org/w/page/13246922/Directory%20Indexing

⑦ Role Based Access Control (RBAC) and Role Based Security, NIST,
http://csrc.nist.gov/groups/SNS/rbac/

⑧ CAPEC-101 Server Side Includes (SSI) Injection, CAPEC,
http://capec.mitre.org/data/definitions/101.html

2.5 암호키 관리

유형	보안기능
설계항목	암호키 관리
설명	암호키 생성, 분배, 접근, 파기 등 암호키 생명주기별 암호키 관리방법을 안전하게 설계한다.
보안대책	① DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다. ② 설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉토리에 보관해야 한다.

가. 취약점 개요

사례1 : 하드코드된 암호키

코드 내부에 암호화 키를 하드코딩하여 사용하여 암호화를 수행하면 암호화된 정보가 유출될 가능성이 높아진다.

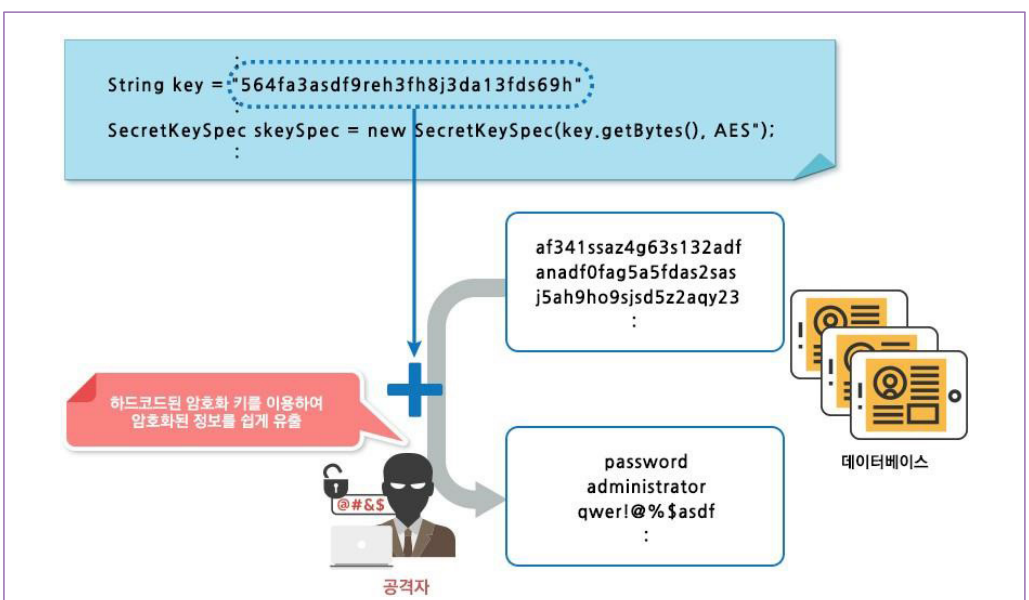


그림 3-26 | 하드코드된 중요정보

사례2 : 주석문 안에 포함된 암호키

주석문 안에 암호키에 대한 설명이 포함된 경우 공격자가 소스코드에 접근할 수 있다면 아주 쉽게 암호키가 노출될 수 있다.

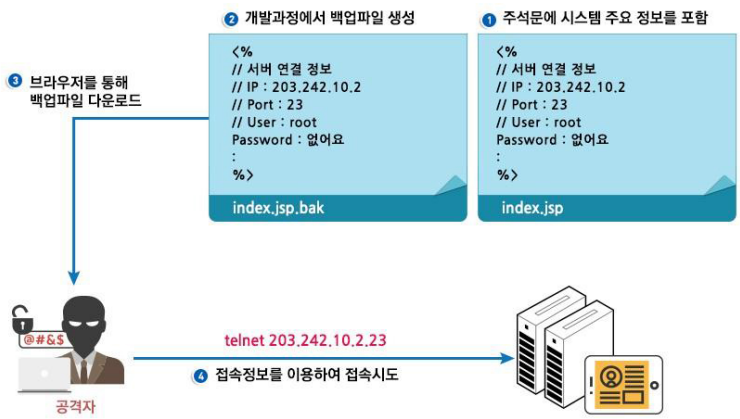


그림 3-27 | 주석문 안에 포함된 시스템 주요정보

나. 설계시 고려사항

① DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호 키 관리 안내서」에서 정의하고 있는 관리 방법을 적용해야 한다.

(ㄱ) 암호키 관리 규칙 생성 시 고려사항

1. DB데이터 암호화에 사용되는 암호키는 데이터가 저장되는 데이터베이스와 물리적으로 분리된 장소에 별도로 보관한다.
2. 암호키를 생성, 분배, 사용, 폐기하는 키의 생명주기관리를 위한 명시적인 암호화 정책을 적용한다.
3. 비밀번호나 암호화키는 메모리에 저장하지 않는다.
4. 비밀번호나 암호키가 메모리에 저장되어야 하는 경우 사용종료 후 메모리를 0으로 초기화 한다.
5. 암호키 생성 및 변경 시 암호키에 대한 백업기능을 구현한다.
6. 암호 알고리즘에 사용되는 키 종류에 따라 사용 유효기한을 설정한다.

(ㄴ) 조직의 보호 목적에 따라 암호키 관리 수준을 지정

NIST[8]에서 제정한 FIPS[9] 140-2[10]의 레벨로써, 조직의 보호목적에 따라 적절히 채택한다.

표 3-17 FIPS 140-2 레벨 분류

레벨	내용
Level 1	암호모듈에 대한 기본적인 보안요구사항만을 충족하여 최소한의 보안을 제공한다.
Level 2	침입자의 불법적인 접근을 방지하고, 침입 이후에 변조를 나타내는 증거를 제공함으로써 물리적인 보안메커니즘을 제공한다.
Level 3	강력한 변조 탐지 및 대응의 일환으로 침입을 감지하면 저장된 키를 삭제한다.
Level 4	암호모듈 외부의 전압이나 온도 등을 감지하여 슈퍼쿨링(Supercooling) 등 환경의 이상 변화시, 암호키를 삭제한다.

(ㄷ) 키 생명주기 기준 암호화 키 관리 프로세스를 구축

키 생성 – 암호키와 비밀번호를 생성, 사용, 관리하는 사람 등을 명시하고 키를 생성하는데 사용하는 프로 그램 등 어떠한 방법으로 생성하는지에 대한 절차를 명시한다.

키 사용 – 암호키와 비밀번호를 어떠한 방법으로 사용하는지에 대한 절차, 생성한 키의 종류에 따른 변경 주기, 인가된 사용자만 키에 접근할 수 있는 접근통제 방법 및 요구사항 등을 명시한다.

키 폐기 – 키의 사용주기가 다 된 경우, 사용 용도가 끝난 경우 등 생성한 키를 폐기하여야 하는 조건을 정하고, 암호키와 비밀번호를 안전하게 폐기하는 절차 및 요구사항 등을 명시한다.

(ㄹ) 키 복구 방안

사용자 퇴사 등으로 인해 사용자 이외의 사람에게 키 복구가 필요한 경우, 암호키는 정보보호 담당자의 관리하에 암호화키 관리대장 등에서 복구하고, 비밀번호는 정보보호 담당자가 임시 비밀번호를 발급하는 등 키 복구에 대한 방안을 마련하도록 한다.

8) 미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)는 국립표준국(NBS, National Bureau of Standards)이라고 알려진 측정 표준 실험실로 미국 상무부 산하의 비규제 기관이다.

9) FIPS(Federal Information Processing Standards)는 미연방 정부 기관 및 정부 계약자를 위한 미국 연방 정부에서 개발하여 공개한 컴퓨터시스템 사용 표준이다.

10) 연방 정보 처리 표준(FIPS) 간행물 140-2 (FIPS PUB 140-2)는 암호화 모듈을 승인하는 데 사용되는 미국 정부 컴퓨터 보안 표준으로 제목은 “암호화 모듈에 대한 보안 요구 사항”이다.

(ㅁ) 암호키 사용 유효기간

암 · 복호화 키의 사용이 일정 기간을 넘은 경우 사용자 인터페이스로 키 사용 기간이 경과했음을 알리고 새로운 키 생성을 권장하도록 설계한다. [표 3-18]는 NIST에서 권고하는 암호키 사용 유효기간이다.

표 3-18 암호키 사용 유효기간(NIST권고안)

키 종류		사용 유효기간	
		송신자 사용기간	수신자 사용기간
대칭키 암호 알고리즘	비밀키	최대 2년	(송신자 사용기간+3년)이하
공개키 암호 알고리즘	암호화 공개키	최대 2년	
	복호화 개인키	최대 2년	
	검증용 공개키	1~3년	
	서명용 개인키	키 크기에 따라 다름	

② 설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉터리에 보관해야 한다.

설정파일 내의 중요정보 암호화에 사용된 암호키는 마스터키를 이용하여 암호화하여 별도의 디렉터리에 보관한다.

다. 진단 세부사항

요구사항 ①

DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다.

진단기준

1. 암호키를 관리하기 위한 정책이 수립되어 있으며, 안전하게 암호키를 관리하기 위한 방법이 설계에 반영되어 있는가?
2. 암호키가 안전하게 관리되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 암호키를 관리하기 위한 정책이 수립되어 있으며, 안전하게 암호키를 관리하기 위한 방법이 설계에 반영되어있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 설정파일(ex. properties, xml 등)에 저장되는 중요정보(DB계정정보, 시스템 중요정보 등)가 암호화되어 저장되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-3 설정파일에 저장된 중요정보의 암·복호화에 사용되는 키는 설정파일과 별도의 위치에 저장되도록 설계 되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 암호키를 안전하게 관리하기 위한 암호키 관리수준을 정의하고 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-5 암호키 생성 및 변경방법을 수립하고, 암호키 백업정책을 정의하고 있는지 확인	요구사항정의서, 아키텍처설계서, 유즈케이스설계서 검토
1-6 암호키 사용기간을 정의하고 사용기간 경과시 대응방법을 정의하고 있는지 확인 - 대칭키 사용기간 : 송신자 2년, (송신자 사용기간 +3년) 이내 - 비대칭키 사용기간: 암호화 공개키 2년, 개인키 2년, 검증용 공개키 1~3년, 서명용 개인키 키 크기에 따라 다름	요구사항정의서, 아키텍처설계서 검토
1-7 프로그램에서 암호키 사용 시 사용 종료후 메모리를 0으로 초기화하도록 코딩 규칙이 정의되어 있는지 확인	개발가이드 검토

2. 암호키가 안전하게 관리되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 암호키가 외부로 유출될 수 있는지를 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

요구사항 ②

설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉토리에 보관해야 한다.

진단기준

1. 설정파일에 저장되는 중요정보가 안전하게 암호화되어 저장되도록 설계되어 있는가?
2. 설정파일의 중요정보가 암호화되어 저장되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 설정파일에 저장되는 중요정보가 안전하게 암호화되어 저장되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 설정파일(ex. properties, xml 등)에 저장되는 중요정보(DB계정정보, 시스템 중요정보 등)가 암호화되어 저장되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-3 설정파일에 저장된 중요정보의 암·복호화에 사용되는 키는 설정파일과 별도의 위치에 저장되도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토

2. 설정파일의 중요정보가 암호화 되어 저장되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 설정파일의 중요정보가 암호화되어 저장되어 있는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토
2-2 설정파일 암호화에 사용된 암호키가 안전한 위치에 저장되어 사용되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트케이스 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	하드코드된 중요정보
보안기능	주석문안에 포함된 시스템 주요 정보

마. 참고자료

① CWE-615 Information Exposure Through Comments, MITRE, http://cwe.mitre.org/data/definitions/615.html

② CWE-321 Use of Hard-coded Cryptographic Key, MITRE, http://cwe.mitre.org/data/definitions/321.html

③ 암호키 관리 안내서, 한국인터넷진흥원, http://www.kisa.or.kr/public/laws/laws3_View.jsp?cPage=1&-mode=view&p_No=259&b_No=259&d_No=83&ST=total&SV=

④ Key Management Cheat Sheet, OWASP, http://www.owasp.org/index.php/Key_Management_Cheat_Sheet

⑤ 2016 OWASP Application Security Verification Standard, OWASP, Cryptography at Rest Verification Re- quirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verificatio n_Stan- dard_Project

⑥ Recommendation for Key Management, NIST SP 800-57 Part1 Revesion4, http://nvlpubs.nist.gov/nistpubs/ SpecialPublications/NIST.SP.800-57pt1r4.pdf

⑦ Use of hard-coded cryptographic key, OWASP, http://www.owasp.org/index.php/Use_of_hard-cod- ed_cryptographic_key

⑧ Cryptographic Storage Cheat Sheet, OWASP, http://www.owasp.org/index.php/Cryptographic_Storage_ Cheat_Sheet

⑨ A basic encryption strategy for storing sensitive data, ITworld, http://www.itworld.com/article/2693828/data-protection/a-basic-encryption-strategy -for-storing-sensitive-data.html

2.6 암호연산

유형	보안기능
설계항목	암호연산
설명	국제표준 또는 검증필 암호모듈로 등재된 안전한 암호 알고리즘을 선정하고 충분한 암호키 길이, 솔트, 충분한 난수 값을 적용한 안전한 암호연산 수행방법을 설계한다.
보안대책	① 대칭키 또는 비대칭키를 이용해서 암·복호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호이용안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다. ② 복호화되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트값을 적용하여 암호화해야 한다. ③ 난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.

가. 취약점 개요

사례1 : 취약한 암호알고리즘 사용

소프트웨어 개발자들은 환경설정 파일에 저장된 비밀번호를 보호하기 위하여 간단한 인코딩 함수를 이용하여 비밀번호를 감추는 방법을 사용하기도 한다. 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수를 사용하면 비밀번호를 안전하게 보호할 수 없다.

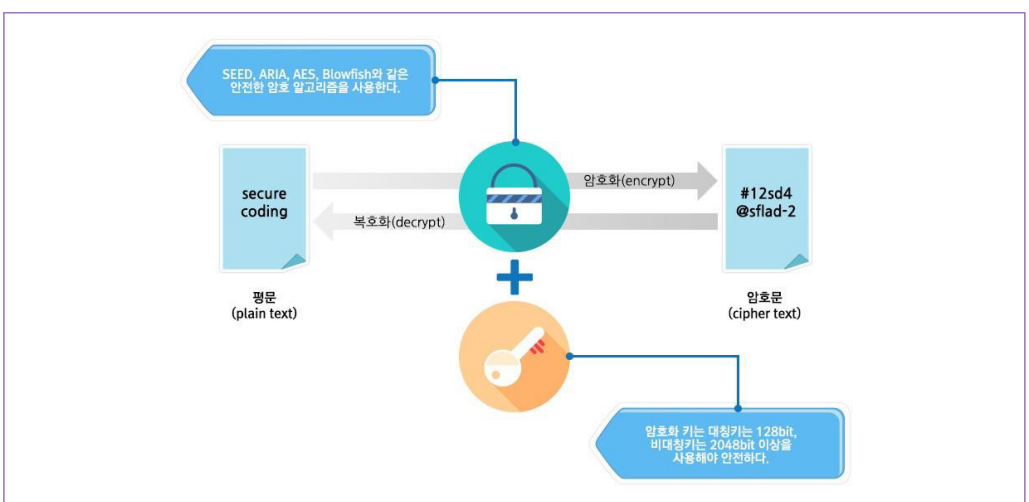


그림 3-28 | 취약한 암호화 알고리즘 사용

사례2 : 충분하지 않은 키 길이 사용

검증된 암호화 알고리즘을 사용하더라도 키 길이가 충분히 길지 않으면 짧은 시간 안에 키를 찾아낼 수 있고 이를 이용해 공격자가 암호화된 데이터나 비밀번호를 복호화 할 수 있게 된다.

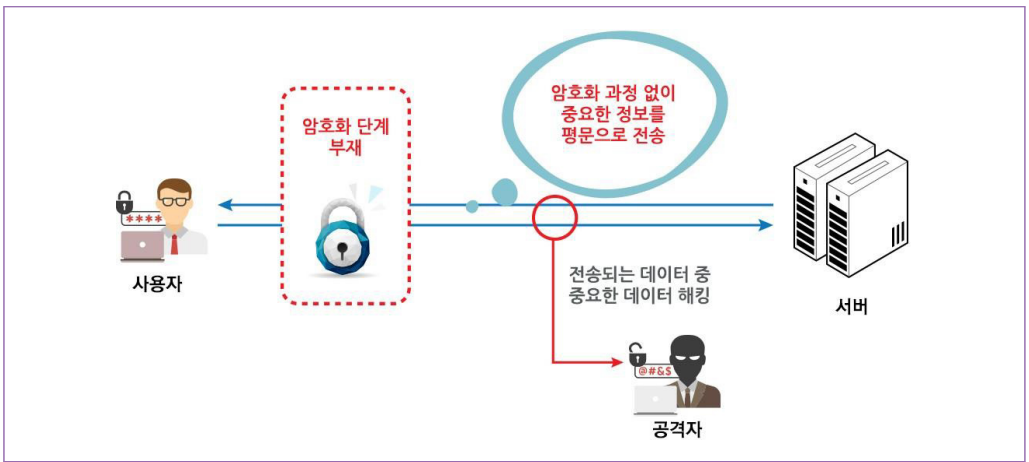


그림 3-29 | 충분하지 않은 키 길이 사용

사례3 : 적절하지 않은 난수 값 사용

예측 가능한 난수를 사용하는 것은 시스템의 보안약점을 유발한다. 예측 불가능한 숫자가 필요한 상황에서 예측 가능한 난수를 사용한다면, 공격자는 SW에서 생성되는 다음 숫자를 예상하여 시스템을 공격하는 것이 가능하다.

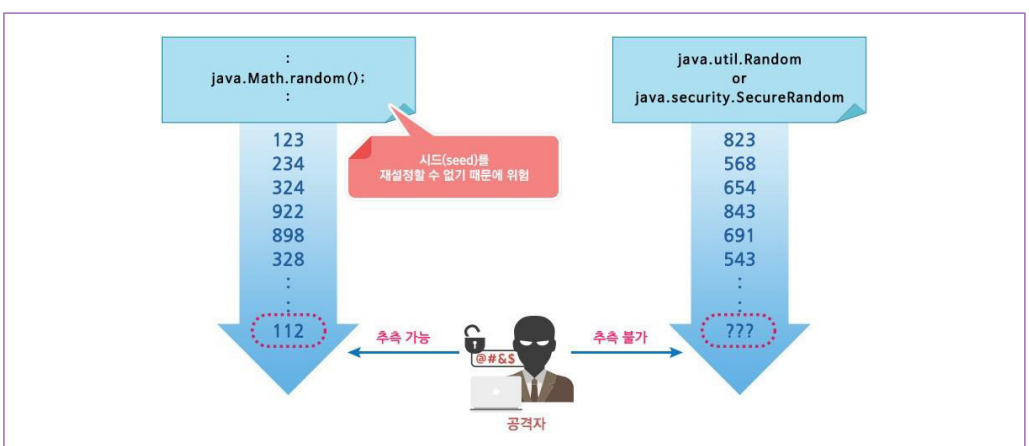


그림 3-30 | 적절하지 않은 난수 값 사용

사례4 : 솔트 없이 사용하는 일방향 해시함수

비밀번호 저장시 일방향 해시함수의 성질을 이용하여 비밀번호의 해시값을 저장한다. 만약 비밀번호를 솔트(Salt)없이 해시하여 저장한다면, 공격자는 레인보우 테이블과 같이 가능한 모든 비밀번호에 대해 해시값을 미리 계산하고, 이를 이용한 전수조사로 비밀번호를 찾을 수 있게 된다.

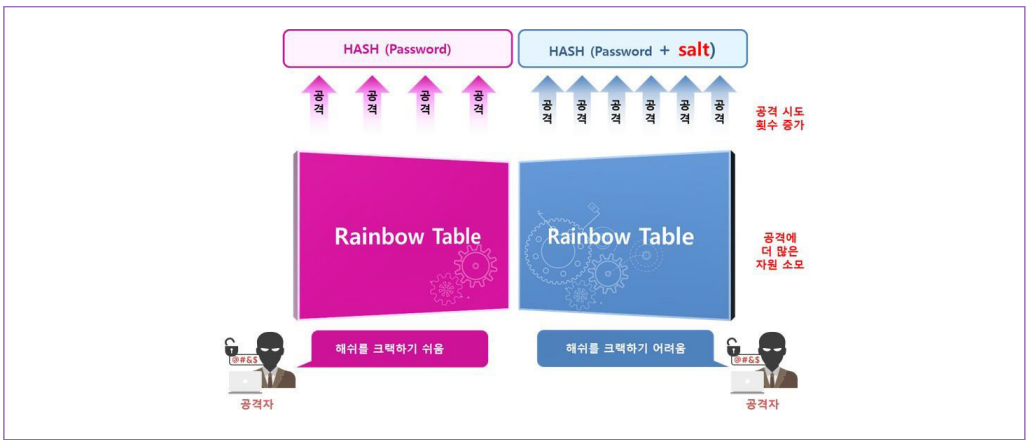


그림 3-31 | 솔트 없이 일방향 해쉬 함수 사용

나. 설계시 고려사항

① 대칭키 또는 비대칭키를 이용해서 암·복호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호 알고리즘 및 키 길이 이용 안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다.

표 3-19 NIST 알고리즘 안전성 유지기간 및 최소 키길이 권고

알고리즘 안전성 보장기간	보안 강도 (비트)	대칭키 알고리즘	비대칭키 알고리즘			타원·회귀 곡선기반 (ex. ECC)
			인수분해 기반 (ex.RSA)	이산대수기반 (ex. KCDSA)		
				공개키	개인키	
~ 10년	80	2TDEA	1024	1024	160	160
11년~30년	112	3TDEA	2048	2048	224	224
30년~	128	AES-128	3072	3072	256	256
	192	AES-192	7680	7680	384	384
	256	AES-256	15360	15360	512	512

표 3-20 국내외 사용 권고 알고리즘

대칭키 암호화 알고리즘	비대칭키 암호화 알고리즘
SEED ARIA-128/192/256 AES-128/192/256 Camelia-128/192/256 Blowfish MISTY1 KASUMI 등	RSA KCDSA(전자서명용) RSAES-OAEP ElGamal ECC ECKCDSA 등

② 복호화되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트 값을 적용하여 암호화해야 한다.

해시함수는 사용 목적에 따라 메시지인증/키 유도/난수 생성용과 단순 해시(메시지 압축)/전자서명용으로 나뉘며, 사용 목적과 보안강도에 따라 선택하여 이용한다.

표 3-21 보안강도에 따른 메시지인증/키유도/난수생성용 해시함수 분류

보안강도(비트)	NIST(미국)	CRYPTREC(일본)	ECRYPT(유럽)	국내
80	SHA-1 SHA-224/256/384/512	SHA-1 SHA-256/384/512 RIPEMD-160	SHA-1 SHA-224/256/384/512 RIPEMD-160 Whirlpool	HAS-160 SHA-1 SHA-256/384/512
112	SHA-1 SHA-224/256 SHA-384/512	SHA-1 SHA-256/384/512 RIPEMD-160	SHA-1 SHA-224/256/384/512 RIPEMD-160 Whirlpool	HAS-160 SHA-1 SHA-256/384/512
128	SHA-1 SHA-224/256 SHA-384/512	SHA-1 SHA-256/384/512 RIPEMD-160	SHA-1 SHA-224/256/384/512 RIPEMD-160 Whirlpool	HAS-160 SHA-1 SHA-256/384/512
192	SHA-256/384/512	SHA-256/384/512	SHA-224/256/384/512 Whirlpool	SHA-256/384/512
256	SHA-256/384/512	SHA-256/384/512	SHA-224/256/384/512 Whirlpool	SHA-256/384/512

표 3-22 보안강도에 따른 단순해시/전자서명용 해시함수 분류

보안강도(비트)	NIST(미국)	CRYPTREC(일본)	ECRYPT(유럽)	국내
80	SHA-1 SHA-224/256/384/512	SHA-1 SHA-256/384/512 RIPEMD-160	SHA-1 SHA-224/256/384/512 RIPEMD-160 Whirlpool	SHA-1 HAS-160 SHA-256/384/512

보안강도(비트)	NIST(미국)	CRYPTREC(일본)	ECRYPT(유럽)	국내
112	SHA-224/256/384/512	SHA-256/384/512	SHA-224/256/384/512 Whirlpool	SHA-256/384/512
128	SHA-256/384/512	SHA-256/384/512	SHA-256/384/512 Whirlpool	SHA-256/384/512
192	SHA-384/512	SHA-384/512	SHA-384/512 Whirlpool	SHA-384/512
256	SHA-512	SHA-512	SHA-512	SHA-512

③ 난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.

FIPS 140-2 인증을 받은 암호모듈의 난수 생성기와 256비트 이상의 시드를 사용하여 난수를 생성한다. 난수의 무작위성을 보장하기 위해 이전 난수생성 단계의 결과를 다음 난수생성 단계의 시드로 사용하는 의사난수생성기를 이용한다.

다. 진단 세부사항

요구사항 ①

대칭키 또는 비대칭키를 이용해서 암호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호 알고리즘 및 키 길이 이용 안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다.

진단기준

1. 안전한 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용하도록 설계되어 있는가?

진단방법

1. 안전한 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토

진단방법	관련산출물 검토예시
1-2 암복호화 기능 설계 시 안전한 암호 알고리즘을 사용하도록 설계되어 있는지 확인 - 대칭키 암호화 알고리즘 : SEED, ARIA, AES 등 - 비대칭키 암호화 알고리즘 : RSA, KCDSA, ECC 등	요구사항정의서, 아키텍처설계서 검토
1-3 암복호화 기능 설계 시 안전한 암호키 길이를 사용하도록 설계되어 있는지 확인 - 대칭키 암호화 알고리즘 키길이 128 비트 이상 - 비대칭키 암호화 알고리즘 키길이 2,048 비트 이상	요구사항정의서, 아키텍처설계서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서 등

요구사항 ③

복호화 되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트값을 적용하여 암호화해야 한다.

진단기준

1. 일방향 암호화 수행을 위한 해시함수 사용 방법이 안전하게 설계되어 있는가?
2. 해시함수로 암호화된 데이터 크랙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 일방향 암호화 수행을 위한 해시함수 사용방법이 안전하게 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 복호화되지 않는 암호화 기능설계시 안전한 암호화 알고리즘을 사용하도록 설계되어 있는지 확인 - SHA-2 계열(SHA-224, SHA-256, SHA-384, SHA-512)	요구사항정의서, 아키텍처설계서 검토
1-3 해시함수의 보안을 강화하기 위해 솔트값(난수발생기를 이용해 생성한 안전한 난수값)을 사용하여 암호화하도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서, DB 설계서 검토

2. 해시함수로 암호화된 데이터 크랙을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 암호화된 데이터를 대상으로 복호화 시도 등을 포함하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, DB설계서, 단위테스트계획서 등

요구사항 ③

난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.

진단기준

1. 난수 생성을 위해 안전한 난수생성 알고리즘을 사용하도록 설계되어 있는가?
2. 생성된 난수의 안전성을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 난수생성을 위해 안전한 난수생성 알고리즘을 사용하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 개발가이드에 안전한 난수 생성알고리즘을 사용하도록 코딩규칙을 정의하고 있는지 확인 - JAVA 프로그램 개발시 java.security.SecureRandom, java.util.Random과 같은 안전한 API 사용 - C 프로그램 개발시 randomize(seed) 함수	개발가이드 코딩규칙 검토

2. 생성된 난수의 안전성을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 생성된 난수가 안전한지를 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트케이스 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	취약한 암호화 알고리즘 사용
보안기능	충분하지 않은 키 길이 사용
보안기능	적절하지 않은 난수 값 사용
보안기능	부적절한 인증서 유효성 검증
보안기능	솔트 없이 일방향 해쉬함수 사용

마. 참고자료

① CWE-327 Use of a Broken or Riscky Cryptographic Algorithm, MITRE, http://cwe.mitre.org/data/defini- tions/327.html

② CWE-326 Inadequate Encryption Strength, MITRE, http://cwe.mitre.org/data/definitions/326.html

③ WASC Insufficient Data Protection Working, WASC, http://projects.webappsec.org/w/page/33923604/Insufficient%20Data%20Protection%20Working

④ 2016 OWASP Application Security Verification Standard, OWASP, Cryptography at Rest Verification Re- quirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Stan- dard_Project

⑤ Cryptographic Storage Cheat Sheet, OWASP, http://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

2.7 중요정보 저장

유형	보안기능
설계항목	중요정보 저장
설명	중요정보(비밀번호, 개인정보 등)를 저장·보관하는 방법이 안전하도록 설계한다.
보안대책	① 중요정보 또는 개인정보는 암호화해서 저장해야 한다. ② 불필요하거나 사용하지 않는 중요정보가 메모리에 남지 않도록 해야 한다.

가. 취약점 개요

사례1 : 중요정보 평문저장

메모리나 디스크에서 처리하는 중요데이터(개인정보, 인증정보, 금융정보)가 제대로 보호되지 않을 경우, 보안이나 데이터의 무결성이 훼손될 수 있다. 특히 프로그램이 개인정보, 인증정보 등의 사용자 중요정보 및 시스템 중요정보를 처리하는 과정에서 이를 평문으로 저장할 경우 공격자에게 민감한 정보가 노출될 수 있는 취약점이다.

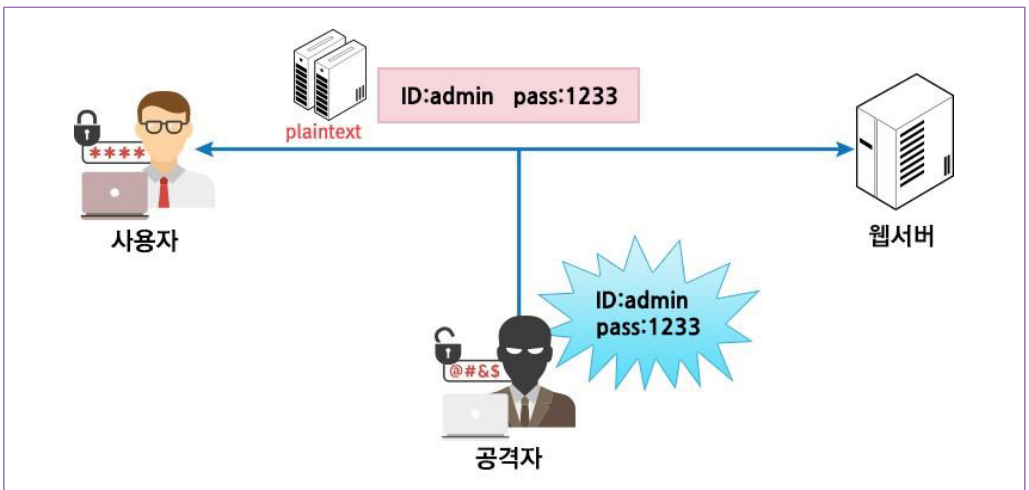


그림 3-32 | 중요정보 평문저장

사례2 : 사용자 하드디스크에 저장된 쿠키를 통한 정보노출

개인정보, 인증정보 등이 영속적인 쿠키(Persistent Cookie)에 저장된다면, 공격자는 쿠키에 접근할 수 있는 보다 많은 기회를 가지게 되며, 이는 시스템을 취약하게 만든다.

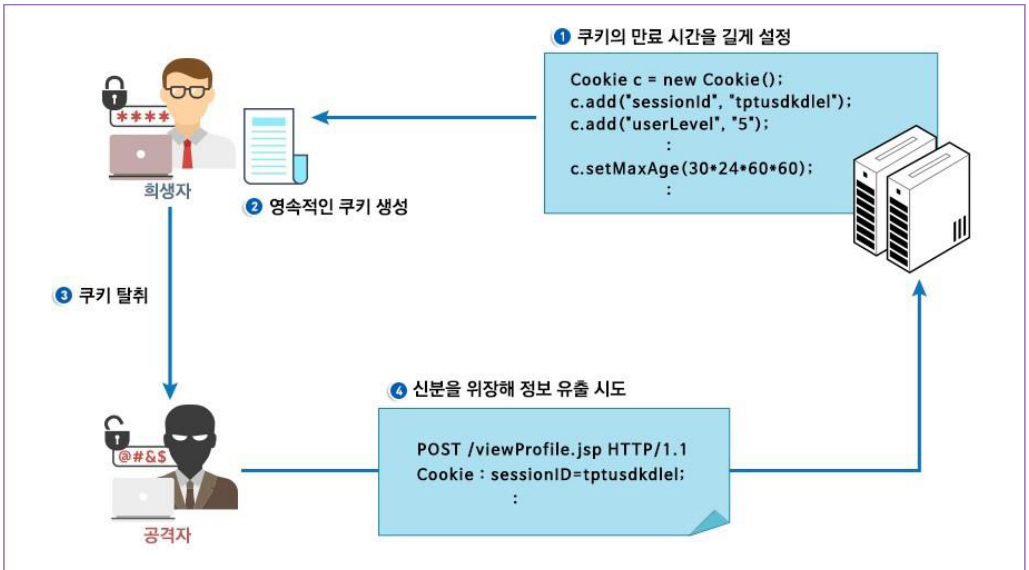


그림 3-33 | 사용자 하드디스크에 저장되는 쿠키를 통한 정보노출

나. 설계시 고려사항

① 중요정보 또는 개인정보는 암호화해서 저장해야 한다.

중요정보가 다뤄지는 "안전영역"을 설정하고 중요정보가 해당 영역 외부로 누출되지 않도록 설계 한다.

서버의 DB나 파일 등에 저장되는 중요정보는 반드시 암호화해서 저장해야 하며 "암호연산" 설계 항목에서 정의하고 있는 안전한 암호 알고리즘과 암호키를 사용한다.

특히 쿠키, HTML5 로컬저장소와 같은 클라이언트 측 하드드라이브에는 중요정보가 저장되지 않도록 설계해야 하며, 부득이하게 중요정보를 저장해야 하는 경우 반드시 클라이언트 측에 저장되는 민감 정보를 암호화한다.

클라이언트 언어인 HTML 코드는 사용자에게 공개되어있는 것과 마찬가지이므로 중요한 로직 및 주석처리는 서버 측 언어에서만 처리되도록 설계해야 한다.

② 불필요하거나 사용하지 않는 중요정보가 메모리에 남지 않도록 해야 한다.

개인정보 또는 특정 금융정보를 처리하는 기능 구현 시 더 이상 필요하지 않은 데이터에 대해 메모리를 초기화하여 중요데이터가 메모리에 남지 않도록 시큐어코딩 규칙을 정의한다.

특히 민감한 정보를 포함하는 페이지는 사용자 측 캐싱을 비활성화하도록 제한적인 캐시 정책을 수립하여야 하며, 부득이 캐싱을 해야 하는 경우 캐싱되는 정보는 암호화하여 저장하도록 설계한다.

인증정보와 같은 민감한 정보를 포함하는 웹 폼을 구현하는 경우 자동완성 기능을 비활성화하도록 시큐어코딩 규칙을 정의한다.

다. 진단 세부사항

요구사항 ①

중요정보 또는 개인정보는 암호화해서 저장해야 한다.

중요정보와 개인정보에 대해 암호화하도록 명시하고, 암호화 대상 정보와 암호화 구현 방법을 명시하고 있는지 확인한다.

진단기준

1. 중요정보 또는 개인정보가 암호화되어서 저장되도록 설계되어 있는가?
2. 저장된 중요정보에 대한 암호화 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 중요정보 또는 개인정보가 암호화되어서 저장되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토

진단방법	관련산출물 검토예시
1-2 암호화하여 저장되어야 하는 중요정보 또는 개인정보가 분류되어 있는지 확인 - 암호화 대상 중요정보(개인정보, 신용정보, 인증정보 등) - 개인정보보호법 : 고유식별정보(주민번호, 운전면허번호, 여권번호, 외국인등록 번호), 바이오정보, 비밀번호 - 정보통신망법 : 주민등록번호, 신용카드번호, 계좌번호 - 위치정보법 : 위치정보 - 시스템 중요정보 : 데이터베이스 및 연동시스템 계정 정보, 주요 설정 정보 - 기타 관련 법률 및 서비스 특성에 따른 주요 정보	요구사항정의서, 아키텍처설계서 검토
1-3 안전한 암호화 기능을 구현하는 프로그램이 설계되어 있거나, 안전한 외부 암호 라이브러리를 사용하도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서, 프로그램명세서, 클래스설계서 검토
1-4 프로그램 내에서 암호화 대상 정보를 암·복호화 하는 경우 개발가이드에 암·복호화 코딩규칙이 정의되도록 설계되어 있는지 확인	개발가이드 검토

2. 저장된 중요정보에 대한 암호화 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 저장된 중요정보가 암호화 되어 저장되었는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 클래스설계서, 단위테스트계획서 등

요구사항 ②

불필요하거나 사용하지 않는 중요정보가 메모리에 남지 않도록 해야 한다.

중요정보 사용 후 메모리에 존재하지 않도록 설계되어 있는지 확인한다.

진단기준

1. 사용하지 않는 중요정보가 메모리에 남아있지 않도록 설계되어 있는가?
2. 사용하지 않는 중요정보의 메모리 잔존을 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 사용하지 않는 중요정보가 메모리에 남아있지 않도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 중요정보 사용 후 메모리 초기화를 수행하여 중요정보가 메모리에 남지 않도록 개발가이드에 코딩 규칙을 정의하고 있는지 확인	개발가이드 검토
1-3 중요정보가 포함된 페이지는 사용자 측에 캐싱되지 않도록 개발가이드 코딩 규칙을 정의하고 있는지 확인	개발가이드 검토

2. 사용하지 않는 중요정보의 메모리 잔존을 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 중요정보가 메모리에 초기화되지 않고 남아있는지를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
2-2 중요정보가 포함된 페이지가 사용자 측에 캐싱되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	암호화 되지 않은 중요정보
보안기능	사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출

마. 참고자료

① CWE-312, Cleartext Storage of Sensitive Information, MITRE, http://cwe.mitre.org/data/definitions/312.html

② CWE-539 Information Exposure Through Persistent Cookies, MITRE, http://cwe.mitre.org/data/definitions/539.html

③ 2016 OWASP Application Security Verification Standard, OWASP, Data Protection Verification Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

④ Sensitive Data, Microsoft Patterns & Practices, MSDN, http://msdn.microsoft.com/en-us/library/ff650867.aspx

⑤ Password in the Clear, W3C, http://www.w3.org/2001/tag/doc/passwordsInTheClear-52

2.8 중요정보 전송

유형	보안기능
설계항목	중요정보 전송
설명	중요정보(비밀번호, 개인정보, 쿠키 등)를 전송하는 방법이 안전하도록 설계한다.
보안대책	① 인증정보와 같은 민감한 정보 전송시 안전하게 암호화해서 전송해야 한다. ② 쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.

가. 취약점 개요

프로그램이 보안과 관련된 민감한 데이터를 평문으로 송·수신할 경우, 통신채널 스니핑으로 인가되지 않은 사용자에게 민감한 데이터가 노출될 수 있다.

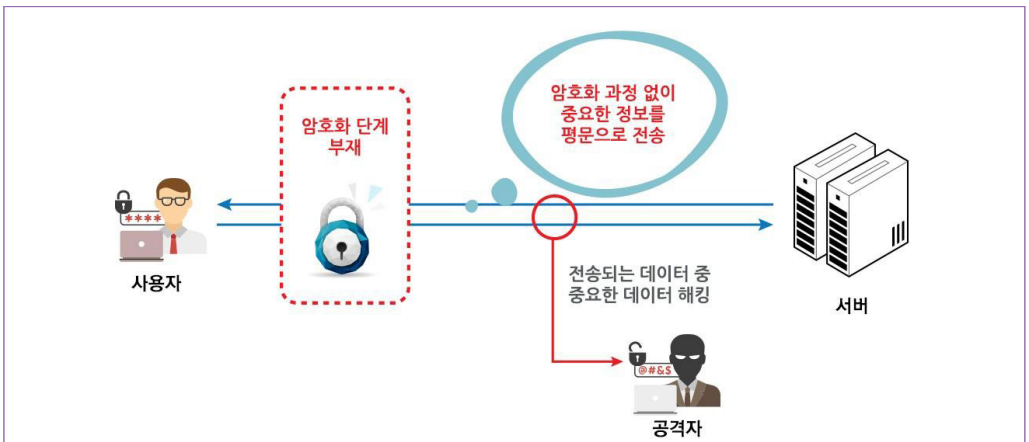


그림 3-34 | 중요정보 평문전송

나. 설계시 고려사항

① 인증정보와 같은 민감한 정보 전송 시 안전하게 암호화해서 전송해야 한다.

분석단계에서 정의된 중요정보를 네트워크로 전송해야 하는 경우 안전한 암호모듈로 암호화한 뒤 전송하거나 안전한 통신 채널을 사용하도록 설계한다. 안전한 암호화는 “암호연산” 설계항목을 충족시키는 암호화 알고리즘이나 암호키를 사용한다.

웹 애플리케이션 설계 시 클라이언트에서 서버로 전달되는 데이터(hidden필드, Ajax변수, 쿠키, 헤더 등) 또는 서버에서 클라이언트로 전달되는 데이터(HTTP 응답헤더 등) 중 불필요하게 많은 데이터가 전송되지 않도록 설계한다.

② 쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.

쿠키에는 중요정보가 포함되지 않도록 설계해야 하지만 부득이 쿠키에 중요정보가 포함되어야 하는 경우 반드시 세션쿠키로 설정되어야 하며, 전달되는 중요정보는 반드시 암호화해서 전송해야 한다.

다. 진단 세부사항

요구사항 ①

인증정보와 같은 민감한 정보 전송 시 안전하게 암호화해서 전송해야 한다.

중요정보를 네트워크로 외부와 송·수신하는 경우 데이터 암호화 또는 통신 구간 암호화를 수행하도록 명시하고 있는지 확인한다.

진단기준

1. 인증정보와 같은 민감한 정보를 전송할 때 안전하게 암호화해서 전송되도록 설계되어 있는가?
2. 네트워크로 전송되는 중요정보의 암호화 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 인증정보와 같은 민감한 정보를 전송할 때 안전하게 암호화해서 전송되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 암호화해야 하는 중요정보(인증정보, 민감한정보, 개인정보 등)가 분류되어 있는지 확인 - 암호화 대상 중요정보(개인정보, 신용정보, 인증정보 등)개인정보보호법 : 고유 식별정보(주민번호, 운전면허번호, 여권번호, 외국인등록번호), 바이오 정보, 비밀번호 - 정보통신망법 : 주민등록번호, 신용카드번호, 계좌번호 - 위치정보법 : 위치정보 - 시스템 중요정보 : 데이터베이스 및 연동시스템 계정 정보, 주요 설정 정보 - 기타 관련 법률 및 서비스 특성에 따른 주요 정보	요구사항정의서, 아키텍처설계서 검토

진단방법	관련산출물 검토예시
1-3 중요정보 전송을 위해 안전한 암호화 정책(TLSv2, VPN 등)이 적용된 암호화 통신 구간을 사용하도록 설계되어 있는지 확인	아키텍처설계서 검토
1-4 암호화 통신 구간을 사용하지 않는 경우 중요정보를 안전한 암호 알고리즘을 사용하여 암호화해서 전송하도록 설계되어 있는지 확인	프로그램명세서, 유즈케이스설계서, 클래스설계서, 개발가이드 검토

2. 네트워크로 전송되는 중요정보의 암호화 여부를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 네트워크 구간의 패킷 스니핑 등으로 암호화되어 전송되어야 하는 중요 정보의 노출 여부를 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 유즈케이스설계서, 클래스설계서, 프로그램명세서, 단위테스트계획서 등

요구사항 ②

쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.

쿠키에 포함되는 중요정보를 식별하고 암호화 하도록 명시하고 있는지 확인한다.

진단기준

1. 쿠키에 포함되는 중요정보는 암호화되어 전송되도록 설계되어 있는가?
2. 쿠키에 포함된 중요정보에 암호화가 적용되었는지에 대한 테스트 계획을 수립하고 있는가?

진단방법

1. 쿠키에 포함되는 중요정보는 암호화되어 전송되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 암호화해야 하는 중요정보(인증정보, 민감정보, 개인정보 등)가 분류되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토

진단방법	관련산출물 검토예시
1-3 중요정보를 포함하는 쿠키 전송을 위해 안전한 암호화 정책(TLSv2, VPN 등)이 적용된 암호화 통신 구간을 사용하도록 설계되어 있는지 확인	아키텍처설계서 검토
1-4 암호화 통신 구간을 사용하지 않는 경우 중요정보를 안전한 암호 알고리즘을 사용하여 암호화해서 전송하도록 설계되어 있는지 확인	프로그램명세서, 유즈케이스설계서, 클래스설계서, 개발가이드 검토

2. 쿠키에 포함된 중요정보에 암호화가 적용되었는지에 대한 테스트 계획을 수립하고 있는가?

진단방법	관련산출물 검토예시
2-1 네트워크 구간의 패킷 스니핑 등으로 쿠키에 포함된 암호화하여 전송되어야 하는 중요정보의 노출 여부를 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 유즈케이스설계서, 클래스설계서, 개발가이드, 프로그램명세서, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	암호화되지 않은 중요정보

마. 참고자료

① CWE-319, Cleartext Transmission of Sensitive Information, MITRE, http://cwe.mitre.org/data/definitions/312.html

② Transport Layer Protection Cheat Sheet, OWASP, http://www.owasp.org/index.php/Transport_Layer_Pro- tection_Cheat_Sheet

③ 2016 OWASP Application Security Verification Standard, OWASP, Communications security Verification Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_ Standard_Project

④ The Transport Layer Security (TLS) Protocol Version 1.2, RFC5246, http://tools.ietf.org/html/rfc5246

⑤ User Privacy Protection Cheat Sheet, OWASP, http://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet#Strong_Cryptography

3. 에러처리

3.1 예외처리

유형	에러처리
설계항목	예외처리
설명	오류메시지에 중요정보(개인정보, 시스템 정보, 민감 정보 등)가 노출되거나, 부적절한 에러 및 오류처리로 인하여 의도치 않은 상황이 발생하지 않도록 설계한다.
보안대책	① 명시적인 예외의 경우 예외처리 블럭을 이용하여 예외 발생시 수행해야 하는 기능이 구현되도록 해야 한다. ② 런타임 예외의 경우 입력값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다. ③ 에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.

가. 취약점 개요

사례1 : 오류 메시지 정보노출

웹 서버에 별도의 에러페이지를 설정하지 않은 경우, 에러 메시지로 서버 데이터 정보 등 공격에 필요한 정보가 노출되는 취약점이다.

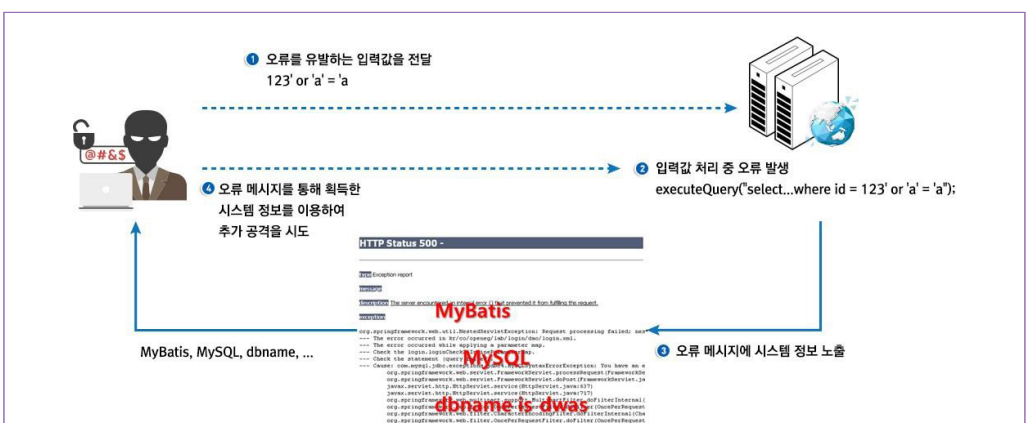


그림 3-35 | 오류 메시지 정보노출

사례2 : 시스템 정보노출

시스템, 관리자, DB정보 등 시스템의 내부 데이터가 공개되면, 공격자에게 또 다른 공격의 빌미를 제공하게 된다.

그림 3-36 | 시스템 데이터 정보노출

나. 설계시 고려사항

① 명시적인 예외의 경우 예외처리 블럭을 이용하여 예외 발생시 수행해야 하는 기능이 구현되도록 해야 한다.

각 프로그래밍 언어별 예외처리 문법에 대한 안전한 사용방법을 기술하고, 모든 개발자가 구현 단계에서 안전하게 예외처리를 할 수 있도록 시큐어코딩 규칙을 정의한다.

[예시] 자바 플랫폼 사용시

프로그램에서 발생된 에러 정보를 로깅하는 Logger API를 활용할 수 있도록 시큐어코딩 규칙을 정의한다.

② 런타임 예외의 경우 입력 값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다.

입력값에 따라 예외가 발생 가능한 경우 입력 값의 범위를 체크하여 사용하도록 시큐어코딩 규칙을 정의한다.

③ 에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.

(ㄱ) 에러가 발생한 경우 지정된 페이지로 사용자에게 에러 공지.

에러가 발생한 경우 프로그램 내에서 지정된 에러페이지로 리다이렉트 되도록 시큐어코딩 규칙을 정의하거나, 웹 애플리케이션 서버 설정하여 특정 에러나 예외사항에 대해 지정된 페이지가 사용자에게 보일 수 있도록 설계한다.

[예시] web.xml 파일을 이용하여 에러시 지정된 페이지 응답

```
<!– error 페이지 –>
<error-page>
        <error-code>404</error-code>
        <location>/WEB-INF/jsp/common/error/404error.jsp</location>
</error-page>
<error-page>
        <error-code>500</error-code>
        <location>/WEB-INF/jsp/common/error/500error.jsp</location>
</error-page>
<error-page>
        <exception-type>java.lang.Throwable</exception-type>
        <location>/WEB-INF/jsp/common/error/error.jsp</location>
</error-page>
```

(ㄴ) 사용자에게 보내지는 오류 메시지에 중요정보가 포함되지 않도록 함.

오류 메시지에 중요정보(개인정보, 시스템정보, 민감정보 등)가 포함되지 않도록 시큐어코딩 규칙을 정의한다.

다. 진단 세부사항

요구사항 ①

명시적인 예외의 경우 예외처리 블럭을 이용하여 예외 발생 시 수행해야 하는 기능이 구현되도록 해야 한다.

예외 발생을 예방하기 위해 입력값을 검증하고 예외 발생시 예외처리 블록을 이용하여 수행되어야 할 기능이 구현되도록 설계되어 있는지 확인한다.

진단기준

1. 프로그램 작성시 명시적인 Exception 상황을 안전하게 처리하도록 설계되어 있는가?
2. 안전한 예외처리를 점검하는 테스트계획이 수립되어 있는가?

진단방법

1. 프로그램 작성시 명시적인 Exception 상황을 안전하게 처리하도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 Exception 유형별 처리 방법을 정의하고 프로그램 구현시 안전한 처리 방법을 개발가이드에 정의하고 있는지 확인 - 오류 메시지로 시스템정보가 노출되지 않도록 프로그램이 작성 되어야함 - 예외 상황에 대해 반드시 처리되도록 프로그램이 작성되어야 함 - 각각의 예외 상황에 대해 분리하여 처리하도록 프로그램이 작성되어야 함	유즈케이스설계서, 클래스설계서, 개발가이드 검토
1-3 Exception 발생에 대한 로깅 정책이 설계에 반영되었는지 확인 - Exception 발생 상황에 대해 시간, 이유 등과 같은 내용이 포함되도록 설계되어 있는지 확인 - 로그에 포함되는 중요정보는 암호화되도록 설계되어 있는지 확인	아키텍처설계서 검토

2. 쿠키에 포함된 중요정보에 암호화가 적용되었는지에 대한 테스트 계획을 수립하고 있는가?

진단방법	관련산출물 검토예시
2-1 예외가 발생 가능한 입력값을 이용하여 예외 발생시 안전하게 처리되는지 여부를 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 단위테스트계획서 등

요구사항 ②

런타임 예외의 경우 입력값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다.

예외 발생을 예방하기 위해 입력값을 검증하고 예외 발생시 예외처리 블록을 이용하여 수행 되어야 할 기능이 구현되도록 설계되어 있는지 확인한다.

진단기준

1. 프로그램 실행시 발생되는 Runtime Exception이 안전하게 처리되도록 설계되어 있는가?
2. 안전한 예외처리를 점검하는 테스트계획이 수립되어 있는가?

진단방법

1. 프로그램 실행시 발생되는 Runtime Exception이 안전하게 처리되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 예외가 발생 가능한 기능 구현 시 입력데이터에 대한 유효한 입력값의 범위 검증하여 예외가 발생하지 않도록 개발가이드로 코딩 규칙을 정의 하고 있는지 확인	개발가이드 검토

2. 안전한 예외처리를 점검하는 테스트계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 예외가 발생 가능한 입력값을 이용하여 예외 상황이 안전하게 처리되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 단위테스트케이스 등

요구사항 ③

에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다. 에러 발생 시 오류 메시지 정보 또는 중요정보가 노출되지 않도록 설계되어 있는지 확인한다.

진단기준

1. Exception 발생시 상세한 에러정보가 사용자에게 노출되지 않도록 설계되어 있는가?
2. 에러로 상세정보가 노출되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. Exception 발생시 상세한 에러정보가 사용자에게 노출되지 않도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 예외 발생 시 스택정보나 중요정보가 외부로 노출되지 않도록 개발가이드에 코딩 규칙이 정의되어 있는지 확인	개발가이드 검토
1-3 Exception 발생 시 지정된 에러페이지를 이용하여 사용자에게 예외사항을 정보를 제공하도록 서버설정 계획이 설계되어 있는지 확인 - 예외에 따라 사용자에게 제공되어야 하는 에러페이지를 작성하여, 지정된 페이지가 응답되도록 서버가 설정되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 Exception 발생에 대한 로그가 생성될 수 있도록 서버설정 계획이 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토

2. 에러로 상세정보가 노출되는지 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 예외 발생 시 상세한 예외사항, 시스템 중요정보 등이 사용자에게 노출되는지 점검할 수 있는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
에러처리	오류 메시지 정보노출

마. 참고자료

① CWE-209 Information Exposure Through an Error Message, MITRE, http://cwe.mitre.org/data/defini- tions/209.html

② CWE-390 Detection of Error Condition Without Action, MITRE, http://cwe.mitre.org/data/definitions/390. html

③ CWE-754 Improper Check for Unusual or Exceptional Conditions, MITRE http://cwe.mitre.org/data/defi- nitions/754.html

④ Error Handling, Auditing and Logging, OWASP, http://www.owasp.org/index.php/Error_Handling,_Audit- ing_and_Logging

⑤ Improper Error Handling, OWASP, http://www.owasp.org/index.php/Improper_Error_Handling

⑥ “Best practices with custom error pages in .Net”, Micro Support, http://support.microsoft.com/default.aspx-?scid=kb;en-us;834452

⑦ WASC Fingerprinting, WASC, http://projects.webappsec.org/w/page/13246925/Fingerprinting

⑧ WASC Information Leakage, WASC, http://projects.webappsec.org/w/page/13246936/Information%20Leak-age

4. 세션통제

4.1 세션 통제

유형	세션통제
설계항목	세션통제
설명	다른 세션간 데이터 공유금지, 세션 ID 노출금지, (재)로그인시 세션ID 변경, 세션종료(비활성화, 유효기간 등) 처리 등 세션을 안전하게 관리할 수 있는 방안을 설계해야 한다.
보안대책	① 세션 간 데이터가 공유되지 않도록 설계해야 한다. ② 세션이 안전하게 관리되도록 해야 한다. ③ 세션ID가 안전하게 관리되도록 해야 한다.

가. 취약점 개요

사례1 : 불충분한 세션관리

인증시 일정한 규칙이 존재하는 세션ID가 발급되거나 세션 타임아웃을 너무 길게 설정한 경우 공격자에 의해 사용자 권한이 도용될 수 있는 취약점이다.

그림 3-37 | 불충분한 세션관리

사례2 : 잘못된 세션에 의한 정보노출

다중 스레드 환경에서는 싱글톤(Singleton)객체 필드에 경쟁조건(Race Condition) 발생할 수 있다. 따라서, 다중 스레드 환경인 java의 서블릿(servlet) 등에서는 정보를 저장하는 멤버변수가 포함되지 않도록 하여, 서로 다른 세션 간에 데이터를 공유하지 않도록 해야 한다.

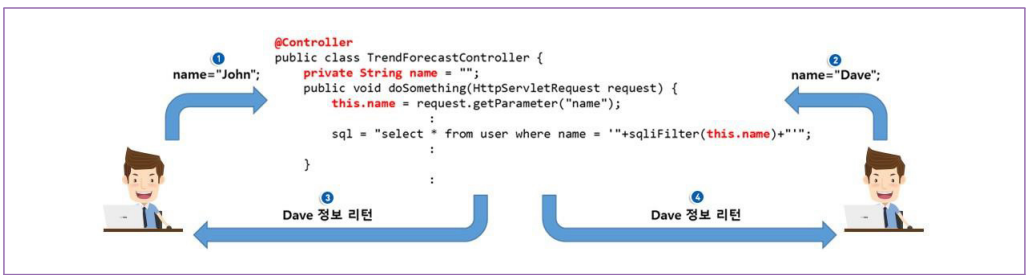


그림 3-38 | 잘못된 세션에 의한 데이터 정보노출

나. 설계 시 고려사항

① 세션 간 데이터가 공유되지 않도록 설계해야 한다.

스레드로 동작하는 웹애플리케이션의 컨트롤러 컴포넌트나, 싱글톤 객체로 생성되는 서비스 컴포넌트를 설계하는 경우 클래스 멤버 변수나 클래스변수는 세션 간에 공유되는 데이터가 되므로 클래스 설계시 읽고 쓰기가 가능한 변수를 사용하지 않도록 설계해야 한다.

② 세션이 안전하게 관리 되도록 해야 한다.

- 시스템 내의 모든 페이지에서 로그아웃이 가능하도록 UI를 설계하고, 로그아웃을 요청하면 사용자에게 할당된 세션을 완전히 제거하는 API를 사용하도록 시큐어코딩 규칙을 정의한다. 예를 들어, Java의 경우 session.invalidate() 메서드를 사용하여 세션에 저장된 정보를 완전히 제거할 수 있다.
- 세션 타임아웃 시간은 중요기능의 경우 2~5분, 위험도가 낮은 애플리케이션의 경우에는 15~30분으로 설정하고, 이전 세션이 종료되지 않은 상태에서 새로운 세션이 생성되지 않도록 해야 한다.
- 웹 브라우저 종료로 인한 세션 종료는 서버 측에서 인지할 수 없으므로, 일정 시간 동안 사용되지 않는 세션 정보는 강제적으로 삭제되도록 설계한다.
- 중복 로그인을 허용하지 않는 경우, 새로운 로그인 세션 생성 시 이전에 생성된 로그인 세션을 종료하거나, 새로이 세션이 연결되지 않도록 검증하는 정책이 설계단계에 고려되어야 한다.

- 세션ID가 포함된 쿠키에 대해 HttpOnly 속성을 설정하여 자바스크립트로 조회할 수 없도록 만들어 XSS공격에 대응하도록 설계한다.
- 사용자가 비밀번호를 변경하는 경우 현재 활성화된 세션을 삭제하고 다시 할당한다.

③ 세션ID가 안전하게 관리되도록 해야 한다.

(ㄱ) 세션ID 생성

- 세션ID는 안전한 서버에서 생성해서 사용되어야 한다.
- 세션ID는 최소 128비트의 길이로 생성되어야 하며, 안전한 난수 알고리즘을 적용하여 예측이 불가능한 값이 사용되어야 한다.

(ㄴ) 세션ID 사용

- URL Rewrite 기능을 사용하는 경우 세션ID가 URL에 노출될 수 있으므로, 사용하지 않도록 설계한다.

(ㄷ) 세션ID 폐기

- 로그인 성공시 로그인 전에 할당받은 세션ID는 파기하고 새로운 값으로 재할당하여 세션ID 고정 공격에 대응하도록 시큐어코딩 규칙을 정의한다.
- 장기간 접속되어 있는 경우 세션ID의 노출위험이 커지므로, 일정시간 주기적으로 세션ID를 재할당하도록 설계한다.

다. 진단 세부사항

요구사항 ①

세션 간 데이터가 공유되지 않도록 설계해야 한다.

세션 간 데이터 공유로 인한 정보노출이나 오류가 발생하지 않도록 설계되어 있는지 확인한다.

진단기준

1. 세션 간 데이터가 공유되지 않도록 클래스가 설계되어 있는가?
2. 세션 간 데이터 공유를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 세션 간 데이터가 공유되지 않도록 클래스가 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 스레드로 동작되는 클래스 설계 시 읽기/쓰기가 가능한 스레드 간 공유 데이터가 설계되어 있는지 확인 - Servlet, Controller의 경우 클래스 멤버변수, 클래스 변수 선언 - JSP의 경우 선언부에 변수 선언	프로그램명세서, 클래스설계서, 유즈케이스명세서 검토

2. 세션 간 데이터 공유를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 여러 개의 세션을 이용하여 공유되는 데이터에 대한 값의 변경 여부를 점검하는 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스명세서, 단위테스트계획서 등

요구사항 ②

세션이 안전하게 관리되도록 해야 한다.

세션생성과 종료, 세션 타임아웃 시간설정 등 안전한 세션 관리방안이 설계되어 있는지 확인한다.

진단기준

1. 세션이 안전하게 관리되도록 설계되어 있는가?
2. 세션이 안전하게 관리되는지를 점검하는 테스트 계획이 수립되어 있는가?

진단방법

1. 세션이 안전하게 관리되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 로그인에 성공하면 세션을 재할당하도록 코딩규칙이 정의되어 있는지 확인	개발가이드 검토
1-3 로그아웃 시 할당된 세션을 완전히 제거하도록 코딩규칙이 정의되어 있는지 확인	개발가이드 검토
1-4 사용자의 일방적인 연결종료에 대해 할당된 세션이 제거되도록 프로그램이 설계되어 있는지 확인	유즈케이스설계서, 클래스설계서 검토
1-5 오랫동안 사용하지 않는 세션에 대해 타임아웃을 설정하여 할당된 세션이 제거되도록 설계되어 있는지 확인 - 중요기능의 경우 2~5분, 위험도가 낮은 애플리케이션의 경우에는 15~30분으로 설정하도록 설계되어 있는지 확인	요구사항정의서, 아키텍처설계서, 유즈케이스설계서, 클래스설계서 검토
1-6 비밀번호 변경 시 재 인증을 수행하여 새로운 세션이 할당되도록 설계되어 있는지 확인	유즈케이스설계서, 클래스설계서 검토

2. 세션이 안전하게 관리되는지를 점검하는 테스트 계획이 수립되어 있는가?

진단방법	관련산출물 검토예시
2-1 로그인/로그아웃을 수행하여 안전하게 세션이 관리되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
2-2 브라우저 강제종료, 오랫동안 사용하지 않는 세션에 대해 안전하게 세션이 관리되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
2-3 비밀번호 변경 시 안전하게 세션이 관리되는지 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 프로그램명세서, 유즈케이스설계서, 클래스설계서, 단위테스트계획서 등

요구사항 ③

세션ID가 안전하게 관리되도록 해야 한다.

세션ID가 안전하게 관리될 수 있도록 설계되어 있는지 확인한다.

진단기준

1. 세션ID가 안전하게 관리되도록 설계되어 있는가?
2. 세션ID노출 여부를 점검하는 테스트 계획을 수립하고 있는가?

진단방법

1. 세션ID가 안전하게 관리되도록 설계되어 있는가?

진단방법	관련산출물 검토예시
1-1 요구사항정의서에 설계항목에 대한 대책이 수립되어 있으며, 아키텍처 설계서에 설계항목 적용계획이 수립되어 있고, 요구사항추적표로 요구사항 추적 가능 여부 확인	요구사항정의서, 요구사항추적표, 아키텍처 설계서 검토
1-2 세션ID 생성을 서버에서 생성하도록 웹서버 환경설정 계획이 수립되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-3 세션ID 전송 방법을 쿠키로 제한하며, 쿠키의 속성에 HttpOnly가 설정되도록 웹서버 환경설정 계획이 수립되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-4 장시간 접속되어 있는 경우 일정 시간 주기적으로 세션ID가 재할당하도록 웹서버 환경설정 계획이 수립되어 있는지 확인	요구사항정의서, 아키텍처설계서 검토
1-5 로그인 수행 전에 할당받은 세션ID는 로그인 성공 후 재할당하도록 코딩규칙 이 정의되어 있는지 확인	개발가이드 검토

2. 세션ID노출 여부를 점검하는 테스트 계획을 수립하고 있는가?

진단방법	관련산출물 검토예시
2-1 자바스크립트를 사용하여 세션ID값 노출을 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
2-2 장시간 접속시 세션ID값의 재할당 여부를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토
2-3 성공적인 로그인 후 세션ID 재할당 여부를 점검하기 위한 테스트 계획이 수립되어 있는지 확인	단위테스트계획서 검토

관련산출물

요구사항정의서, 요구사항추적표, 아키텍처설계서, 개발가이드, 단위테스트계획서 등

라. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
캡슐화	잘못된 세션에 의한 데이터 정보 노출

마. 참고자료

① CWE-488 Exposure of Data Element to Wrong Session, MITRE, http://cwe.mitre.org/data/definitions/488. html

② 2013 OWASP Top 10 – A6 Sensitive Data Exposure, OWASP, https://www.owasp.org/index.php/Top_10_2013

③ WASC Threat Classification Credential and Session Prediction, WASC, http://projects.webappsec.org/w/page/13246918/Credential%20and%20Session%20Prediction

④ 2016 OWASP Application Security Verification Standard, OWASP, Session Management Verification Re-quirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Stan- dard_Project

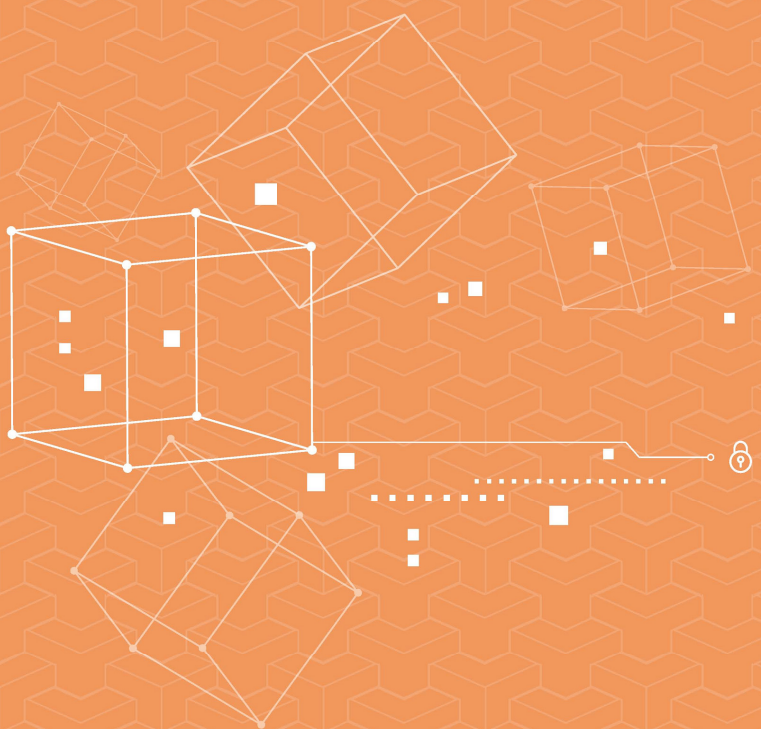
⑤ Session Management Cheat Sheet, OWASP, http://www.owasp.org/index.php/Session_Management_ Cheat_Sheet

⑥ "HTTP State Management Mechanism", RFC 6265, IETF, http://tools.ietf.org/html/rfc6265

⑦ Unauthenticated Session Fixation Attacks, Christian Schneider, http://www.christian-schneider.net/UnauthenticatedSessionFixationAttacks.html#main

⑧ Session Fixation Attacks and Protections in Web Applications, Raul Siles, http://media.blackhat.com/bh-eu-11/Raul_Siles/BlackHat_EU_2011_Siles_SAP_Session-WP.pdf

⑨ Insufficient Session-ID Length, OWASP, http://www.owasp.org/index.php/Insufficient_Session-ID_Length



소프트웨어 보안약점 진단가이드

KOREA INTERNET & SECURITY AGENCY

PART 4

제4장 구현단계 보안약점 진단

PART 4

제4장 구현단계 보안약점 진단

| 제 1 절 | 입력데이터 검증 및 표현

프로그램 입력값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못된 형식지정, 일관되지 않은 언어셋 사용 등으로 인해 발생되는 보안약점으로 SQL 삽입, 크로스사이트 스크립트(XSS) 등의 공격을 유발할 수 있다.

1. SQL 삽입

가. 개요

데이터베이스(DB)와 연동된 웹 응용프로그램에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 폼 및 URL 입력란에 SQL 문을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안약점을 말한다.

취약한 웹 응용프로그램에서는 사용자로부터 입력된 값을 필터링 과정 없이 넘겨받아 동적쿼리(Dynamic Query)[11]를 생성하기 때문에 개발자가 의도하지 않은 쿼리가 생성되어 정보유출에 악용될 수 있다.

나. 보안대책

PreparedStatement[12]객체 등을 이용하여 DB에 컴파일 된 쿼리문(상수)을 전달하는 방법을 사용한다. PreparedStatement를 사용하는 경우에는 DB 쿼리에 사용되는 외부입력값에 대하여 특수문자 및 쿼리 예약어를 필터링하고, 스트러츠(Struts), 스프링(Spring) 등과 같은 프레임워크를 사용하는 경우에는 외부입력값 검증모듈 및 보안모듈을 상황에 맞추어 적절하게 사용한다.

11) 동적쿼리(Dynamic Query) : 컬럼이나 테이블명을 바꿔 SQL쿼리를 실시간 생성해 DB에 전달하여 처리하는 방식

12) PreparedStatement : 컴파일된 쿼리 객체로 MySQL, Oracle, DB2, SQL Server 등에서 지원하며, Java의 JDBC, Perl의 DBI, PHP의 PDO, ASP의 ADO를 이용하여 사용가능

다. 코드예제

다음은 안전하지 않은 코드의 예로, 외부로부터 입력받은 gubun의 값을 아무런 검증과정을 거치지 않고 SQL 쿼리를 생성하는데 사용하고 있다. 이 경우 gubun의 값으로 a' or 'a' = 'a 를 입력하면 조건절이 b_gubun = 'a' or 'a' = 'a' 로 바뀌어 쿼리의 구조가 변경되어 board 테이블의 모든 내용이 조회된다.

안전하지 않은 코드의 예 JDBC API

```
1: //외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
2: String gubun = request.getParameter("gubun");
3: ......
4: String sql = "SELECT * FROM board WHERE b_gubun = '" + gubun + "'";
5: Connection con = db.getConnection();
6: Statement stmt = con.createStatement();
7: //외부로부터 입력받은 값이 검증 또는 처리 없이 쿼리로 수행되어 안전하지 않다.
8: ResultSet rs = stmt.executeQuery(sql);
```

이를 안전한 코드로 변환하면 다음과 같다. 파라미터(Parameter)를 받는 PreparedStatement 객체를 상수 스트링으로 생성하고, 파라미터 부분을 setString, setParameter등의 메소드로 설정하여, 외부의 입력이 쿼리문의 구조를 바꾸는 것을 방지해야 한다.

안전한 코드의 예 JDBC API

```
1: String gubun = request.getParameter("gubun");
2: ......
3: //1. 사용자에 의해 외부로부터 입력받은 값은 안전하지 않을 수 있으므로, PreparedStatement
   사용을 위해 ?문자로 바인딩 변수를 사용한다.
4: String sql = "SELECT * FROM board WHERE b_gubun = ?";
5: Connection con = db.getConnection();
6: //2. PreparedStatement 사용한다.
7: PreparedStatement pstmt = con.prepareStatement(sql);
8: //3.PreparedStatement 객체를 상수 스트링으로 생성하고, 파라미터 부분을 setString등의 메소드로
   설정하여 안전하다.
9: pstmt.setString(1, gubun);
10: ResultSet rs = pstmt.executeQuery();
```

MyBatis Data Map은 외부에서 입력되는 값이 SQL 질의문을 연결하는 문자열로 사용되는 경우에 의도하지 않은 정보가 노출될 수 있는 공격 형태이다.

안전하지 않은 코드의 예 MyBatis

```
1: 〈?xml version="1.0" encoding="UTF-8" ?〉
2: 〈!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd"〉
3: ......
4: 〈select id="boardSearch" parameterType="map" resultType="BoardDto"〉
5: //$기호를 사용하는 경우 외부에서 입력된 keyword값을 문자열에 결합한 형태로 쿼리에 반영되므로
   안전하지 않다.
6:     select * from tbl_board where title like '%${keyword}%' order by pos asc
7: 〈/select〉1
```

외부 입력값을 MyBatis 쿼리맵에 바인딩할 경우 $ 기호가 아닌 # 기호를 사용해야 한다. $ 기호를 사용하는 경우 입력값을 문자열에 결합하는 형태로 쿼리에 반영하므로 쿼리문이 조작될 수 있다.

안전한 코드의 예 MyBatis

```
1: 〈?xml version="1.0" encoding="UTF-8" ?〉
2: 〈!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd"〉
3: ......
4: 〈select id="boardSearch" parameterType="map" resultType="BoardDto"〉
5: //$ 대신 #기호를 사용하여 변수가 쿼리맵에 바인딩 될 수 있도록 수정하는 것이 안전하다.
6:     select * from tbl_board where title like '%'||#{keyword}||'%' order by pos asc
7: 〈/select〉
```

Hibernate의 경우 기본으로 PreparedStatement를 사용하지만, 아래 코드와 같이 파라미터 바인딩(binding)없이 사용할 경우 외부로부터 입력받은 값에 의해 쿼리의 구조가 변경 될 수 있다.

안전하지 않은 코드의 예 Hibernate

```
1: import org.hibernate.Query
2: import org.hibernate.Session
3: .....
4: //외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
5: String name = request.getParameter("name");
6: //Hiberate는 기본으로 PreparedStatement를 사용하지만, 파라미터 바인딩 없이 사용 할 경우
   안전하지 않다.
7: Query query = session.createQuery("from Student where studentName = '" + name
   + "' ");
```

아래 코드와 같이 외부 입력값이 위치하는 부분을 **? 또는 :명명된 파라미터 변수**로 설정하고, 실행 시에 해당 파라미터가 전달되는 바인딩(binding)을 함으로써 외부의 입력이 쿼리의 구조를 변경시키는 것을 방지할 수 있다.

안전한 코드의 예 Hibernate

```
1: import org.hibernate.Query
2: import org.hibernate.Session
3: .....
4: String name = request.getParameter("name");
5: //1. 파라미터 바인딩을 위해 ?를 사용한다.
6: Query query = session.createQuery("from Student where studentName = ? ");
7: //2. 파라미터 바인딩을 사용하여 외부 입력값에 의해 쿼리 구조 변경을 못하게 사용하였다.
8: query.setString(0, name);
```

안전한 코드의 예 Hibernate

```
1: import org.hibernate.Query
2: import org.hibernate.Session
3: .....
```

안전한 코드의 예 Hibernate

```
4: String name = request.getParameter("name");
5: //1. 파라미터 바인딩을 위해 명명된 파라미터 변수를 사용한다.
6: Query query = session.createQuery("from Student where studentName = :name ");
7: //2. 파라미터 바인딩을 사용하여 외부 입력값에 의해 쿼리 구조 변경을 못하게 사용하였다.
8: query.setParameter("name", name);
```

다음 C# 코드는 외부 입력값을 SQL 쿼리에 직접 사용하고 있어, 쿼리의 구조가 변경될 위험이 있습니다.

안전하지 않은 코드의 예 C#

```
1: public void ButtonClickBad(object sender, EventArgs e)
2: {
3:   string connect = "MyConnString";
4:   string usrinput = Request["ID"];
5: // 외부로부터 입력받은 값을 SQL 쿼리에 직접 사용하는 것은 안전하지 않다.
6:   string query = "Select * From Products Where ProductID = " + usrinput;
7:   using (var conn = new SqlConnection(connect))
8:   {
9:     using (var cmd = new SqlCommand(query, conn))
10:   {
11:     conn.Open();
12:     cmd.ExecuteReader(); /* BUG */
13:   }
14:   }
15: }
```

파라미터 바인딩을 사용하여 쿼리의 구조가 변경될 위험을 제거해야 합니다.

안전한 코드의 예 C#

```
1: void ButtonClickGood(object sender, EventArgs e)
2: {
3:   string connect = "MyConnString";
4: string usrinput = Request["ID"];
5: //파라미터 바인딩을 위해 @을 사용합니다. 외부입력 값에 의해 쿼리 구조 변경을 할 수 없습니다.
6:   string query = "Select * From Products Where ProductID = @ProductID";
7:   using (var conn = new SqlConnection(connect))
8:   {
9:     using (var cmd = new SqlCommand(query, conn))
10:   {
11:     cmd.Parameters.AddWithValue("@ProductID",
    Convert.ToInt32(Request["ProductID"]);
12:     conn.Open();
13:     cmd.ExecuteReader();
14:   }
15: }
```

라. 진단방법

Statement statement[13] 객체로 쿼리가 실행되는 부분을 확인하고(①), Statement 객체가 Pre-paredStatement 객체인지 확인한다(②). PreparedStatement 객체를 사용하고 setString 등의 메소 드로 외부 입력값을 설정하는 경우에는 기본적으로 안전하다고 판정하지만, 쿼리에 사용되는 변수가 외부 입력값인 경우엔 적절한 필터링 모듈이 존재하는지 추가로 확인해야 한다(③). 즉, 쿼리 생성과 관련된 외부 입력값에 대한 필터링 모듈이 반드시 존재해야 하며, 그 외에도 관련 프레임워크에서 적절한 조치가 이루어질 경우에 안전하다고 판정한다.

일반적인 진단의 예

```
1:  class Login {
2:   public Connection getConnection() throws SQLException {
3:    DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
4:    String dbConnection = PropertyManager.getProperty("db.connection");
5:   return DriverManager.getConnection(dbConnection);
6:   }
7:   // 외부입력 값(username, password)을 제공받음
8:   public void doPrivilegedAction(String username, char[] password) throws SQLException {
9:   Connection connection = getConnection();
10:  if (connection == null) {
11:    // handle error
12:    }
13:    try {
14:    String pwd = HashUtil.hashPassword(password);
15:    String sqlString = "SELECT * FROM db_user WHERE username = '"+ username
       + "' AND password = '" + pwd + "'"; ····· ③
16:    Statement stmt = connection.createStatement(); ····· ②
17:    ResultSet rs = stmt.executeQuery(sqlString); ····· ①
18:    if (!rs.next()) {
19:    throw new SecurityException("User name or password incorrect");
20:    }
21:    // Authenticated; proceed
22:    } finally {
23:    // …
24:   }
25: }
26: }
```

13) statement : 자바에서 사용하는 객체 중 하나

다음의 예제를 살펴보면, 5라인에서 PreparedStatement 객체를 사용하고 있으나 객체를 상수 스트링으로 생성하여 파라미터 부분을 setString 메소드로 설정하지 않아, 4라인에서 사용자의 외부 입력 값이 쿼리문의 구조를 바꾸게 되어 취약하다고 판정한다.

정탐코드의 예

```
1: …
2: String tableName = props.getProperty(“jdbc.tableName”);
3: String name = props.getProperty(“jdbc.name”);
4: String query = “SELECT * FROM “ + tableName + “ WHERE Name =‘” + name + “’”;
5: stmt = con.prepareStatement(query);
6: rs = stmt.executeQuery();
7: …
```

다음의 예제를 살펴보면, 15라인에서 사용자의 입력값이 변수 pid에 저장되고, 19라인에서 commentDao.getProjectCommentTblByWhere 함수의 파라미터로 사용된다. 해당 함수의 134라인에서 문자열을 만들게 되고, 138번 라인에서 해당 문자열을 이용해서 쿼리를 실행하므로 취약하다고 판정 한다.

정탐코드의 예

```
〈 projectDetail.jsp 〉
14: 〈%
15: String pid = request.getParameter(“projectid”) == null ? “” :
    request.getParame- ter(“projectid”);
16:MashupProjectDao mashupDao = (MashupProjectDao)
   DAOHelper.getMashupPro- jectDao(application);
17: ProjectCommentDao commentDao = (ProjectCommentDao)
    DAOHelper.getProject- CommentDao(application);
18: MashupProjectTbl prjTbl = mashupDao.getMashupProjectTbl(pid);
19: ArrayList commentList =
    commentDao.getProjectCommentTblByWhere(“where projectid=” + pid +  “ order by
    writedate desc”);
20: String loginID = session.getAttribute(“userid”) == null ? “” : (String)
    session.getAt- tribute(“userid”);
21: int category = prjTbl.getCategory();
22: String categoryStr = “”;
```

정탐코드의 예

```
〈 ProjectCommentDAO.java 〉
131:public ArrayList getProjectCommentTblByWhere(String where) {
132:   String sql_ =
       this.getMessageSourceAccessor().getMessage("ProjectCommentDao.
       getProjectCommentTbl");
133:    ArrayList ret = new ArrayList();
134:    sql_ += " " + where;
135:
136:    try {
137:     JdbcTemplate template_ = this.getJdbcTemplate();
138:     List tmp = template_.query(sql_, new ProjectCommentRowMapper());
139:     ret = Util.trimToResize(tmp);
140:    } catch (DataAccessException e) {
```

[SQL 삽입 : JDO]

외부의 입력값에 대한 적절한 검사 과정을 거치지 않고, JDO(Java Data Objects) API의 SQL 또는 JDOQL 쿼리 생성을 위한 문자열로 사용하면, 공격자가 프로그래머가 의도하지 않았던 문자열을 전달함으로써 쿼리의 의미를 왜곡시키거나 그 구조를 변경하여 임의의 쿼리 명령어를 수행할 수 있다.

다음의 예제에서는 공격자가 외부의 입력(name) 값을 foo'; DELETE FROM MYTABLE; --로 주게 되면, 다음과 같은 쿼리가 실행되어 테이블이 삭제될 수 있으므로 취약하다고 판정한다.

SELECT col1 FROM MYTABLE WHERE name = 'foo' ; DELETE FROM MYTABLE; --'

정탐코드의 예

```
1: …
2: public class U9102 implements ContactDAO {
3:   public List<Contact> listContacts() {
4:     PersistenceManager pm = getPersistenceManagerFactory().getPersistenceMan ager();
5:     String query = "select from " + Contact.class.getName();
6:     try {
7:       Properties props = new Properties();
8:       String fileName = "contacts.txt";
```

정탐코드의 예

```
9:        FileInputStream in = new FileInputStream(fileName);
10:
11:       if (in != null) {
12:       props.load(in);
13:       in.close();
14:      }
15:      // 외부로부터 입력을 받는다
16:      String name = props.getProperty("name");
17:      if (name != null) {
18:       query += " where name = '" + name + "'";
19:      }
20:    } catch (IOException e) {
21:      …
22:    }
23:    // 외부입력 값이 JDO 객체의 인자로 사용된다.
24:    return (List<Contact>) pm.newQuery(query).execute();
25:  }
26: }
27: …
```

[SQL 삽입 : Persistence]

J2EE Persistence API를 사용하는 응용프로그램에서 외부의 입력값을 검증 없이 쿼리에 그대로 사용하면, 쿼리의 의미를 왜곡시키거나 그 구조를 변경함으로써 임의의 쿼리가 실행될 수 있다. 공격자가 외부 입력(id)의 값으로 foo'; DELETE FROM MYTABLE; --을 주게 되면, 다음과 같은 쿼리가 실행되어 테이블이 삭제될 수 있으므로 취약하다고 판정한다.

SELECT col1 FROM MYTABLE WHERE name = 'foo' ; DELETE FROM MYTABLE; --'

정탐코드의 예

```
1:  public class U9103 implements ServletContextListener {
2:    public List<?> getAllItemsInWildcardCollection() {
3:      EntityManager em = getEntityManager();
4:      List<U9103> r_type = null;
5:       try {
```

정탐코드의 예

```
6:    Properties props = new Properties();
7:    String fileName = "conditions.txt";
8:     FileInputStream in = new FileInputStream(fileName);
9:     props.load(in);
10:    // 외부로부터 입력을 받는다.
11:    String id = props.getProperty("id");
12:    // 외부 입력 값이 query의 인자로 사용이 된다.
13:    Query query = em.createNativeQuery("SELECT OBJECT(i) FROM Item i WHERE
       i.itemID > " + id);
14:    List<U9103> items = query.getResultList();
15:    …
16:    return r_type;
17:  }
18: …
```

[SQL 삽입 : mybatis Data Map]

외부에서 입력된 값이 쿼리의 인자값으로만 사용되지 않고, 쿼리 명령어에 연결되는 문자열 로 사용되면, 공격자가 의도하지 않았던 문자열을 전달함으로써 쿼리의 의미를 왜곡시키거나, 그 구조를 변경하여 임의의 DB 명령어를 수행할 수 있다.

다음의 예제는 mybatis Data Map에서 사용하는 쿼리 설정파일(XML)이다. 정의된 쿼리 중 delStu- dent 명령어 선언에서 쿼리에 삽입되는 인자들 중 $name$으로 전달되는 문자열 값은 그대로 연결되어 쿼리가 만들어진다. 따라서 만약 NAME의 값으로 ' OR 'x'='x 을 전달하면 다음과 같은 쿼리가 실행되어 테이블의 모든 원소를 삭제할 수 있으므로 취약하다고 판정한다.

DELETE STUDENTS WHERE NUM = #num# and NAME = '' OR 'x'='x'

정탐코드의 예

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN"
   "http://www. ibatis.com/dtd/sql-map-2.dtd">
3: <sqlMap namespace="Student">
4:  <resultMap id="StudentResult" class="Student">
```

정탐코드의 예

```
5:  〈result column="ID" property="id" /〉
6:  〈result column="NAME" property="name" /〉
7:  〈/resultMap〉
8: 〈select id="listStudents" resultMap="StudentResult"〉
9:  SELECT NUM, NAME
10: FROM STUDENTS
11: ORDER BY NUM
12:〈/select〉
13:〈select id="nameStudent" parameterClass="Integer" resultClass="Student"〉
14: SELECT NUM, NAME
15: FROM STUDENTS
16: WHERE NUM = #num#
17:〈/select〉
18:〈!-- dynamic SQL 사용 --〉
19:〈delete id="delStudent" parameterClass="Student"〉
20: DELETE STUDENTS
21: WHERE NUM = #num# AND NAME = '$name$'
22:〈/delete〉
23:〈/sqlMap〉
```

[SQL 삽입 : Hibernate]

외부의 신뢰할 수 없는 입력을 적절한 검사 과정을 거치지 않고 Hibernate API의 SQL 쿼리 생성을 위한 문자열로 사용하면, 공격자가 프로그래머가 의도하지 않았던 문자열을 전달함으로써 쿼리의 의미를 왜곡시키거나 그 구조를 변경하여 임의의 DB 명령어가 수행되도록 할 수 있다.

다음의 예제에서는 외부의 입력(name)을 아무 검증과정 없이 쿼리에 그대로 사용하고 있다. 만일, 외부의 입력으로 "n' or '1'='1"과 같은 문자열이 입력되면, 다음과 같은 쿼리가 실행되어 테이블 내의 모든 레코드가 반환될 수 있으므로 취약하다고 판정한다.

"from Student where studentName='n' or '1'='1'"

정탐코드의 예

```
1: import org.hibernate.Query
2: import org.hibernate.Session
```

정탐코드의 예

```
3: ......
4: String name = request.getParameter("name");
5: Query query = session.createQuery("from Student where studentName = '" + name
   + "' ");
```

아래 코드에서는 외부 입력값이 위치하는 부분을 ?로 설정하고, 실행 시에 해당 파라미터가 전달되도록 수정함으로써 외부의 입력(name)이 쿼리의 구조를 변경시키는 것을 방지하고 있어 취약하지 않다.

오탐코드의 예

```
1: import org.hibernate.Query
2: import org.hibernate.Session
3: ......
4: String name = request.getParameter("name");
5: Query query = session.createQuery("from Student where studentName = ? ");
6: query.setString(0, name);
```

오탐코드의 예

```
1: import org.hibernate.Query
2: import org.hibernate.Session
3: ......
4: String name = request.getParameter("name");
5: Query query = session.createQuery("from Student where studentName = :name ");
6: query.setParameter("name", name);
```

마. 참고자료

[1] CWE-89 SQL Injection, MITRE,
http://cwe.mitre.org/data/definitions/89.html

[2] Threat and Vulnerability, "SQL Injection", Microsoft,
http://technet.microsoft.com/en-us/library/ms161953%28v=SQL.105%29.aspx

[3] Input validation and Data Sanitization, Threat and Vulnerability, Prevent SQL Injection, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS00-J.+Prevent+SQL+injection

[4] SQL Injection Prevention Cheat Sheet, OWASP
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

2. 코드삽입

가. 개요

공격자가 소프트웨어의 의도된 동작을 변경하도록 임의 코드를 삽입하여 소프트웨어가 비정상적으로 동작하도록 하는 보안약점을 말한다. 코드 삽입은 프로그래밍 언어 자체의 기능에 의해서만 제한된다는 점에서 운영체제 명령어 삽입과 다르다.

취약한 프로그램에서 사용자의 입력 값에 코드가 포함되는 것을 허용할 경우, 공격자는 개발자가 의도하지 않은 코드를 실행하여 권한을 탈취하거나 인증 우회, 시스템 명령어 실행 등을 할 수 있다.

나. 보안대책

동적코드를 실행할 수 있는 함수를 사용하지 않는다. 필요 시, 실행 가능한 동적코드를 입력 값으로 받지 않도록, 외부 입력 값에 대하여 화이트리스트 방식으로 구현한다. 또는 유효한 문자만 포함하도록 동적 코드에 사용되는 사용자 입력 값을 필터링 한다.

다. 코드예제

다음 예제의 소스코드는 javax.script.ScriptEngineManager을 사용하여 ScriptEnigneManeger()로 사용자의 입력을 실행하여 출력한다. 이 경우, 공격자는 조작된 인수를 입력한 공격코드를 이용하여 새로운 파일을 만들거나 덮어씌울 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: public class CodeInjectionController {
2: @RequestMapping(value = "/execute", method = RequestMethod.GET)
3:       public String execute(@RequestParam("src") String src)
4:       throws ScriptException {
5:               ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
6:               ScriptEngine scriptEngine= scriptEngineManager.getEngineByName("javascript");
7:               // 외부 입력값인 src를 javascript eval 함수로 실행하고 있어 안전하지 않다.
8:               String retValue = (String)scriptEngine.eval(src);
9:               return retValue;
10:      }
11: }
```

다음 예제는 외부 입력 값을 javascript의 new Function()으로 동적으로 코드를 실행할 수 있다.

안전하지 않은 코드의 예 JSP

```
1: 〈body〉
2: 〈%
3:        String name = request.getparameter("name");
4: %〉
5: ...
6:〈script〉
7: // 외부 입력값인 name을 javascript new Function()을 이용하여 문자열을 함수로 실행하고 있다.
8:       (new Function(〈%=name%〉))();
9: 〈/script〉
10: 〈/body〉
```

외부 입력 값에 실행이 가능한 코드가 포함되어 있을 경우 입력 값을 필터링 하여 사전에 검증하는 코드를 추가하면 코드삽입을 완화할 수 있다. 이런 조치를 취할 경우 입력 값의 형태에 따라 정규 표현식을 변형하여 적용해야 한다.

안전한 코드의 예 JAVA

```
1: @RequestMapping(value = "/execute", method = RequestMethod.GET)
2: public String execute(@RequestParam("src") String src) throws ScriptException {
3:        // 정규식을 이용하여 특수문자 입력시 예외를 발생시킨다.
4:        if (src.matches("[\\w]*") == false) {
5:                throw new IllegalArgumentException();
6:        }
7:        ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
8:        ScriptEngine scriptEngine = scriptEngineManager.getEngineByName("javascript");
9:        String retValue = (String)scriptEngine.eval(src);
10:       return retValue;
11: }
```

스크립트 실행이 필요한 경우는 화이트리스트 방식을 적용하여 유효한 문자인 경우에만 실행되도록 하고 그 외의 경우는 모두 예외 처리한다.

안전한 코드의 예 JAVA

```
1: @RequestMapping(value = "/execute", method = RequestMethod.GET)
2: public String execute(@RequestParam("src") String src) throws ScriptException {
3:         // 유효한 문자 "_" 일 경우 실행할 메소드 호출한다.
4:         if (src.matches("UNDER_BAR") == true) {
5:                 ...
6:                 // 유효한 문자 "$" 일 경우 실행할 메소드 호출한다.
7:                 } else if (src.matches("DOLLAR") == true) {
8:                         ...
9:                 // 유효하지 않은 특수문자 입력시 예외를 발생시킨다.
10:                } else {
11:                        throw new IllegalArgumentException();
12:                }
13:        ...
14: }
```

라. 진단방법

각 언어에서 제공하고 있는 동적실행 함수를 확인한다(①). 아래 예제에서는 자바 코드에서 자바 스크립트를 사용하고 있다. 동적코드 실행에 사용되는 데이터가 신뢰할 수 있는 값인지 확인한다.(②) 이 때, 데이터가 신뢰할 수 없는 값이나 별도의 검증절차가 없으면 취약하다고 판단한다.

일반적인 진단의 예

```
1: public class CodeInjection {
2:         // eval 실행 데이터가 프로그램 실행 파라미터이다.
3:         public static void main(String[] args) throws ScriptException {  ····· ②
4:                 ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
5:                 ScriptEngine scriptEngine= scriptEngineManager.getEngineByName("javascript");
6:                 String src = "print('" + args[1] + "');";
7:                 scriptEngine.eval(src);  ····· ①
8:         }
9: }
```

다음 예제에서는 6라인에서 동적코드 실행 함수인 ScriptEngine.eval함수를 사용하고 있으며, 11라인의 실행코드 데이터 name은 사용자의 입력 값이므로 취약하다고 판정한다.

정탐코드의 예

```
1: class ACC {
2:         public static void executeScript(final String name) throws ScriptException {
3:                 ScriptEngineManager manager = new ScriptEngineManager();
4:                 ScriptEngine engine = manager.getEngineByName("javascript");
5:                 // eval을 이용해서 변수 name 값을 표시한다.
6:                 engine.eval("print('" + name + "')");
7:         }
8: }
9:         ...
10: @RequestMapping(value = "/print", method = RequestMethod.POST)
11: public void print(@RequestParam("name") String name) throws ScriptException{
12:       // 사용자 입력값인 name을 이용해서 eval 을 사용하는 함수를 호출한다.
13:       ACC.executeScript(name);
14:       ...
15:}
```

PHP의 eval() 함수는 PHP 코드를 실행하는 함수이며, 괄호 안의 문자열을 실행한다. 아래 코드는 공격자가 2라인의 arg 파라미터에 아래와 같은 명령어를 추가하여 PHP 삽입코드를 실행될 수 있으나 이를 사전에 검증하지 않고 3라인에서 코드를 실행하기 때문에 취약한 것으로 판단한다.

[삽입코드의 예]

```
/vul.php?arg=1;phpinfo()
/vul.php?arg=1;system("uname -a")
/vul.php?arg=1;system("cat /etc/passwd")
/vul.php?arg=1;system("ps -ef")
```

정탐코드의 예

```
vul.php
1: $myvar = 'somevalue';
2: $x = $_GET['arg'];
3: eval('$myvar = ' . $x . ';');
```

아래 예에서는 사용자 메시지를 파일에 쓰고 사용자가 볼 수 있도록 한다. 이 때, 공격자는 3~4라인의 name과 message 파라미터에 악의적인 메시지를 입력하면 필터링 과정 없이 6라인에서 파일에 저장된다. PHP가 파싱하고 실행하는 과정에서 메시지 조회 기능을 수행하게 될 경우 10라인의 메시지 출력 시 저장된 파일을 보여주게 되어 의도하지 않은 PHP 명령어가 실행될 수 있으므로 취약하다고 판단한다.

정탐코드의 예

```
1: $MessageFile = "cwe-94/messages.out";
2: if ($_GET["action"] == "NewMessage") {
3:         $name = $_GET["name"];
4:         $message = $_GET["message"];
5:         $handle = fopen($MessageFile, "a+");
6:         fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
7:         fclose($handle);
8:         echo "Message Saved!<p>\n";
9: } else if ($_GET["action"] == "ViewMessages") {
10:        include($MessageFile);
11:}
```

대부분의 경우 애플리케이션을 보호하기 위해 잠재적으로 위험한 모든 기호를 제거하면 문제 발생을 줄일 수 있다. arg 파라미터가 eval() 함수에서 사용되기 전에 아래 예제의 3라인과 같이 잠재적으로 위험한 기호를 제거하기 때문에 취약하지 않다고 판정한다.

오탐코드의 예

```
1 $myvar = "varname";
2 $x = $_GET['arg'];
3: $x=preg_replace("/[w\\^a-z0-9]/i", "", $x);
4: eval("\$myvar = \$x;");
```

다음의 예제코드는 외부로부터 입력 값을 받아 사용하기 전에 검증을 수행하기 때문에 취약하지 않다고 판정한다.

오탐코드의 예

```
1: class ACC {
2:         public static void executeScript(final String name) throws ScriptException {
3:                 if (!Filter.filterScript(name)) {
4:                         throw new IllegalArgumentException();
5:                 }
6:         ScriptEngineManager manager = new ScriptEngineManager();
7:         ScriptEngine engine = manager.getEngineByName("javascript");
8:         engine.eval("print('" + name + "')");
9:         ...
10:        }
11:...
12: @Controller
13: public class evalPrint {
14:         @RequestMapping(value = "/print", method = RequestMethod.POST)
15:         public Student upload(String name) throws ClassNotFoundException, IOException {
16:                 ACC.executeScript(name);
17:                 ...
18:                 public class Filter {
19:                         public static boolean filterScript(String name) {
20:                                 boolean returnValue = false;
21:                                 // 알파벳, 숫자, '_' 만을 허용한다.
22:                                 if (name.matches("[\\w]*")) {
23:                                         returnValue = true;
24:                                 }
25:                         return returnValue;
26:                 }
27:        }
```

다음의 예제코드는 동적 실행함수를 사용하였으나 외부 입력값을 실행하지 않아 취약하지 않다고 판정한다.

오탐코드의 예

```
1: 〈script〉
2:        var obj = {};
3:        for(var i = 0; i 〈 5; i++){
4:        eval("obj.test" + i + "=" + i);
5:        }
6:        console.log(obj);
7: 〈/script〉
```

마. 참고자료

① CWE-94: Improper Control of Generation of Code ('Code Injection') , MITRE, http://cwe.mitre.org/data/definitions/94.html

② CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'), MITRE, http://cwe.mitre.org/data/definitions/95.html

③ Code Injection Software Attack, OWASP, https://owasp.org/www-community/attacks/Code_Injection

3. 경로 조작 및 자원 삽입

가. 개요

검증되지 않은 외부입력값으로 파일 및 서버 등 시스템 자원에 대한 접근 혹은 식별을 허용할 경우, 입력값 조작으로 시스템이 보호하는 자원에 임의로 접근할 수 있는 보안약점이다. 경로조작 및 자원삽입 약점을 이용하여 공격자는 자원의 수정, 삭제, 시스템 정보누출, 시스템 자원 간 충돌로 인한 서비스 장애 등을 유발시킬 수 있다.

즉, 경로 조작 및 자원 삽입으로 공격자가 허용되지 않은 권한을 획득하여, 설정에 관계된 파일 을 변경하거나 실행시킬 수 있다.

나. 보안대책

외부의 입력을 자원(파일, 소켓의 포트 등)의 식별자로 사용하는 경우, 적절한 검증을 거치도록 하거나, 사전에 정의된 적합한 리스트에서 선택되도록 한다. 특히, 외부의 입력이 파일명인 경우에는 경로순회(directory traversal)[14] 공격의 위험이 있는 문자(“ / \ .. 등)를 제거할 수 있는 필터를 이용한다.

다. 코드예제

외부 입력값(P)이 버퍼로 내용을 옮길 파일의 경로설정에 사용되고 있다. 만일 공격자에 의해 P의 값으로 ../../../rootFile.txt와 같은 값을 전달하면 의도하지 않았던 파일의 내용이 버퍼에 쓰여 시스템에 악영향을 준다.

14) 경로순회 : 상대경로 참조 방식(“./”,“../”등)을 이용해 다른 디렉토리의 중요파일에 접근하는 공격방법으로 경로추적이라고도 한다.

안전하지 않은 코드의 예 JAVA

```
1: //외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
2: String fileName = request.getParameter("P");
3: BufferedInputStream bis = null;
4: BufferedOutputStream bos = null;
5: FileInputStream fis = null;
6: try {
7:     response.setHeader("Content-Disposition", "attachment;filename="+fileName+";");
8:     ...
9:     //외부로부터 입력받은 값이 검증 또는 처리 없이 파일처리에 수행되었다.
10:    fis = new FileInputStream("C:/datas/" + fileName);
11:    bis = new BufferedInputStream(fis);
12:    bos = new BufferedOutputStream(response.getOutputStream());
```

외부 입력값에 대하여 상대경로를 설정할 수 없도록 경로순회 문자열(/ \ & .. 등)을 제거하고 파일의 경로설정에 사용한다.

안전한 코드의 예 JAVA

```
1:  String fileName = request.getParameter("P");
2:  BufferedInputStream bis = null;
3:  BufferedOutputStream bos = null;
4:  FileInputStream fis = null;
5:  try {
6:      response.setHeader("Content-Disposition", "attachment;filename="+fileName+";");
7: ...
8:  // 외부 입력받은 값을 경로순회 문자열(./\)을 제거하고 사용해야한다.
9:      filename = filename.replaceAll("\\.", "").replaceAll("/", "").replaceAll("\\\\", "");
10:     fis = new FileInputStream("C:/datas/" + fileName);
11:     bis = new BufferedInputStream(fis);
12:     bos = new BufferedOutputStream(response.getOutputStream());
13:     int read;
14:     while((read = bis.read(buffer, 0, 1024)) != -1) {
```

안전한 코드의 예 JAVA

```
15:         bos.write(buffer,0,read);
16:     }
17: }
```

인자값이 파일 이름인 경우에는 애플리케이션에서 정의(제한)한 디렉토리 c:\help_files\에서 파일을 읽어서 출력하지만, args[0]의 값으로 "..\..\..\windows\system32\drivers\etc\hosts"와 같이 경로조작 문자열을 포함한 입력이 들어오는 경우 접근이 제한된 경로의 파일을 열람할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: public class ShowHelp {
2:     private final static String safeDir = "c:\\help_files\\";
3:     public static void main(String[] args) throws IOException {
4:         String helpFile = args[0];
5:         try (BufferedReader br = new BufferedReader(new FileReader(safeDir + helpFile))) {
6:             String line;
7:             while ((line = br.readLine()) != null) {
8:                 System.out.println(line);
9:             }
10:            ...
11:        }
```

외부 입력값으로 파일 경로를 조합하여 파일 시스템에 접근하는 경로를 만들지 말아야 한다.

외부에서 입력되는 값에 대하여 null 여부를 체크하고, 외부에서 입력되는 파일 이름에서 경로조작 문자열 제거 조치 후 사용하도록 한다.

안전한 코드의 예 JAVA

```
1: public class ShowHelpSolution {
2:     private final static String safeDir = "c:\\help_files\\";
3:     //경로조작 문자열 포함 여부를 확인하고 조치 후 사용하도록 한다.
```

안전한 코드의 예 JAVA

```
4:    public static void main(String[] args) throws IOException {
5:     String helpFile = args[0];
6:     if (helpFile != null) {
7:        helpFile = helpFile.replaceAll("\\.{2,}[/\\\\]", "");
8:     }
9:     try (BufferedReader br = new BufferedReader(new FileReader(safeDir + helpFile))) {
10:    ...
```

다음 C# 코드는 외부 입력값을 파일명에 바로 사용하고 있다. 이는 의도치 않은 파일의 손상을 가져올 수 있다.

안전하지 않은 코드의 예 C#

```
1: //외부 입력 값이 검증 없이 파일처리에 사용 되었다.
2: string file = Request.QueryString["path"];
3:  if (file != null)
4:  {
5:      File.Delete(file);
6:  }
```

외부 입력값에 경로 조작 문자열을 제거하여 조작 위험을 없앨 수 있다.

안전한 코드의 예 C#

```
1: string file = Request.QueryString["path"];
2:  if (file != null)
3: {
4: //경로조작 문자열이 있는지 확인하고 파일 처리를 하도록 한다.
5:    if (file.IndexOf('\\') > -1 || file.IndexOf('/') > -1)
6:    {
7:     Response.Write("Path Traversal Attack");
8:    }
```

안전한 코드의 예 C#

```
9:    else
10:   {
11:     File.Delete(file);
12:   }
13: }
```

아래 C 코드는 외부 입력 값을 파일 경로로 바로 사용하고 있다. 이는 공격자가 환경 변수 reportfile을 조작하여 디렉토리 경로를 조작할 수 있다.

안전하지 않은 코드의 예 C

```
1:  char* filename = getenv("reportfile");
2:  FILE *fin = NULL;
3:  // 외부 설정 값에서 받은 파일 이름을 그대로 사용한다.
4:  fin = fopen(filename, "r");
5:  while (fgets(buf, BUF_LEN, fin)) {
6:    // 파일 내용 출력
7: }
```

아래 C 코드는 외부에서 불러온 파일 이름을 그대로 사용하지 않고 경로 조작 가능성이 있는 문자열을 검증하고 사용한다.

안전한 코드의 예 C

```
1: FILE *fin = NULL;
2:  regex_t regex;
3:  Int ret;
4:  char* filename = getenv("reportfile");
5:  ret = regcomp(&regex, ".*\\.\\..*", 0);
6:  // 경로 조작 가능성 있는 문자열 탐지
7:  ret = regexec(&regex, filename, 0, NULL, 0);
8:  If (!ret) {
```

안전한 코드의 예 C

```
9:   // 경로 조작 문자열 발견, 오류 처리
10: }
11: // 필터링된 파일 이름으로 사용
12: fin = fopen(filename, "r");
13: while (fgets(buf, BUF_LEN, fin)) {
// 파일 내용 출력
15: }
```

라. 진단방법

디렉토리 경로나 자원에 대한 접근 경로 생성, 자원을 제어하기 위한 코드는 외부의 입력값이 경로 생성과 자원제어에 직접적으로 영향을 주어서는 안된다. 따라서, 외부 입력값이 해당 작업에 직접적인 영향을 미치는지 확인한다. 이를 효과적으로 판단하기 위한 방법은 다음과 같다. 먼저 경로 생성과 자원제어를 위한 코드에 외부 입력값을 사용하지지 여부를 확인한 후, 외부 입력값을 사용하는 경우 적절한 필터링이 이루어지거나 리스트의 값이 선택되는 지 확인한다. 또한, 시스템 경로를 얻기 위해 경로를 직접 입력하는 대신 시스템 함수를 호출하여 경로를 생성하는 것이 바람직하므로, 이러한 함수가 사용되는지 확인한다.

[경로 조작]

파일객체를 사용하는지 확인하고(①), 해당 파일에 대한 접근이 외부에서 직접 접근하는지 확인한다(②). 파일객체가 가리키는 경로에 외부 입력 값이 경로 순회 문자열에 대한 제거없이 사용되면 경로의 변경이 가능하게 되어 취약하다고 판정한다.

일반적인 진단의 예

```
1: …
2: String basePath = "/web/data/";
3: String filename = request.getParameter("filename");  ……②
4: String fullPath = basePath + filename;
5: …
6: File f = new File(fullPath);  ……①
7: if (f.isFile()) {
8:     FileInputStream in = new FileInputStream(fullPath);
9:     ServletOutputStream os = response.getOutputStream();
```

일반적인 진단의 예

```
10:  byte[] buf = new byte[1024];
11:  int len = 0;
12:  while ((len = in.read(buf)) > 0) {
13:    os.write(buf, 0, len);
14:            }
15:  }
16:  …
```

다음의 예제는 4라인에서 외부에서 파일경로를 직접 접근하는 변수에 대해서 9~11라인에서 경로순회 문자 필터링을 하였으나 13~14라인에서 컨텍스트 경로에 접근이 가능하도록 되어있어 웹 및 서블릿 설정 파일에 접근이 가능하다.

정탐코드의 예

```
1: …
2: @RequestMapping(value = "/Download.do")
3: public void downloadFile(
4:      @RequestParam(value = "filePath", required = true) String filePath
5:      , HttpServletRequest request
6:      , HttpServletResponse response) throws Exception {
7: …
8:   filePath = filePath.replaceAll("\\.", "");
9:   filePath = filePath.replaceAll("/", "");
10: filePath = filePath.replaceAll("\\\\", "");
11: String targetFile = getServletContext().getRealPath("/") + filePath;
12: File file = new File(targetFile);
13:…
```

[자원 삽입]

파일명, 소켓의 포트 등과 같은 자원을 사용하는지 확인하고(①), 해당 자원에 대한 접근이 외부에서 직접 접근하는지 확인한다(②). 직접 접근하지 않고 매핑표나 리스트를 가질 경우엔 기본적으로 안전하다고 판단하고, 그 외의 경우엔 취약하다고 판정한다.

일반적인 진단의 예

```
1: public class U99 {
2:   public void f() throws IOException {
3:     int def = 1000;
4:     ServerSocket serverSocket;
5:     Properties props = new Properties();
6:     String fileName = "file_list";
7:     FileInputStream in = new FileInputStream(filename);
8:     props.load(in);
9:
10:    String service = props.getProperty("Service No");
11:    int port = Integer.parseInt(service);  ..... ②
12:
13:    if (port != 0)
14: serverSocket = new ServerSocket(port + 3000);  ..... ①
15:    else
16: serverSocket = new ServerSocket(def + 3000);  ..... ①
17:    ServerSocket.close();
18:    }
19: }
```

중요 자원(파일, 소켓 등)을 사용할 때, 식별자를 구성하는 외부 입력값에 대한 검증 프로세스가 존재하지 않으면 취약한 것으로 판정한다.

다음의 예제를 살펴보면, EgovFileCmprs.jsp에서 10라인에 외부 입력값인 source를 받아서 13라인에서 static 함수인 EgovFileCmprs.cmprsFile를 호출하고 있다. 이 함수 내에서 7라인에서 source 파라미터를 source1 변수에 저장하고, 9라인에서 그 변수를 사용하여 파일을 생성하고 있어 취약하다고 판정한다.

정탐코드의 예

```
〈 EgovFileCmprs.jsp 〉
8: 〈%
9: boolean isCompressed = false;
```

정탐코드의 예

```
10:String source = request.getParameter("source");
11:String target = request.getParameter("target");
12:if (source != null && source.length() > 0 && target != null && target.length() > 0) {
13:isCompressed = EgovFileCmprs.cmprsFile(source, target);
14:…
15:%>
```

〈 EgovFileCmprs.java 〉

```
1: public static boolean cmprsFile(String source, String target) throwsException {
2: boolean result = false;
3: int cnt = 0;
4: FileInputStream finput = null;
5: FileOutputStream foutput = null;
6: ZipOutputStream zoutput = null;
7: String source1 = source.replace('\\', FILE_SEPARATOR).replace('/',
   FILE_SEPARA- TOR);
8: String target1 = target.replace('\\', FILE_SEPARATOR).replace('/', FILE_SEPARATOR);
9: File srcFile = new File(source1);
```

다음의 예제에서 사용되는 getRealPath(), getContextPath() 등 내부 함수는 소스가 안전하므로 취약하지 않다고 판정한다.

오탐코드의 예

```
1: File file = new File(getServletConfig().getServletContext().getRealPath("/")+
2: "/jsp/gis/gcc/cmm/xml/"+(String)tempChartImgList.get(k));
```

다음의 예제에서 사용되는 getRealPath(), getContextPath() 등 내부 함수는 소스가 안전하므로 취약 하지 않다고 판정한다.

오탐코드의 예

```
1: File[] f_list = f.listFiles( new FilenameFilter() {
2: public boolean accept(File dir, String name) {
3: String org_code = SystemierConfig.getPropertiesBean().getProperty("is.OrgCode");
```

오탐코드의 예

```
4: if(name.matches(org_code + ".*CVDTST.*" + ".txt")) {
5: return true;
6: }
7: return false;
8: }
9: });
10:BufferedReader in = new BufferedReader( new FileReader(f_list[0].getPath()) );
```

일반적으로 프레임워크를 사용하면서 시스템 프로퍼티에서 가져오는 명령어는 취약하지 않다. 하지만 시스템 프로퍼티 등 공용으로 사용하는 프로퍼티가 아닌 개별 사용 프로퍼티일 경우는 정탐, 시스템 프로퍼티 사용은 오탐으로 처리한다.

마. 참고자료

[1] CWE-99 Resource Injection, MITRE,
http://cwe.mitre.org/data/definitions/99.html

[2] CWE-22 Path Traversal, MITRE,
http://cwe.mitre.org/data/definitions/22.html

[3] Path Traversal, OWASP,
https://www.owasp.org/index.php/Path_Traversal

4. 크로스사이트 스크립트

가. 개요

웹 페이지에 악의적인 스크립트를 포함시켜 사용자 측에서 실행되게 유도할 수 있다. 예를 들어, 검증되지 않은 외부 입력이 동적 웹페이지 생성에 사용될 경우, 전송된 동적 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 수행되어 정보유출 등의 공격을 유발할 수 있다.

나. 보안대책

외부입력값에 스크립트가 삽입되지 못하도록 문자변환 함수 또는 메서드를 사용하여 〈 〉 & " 등을 < > & "로 치환한다. HTML태그를 허용하는 게시판에서는 허용되는 HTML 태그들을 화이트리스트로 만들어 해당 태그만 지원하도록 한다.

다. 코드예제

크로스사이트 스크립트(XSS)는 크게 3가지 공격 방법이 존재한다.

Reflected XSS 공격은 검색 결과, 에러 메시지 등으로 서버가 외부에서 입력받은 악성 스크립트가 포함된 URL 파라미터 값을 사용자 브라우저에서 응답할 때 발생한다. 공격 스크립트가 삽입된 URL을 사용자가 쉽게 확인할 수 없도록 변형하여, 이메일, 메신저, 파일 등으로 실행을 유도하는 공격이다.

Stored XSS 공격은 웹 사이트의 게시판, 코멘트 필드, 사용자 프로필 등의 입력 form으로 악성 스크립트를 삽입하여 DB에 저장되면, 사용자가 사이트를 방문하여 저장되어 있는 페이지에 정보를 요청할 때, 서버는 악성 스크립트를 사용자에게 전달하여 사용자 브라우저에서 스크립트가 실행 되면서 공격한다.

DOM기반 XSS 공격은 외부에서 입력받은 악성 스크립트가 포한된 URL 파라미터 값이 서버를 거치지 않고, DOM 생성의 일부로 실행되면서 공격한다. Reflected XSS 및 Stored XSS 공격은 서버 애플리케이션 취약점으로 인해, 응답 페이지에 악성 스크립트가 포함되어 브라우저로 전달되면서 공격하는 것인 반면, DOM기반 XSS는 서버와 관계없이 발생하는 것이 차이점이다.

안전하지 않은 코드의 예 JAVA

```
1: 〈% String keyword = request.getParameter("keyword"); %〉
2: //외부 입력값에 대하여 검증 없이 화면에 출력될 경우 공격스크립트가 포함된 URL을 생성 할 수 있어 안전하지 않다.(Reflected XSS)
3: 검색어 : 〈%=keyword%〉
```

안전하지 않은 코드의 예 JAVA

```
1: //게시판 등의 입력form으로 외부값이 DB에 저장되고, 이를 검증 없이 화면에 출력될 경우 공격스크립트가 실행되어 안전하지 않다.(Stored XSS)
2: 검색결과 : ${m.content}
3: 〈script type="text/javascript"〉
4: //외부 입력값에 대하여 검증 없이 브라우저에서 실행되는 경우 서버를 거치지 않는 공격스크립트가 포함된 URL을 생성 할 수 있어 안전하지 않다. (DOM 기반 XSS)
5: document.write("keyword:" + 〈%=keyword%〉);
6: 〈/script〉
```

외부 입력값 파라미터나 게시판 등의 form에 의해 서버의 처리 결과를 사용자 화면에 출력하는 경우, 입력값에 대해서 문자열 치환 함수를 이용하여 스크립트 문자열을 제거하거나, JSTL을 이용하여 출력하거나, 잘 만들어진 외부 XSS 방지 라이브러리를 활용하는 것이 안전하다.

크로스사이트 스크립트의 경우 동작 상황에 따라 동일한 조치방법을 사용하면, 크로스사이트 스크립트 방지는 되더라도 원하는 동작이 정상적으로 되지 않을 수 있기 때문에, 잘 만들어진 외부 XSS방지 라이브러리를 이용하여 각 동작 상황에 따라 적절하게 사용하는 것을 권장한다.

안전한 코드의 예 JAVA

```
1: 〈% String keyword = request.getParameter("keyword"); %〉
2: // 방법1. 입력값에 대하여 스크립트 공격가능성이 있는 문자열을 치환한다.
3: keyword = keyword.replaceAll("&", "&amp;");
4: keyword = keyword.replaceAll("<", "&lt;");
5: keyword = keyword.replaceAll(">", "&gt;");
6: keyword = keyword.replaceAll("\"", "&quot;");
```

안전한 코드의 예 JAVA

```
7: keyword = keyword.replaceAll("'", "&#x27;");
8: keyword = keyword.replaceAll("/", "&#x2F;");
9: keyword = keyword.replaceAll("(", "&#x28;");
10:keyword = keyword.replaceAll(")", "&#x29;");
11:검색어 : 〈%=keyword%〉
12://방법2. JSP에서 출력값에 JSTL c:out 을 사용하여 처리한다.
13:〈%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%〉
14:〈%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %〉
15:검색결과 : 〈c:out value="${m.content}"/〉
16:〈script type="text/javascript"〉
17: //방법3. 잘 만들어진 외부 라이브러리를 활용(NAVER Lucy-XSS-Filter, OWASP ESAPI,
    OWASP Java-Encoder-Project)
18:document.write("keyword:“ +
    〈%=Encoder.encodeForJS(Encoder.encodeForHTML(keyword))%〉);
19: 〈/script〉
```

다음 C# 코드는 외부 입력값을 출력에 바로 사용하고 있다. 이는 XSS 공격을 유발 할 수 있다.

안전하지 않은 코드의 예 C#

```
1: string usrInput = Request.QueryString["ID"];
2: // 외부 입력 값이 검증 없이 화면에 출력 됩니다.
3: string str = "ID : " + usrinput;
4: Request.Write(str);
```

AntiXss 패키지 등을 사용하여 XSS 공격을 예방할 수 있다.

안전한 코드의 예 C#

```
1: string usrInput = Request.QueryString["ID"];
2: string str = "ID : " + usrinput;
3: //AntiXss 패키지 등을 이용하여 외부 입력값을 필터링 합니다.
```

안전한 코드의 예 C#

```
4: var sanitizedStr = Sanitizer.GetSafeHtmlFragment(str);
5: Request.Write(sanitizedStr);
```

아래 C 코드 예제는 외부 입력값으로 사용자로부터 받은 입력값을 검증 없이 바로 cgi에 출력하는 화면이다.

안전하지 않은 코드의 예 C

```
1: int XSS(int argc, char* argv[]) {
2:     unsigned int i = 0;
3:     char data[1024];
4:     …
5:     // cgiFromString으로 받아온 사용자 입력값이 검증 없이 화면에 출력됩니다.
6:     cgiFromString("user input", data, sizeof(data));
7:     fprintf(cgiOut, "Print user input = %s<br/>", data);
8:     fprintf(cgiOut, "</body></html>\n");
9:     return 0;13:
10: }
```

cgi에 출력하기 전에 사용자 입력값을 검증하여야 한다.

안전한 코드의 예 C

```
1: cgiFromString("user input", data, sizeof(data));
2: // data에 위험한 문자열을 검사하는 코드를 추가한다.
3: if(strchr(p, '<')) return;
4: if(strchr(p, '>')) return;
5: …
6: fprintf(cgiOut, "Print user input = %s<br/>", data);
7: fprintf(cgiOut, "</body></html>\n");
```

라. 진단방법

웹 페이지로 출력하는 변수값이 존재하는지 확인하고(①), 해당 변수값이 외부 입력값 또는 사용자 입력 데이터 폼에 의해 저장된 데이터 베이스 값인지 확인한(②) 후 변수값이 적절하게 필터를 거치는지 확인한다. 적절한 필터를 거친 후 출력되는 경우나 프레임워크 등을 사용하여 자체적인 검증 기능이 존재하면 안전하지만 그 외에는 취약하다.

일반적인 진단의 예

```
1:  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2:  <html>
3:  <head>
4:  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5:  </head>
6:  <body>
7:  <h1>XSS Sample</h1>
8:  <%
9:  // 외부로부터 이름을 받음
10: String name = request.getParameter("name"); ····· ②
11: String title = null;
12: …
13: rs = pstmt.executeQuery(query);
14: while(rs.next()){
15: title = rs.getString("title")) ····· ②
16: …}
17: %>
18:
19: <!-- 외부로부터 받은 name이 적절한 필터없이 그대로 출력 -->
20: <p>NAME:<%=name%></p> ····· ①
21: <p>TITLE:<%=title%></p> ····· ①
22: </body>
23: </html>
```

[크로스사이트 스크립트 : Reflected]

다음의 예제에서 @ModelAttribute("apApsCommonCodeVO") ApApsCommonCodeVO apApsCommonCodeVO 어노테이션에 따라 apsApsCommonCodeVO는 외부 입력값이다. 20라인에서 외부 입력값의 target값을 저장하고 26번째 라인에서 외부 입력값을 response의 output stream에 출력하고 있다. 그 과정에서 입력값에 대한 필터링이 없으므로 취약한 것으로 판정한다.

정탐코드의 예

```
16: @RequestMapping("/aps/common/makeCommonCode.do")
17: public void makeCommonCode(HttpSession session,
    @ModelAttribute("apApsCom- monCodeVO")
18: ApApsCommonCodeVO apApsCommonCodeVO, HttpServletRequest request,
    HttpServletResponse
19: response, ModelMap model) throws Exception {
20: String target = apApsCommonCodeVO.getTarget();
21: List resultList = apApsExcelService.selectCommonCode(apApsCommonCodeVO);
22: JSONArray jsonArray = new JSONArray();
23: jsonArray = JSONArray.fromObject(resultList);
24: response.setContentType("text/xml;charset=utf-8");
25: PrintWriter printWriter = response.getWriter();
26: printWriter.print(target);
27: printWriter.print(jsonArray.toString());
28: printWriter.flush();
29: printWriter.close();
30: }
```

다음의 예제에서는 2라인에서 외부 입력값으로 request.getQueryString()를 입력받아 6라인에서 URL을 구성하여 출력하고 있어 취약하다.

정탐코드의 예

```
1: …
2: 〈%@ taglib prefix="c" url="http://java.sun.com/jsp/jstl/core"%〉
3: 〈%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%〉
4: …
5: 〈c:out value="${param.name}" escapeXml="false" /〉
```

다음의 예제에서는 JSTL의 〈c:out〉을 이용하여 외부 값을 출력할 때 escapeXml값을 false로 설정하였기 때문에 스크립트가 실행될 수 있어 취약한 것으로 판정한다.

정탐코드의 예

```
1:  〈%
2:  String param = request.getParameter("param");
3:  if ( param != null ) {
4:        param = param.replaceAll("〈script〉","");
5:        param = param.replaceAll("〈/script〉","");
6:  }
7:  %〉
8:  …
9:  〈p〉 제목 : 〈%=title%〉 〈/p〉
```

다음의 예제에서는 필터링을 사용하였으나 〈script〉 문장을 공백으로 치환하였기 때문에 〈sc〈script〉ript〉 등으로 공격할 경우 스크립트가 실행되어 취약한 것으로 판정한다.

정탐코드의 예

```
1:  〈%
2:   String query = request.getQueryString();
3:   String nextUrl = request.getRequestURL().toString() + query;
4:  %〉
5:  ….
6:  〈input type="hidden" name="nextUrl" value= "〈%=nextUrl%〉" /〉
```

다음의 예제에서는 외부 입력값이 html 주석으로 되어있어 수행되지 않으나, 공격시 주석을 닫는 부분과 함께 공격 스크립트를 삽입(--〉〈script〉…〈/script〉)하면 취약하다.

정탐코드의 예

```
1:  〈%
2:  String param = request.getParameter("param");
3:  %〉
4:  〈!-- 〈%=param%〉 --〉
```

다음의 예제에서는 외부 입력값에 스크립트 공격을 시도하더라도, 해당 스크립트 공격이 URL로 포함되어 이동하기 때문에, 공격이 수행되지 않아 안전한 것으로 판정한다.

오탐코드의 예

```
1:  1: 〈%=respose.sendRedirect(requst.getParameter("url"))%〉
```

다음의 예제에서는 외부 입력값에 스크립트 공격을 시도하더라도, 해당 스크립트 공격이 URL로 포함되어 이동하기 때문에, 공격이 수행되지 않아 안전한 것으로 판정한다.

오탐코드의 예

```
1:  〈script language="javascript"〉
2:  location.href('〈%=request.getParameter("url")%〉');
```

다음의 예제에서와 같이 JSTL의 〈c:out〉을 이용하여 외부 값을 출력할 때는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  〈%@ taglib prefix="c" url="http://java.sun.com/jsp/jstl/core"%〉
2:  〈%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%〉
3:  …
4:  〈c:out value="${param.name}"/〉
```

다음의 예제에서와 같이 JSTL의 escapeXml을 이용하여 외부 값을 출력할 때는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  〈%@ taglib prefix="c" url="http://java.sun.com/jsp/jstl/core"%〉
2:  〈%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%〉
3:  …
4:  ${fn:escapeXml(param.name)}
```

다음의 예제와 같이 출력값이 문자형이 아닌 경우에는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  Int pageNm = Integer.parseInt(request.getParameter("pageNm");
2:  〈input type=text size=10 name="pageNm" value='〈%=pageNm%〉' /〉
```

다음의 예제와 같이 subString을 이용하여 글자 수가 짧은 글자 수 이하로 제한된 경우에는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  〈%
2:  // 외부로 부터 입력을 받는다.
3:  String date = request.getParameter("date");
4:  String year = date.substring(0, 4);
5:  String month = date.substring(4, 6);
6:  String day = date.substring(6, 8);
7:  out.println(year + "년" + month + "월" + day + "일" );
8:  %〉
```

다음의 예제와 같이 출력값이 프로그램 내부에서 정해진 경우에는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  g1 = new WKTReader().read(geometryParam);
2:  p1 = (Polygon) g1.buffer(Double.parseDouble(bufferSizeParam));
3:  response.setContentType("text/plain;charset=EUC-KR");
4:  response.setHeader("Cache-Control", "no-cache");
5:  response.getWriter().write(p1.toText());
```

다음의 예제와 같이 웹페이지가 아닌 System 콘솔에 출력하는 경우에는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  //파라메터 설정
```

오탐코드의 예

```
2:  String exam_tgt_se = request.getParameter("exam_tgt_se");
3:  String mw_take_no = request.getParameter("mw_take_no");
4:  String apv_perm_reg_mgt_no = request.getParameter("apv_perm_reg_mgt_no");
5:  String mw_afr_no = request.getParameter("mw_afr_no");
6:
7:  System.out.println("exam_tgt_se : " + exam_tgt_se);
8:  System.out.println("mw_take_no : " + mw_take_no);
9:  System.out.println("apv_perm_reg_mgt_no : " + apv_perm_reg_mgt_no);
```

다음의 예제와 같이 사용자 입력값이 아닌 경우에는 안전한 것으로 판정한다.

오탐코드의 예

```
1:  var CompPath = "%XPLATFORM%\\" + sKey + "\\component";
2:  //XLauncher.componentpath = CompPath;
3:  XLauncher.onlyone = "false";
4:  var iconStr = "<%= request.getRequestURL() %>";
```

[크로스사이트 스크립트 : Stored]

다음의 예제와 같이 게시판 제목, 내용 또는 사용자 정보와 같이 입력 폼에 의해 데이터베이스에 저장 된 데이터가 웹페이지에 출력될 경우 취약하며, 메뉴 이름 등과 같이 사용자 입력으로 수정되지 않는 데이터베이스에 저장된 값이 출력될 경우 취약하지 않는다고 판정한다. 데이터베이스가 할당된 변수가 사용자 입력으로 수정되는 변수 여부 판단은 정적도구를 사용하여 판단하기 어렵다.

오탐코드의 예

```
1:  <%
2:  …
3:  rs = pstmt.executeQuery(query);
4:  while(rs.next()){
5:  board_contents = rs.getString("board_contents"))
6:  … }
7:  %>
8:  …
```

오탐코드의 예

```
9:  〈p〉 제목 : 〈%=board_contents%〉 〈/p〉
```

[크로스사이트 스크립트 : DOM]

다음의 예제에서는 request.getParameter()에서 전달된 외부의 입력(name)이 document.write()의 인자 값 생성에 그대로 사용되어 취약한 것으로 판정한다.

정탐코드의 예

```
1: …
2: 〈%
3: // 외부로 부터 입력을 받는다.
4: String name = request.getParameter("name");
5: %〉
6: 〈SCRIPT language="javascript"〉
7: // 외부의 입력을 그대로 출력한다.
8: document.write("name:" + 〈%=name%〉 );
```

다음의 예제에서는 request.getParameter()에서 전달된 외부의 입력(name)이 필터링되고 있으나 ;) alert(documement.cookie);〈%--를 파라미터에 입력할 경우 악성 스크립트가 실행된다. 크로스사이트 스크립트 : DOM을 방지하기 위해 :, %, - 등도 필터링 문자에 추가한다.

정탐코드의 예

```
1:  …
2:  〈%
3:  String name = request.getParameter("name");
4: name = name.replaceAll("〈","&lt;");
5: name = name.replaceAll("〉","&gt;");
6:    name = name.replaceAll("&","&amp;");
7:  name = name.replaceAll(""","&quot;");
8: name = name.replaceAll("'",
     "&#x27;"); 9: name = name.replaceAll("/",
     "&#x2F;"); 10: %〉
11: 〈SCRIPT language="javascript"〉
12: document.write("name:" + 〈%=name%〉 );
```

마. 참고자료

[1] CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), MITRE,
http://cwe.mitre.org/data/definitions/79.html

[2] Properly encode or escape output, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS51-J.+Properly+encode+or+escape+output

[3] XSS (Cross Site Scripting) Prevention Cheat Sheet, OWASP,
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

[4] DOM based XSS Prevention Cheat Sheet, OWASP,
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

[5] "Understanding Malicious Content Mitigation for Web Developers"
http://www.cert.org/tech_tips/malicious_code_mitigation.html

5. 운영체제 명령어 삽입

가. 개요

적절한 검증절차를 거치지 않은 사용자 입력값이 운영체제 명령어의 일부 또는 전부로 구성되어 실행되는 경우, 의도하지 않은 시스템 명령어가 실행되어 부적절하게 권한이 변경되거나 시스템 동작 및 운영에 악영향을 미칠 수 있다.

일반적으로 명령어 라인의 파라미터나 스트림 입력 등 외부 입력을 사용하여 시스템 명령어를 생성하는 프로그램이 많이 있다. 하지만 이러한 경우 외부 입력 문자열은 신뢰할 수 없기 때문에 적절한 처리를 해주지 않으면, 공격자가 원하는 명령어 실행이 가능하게 된다.

나. 보안대책

웹 인터페이스로 서버 내부로 시스템 명령어를 전달시키지 않도록 응용프로그램을 구성하고, 외부에서 전달되는 값을 검증 없이 시스템 내부 명령어로 사용하지 않는다. 외부 입력에 따라 명령어를 생성하거나 선택이 필요한 경우에는 명령어 생성에 필요한 값들을 미리 지정해 놓고 외부 입력에 따라 선택하여 사용한다.

다. 코드예제

다음의 예제는 Runtime.getRuntime().exec()명령어로 프로그램을 실행하며, 외부에서 전달되는 인자값은 명령어의 생성에 사용된다. 그러나 해당 프로그램에서 실행할 프로그램을 제한하지 않고 있기 때문에 외부의 공격자는 가능한 모든 프로그램을 실행시킬 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: public static void main(String args[]) throws IOException {
2: // 해당 프로그램에서 실행할 프로그램을 제한하고 있지 않아 파라미터로 전달되는 모든 프로그램이
   실행될 수 있다.
3:    String cmd = args[0];
4:    Process ps = null;
5:    try {
6:       ps = Runtime.getRuntime().exec(cmd);
7:       ...
```

다음의 예제와 같이 미리 정의된 파라미터의 배열을 만들어 놓고, 외부의 입력에 따라 적절한 파라미터를 선택하도록 하여, 외부의 부적절한 입력이 명령어로 사용될 가능성을 배제하여야 한다.

안전한 코드의 예 JAVA

```
1: public static void main(String args[]) throws IOException {
2: // 해당 어플리케이션에서 실행할 수 있는 프로그램을 노트패드와 계산기로 제한하고 있다.
3:     List<String> allowedCommands = new ArrayList<String>();
4:     allowedCommands.add("notepad");
5:     allowedCommands.add("calc");
6:     String cmd = args[0];
7:     if (!allowedCommands.contains(cmd)) {
8:         System.err.println("허용되지 않은 명령어입니다.");
9:         return;
10:    }
11:    Process ps = null;
12:    try {
13:        ps = Runtime.getRuntime().exec(cmd);
14:        .....
```

아래 코드는 외부입력값을 검증하지 않고 그대로 명령어로 실행하기 때문에 공격자의 입력에 따라 의도하지 않은 명령어가 실행될 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: //외부로 부터 입력 받은 값을 검증 없이 사용할 경우 안전하지 않다.
2: String date = request.getParameter("date");
3: String command = new String("cmd.exe /c backuplog.bat");
4: Runtime.getRuntime().exec(command + date);
```

운영체제 명령어 실행 시에는 아래와 같이 외부에서 들어오는 값에 의하여 멀티라인을 지원하는 특수문자(| ; & :)나 파일 리다이렉트 특수문자(〉 〉〉)등을 제거하여 원하지 않은 운영체제 명령어가 실행될 수 없도록 필터링을 수행한다.

안전한 코드의 예 JAVA

```
1: String date = request.getParameter("date");
2: String command = new String("cmd.exe /c backuplog.bat");
3: //외부로부터 입력 받은 값을 필터링으로 우회문자를 제거하여 사용한다.
4: date = date.replaceAll("|","");
5: date = date.replaceAll(";","");
6: date = date.replaceAll("&","");
7: date = date.replaceAll(":","");
8: date = date.replaceAll(">","");
9: Runtime.getRuntime().exec(command + date);
```

다음 C# 코드는 외부 입력값을 프로세스가 실행할 파일 이름에 직접 사용하고 있다. 이는 공격자의 입력에 따라 의도하지 않은 명령어가 실행될 수 있다.

안전하지 않은 코드의 예 C#

```
1: //외부 입력값이 프로세스가 실행할 파일 이름을 지정하고 있습니다.
2: string fileName = PgmTextBox.Text;
3: ProcessStartInfo proStartInfo = new ProcessStartInfo();
4: proStartInfo.FileName = fileName;
5: Process.Start(proStartInfo);
```

적절한 정규식이나, white list 등을 이용하여 검증을 한 후 사용하도록 한다.

안전한 코드의 예 C#

```
1: string fileName = PgmTextBox.Text;
2: //외부 입력값에 대해 정규식 등을 이용하여 검증을 해줘야 합니다.
```

안전한 코드의 예 C#

```
3:  if (Regex.IsMatch(fileName, "properRegexHere"))
4:  {
5:    ProcessStartInfo proStartInfo = new ProcessStartInfo();
6:    proStartInfo.FileName = fileName;
7:    Process.Start(proStartInfo);
8:  }
```

공격자의 입력에 따라 의도하지 않은 명령어가 실행될 수 있다.

안전하지 않은 코드의 예 C

```
1:  int main(int argc, char* argv[]) {
2:     char cmd[CMD_LENGTH];
3:     if (argc < 1 ) {
4:        // error
5:     }
6:     // 외부 입력값으로 커맨드를 직접 수행
7:     cmd_data = argv[1];
8:    snprintf(cmd, CMD_LENGTH, "cat %s", cmd_data);
9:    system(cmd);
10:   ……
11: }
```

운영체제 명령어 실행 시에는 아래와 같이 외부에서 들어오는 값에 의하여 멀티라인을 지원하는 특수문자(| ; & :)나 파일 리다이렉트 특수문자(〉 〉〉)등을 제거하여 원하지 않은 운영체제 명령어가 실행될 수 없도록 필터링을 수행한다.

안전한 코드의 예 C

```
1:  int main(int argc, char* argv[]) {
2:     char cmd[CMD_LENGTH];
3:     int len = 0;
```

안전한 코드의 예 C

```
4:    if (argc < 1 ) {
5:      // error
6:    }
7:    // 외부 입력값으로 커맨드를 직접 수행
8:    cmd_data = argv[1];
9:    len = strlen(cmd_data);
10:   for (int i = 0; I < len; i++) {
11:     if (cmd_data[i] == '|' || cmd_data[i] == '&' ||
12:     cmd_data[i] == ';' || cmd_data[i] == ':' || cmd_data[i] == '>') {
13:       // 멀티라인을 지원하는 특수문자나 파일 리다이렉트 특수문자가 존재하여
14:       // 안전하지 않음
15:       return -1;
16:   }
17:  }
18:  snprintf(cmd, CMD_LENGTH, "cat %s", cmd_data);
19:  system(cmd);
20:  ……
21: }
```

라. 진단방법

운영체제 명령어(exec(), system(), Runtime.getRuntime().exec 등)를 실행할 수 있는 함수가 호출되는지 확인하고(①) 외부에서 전달되는 값이 시스템 내부명령어의 일부 또는 전부로 사용되는지 확인한다(②). 정해진 후보군에서 선택된 값(White List)이거나 적절하게 검증하면 안전하고 그 외에는 취약하다.

일반적인 진단의 예

```
1:  public void f() throws
    IOException { 2:
          Properties props = new
    Properties(); 3:
          String fileName = "file_list";
```

일반적인 진단의 예

```
4:    FileInputStream in = new FileInputStream(fileName);
5:    props.load(in);
6:
7:    // 외부에서 전달된 값이 시스템 내부 명령어의 일부로 사용됨.
8:    String version = props.getProperty("dir_type") ····· ②
9:    String cmd = new String("cmd.exe /c rmanDB.bat ");
10:
11:   // 입력값 검증 없이 내부 명령어의 일부로 사용됨.
12:   Runtime.getRuntime().exec(cmd + "c:\\prog_cmd\\" + version); ····· ①
13:   }
```

다음의 예제에서 외부 입력값인 processMon의 값을 파라미터로 getProcessId를 호출하고, 19라인에서 execStr 문자열을 만들며 그 문자열이 28라인에서 명령어로 쓰이고 있어 취약한 것으로 판정한다.

정탐코드의 예

```
〈 EgovProcessMonController.java 〉
1: @RequestMapping("/utl/sys/prm/selectProcessSttus.do")
2: public String selectProcessSttus(@ModelAttribute("processMonVO") ProcessMonVO
   processMonVO, ModelMap model) throws Exception {
4: …
5: }
〈 ProcessMonChecker.java 〉
14:public staticString getProcessId(String processNm) throws Exception {
15:       Process p = null;
16:       String procsSttus =
null; 17:BufferedReader buf =
null; 18:String result = null;
19:       String execStr = "tasklist /fo table /nh /fi \"imagename eq "+processNm+"\"";
20:       int cnt = 0;
21:       String str = null;
22:       try {
23:        if (Globals.OS_TYPE == null) {
24:        throw new RuntimeException("Globals.OS_TYPE property value is needed");
```

정탐코드의 예

```
25: }
26: if ("WINDOWS".equals(Globals.OS_TYPE)) {
27:   cnt = -1;
28:   p = Runtime.getRuntime().exec(execStr);
29: }
30:}
```

프레임워크 특성상 Property 사용은 금지할 수 없다. 하지만 시스템 프로퍼티 등 공용으로 사용하는 프로퍼티가 아닌 개별 사용 프로퍼티일 경우는 취약한 것으로, 시스템 프로퍼티를 사용하는 경우는 안전한 것으로 판정한다.

다음의 예제에서 5라인 cmdStr 변수는 Property에서 값을 가져 온 이후 7라인에서 문자열을 만드는데 사용되고, 해당 문자열이 9라인에서 명령어로 사용되므로 안전한 것으로 판정한다.

오탐코드의 예

```
1: public static floatgetMoryFreeCpcty() throws Exception {
2: float cpcty = 0;
3: Process p = null;
4: try {
5: String cmdStr = EgovProperties.getPathProperty(Globals.SERVER_CONF_PATH,
6: "SHELL."+Globals.OS_TYPE+".getMoryInfo");
7: String[] command = {cmdStr.replace('\\', FILE_SEPARATOR).replace('/', FILE_SEPA-
   RATOR),"FREE"}
8: }
9: p = Runtime.getRuntime().exec(command);
```

사용자의 입력값이 운영체제 명령어를 실행하는데 들어가지 않을 때에는 안전한 것으로 판정한다.

다음의 예제에서 fname은 srcFile.getName()의 결과값인 파일의 이름이므로 명령어에 삽입되어 공격할 수 있는 file separator나 ".." 등의 문자열이 없으므로 취약하지 않은 것으로 판정한다.

오탐코드의 예

```
1: public static String getOwner(String file) throws Exception{
2:   String owner = "";
3:   String src = file.replace('\\', FILE_SEPARATOR).replace('/', FILE_SEPARATOR);
4:   BufferedReader b_err=null;
5:   BufferedReader b_out=null;
6:   try {
7:    File srcFile = newFile(src);
8:    if (srcFile.exists()) {
9:     String parentPath = srcFile.getParent();
10:    String fname = srcFile.getName();
11:    Process p = null;
12:    String cmdStr = EgovProperties.getProperty(Globals.SHELL_FILE_PATH,
       "SHELL."+Globals.OS_TYPE+".getDrctryOwner");
13:    String[] command = {cmdStr.replace('\\', FILE_SEPARATOR).replace('/',
    FILE_SEP- ARATOR), parentPath.replace('\\', FILE_SEPARATOR).replace('/',
    FILE_SEPARATOR), fname};
14:    p = Runtime.getRuntime().exec(command);
15:  }
16: }
17:}
```

마. 참고자료

[1] CWE-78 OS Command Injection, MITRE,
http://cwe.mitre.org/data/definitions/78.html

[2] Sanitize untrusted data passed to the Runtime.exec() method, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS07-J.+Sanitize+untrusted+data+passed+to+the+Runtime.exec()+method?focusedCommentId=6465 1588#comment-64651588

[3] Do not call system(), CERT,
http://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=2130132

[4] Reviewing Code for OS Injection, OWASP
https://www.owasp.org/index.php/Reviewing_Code_for_OS_Injection

6. 위험한 형식 파일 업로드

가. 개요

서버 측에서 실행될 수 있는 스크립트 파일(asp, jsp, php 파일 등)이 업로드 가능하고, 이 파일을 공격자가 웹으로 직접 실행시킬 수 있는 경우, 시스템 내부명령어를 실행하거나 외부와 연결하여 시스템을 제어할 수 있는 보안약점이다.

나. 보안대책

화이트 리스트 방식으로 허용된 확장자만 업로드를 허용한다. 업로드 되는 파일을 저장할 때에는 파일명과 확장자를 외부사용자가 추측할 수 없는 문자열로 변경하여 저장하며, 저장 경로는 'web document root' 밖에 위치시켜서 공격자의 웹으로 직접 접근을 차단한다. 또한 파일 실행여부를 설정할 수 있는 경우, 실행 속성을 제거한다.

다. 코드예제

업로드할 파일에 대한 유효성을 검사하지 않으면, 위험한 유형의 파일을 공격자가 업로드하거나 전송할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1:    MultipartRequest multi
   = new MultipartRequest(request,savePath,sizeLimit,"euc-kr",new DefaultFileRenamePolicy());
2:    .....
3: //업로드 되는 파일명을 검증 없이 사용하고 있어 안전하지 않다.
4:    String fileName = multi.getFilesystemName("filename");
5:    .....
6:    sql = " INSERT INTO board(email,r_num,w_date,pwd,content,re_step,re_num,filename)
      "+ " values ( ?, 0, sysdate(), ?, ?, ?, ?, ? ) ";
7:    PreparedStatement pstmt = con.prepareStatement(sql);
8:    pstmt.setString(1, stemail);
9:    pstmt.setString(2, stpwd);
10:   pstmt.setString(3, stcontent);
```

안전하지 않은 코드의 예 JAVA

```
11:  pstmt.setString(4, stre_step);
12:  pstmt.setString(5, stre_num);
13:  pstmt.setString(6, fileName);
14:  pstmt.executeUpdate();
15:  Thumbnail.create(savePath+"/"+fileName, savePath+"/"+"s_"+fileName, 150);
```

아래 코드는 업로드 파일의 확장자를 검사하여 허용되지 않은 확장자인 경우 업로드를 제한하고 있다.

안전한 코드의 예 JAVA

```
1:  MultipartRequest multi
    = new MultipartRequest(request,savePath,sizeLimit,"euc-kr",new DefaultFileRenamePolicy());
2:  .....
3:  String fileName = multi.getFilesystemName("filename");
4:  if (fileName != null) {
5:    //1.업로드 파일의 마지막 "." 문자열의 기준으로 실제 확장자 여부를 확인하고, 대소문자 구별을
    해야한다.
6:     String fileExt = FileName.substring(fileName.lastIndexOf(".")+1).toLowerCase();
7:     //2.되도록 화이트 리스트 방식으로 허용되는 확장자로 업로드를 제한해야 안전하다.
8:     if (!"gif".equals(fileExt) && !"jpg".equals(fileExt) && !"png".equals(fileExt)) {
9:          alertMessage("업로드 불가능한 파일입니다.");
10:         return;
11:    }
12: }
13: .....
14: sql = " INSERT INTO board(email,r_num,w_date,pwd,content,re_step,re_num,filename)
    " + " values ( ?, 0, sysdate(), ?, ?, ?, ?, ? ) ";
15:  PreparedStatement pstmt = con.prepareStatement(sql);
16:  .....
17:  Thumbnail.create(savePath+"/"+fileName, savePath+"/"+"s_"+fileName, 150);
```

업로드할 파일에 대한 유효성을 검사하지 않으면, 위험한 유형의 파일을 공격자가 업로드하거나 전송할 수 있다.

안전하지 않은 코드의 예 C#

```
1: string fn = Path.GetFileName(FileUploadCtr.FileName);
2: //업로드 하는 파일명을 검증 없이 사용하고 있습니다.
3: FileUploadCtr.SaveAs(fn);
4: StatusLabel.Text = "Upload status: File uploaed!";
```

파일 타입과 크기 등을 검사하여 제한하도록 한다.

안전한 코드의 예 C#

```
1: //파일 타입과 크기를 제한합니다.
2: if (FileUploadCtr.PostedFile.ContentType == "image/jpeg")
3: {
4:     if (FileUploadCtr.PostedFile.ContentLength < 102400)
5:     {
6:       string fn = Path.GetFileName(FileUploadCtr.FileName);
7:       FileUploadCtr.SaveAs(Server.MapPath("~/") + fn);
8:       StatusLabel.Text = "Upload status: File uploaed!";
9:     }
10:      else
11:         StatusLabel.Text = "Upload Status: The File has to be less than 100 kb!";
12: }
13: else
14:   StatusLabel.Text = "Upload Status: Only JPEG files are accepted!";
```

라. 진단방법

외부 입력값에서 파일명을 얻어오는 부분이 존재하는지 확인하고(①), 허용된 확장자에 대해서만 파일 업로드를 허용하는지 확인한다(②). 제어문 등을 사용하여 허용된 파일만 업로드 될 경우와 업로드된 파일명을 외부에서 알 수 없는 형태로 변경할 경우엔 안전하지만 그 외에는 취약하다.

일반적인 진단의 예

```
1:  public void upload(HttpServletRequest request) throws ServletException {
2:  // MultipartHttpServletRequest를 케스팅
3:  MultipartHttpServletRequest mRequest = (MultipartHttpServletRequest) request;
4:
5:  String next = (String) mRequest.getFileNames().next();
6:  MultipartFile file = mRequest.getFile(next);
7:
8:  // MultipartFile로 부터 file 이름을 얻어옴
9:  String fileName = file.getOriginalFilename();  ····· ①
10:
11: // upload 파일에 대한 확장자 유효성 체크를 하지 않음
12: File uploadDir = new File("/app/webapp/data/upload/notice");
13: String uploadFilePath = uploadDir.getAbsolutePath()+"/"+fileName;  ····· ②
14:
15: /* 이하 file upload 루틴 */
16: }
17: }
```

getOriginalFilename() 함수를 사용하여 파일을 업로드하면서 해당 파일의 확장자를 체크하지 않는 경우엔 취약하다고 판정한다.

다음의 예제는 업로드된 파일의 확장자를 체크하고 있지 않으므로 취약한 것으로 판정한다.

정탐코드의 예

```
1:  while (itr.hasNext()) {
2:    Entry<String, List<MultipartFile>> entry = itr.next();
3:    fileList = entry.getValue();
4:
```

정탐코드의 예

```
5:    for(int i=0, s=fileList.size(); i<s; i++) {
6:     file = (MultipartFile)fileList.get(i);
7:     String orginFileName = file.getOriginalFilename();
8:     //-----------------------------------
9:     // 원 파일명이 없는 경우 처리
10:    // (첨부가 되지 않은 input file type)
11:    //-----------------------------------
12:    if ("".equals(orginFileName)) continue;
13:    ////---------------------------------
14:
15:    int index = orginFileName.lastIndexOf(".");
16:    //String fileName = orginFileName.substring(0, index);
17:    String fileExt = orginFileName.substring(index + 1);
18:    String newName = KeyStr + EgovStringUtil.getTimeStamp() + fileKey;
19:    long _size = file.getSize();
20:
21:    if (!"".equals(orginFileName)) {
22:         filePath = storePathString + File.separator + newName;
23:         file.transferTo(new File(filePath));
24:       }
25:       fvo = new FileVO();
26:       fvo.setFileExtsn(fileExt);
27:       fvo.setFileStreCours(storePathString);
28:       fvo.setFileMg(Long.toString(_size));
29:       fvo.setOrignlFileNm(orginFileName);
30:       fvo.setStreFileNm(newName);
31:       fvo.setAtchFileId(atchFileIdString);
32:       fvo.setFileSn(String.valueOf(fileKey));
33:
34:       //writeFile(file, newName, storePathString);
35:       result.add(fvo);
36:
37:       fileKey++;
38:    }
39:   }
40:   return result;
```

다음의 예제는 파일의 확장자를 체크하여 필터링하고 있으나, 확장자의 대소문자구분을 하지 않아 JSP, Jsp, ASP, PHP, Php 등의 확장자가 업로드 가능하다.

정탐코드의 예

```
1: …
2: String fileName = file.getOriginalFilename();
3:   if ( fileName != null ) {
4:     if ( !fileName.endsWith(“.jsp”) ) {
5:       /* file 업로드 루틴 */
6:     }
7:   }
8: …
```

다음의 예제는 파일의 확장자를 체크하여 필터링하고 있으므로 안전하다.

오탐코드의 예

```
1:      String orginFileName = file.getOriginalFilename();
2:      //-------------------------------------
3:      // 원 파일명이 없는 경우 처리
4:      // (첨부가 되지 않은 input file type)
5:      //-------------------------------------
6:      if (“”.equals(orginFileName)) { continue }
7:      ////-----------------------------------
8:      int index = orginFileName.lastIndexOf(“.”);
9:      //String fileName = orginFileName.substring(0, index);
10:       String fileExt = orginFileName.substring(index + 1);11:      /* 확장자 체크 */
12:       for(Object fileExclusionExt : this.fileExclusionExtension) {
13:          if( ((String) fileExclusionExt).equals(fileExt.toLowerCase())){
14:            throw new Exception(“egume.message.error.file.exclusion.extension”);
15:          }
16:   }
```

다음의 예제와 같이 getOriginalFilename()의 리턴 값을 저장하여 사용하지 않으면, 파일명을 사용하기 위해 다른 곳에서 해당 함수를 호출할 것이므로 취약하지 않다고 판정한다.

오탐코드의 예

```
1:    if (!"".equals(file.getOriginalFilename())) {
2:      zipManageService.insertExcelZip(file.getInputStream());
```

마. 참고자료

[1] CWE-434 Unrestricted Upload of File with Dangerous Type, MITRE, http://cwe.mitre.org/data/definitions/434.html

[2] Prevent arbitrary file upload, CERT, http://www.securecoding.cert.org/confluence/display/java/IDS56-J.+Prevent+arbitrary+file+upload

[3] Secure File Upload Check List With PHP, Clay, http://hungred.com/useful-information /secure-file-upload-check-list-php/

[4] Unrestricted File Upload, OWASP https://www.owasp.org/index.php/Unrestricted_File_Upload

7. 신뢰되지 않는 URL 주소로 자동접속 연결

가. 개요

사용자로부터 입력되는 값을 외부사이트의 주소로 사용하여 자동으로 연결하는 서버 프로그램은 피싱(Phishing) 공격에 노출되는 취약점을 가질 수 있다.

일반적으로 클라이언트에서 전송된 URL 주소로 연결하기 때문에 안전하다고 생각할 수 있으나, 공격자는 해당 폼의 요청을 변조함으로써 사용자가 위험한 URL로 접속할 수 있도록 공격할 수 있다.

나. 보안대책

자동 연결할 외부 사이트의 URL과 도메인은 화이트 리스트로 관리하고, 사용자 입력값을 자동 연결할 사이트 주소로 사용하는 경우에는 입력된 값이 화이트 리스트에 존재하는지 확인해야 한다.

다. 코드예제

다음과 같은 코드가 서버에 존재할 경우 공격자는 아래와 같은 링크에 희생자가 접근하도록 함으로써 희생자가 피싱 사이트 등으로 접근하도록 할 수 있다.

Click

안전하지 않은 코드의 예 JAVA

```
1: String id = (String)session.getValue("id");
2: String bn = request.getParameter("gubun");
3: //외부로부터 입력받은 URL이 검증 없이 다른 사이트로 이동이 가능하여 안전하지 않다.
4: String rd = request.getParameter("redirect");
5: if (id.length() > 0) {
6:     String sql = "select level from customer where customer_id = ? ";
7:     conn = db.getConnection();
8:     pstmt = conn.prepareStatement(sql);
9:     pstmt.setString(1, id);
10:    rs = pstmt.executeQuery();
```

안전하지 않은 코드의 예 JAVA

```
11:    rs.next();
12:    if ("0".equals(rs.getString(1)) && "01AD".equals(bn)) {
13:        response.sendRedirect(rd);
14:        return;
15:    }
```

다음의 예제와 같이, 외부로 연결할 URL과 도메인들은 화이트 리스트를 작성한 후, 그 중에서 선택하도록 함으로써 안전하지 않은 사이트로의 접근을 차단할 수 있다.

안전한 코드의 예 JAVA

```
1: //이동 할 수 있는 URL범위를 제한하여 피싱 사이트등으로 이동 못하도록 한다.
2: String allowedUrl[] = { "/main.do", "/login.jsp", "list.do" };
3: ......
4: String rd = request.getParameter("redirect");
5:  try {
6:     rd = allowedUrl[Integer.parseInt(rd)];
7:  } catch(NumberFormatException e) {
8:     return "잘못된 접근입니다.";
9:  } catch(ArrayIndexOutOfBoundsException e) {
10:    return "잘못된 입력입니다.";
11:}
12:if (id.length() > 0) {
13:......
14:    if ("0".equals(rs.getString(1)) && "01AD".equals(bn)) {
15:        response.sendRedirect(rd);
16:        return;
17:  }
```

외부 입력 값으로 받은 URL로 검증 없이 연결되는 경우, 공격자의 입력에 따라 피싱사이트 등으로 연결될 수 있다.

안전하지 않은 코드의 예 C#

```
1: // 외부 입력값으로 받은 URL 을 검증 없이 연결하고 있습니다.
2: string url = Request["dest"];
3: Response.Redirect(url);
```

로컬 URL 검증이나 white list 등을 이용하여 검증이 필요하다.

안전한 코드의 예 C#

```
1: public void AttackOpenRedirect()
2: {
3:    string url = Request["dest"];
4:    // 외부 입력값이 로컬 URL인지 확인합니다. MVC 3 이상의 프레임워크를 사용할 경우,
   System.Web.Mvc 에 정의되어있는 Url.isLocalUrl 을 바로 사용할 수 있습니다.
5:    if(isLocalUri(url)) Response.Redirect(url);
6: }
7: private bool IsLocalUrl(string url)
8: {
9:  if(string.IsNullOrEmpty(url))
10: {
11:  return false;
12: }
13: Uri absoluteUri;
14: if(Uri.TryCreate(url, UriKind.Absolute, out absoluteUri))
15: {
16:       return String.Equals(this.Request.Url.Host, absoluteUri.Host,
   StringComparison.OrdinalIgnoreCase);
17: }
18: else
19: {
20:  bool isLocal = !url.StartsWith("http:",StringComparison.OrdinalIgnoreCase)
      && !url.StartsWith("https:", StringComparison.OrdinalIgnoreCase)
      && Uri.IsWellFormedUriString(url, UriKind.Relative);
21:  return isLocal;
22: }
}
```

라. 진단방법

외부 사이트로 리다이렉션하는 함수나 메소드(java의 경우 response.sendRedirect 메소드)가 존재하는지 확인하고(①), 리다이렉션 하는 함수나 메소드의 인자값(url)이 외부 입력값인지 확인한다(②). 일반적으로 외부에서 입력받는 변수가 아닌 경우 안전하다고 판정한다. 외부에서 입력받는 변수인 경우, 리다이렉션 하는 함수나 메소드의 인자값(url)을 관리하는지 확인해서 유효값 검사 및 화이트 리스트로 관리하는 경우 안전한 것으로 판정한다.

일반적인 진단의 예

```
1:  public class U601 extends HttpServlet {
2:    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
3:      String query = request.getQueryString(); ····· ②
4:      if(query.contains("url")) {
5:        String url = request.getParameter("url");
6:        // url에 대한 유효성 점검없이 sendRedirect의 인자로 사용
7:        if(url != null) {
8:        url = url.replaceAll("\r","").replaceAll("\n","");
9:        response.sendRedirect(url); ····· ①
10:     }
11:   }
12:  }
13: }
```

외부 입력값을 이용하여 URL을 이동하는 response.sendRedirect(String url) 함수가 사용된다면 취약하다.

다음의 예제에서 5, 6 라인에서 code, action을 request에서 받아온다. 이후 8번째 라인에서 페이지 이동 URL에 code, action 값을 사용하여 URL을 만든다.

정탐코드의 예

```
1: <%
2: //redirect
3: String code = nvl(request.getAttribute("redirectCode"));
```

정탐코드의 예

```
4: String action = nvl(request.getAttribute("action"));
5: logger.debug("〉〉〉 redirectCode - "
   + code); 6:       logger.debug("〉〉〉 redirectAction -
   " + action); 7: ·····
8:          se.sendRedirect(CP + action + "?redirectCode=" + code);
9:            return;
10:         %〉
```

다음의 예제에서 HttpRequest.getContextPath() 함수는 내장함수로 context path를 리턴하므로 안전한 URL이다.

오탐코드의 예

```
1:  1: response.sendRedirect(request.getContextPath() + "/login.do");
```

마. 참고자료

[1] CWE-601 URL Redirection to Untrusted Site, MITRE, http://cwe.mitre.org/data/definitions/601.html

[2] Unvalidated Redirects and Forwards Cheat Sheet, OWASP https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

8. 부적절한 XML 외부개체 참조

가. 개요

XML 문서에는 DTD(Document Type Definition)를 포함할 수 있으며, DTD는 XML 엔티티(entitiy)*를 정의한다. 부적절한 XML 외부개체 참조 보안약점은 서버에서 XML 외부 엔티티를 처리할 수 있도록 설정된 경우에 발생할 수 있다.

취약한 XML parser가 외부 값을 참조하는 XML 값을 처리할 때, 공격자가 삽입한 공격 구문이 동작되어 서버 파일 접근, 불필요한 자원 사용, 인증 우회, 정보 노출 등이 발생 할 수 있다.

* 반복적인 문장이나 문자열을 저장해놓고 쉽게 참조할 수 있도록 함

나. 보안대책

로컬 정적 DTD를 사용하도록 설정하고, 외부에서 전송된 XML문서에 포함된 DTD를 완전하게 비활성화해야 한다. 비활성화를 할 수 없는 경우에는 외부 엔티티 및 외부 문서 유형 선언을 각 파서에 맞는 고유한 방식으로 비활성화 한다.

다. 코드예제

다음의 예제는 XML 소스를 읽어서 분석하는 소스코드이다. 공격자가 아래와 같이 XML 외부 엔티티를 참조하는 recivedXML 데이터를 전송하고, 이를 파싱할 때 /etc/passwd 파일을 참조할 수 있다.

receivedXML

```
〈?xml version="1.0" encoding="ISO-8859-1"?〉
〈!DOCTYPE foo [
〈!ELEMENT foo ANY 〉
〈!ENTITY xxe SYSTEM "file:///etc/passwd" 〉]〉〈foo〉&xxe;〈/foo〉
```

안전하지 않은 코드의 예 JAVA

```
1: public void unmarshal(File receivedXml)
2:throws JAXBException, ParserConfigurationException, SAXException, IOException {
3:        JAXBContext jaxbContext = JAXBContext.newInstance( Student.class );
4:        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
5:        // 입력받은 receivedXml 을 이용하여 Document를 생성한다.
6:        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
7:        dbf.setNamespaceAware(true);
8:        DocumentBuilder db = dbf.newDocumentBuilder();
9:        Document document = db.parse(receivedXml);
10:       // 외부 엔티티로 만들어진 document를 이용하여 마샬링을 수행하여 안전하지 않다.
11:       Student employee = (Student) jaxbUnmarshaller.unmarshal( document );
12: }
```

다음 예제는 class XXE에서 외부개체 참조 제한 설정 없이 secure.xml을 참조하고 있다. 이 때, secure.xml이 아래와 같을 때 /dev/tty 콘솔이 실행되어 입력 요청을 대기하는 서비스 거부 공격이 발생할 수 있다.

secure.xml

```
〈?xml version="1.0"?〉
〈!DOCTYPE foo SYSTEM "file:/dev/tty"〉
〈foo〉bar〈/foo〉
```

안전하지 않은 코드의 예 JAVA

```
1:import javax.xml.parsers.SAXParsers;
2:import javax.xml.parsers.SAXParserFactory;
3:
4:class XXE {
5:        public static void main(String[] args)
6:        throws FileNotFoundException, ParserConfigurationException, SAXException, IOException{
```

안전하지 않은 코드의 예 JAVA

```
7:SAXParserFactory factory = SAXParserFactory.newInstance();
8:              SAXParser saxParser = factory.newSAXParser();
9:              // 외부개체 참조 제한 설정 없이 secure.xml 파일을 읽어서 파싱하여 안전하지 않다.
10:             saxParser.parse(new FileInputStream("secure.xml"), new DefaultHandler());
11:   }
12: }
```

다음의 JAXP DocumentBuilderFactory를 사용하는 경우 아래와 같이 제한 설정 추가할 수 있다.

안전한 코드의 예 JAVA

```
1:DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
2:// XML 파서가 doctype을 정의하지 못하도록 설정한다.
3:dbf.setFeature("http://apache.org/xml/featuresdisallow-doctype-decl", true);
4:// 외부 일반 엔티티를 포함하지 않도록 설정한다.
5:dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
6:// 외부 파라미터도 포함하지 않도록 설정한다.
7:dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
8:// 외부 DTD 비활성화한다.
9:dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
10:// XIncude를 사용하지 않는다.
11:dbf.setXIncludeAware(false);
12:// 생성된 파서가 엔티티 참조 노드를 확장하지 않도록 한다.
13:dbf.setExpandEntityReferences(false);
14:DocumentBuilder db = dbf.newDocumentBuilder();
15:Document document = db.parse(receivedXml);
16:Model model = (Model) u.unmarshal(document);
```

PHP에서는 libxml_disable_entity_loader 함수를 이용하여 외부 엔티티 사용을 비활성화 할 수 있다.

안전한 코드의 예 JAVA

```
1:value = libxml_disable_entity_loader(true);
2:$dom = new DOMDocument();
3:$dom -> loadXML($xml);
4:libxml_disable_entity_loader($value);
```

라. 진단방법

XML 파일을 파싱하고 있는지 확인한 후(①) 신뢰할 수 없는 외부 입력 값의 파일에 대한 외부 엔티티를 비활성화하는 코드가 없을 경우 취약하다고 판정한다.

일반적인 진단의 예

```
1:public class G06 extends HttpServlet {
2:      private ServletFileUpload uploader = null;
3:      ...
4:      protected void doPost(HttpServletRequest request, HttpServletResponse response)
5:      throws ServletException, IOException {
6:      ...
7:      try {
8:      List<FileItem> fileItemsList = uploader.parseRequest(request);
9:      Iterator<FileItem> fileItemsIterator = fileItemsList.iterator();
10:     while (fileItemsIterator.hasNext()) {
11:     FileItem fileItem = fileItemsIterator.next();
12:     File xmlFile = new File(
13:     request.getServletContext().getAttribute("FILES_DIR") + File.separator+ fileItem.getName());
```

일반적인 진단의 예

```
14:     fileItem.write(xmlFile);
15:     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
16:     DocumentBuilder db = dbf.newDocumentBuilder();
17:     Document document = db.parse(xmlFile); ····· ①
18:     changeConfiguration(document);
19:             ...
20:}
```

다음의 SAXParserFactory 사용하는 예제에서는 사용자 입력 값인 XML file을 사용하고 있으나, 외부 엔티티 사용을 비활성화하지 않으므로 취약하다고 판정한다.

정탐코드의 예

```
1:@RequestMapping(value = "/xmlupload", method = RequestMethod.POST)
2:public Student xmlUpload(@RequestParam("file") MultipartFile multipartFile)
3:File xmlFile = new File(multipartFile.getOriginalFilename());
4:multipartFile.transferTo(xmlFile);
5:SAXParserFactory factory = SAXParserFactory.newInstance();
6:SAXParser saxParser = factory.newSAXParser();
7:SAXParseraxParser.parse(new FileInputStream(xmlFile), new DefaultHandler());
```

다음의 DocumentBuilderFactory를 사용하는 예제에서는 3~4라인과 같은 제한 설정을 추가할 수 있다.

오탐코드의 예

```
1 String xml = "xxe.xml";
2 DocumentBuilderFactory df = DocumentBuilderFactory.newInstance();
3 df.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
4 df.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
5 DocumentBuilder builder = df.newDocumentBuilder();
6 Document document = builder.parse(new InputSource(xml));
7 DOMSource domSource = new DOMSource(document);
```

다음의 DocumentBuilderFactory를 사용하는 예제에서는 3~4라인과 같은 제한 설정을 추가할 수 있다.

오탐코드의 예

```
1:String xml = "xxe.xml";
2:DocumentBuilderFactory df = DocumentBuilderFactory.newInstance();
3:df.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
4:df.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
5:DocumentBuilder builder = df.newDocumentBuilder();
6:Document document = builder.parse(new InputSource(xml));
7:DOMSource domSource = new DOMSource(document);
```

SAXParserFactory 예제의 경우 5~6라인과 같이 제한설정을 추가할 수 있다.

오탐코드의 예

```
1:String xml = "xxe.xml";
2:SaxHandler handler = new SaxHandler();
3:SAXParserFactory factory = SAXParserFactory.newInstance();
4:SAXParser parser = factory.newSAXParser();
5:parser.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
6:parser.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
7:parser.parse(xml, handler);
```

SchemaFactory의 경우 SchemaFactory 또는 Validator 클래스의 setProperty()를 사용하여 제한설정을 추가할 수 있다.

오탐코드의 예

```
1:SchemaFactory factory =
SchemaFactory.newInstance("XMLConstants.W3C_XML_SCHEMA_NS_URI");
2:factory.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
3:factory.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
4:Schema schema = factory.newSchema(Source);
```

오탐코드의 예

```
1:SchemaFactory factory =
SchemaFactory.newInstance("XMLConstants.W3C_XML_SCHEMA_NS_URI");
2:Schema schema = factory.newSchema();
3:Validator validator = schema.newValidator();
4:validator.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
5:validator.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
```

XMLInputFactory 클래스의 setProperty()를 사용하여 제한설정을 추가할 수 있다.

오탐코드의 예

```
1:XMLInputFactory factory = XMLInputFactory.newFactory();
2:factory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
3:factory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, false);
4:XMLEventReadereventReader= factory.createXMLEventReader(new FileReader("xxe.xml"));
```

마. 참고자료

① CWE-611: Improper Restriction of XML External Entity Reference, MITRE, https://cwe.mitre.org/data/definitions/611.html

② XML Entity Prevention Cheet Sheet, OWASP https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

9. XML 삽입

가. 개요

검증되지 않은 외부 입력 값이 XQuery 또는 XPath 쿼리문을 생성하는 문자열로 사용되어 공격자가 쿼리문의 구조로 임의로 변경하고 임의의 쿼리를 실행하여 허가되지 않은 데이터를 열람하거나 인증 절차를 우회할 수 있는 보안약점이다.

나. 보안대책

XQuery 또는 Xpath 쿼리에 사용되는 외부 입력데이터에 대하여 특수문자 및 쿼리 예약어를 필터링하고 파라미터화된 쿼리문을 지원하는 XQuery를 사용한다.

다. 코드예제

· XQuery 삽입

다음의 예제에서는 executeQuery로 생성하는 쿼리의 파라미터의 일부로 외부입력값 (name)을 사용하고 있다. 만일 something' or '1'='1 을 name의 값으로 전달하면 다음과 같은 쿼리문을 수행할 수 있으며, 이로써 파일 내의 모든 값을 출력할 수 있게 된다.

doc('users.xml')/userlist/user[uname='something' or '1'='1']

안전하지 않은 코드의 예 JAVA

```
1: // 외부 입력 값을 검증하지 않고 XQuery 표현식에 사용한다.
2: String name = props.getProperty("name");
3: .....
4: // 외부 입력 값에 의해 쿼리 구조가 변경 되어 안전하지 않다.
5: String es = "doc('users.xml')/userlist/user[uname='"+name+"']";
6: XQPreparedExpression expr = conn.prepareExpression(es);
7: XQResultSequence result = expr.executeQuery();
```

다음의 예제에서는 외부 입력값을 받고 해당 값 기반의 XQuery상의 쿼리 구조를 변경시키지 않는 bindString 함수를 이용함으로써 외부 입력값으로 쿼리 구조가 변경될 수 없도록 한다.

안전한 코드의 예 JAVA

```
1: // blingString 함수로 쿼리 구조가 변경되는 것을 방지한다.
2: String name = props.getProperty("name");
3: .....
4: String es = "doc('users.xml')/userlist/user[uname='$xname']";
5: XQPreparedExpression expr = conn.prepareExpression(es);
6: expr.bindString(new QName("xname"), name, null);
7: XQResultSequence result = expr.executeQuery();
```

다음의 C# 코드도 외부입력 값을 이용하여 XQuery 문을 만들고 있다.

안전하지 않은 코드의 예 C#

```
//외부 입력 값으로 XQuery 문을 만듭니다.
1:String squery =
2:      "for $user in doc(users.xml)//user[username='"
3:    + UserTextBox.Text
4:    + "'and pass='"
5:    + PwdTextBox.Text
6:    + "'] return $user";
7:Processor processor = new Processor();
8:XQueryCompiler compiler = processor.NewXQueryCompiler();
9:XdmNode indoc = processor.NewDocumentBuilder().Build(new
Uri(Server.MapPath("users.xml")));
10:  using (StreamReader query = new StreamReader(squery))
11:  {
12:  XQueryCompiler compiler = processor.NewXQueryCompiler();
13:  XQueryExecutable exp = compiler.Compile(query.ReadToEnd());
14:  XQueryEvaluator eval = exp.Load();
15:  eval.ContextItem = indoc;
16:  Serializer qout = new Serializer();
```

안전하지 않은 코드의 예 C#

```
17:   qout.SetOutputProperty(Serializer.METHOD, "xml");
18:   qout.SetOutputProperty(Serializer.DOCTYPE_PUBLIC, "-//W3C//DTD XHTML
1.0 Strict//EN");
19:   qout.SetOutputProperty(Serializer.DOCTYPE_SYSTEM,
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd");
20:   qout.SetOutputProperty(Serializer.INDENT, "yes");
21:   qout.SetOutputProperty(Serializer.OMIT_XML_DECLARATION, "no");
22:   qout.SetOutputWriter(Response.Output);
//별다른 검증 없이 XML 데이터에 접근합니다.
23:   eval.Run(qout):27:
   }
```

문자열 필터링으로 쿼리문 조작을 막을 수 있다.

안전한 코드의 예 C#

```
1:String squery =
2:     "for $user in doc(users.xml)//user[username='"
3:    + UserTextBox.Text
4:    + "'and pass='"
5:    + PwdTextBox.Text
6:    + "'] return $user";
   // 문자열 필터링으로 위험한 문자열을 제거해 줍니다.
7:    string validatedQuery = squery.Replace('/','*');
8: Processor processor = new Processor();
9: XQueryCompiler compiler = processor.NewXQueryCompiler();
10:          XdmNode indoc = processor.NewDocumentBuilder().Build(new
   Uri(Server.MapPath("users.xml")));
11:using (StreamReader query = new StreamReader(validatedQuery))
12:{ // tainted value propagated
13:    XQueryCompiler compiler = processor.NewXQueryCompiler();
14:    XQueryExecutable exp = compiler.Compile(query.ReadToEnd()); //
xquery created
```

안전한 코드의 예 C#

```
15:    XQueryEvaluator eval = exp.Load();
16:    eval.ContextItem = indoc;
17:    Serializer qout = new Serializer();
18:    qout.SetOutputProperty(Serializer.METHOD, "xml");
19:    qout.SetOutputProperty(Serializer.DOCTYPE_PUBLIC, "-//W3C//DTD XHTML
    1.0 Strict//EN");
20:    qout.SetOutputProperty(Serializer.DOCTYPE_SYSTEM,
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd");
21:    qout.SetOutputProperty(Serializer.INDENT, "yes");
22:    qout.SetOutputProperty(Serializer.OMIT_XML_DECLARATION, "no");

23:    qout.SetOutputWriter(Response.Output);
24:    eval.Run(qout);
25:}
```

· XPath 삽입

아래 예제에서는 nm과 pw에 대한 입력값 검증을 수행하지 않으므로 nm의 값으로 “tester”, pw의 값으로 “x’ or ‘x’=‘x”을 전달하면 아래와 같은 질의문이 생성되어 인증과정을 거치지 않고 로그인할 수 있다.

“//users/user[login/text()=‘tester’ and password/text()=‘x’ or //‘x’=‘x’]/home_dir/text()”

안전하지 않은 코드의 예 JAVA

```
// 프로퍼티로부터 외부 입력값 name과 password를 읽어와 각각 nm, pw변수에 저장
1:String nm = props.getProperty("name");
2:String pw = props.getProperty("password");
3:.....
4:XPathFactory factory = XPathFactory.newInstance();
5:XPath xpath = factory.newXPath();
6:.....
// 검증되지 않은 입력값 외부 입력값 nm, pw를 사용하여 안전하지 않은 질의문이 작성되어 expr 변수에
   저장된다.
```

안전하지 않은 코드의 예 JAVA

```
7:XPathExpression expr = xpath.compile("//users/user[login/text()='"+nm+"' and
  password/text()='"+pw+"']/home_dir/text()");
// 안전하지 않은 질의문이 담긴 expr을 평가하여 결과를 result에 저장한다.
8:Object result = expr.evaluate(doc, XPathConstants.NODESET);
// result의 결과를 NodeList 타입으로 변환하여 nodes 저장한다.
9:NodeList nodes = (NodeList) result;
10:for (int i=0; i<nodes.getLength(); i++) {
11:    String value = nodes.item(i).getNodeValue();
12:    if (value.indexOf(">") < 0) {
   // 공격자가 이름과 비밀번호를 확인할 수 있다.
13:      System.out.println(value);
14:  }:
15:}
```

다음의 예제는 XQuery를 사용하여 미리 쿼리 골격을 생성함으로써 외부입력으로 인해 쿼리 구조가 바뀌는 것을 막을 수 있다.

안전한 코드의 예 JAVA

[login.xq 파일]

```
declare variable $loginID as xs:string external;
declare variable $password as xs:string external;
//users/user[@loginID=$loginID and @password=$password]
// XQuery를 이용한 XPath Injection 방지
1:String nm = props.getProperty("name");
2:String pw = props.getProperty("password");
3:Document doc = new Builder().build("users.xml");
//파라미터화된 쿼리가 담겨있는 login.xq를 읽어와서 파라미터화된 쿼리를 생성한다.
4:XQuery xquery = new XQueryFactory().createXQuery(new File("login.xq"));
5:Map vars = new HashMap();
//검증되지 않은 외부값인 nm, pw를 파라미터화된 쿼리의 파라미터로 설정한다.
6:vars.put("loginID", nm);
7:vars.put("password", pw);
// 파라미터화된 쿼리를 실행하므로 외부값을 검증 없이 사용하여도 안전하다.
```

안전한 코드의 예 JAVA

```
8:Nodes results = xquery.execute(doc, null, vars).toNodes();
9:for (int i=0; i<results.size(); i++) {
10:     System.out.println(results.get(i).toXML());21:
11:}
```

파라미터화된 쿼리를 지원하는 XQuery 구문으로 대체할 수 없는 경우에는 XPath 삽입을 유발할 수 있는 문자들을 입력값에서 제거하고 XPath 구문을 생성, 실행하도록 한다.

안전한 코드의 예 JAVA

```
// XPath 삽입을 유발할 수 있는 문자들을 입력값에서 제거
1:public String XPathFilter(String input) {
2:   if (input != null) return input.replaceAll("['\\[]", "");
3:   else return "";
4:}
5:.....
// 외부 입력값에 사용
6:String nm = XPathFilter(props.getProperty("name"));
7:String pw = XPathFilter(props.getProperty("password"));
8:      .....
9:XPathFactory factory = XPathFactory.newInstance();
10:XPath xpath = factory.newXPath();
11:         .....
//외부 입력값인 nm, pw를 검증하여 쿼리문을 생성하므로 안전하다.
12:XPathExpression expr = xpath.compile("//users/user[login/text()='"+nm+"' and
   password/text()='"+pw+"']/home_dir/text()");
13:Object result = expr.evaluate(doc, XPathConstants.NODESET);
14:NodeList nodes = (NodeList) result;
15:for (int i=0; i<nodes.getLength(); i++) {
16:   String value = nodes.item(i).getNodeValue();
17:   if (value.indexOf(">") < 0) {
18:        System.out.println(value);
19:   }
20:}
21:   .....
```

아래 코드는 입력값을 검증하지 않고 입력값을 XPath 구문 생성 및 실행에 사용하고 있다. 입력값으로 any' or 'a' = 'a 와 같이 XPath 구문을 조작하는 문자열을 전달하는 경우 //food [name='any' or 'a' = 'a']/price 같이 구문이 만들어지고 실행되어 모든 food의 가격(price)가 조회되게 된다.

안전하지 않은 코드의 예 JAVA

```
1:public static void main(String[] args) throws Exception {
2:   if (args.length <= 0) {
3:      System.err.println("가격을 검색할 식품의 이름을 입력하세요.");
4:      return;
5:   }
6:   String name = args[0];
7:   DocumentBuilder docBuilder =DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = docBuilder.parse("http://www.w3schools.com/xml/simple.xml");
8:   XPath xpath = XPathFactory.newInstance().newXPath();
// 프로그램의 커맨드 옵션으로 입력되는 외부값 name을 사용하여 쿼리문을 직접
// 작성하여 수행하므로 안전하지 않다.
9:   NodeList nodes = (NodeList) xpath.evaluate("//food[name='" + name + "']/price",
     doc, XPathConstants.NODESET);
10:  for (int i = 0; i < nodes.getLength(); i++) {
11:    System.out.println(nodes.item(i).getTextContent());
12: }
13:}
```

외부 입력값을 XPath 구문 생성 및 실행에 사용하는 경우 XPath 구문을 조작할 수 있는 문자열을 제거하고 사용할 수 있도록 한다.

안전한 코드의 예 JAVA

```
1:public static void main(String[] args) throws Exception {
2:   if (args.length <= 0) {
3:      System.err.println("가격을 검색할 식품의 이름을 입력하세요.");
4:      return;
```

안전한 코드의 예 JAVA

```
5:  }
//프로그램의 커맨드 옵션으로 입력되는 외부값 name에서 XPath 구문을 조작할 수
// 는 문자를 제거하는 검증을 수행하여 안전하다.
6:  String name = args[0];
7:  if (name != null) {
8:      name = name.replaceAll("[()\\-'\\[\\]:,*/]", "");
9:  }
10:  DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
11:  Document doc = docBuilder.parse("http://www.w3schools.com/xml/simple.xml");
12:  XPath xpath = XPathFactory.newInstance().newXPath();
13:  NodeList nodes = (NodeList) xpath.evaluate("//food[name='" + name + "']/price",
   doc, XPathConstants.NODESET);
14:  for (int i = 0; i < nodes.getLength(); i++) {
15: System.out.println(nodes.item(i).getTextContent());
16:  }
17:}
```

다음의 예제는 외부입력을 사용하여 XPath 식에 바로 사용합니다. 이는 공격자의 입력에 따라 XQuery의 구조를 바꾸어 예기치 않은 공격이 발생할 수 있습니다.

안전하지 않은 코드의 예 C#

```
1:string acctID = Request["acctID"];
2:string query = null;
3:if(acctID != null)
4:{
5:  StringBuffer sb = new StringBuffer("/accounts/account[acctID='");
6:  sb.Append(acctID);
7:  sb.Append("']/email/text()");
8:  query = sb.ToString();
9:}
10: XPathDocument docNav = new XPathDocument(myXml);
```

안전하지 않은 코드의 예 C#

```
11:XPathNavigator nav = docNav.CreateNavigator();
//외부 입력값을 검증 없이 사용하고 있습니다.
12:nav.Evaluate(query);
```

다음의 예제는 XPathExpression을 사용하여 미리 쿼리 골격을 생성함으로써 외부입력으로 인해 쿼리 구조가 바뀌는 것을 막을 수 있다.

안전한 코드의 예 C#

```
1:string xpath = "/accounts/account[@acctID=$acctID]/email/text()";
2:XPathExpression expr = DynamicContext.Compile(xpath);
3:DynamicContext ctx = new DynamicContext();
4:ctx.AddVariable("acctID", AccountIDTextBox.Text);
5:expr.SetContext(ctx);
6:XPathNodeIterator data = nav.Select(expr);
```

라. 진단방법

· XQuery 삽입

XQuery가 실행되는 부분을 확인하고(①), XQuery 쿼리스트링에 사용되는 변수가 외부 입력값 여부를 확인한 후(②), 변수에 대한 필터링 모듈이 존재하는지 확인한다. 필터링 모듈이 존재하거나 관련 프레임워크에서 적절히 조치할 경우 안전한 것으로 판정한다.

일반적인 진단의 예

```
1:  …
2:  // 외부로부터 입력을 받음
3:  String name = props.getProperty("name"); ……②
4:  Hashtable env = new Hashtable();
5:  env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
6:  env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=rootDir");
```

일반적인 진단의 예

```
7:  javax.naming.directory.DirContext ctx = new InitialDirContext(env);
8:  javax.xml.xquery.XQDataSource xqds = (javax.xml.xquery.XQDataSource)
    ctx.lookup("x- qj/personnel");
9:  javax.xml.xquery.XQConnection conn = xqds.getConnection();
10:
11: String es = "doc('users.xml')/userlist/user[uname='" + name + "']";  ····· ②
12: // 입력값이 Xquery의 인자로 사용
13: XQPreparedExpression expr = conn.prepareExpression(es);
14: XQResultSequence result = expr.executeQuery();  ····· ①
15: while (result.next()) {
16: String str = result.getAtomicValue();
17: if (str.indexOf('>') < 0) {
18: System.out.println(str);
19: }
20: …
```

다음의 코드에서는 외부의 입력(name)값을 executeQuery를 사용한 쿼리생성의 문자열 인자 생성에 사용하고 있다. 만일 다음과 something' or '='1 을 name의 값으로 전달하면 다음과 같은 쿼리문을 수행할 수 있으며, 이로써 파일 내의 모든 값을 출력할 수 있게 되어 취약하다.

(doc('users.xml')/userlist/user[uname='something' or '=')

정탐코드의 예

```
1:  …
2:  // 외부로부터 입력을 받음
3:  String name = props.getProperty("name");
4:  Hashtable env = new Hashtable();
5:  env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
6:  env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=rootDir");
7:  javax.naming.directory.DirContext ctx = new InitialDirContext(env);
8:  javax.xml.xquery.XQDataSource xqds = (javax.xml.xquery.XQDataSource)
    ctx.lookup("x- qj/personnel");
```

정탐코드의 예

```
9: javax.xml.xquery.XQConnection conn = xqds.getConnection();
10:
11:String es = "doc('users.xml')/userlist/user[uname='" + name + "']";
12:// 입력 값이 Xquery의 인자로 사용
13:XQPreparedExpression expr = conn.prepareExpression(es);
14:XQResultSequence result = expr.executeQuery();
15:while (result.next()) {
16:String str = result.getAtomicValue();
17:if (str.indexOf('〉') 〈 0) {
18:System.out.println(str);
19:}
20:…
```

· XPath 삽입

XPath 객체로 쿼리 스트링이 컴파일 되는 부분을 확인하고(①), XPath 쿼리스트링에 사용되는 변수가 외부 입력값 인지 확인한 후(②) 변수에 대한 필터링 모듈이 존재하는지 확인한다. 필터링 모듈이 존재하거나 관련 프레임워크에서 적절하게 조치된 경우엔 안전하다고 판정한다.

일반적인 진단의 예

```
1:  …
2:  String acctID = request.getParameter("acctID");  ····· ②
3:  String query = null;
4:  if(acctID != null) {
5:  StringBuffer sb = new StringBuffer("/accounts/account[acctID='");
6:  sb.append(acctID);  ····· ②
7:  sb.append("']/email/text()");
8:  query = sb.toString();
```

일반적인 진단의 예

```
9:  }
10: DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
11: domFactory.setNamespaceAware(true);
12: DocumentBuilder builder = domFactory.newDocumentBuilder();
13: Document doc = builder.parse("accounts.xml");
14: XPathFactory factory = XPathFactory.newInstance();
15: XPath xpath = factory.newXPath();
16: XPathExpression expr = xpath.compile(query); ····· ①
17: Object result = expr.evaluate(doc, XPathConstants.NODESET);
18: …
```

다음의 예제에서는 name의 값으로 “”user1””, passwd의 값으로 “”’ or “=’””을 전달하면 다음과 같은 쿼리문이 생성되어 인증과정을 거치지 않고 로그인할 수 있어 취약하다고 판정한다.

(//users/user[login/text()=’user1’ or ‘’=’’ and password/text() = “” or “”=””]/home_dir/text())

정탐코드의 예

```
1:  …
2:  // 외부로부터 입력을 받음
3:  String name = props.getProperty("name");
4:  String passwd = props.getProperty("password");
5:  …
6:  XPathFactory factory = XPathFactory.newInstance();
7:  XPath xpath = factory.newXPath();
8:  …
9:  // 외부 입력이 xpath의 인자로 사용
10: XPathExpression expr = xpath.compile("//users/user[login/text()='" + name + "' and
    password/text() = '" + passwd + "']/home_dir/text()");
11: Object result = expr.evaluate(doc, XPathConstants.NODESET);
12: NodeList nodes = (NodeList) result;
13: for (int i = 0; i < nodes.getLength(); i++) {
14: String value = nodes.item(i).getNodeValue();
15: if (value.indexOf(">") < 0) {
```

정탐코드의 예

```
16:  System.out.println(value);
17:  }
18:  …
```

마. 참고자료

[1] CWE-652 XQuery Injection, MITRE,
http://cwe.mitre.org/data/definitions/652.html

[2] Prevent XML Injection, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS16-J.+Prevent+XML+Injection

[3] CWE-643 XPath Injection, MITRE,
http://cwe.mitre.org/data/definitions/643.html

[4] Prevent XPath Injection, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS53-J.+Prevent+XPath +Injection

[5] XPATH Injection, OWASP,
https://www.owasp.org/index.php/XPATH_Injection

10. LDAP 삽입

가. 개요

공격자가 외부 입력으로 의도하지 않은 LDAP(Lightweight Directory Access Protocol) 명령어를 수행할 수 있다. 즉, 웹 응용프로그램이 사용자가 제공한 입력을 올바르게 처리하지 못하면, 공격자가 LDAP 명령문의 구성을 바꿀 수 있다. 이로 인해 프로세스가 명령을 실행한 컴포넌트와 동일한 권한(Permission)을 가지고 동작하게 된다.

외부입력값을 적절한 처리 없이 LDAP 쿼리문이나 결과의 일부로 사용하는 경우, LDAP 쿼리문이 실행될 때 공격자는 LDAP 쿼리문의 내용을 마음대로 변경할 수 있다.

나. 보안대책

DN(Distinguished Name)과 필터에 사용되는 사용자 입력값에는 특수문자가 포함되지 않도록 특수 문자를 제거한다. 만약 특수문자를 사용해야 하는 경우에는 특수문자(= + 〈 〉 # ; \ 등)가 실행 명령이 아닌 일반문자로 인식되도록 처리한다.

다. 코드예제

userSN과 userPassword 변수의 값으로 *을 전달할 경우 필터 문자열은 “(&(sn=S*)(userPassword=*))”가 되어 항상 참이 되며 이는 의도하지 않은 동작을 유발시킬 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: private void searchRecord(String userSN, String userPassword) throws
   NamingException {
2:    Hashtable<String, String> env = new Hashtable<String, String>();
3:    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
4:    try {
5:       DirContext dctx = new InitialDirContext(env);
6:       SearchControls sc = new SearchControls();
7:       String[] attributeFilter = { "cn", "mail" };
8:       sc.setReturningAttributes(attributeFilter);
```

안전하지 않은 코드의 예 JAVA

```
9:      sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
10:     String base = "dc=example,dc=com";
//userSN과 userPassword 값에 LDAP필터를 조작할 수 있는 공격 문자열에 대한 검증이 없어 안전하지 않다.
11:      String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";
12:     NamingEnumeration<?> results = dctx.search(base, filter, sc);
13:     while (results.hasMore()) {
14:       SearchResult sr = (SearchResult) results.next();
15:       Attributes attrs = sr.getAttributes();
16:       Attribute attr = attrs.get("cn");
17:       .....
18:     }
19:     dctx.close();
20:  } catch (NamingException e) { … }
21:}
```

검색을 위한 필터 문자열로 사용되는 외부의 입력에서 위험한 문자열을 제거하여 위험성을 부분적으로 감소시킬 수 있다.

안전한 코드의 예 JAVA

```
1: private void searchRecord(String userSN, String userPassword) throws NamingException {
2:    Hashtable<String, String> env = new Hashtable<String, String>();
3:    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
4:    try {
5:       DirContext dctx = new InitialDirContext(env);
6:       SearchControls sc = new SearchControls();
7:       String[] attributeFilter = {"cn", "mail" };
8:       sc.setReturningAttributes(attributeFilter);
9:       sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
10:       String base = "dc=example,dc=com";
// userSN과 userPassword 값에서 LDAP 필터를 조작할 수 있는 문자열을 제거하고 사용
```

안전한 코드의 예 JAVA

```
11:     if (!userSN.matches("[\\w\\s]*") || !userPassword.matches("[\\w]*")) {
12:         throw new IllegalArgumentException("Invalid input");
13:     }
14:     String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";
15:     NamingEnumeration<?> results = dctx.search(base, filter, sc);
16:     while (results.hasMore()) {
17:       SearchResult sr = (SearchResult) results.next();
18:       Attributes attrs = sr.getAttributes();
19:       Attribute attr = attrs.get("cn");
20:       ......
21:     }
22:     dctx.close();
23:  } catch (NamingException e) { … }
24:}
```

다음은 인증하지 않은 익명 바인딩으로 LDAP 쿼리를 실행하는 C# 코드 예제이다.

안전하지 않은 코드의 예 C#

```
1: static void SearchRecord(string userSN, string userPW)
2: {
3: try {
4:    DirectoryEntry oDE;
5:    oDE = new DirectoryEntry(GetStrPath());
   // 인증을 하지않은 익명 바인딩으로 LDAP 쿼리를 실행
6:    foreach(DirectoryEntry objChildDE om oDE.Children) {
7:      ...
8:    }
9:              } catch (NamingException e) { ... }
10:}
```

익명 바인딩을 사용하지 않고 인증을 진행해야 한다.

안전한 코드의 예 C#

```
1: static void SearchRecord(string userSN, string userPW)
2: {
3: try {
4:   DirectoryEntry oDE;
5:   oDE = new DirectoryEntry(GetStrPath(), userSN, userPW);
   // userSN과 userPW 로 인증 후에 LDAP 쿼리를 실행
6:   foreach(DirectoryEntry objChildDE om oDE.Children) {
7:   ...
8:   }
9: } catch (NamingException e) { ... }
```

아래 C 코드는 외부에서 불러온 filter를 LDAP 탐색에 그대로 사용하고 있다. LDAP 필터에 OR 연산 등을 사용하여 의도치 않은 동작이 발생할 수 있다.

안전하지 않은 코드의 예 C

```
1: void LDAPInjection() {
2:   char *filter = getenv("Filter");
3:   int error_code;
4:   LDAP *ld = NULL;
5:   LDAPMessage *result;
// 외부에서 불러온 filter를 검증 없이 사용
6:   error_code = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter,
   NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
7: }
```

외부 입력이 아닌 고정된 LDAP 쿼리문을 사용하여 문제를 해결할 수 있다.

안전한 코드의 예 C

```
1: void LDAPInjection() {
2: char *filter = getenv("Filter");
3: int error_code;
4: int i;
5: LDAP *ld = NULL;
6: LDAPMessage *result;
// 정보를 알고 싶은 사용자의 이름을 고정 값으로 사용
7:  for(i = 0; *(filter + i) != 0; i++) {
// 공격 가능한 문자열 검사
8: switch(*(filter + i)) {
9:   case '*':
10:   case '(':
11:   case ')':
12: …
13:      return;
14:}
15:}
16:error_code = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter,
NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
```

라. 진단방법

LDAP 조회 쿼리가 실행됨을 확인하고(①), LDAP 조회문의 필터에 사용되는 변수가 외부 입력값인지 확인한 후(②), 해당 변수에 대한 필터링 모듈이 존재하는지 확인한다(③). 필터링 모듈이 존재하거나 관련 프레임워크에서 적절히 조치할 경우엔 안전한 것으로 판정한다.

일반적인 진단의 예

```
1:  …DirContext ctx = new InitialDirContext(env);
2:  String managerName = request.getParameter("managerName"); ……③
```

일반적인 진단의 예

```
3:  //retrieve all of the employees who report to a manager
4:  String filter = "(manager=" + managerName + ")"; ····· ②
5:  NamingEnumeration employees =
    ctx.search("ou=People,dc=example,dc=com", filter); ····· ①
6:  …
```

다음의 예제에서는 외부의 입력(name)이 검색을 위한 필터 문자열의 생성에 사용되고 있다. name 변수의 값으로 "*"을 전달할 경우, 필터 문자열은 "(name=*)"가 할당되어 항상 참이 되므로, 인증을 우회하거나 응용프로그램이 의도하지 않은 동작을 하게 되므로 취약한 것으로 판정한다.

정탐코드의 예

```
1:  public void f() {
2:  Hashtable env = new Hashtable();
3:  env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
4:  env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=rootDir");
5:  try {
6:  javax.naming.directory.DirContext ctx = new InitialDirContext(env);
7:  // 프로퍼티를 만들고 외부 파일을 로드한다.
8:  Properties props = new Properties();
9:  String fileName = "ldap.properties";
10: FileInputStream in = new FileInputStream(fileName);
11: props.load(in);
12: // LDAP Search를 하기 위해 name을 읽는다
13: String name = props.getProperty("name");
4:  String filter = "(name =" + name + ")";
15: // LDAP search가 name값에 대한 여과없이 그대로 통과되어 검색이 되어진다.
16: NamingEnumeration answer = ctx.search("ou=NewHires", filter, new SearchCon-
    trols(
    ));
17: printSearchEnumeration(answer);
18: ctx.close();
19: }
20: catch (NamingException e) { … }
21: …
```

외부의 입력(name)이 검색을 위한 base 문자열의 생성에 사용되고 있다. 이 경우 임의의 루트 디렉터리를 지정하여 정보에 접근할 수 있으며, 적절한 접근제어가 동반되지 않을 경우 정보 누출이 발생할 수 있다.

정탐코드의 예

```
1:  …
2:   try {
3:    …
4:    // 외부로부터 입력을 받는다.
5:    String name = props.getProperty(""ldap.properties");
6:    // 입력 값에 대한 BasicAttribute를 생성한다.
7:    BasicAttribute attr = new BasicAttribute("name", name);
8:    // 외부 입력 값이 LDAP search의 인자로 사용이 된다.
9:    NamingEnumeration answer = ctx.search("ou=NewHires", attr.getID(), new Search-
      Controls());
10:   printSearchEnumeration(answer);
11:   ctx.close();
12:  }
13:  catch (NamingException e) { … }
14: }
15:
16: public void printSearchEnumeration(NamingEnumeration value) {
17:  try {
18:   while (value.hasMore()) {
19:   SearchResult sr = (SearchResult) value.next();
20:   System.out.println(">>>" + sr.getName() + "\n" + sr.getAttributes());
21:  }
22: }
23:  catch (NamingException e) { … }
24: …
```

마. 참고자료

[1] CWE-90 LDAP Injection, MITRE,
http://cwe.mitre.org/data/definitions/90.html

[2] Prevent LDAP injection, CERT,
http://www.securecoding.cert.org/confluence/display/java/IDS54-J.+Prevent+LDAP+injection

[3] LDAP injection, OWASP
https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet

[4] "LDAP Resources"
http://ldapman.org/

11. 크로스사이트 요청 위조

가. 개요

특정 웹사이트에 대해서 사용자가 인지하지 못한 상황에서 사용자의 의도와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 요청하게 하는 공격을 말한다. 웹 응용프로그램이 사용자로부터 받은 요청에 대해서 사용자가 의도한 대로 작성되고 전송된 것인지 확인하지 않는 경우 발생 가능하고 특히 해당 사용자가 관리자인 경우 사용자 권한관리, 게시물 삭제, 사용자 등록 등 관리자 권한으로만 수행 가능한 기능을 공격자의 의도대로 실행시킬 수 있게 된다.

공격자는 사용자가 인증한 세션이 특정 동작을 수행하여도 계속 유지되어 정상적인 요청과 비정상적인 요청을 구분하지 못하는 점을 악용한다. 웹 응용프로그램에 요청을 전달할 경우, 해당 요청의 적법성을 입증하기 위하여 전달되는 값이 고정되어 있고 이러한 자료가 GET 방식으로 전달된다면 공격자가 이를 쉽게 알아내어 원하는 요청을 보냄으로써 위험한 작업을 요청할 수 있게 된다.

나. 보안대책

입력화면 폼 작성시 GET 방식보다는 POST 방식을 사용하고 입력화면 폼과 해당 입력을 처리하는 프로그램 사이에 토큰을 사용하여, 공격자의 직접적인 URL 사용이 동작하지 않도록 처리한다. 특히 중요한 기능에 대해서는 사용자 세션검증과 더불어 재인증을 유도한다.

다. 코드예제

클라이언트로부터의 요청(request)에 대해서 정상적인 요청 여부인지를 검증하지 않고 처리하는 경우, 크로스사이트 요청 위조 공격에 쉽게 노출될 수 있다.

안전하지 않은 코드의 예

```
// 어떤 형태의 요청이던지 기본적으로 CSRF 취약점을 가질 수 있다.
```

정상 요청 여부를 판단하기 위해 토큰을 이용한다. 사용자가 입력(신청) 페이지를 요청하면 임의의 토큰을 생성한 후 세션에 저장하고, 입력(신청) 페이지에 생성한 토큰을 HIDDEN 필드 항목의 값으로 설정한다. 입력(신청)을 처리하는 페이지에서는 입력(신청) 페이지에서 요청 파라미터로 전달된 HIDDEN 필드의 토큰 값과 세션에 저장된 토큰 값을 비교하여 일치하는 경우에만 정상 요청으로 판단하여 입력(신청)이 처리될 수 있도록 한다.

안전한 코드의 예 JAVA

```
// 입력화면이 요청되었을 때, 임의의 토큰을 생성한 후 세션에 저장한다.
1:session.setAttribute("SESSION_CSRF_TOKEN", UUID.randomUUID().toString());
// 입력화면에 임의의 토큰을 HIDDEN 필드항목의 값으로 설정해 서버로 전달되도록 한다.
2:<input type="hidden" name="param_csrf_token" value="${SESSION_CSRF_TOKEN}" />
// 요청 파라미터와 세션에 저장된 토큰을 비교해서 일치하는 경우에만 요청을 처리한다.
3:String pToken = request.getParameter("param_csrf_token");
4:String sToken = (String)session.getAttribute("SESSION_CSRF_TOKEN");
5:if (pToken != null && pToken.equals(sToken) {
   // 일치하는 토큰이 존재하는 경우 -> 정상 처리
6:          .....
7:} else {
   // 토큰이 없거나 값이 일치하지 않는 경우 -> 오류 메시지 출력
8:          .....
9:}
```

AntiForgeryToken()등을 이용하여 크로스사이트 요청 위조를 방지해야 한다.

안전한 코드의 예 C#

```
1: @using (Html.BeginForm("PostTest","Home",FormMethod.Post,null))
2: {
3:       //AntiForgeryToken()을 이용해 크로스사이트 요청 위조를 방지
4:       @Html.AntiForgeryToken()
5:       <input type="submit" value="Html PsBk Click" />
6: }
```

라. 진단방법

사용자 권한변경, 신규정보 등록 등 주요 기능을 확인하고(①), 해당 기능 수행시 권한확인 절차 존재 여부 확인한다(②). 권한확인 절차가 없거나 권한 확인 방법이 세션쿠키, 사용자 IP, SSL 인증과 같이 자동 제출되는 자격증명에 의존하는 경우 취약하다.

일반적인 진단의 예

```
1:  …
2:  int level = request.getParameter("level"); ····· ①
3:  int group = request.getParameter("group"); ····· ①
4:  String id = request.getParameter("id"); ····· ①
5:  String sql="update member set level=? , group=? where id=?;"; ····· ①
6:  pstmt =con.prepareStatement(sql);
7:  pstmt.setInt(1, level);
8:  pstmt.setString(2, group);
9:  pstmt.setInt(3, id);
10: pstmt.executeUpdate(); ····· ②
11: …
```

다음의 예제를 살펴보면, 2번째 라인에서 임의의 토큰을 생성한 후 세션에 저장하고, 5번째 라인에서 생성한 토큰을 HIDDEN 필드 항목의 값으로 설정한다. 그리고 10번째 라인에서 HIDDEN 필드의 토큰 값과 세션에 저장된 토큰 값을 비교하여 일치하는 경우에만 정상 요청으로 판단하여 처리하고 있기 때문에 오탐으로 판단한다.

오탐코드의 예

```
1:  // 입력화면이 요청되었을 때, 임의의 토큰을 생성한 후 세션에 저장한다.
2:  session.setAttribute("SESSION_CSRF_TOKEN", UUID.randomUUID().toString());
3:
4:  // 입력화면에 임의의 토큰을 HIDDEN 필드항목의 값으로 설정해 서버로 전달되도록 한다.
5:  <input type="hidden" name="param_csrf_token" value="${SESSION_CSRF_TOKEN}" />
6:
7:  // 요청 파라미터와 세션에 저장된 토큰을 비교해서 일치하는 경우에만 요청을 처리한다.
8:  String pToken = request.getParameter("param_csrf_token");
9:  String sToken = (String)session.getAttribute("SESSION_CSRF_TOKEN");
```

오탐코드의 예

```
10: if (pToken != null && pToken.equals(sToken) {
11:     // 일치하는 토큰이 존재하는 경우 -> 정상 처리
12:                 ......
13: } else {
14:    // 토큰이 없거나 값이 일치하지 않는 경우 -> 오류 메시지 출력
15:                                 ..
                                    ..
                                    ..
16: }
```

마. 참고자료

[1] CWE-352 Cross-Site Request Forgery(CSRF), MITRE, http://cwe.mitre.org/data/definitions/352.html

[2] "Security Corner: Cross-Site Request Forgeries", Chris Shiflett, http://shiflett.org/articles/cross-site-request-forgeries

[3] Cross-Site_Request_Forgery_(CSRF), https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

12. 서버사이드 요청 위조

가. 개요

적절한 검증절차를 거치지 않은 사용자 입력 값을 서버간의 요청에 사용하여 악의적인 행위가 발생할 수 있는 보안약점이다.

외부에 노출된 웹 서버에 취약한 애플리케이션이 존재하는 경우 공격자는 URL 또는 요청문을 위조하여 접근통제를 우회하는 방식으로 비정상적인 동작을 유도하거나 신뢰된 네트워크에 있는 데이터를 획득할 수 있다.

나. 보안대책

식별할 수 있는 범위 내에서 사용자의 입력 값을 다른 시스템의 서비스 호출에 사용하는 경우, 사용자의 입력 값을 화이트리스트 방식으로 필터링한다.

사용자가 지정하는 무작위의 URL을 받아들여야 한다면 내부의 URL을 블랙리스트로 지정하여 필터링 한다. 또한 동일한 내부 네트워크에 있더라도 기기 인증, 접근권한을 확인하여 요청이 이루어질 수 있도록 한다.

다. 코드예제

다음 예제는 사용자로부터 입력받은 값의 검증 없이 웹페이지를 접속하도록 구현되어 있다. 이 때, 공격자는 URL을 조작하여 내부 서버에 질의 하게 하여 데이터를 획득할 수 있다.

[참고 : 삽입 코드의 예]

설명	삽입 코드의 예
내부망 중요 정보 획득	• http://site_example.com/connect?url=http://192.168.0.45/member/list.json
외부 접근 차단된 admin 페이지 접근	• http://site_example.com/connect?url=http://192.168.0.45/admin
도메인 체크를 우회하여 중요 정보 획득	• http://site_example.com/connect?url=http://site_example.com:x@192.168.0.45/member/list.json
단축 URL을 이용한 Filter 우회	• http://site_example.com/connect?url=http://bit.ly/sdjk3kjhkl3
도메인을 사설IP로 설정해 중요정보 획득	• http://site_example.com/connect?url=http://internal.site.com/member/list.json
서버내 파일 열람	• http://site_example.com/connect?url=http://attack/fileview.html

안전하지 않은 코드의 예 JAVA

```
1:protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
2:      // 사용자 입력값(url)을 검증없이 사용하여 안전하지 않다.
3:      URL url = new URL(req.getParameter("url"));
4:      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
5:}
```

다음은 사전에 정의된 URL 목록을 맵(Map) 객체에 정의하고 키 값으로 입력받아 키에 매칭되는 URL만 사용할 수 있으므로 URL값을 임의로 조작할 수 없다.

안전한 코드의 예 JAVA

```
1:public class Connect {
2:      // key, value 형식으로 URL의 리스트를 작성한다.
3:      private Map<String, URL> urlMap;
4:      protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
5:          // 사용자에게 urlMap의 key를 입력받아 urlMap에서 URL값을 참조한다.
6:          URL url = urlMap.get(req.getParameter("url"));
7:          // urlMap에서 참조한 값으로 Connection을 만들어 접속한다.
8:          HttpURLConnection conn = (HttpURLConnection) url.openConnection();
9:      }
10: }
```

라. 진단방법

다른 시스템의 서비스를 호출하는 함수가 존재하는지 확인하고(①) 다른 시스템을 호출할 때 사용되는 입력 값이(②) 신뢰할 수 있는 값인지 확인한다. 만약 입력 값이 신뢰할 수 없고 별도의 검증절차가 없으면 안전하지 않다고 판정한다.

일반적인 진단의 예

```
1:public class G03 extends HttpServlet {
2:      protected void doPost(HttpServletRequest req, HttpServletResponse res)
3:      throws ServletException, IOException {
4:              String url = req.getParameter("url");  ····· ②
5:
6:              InputStream inputStream = null;
7:              OutputStream outputStream = null;
8:              //사용자에게 url을 입력 받는다.
9:              URL u = new URL(url);
10:             res.setHeader("content-disposition","attachment;fileName=“”");
11:
12:             int length;
13:             byte[] bytes = new byte[1024];
14:             // 입력받은 URL을 stream으로 생성한다.
15:             inputStream = u.openStream();  ····· ①
16:             outputStream = res.getOutputStream();
17:             while ((length = inputStream.read(bytes)) > 0) {
18:                     outputStream.write(bytes, 0, length);
19:             }
20:
21:     }
22:}
```

다음 예제에서는 openConnection에 사용하는 URL을 properties에서 참조하고 있다. properties는 다른 공격 등으로 위변조 될 수 있으므로 취약하다고 판정한다.

정탐코드의 예

```
1:public class ConnectProperties {
2:      FileReader newFile = new FileReader("File.properties");
3:      Properties properties = new Properties();
4:      properties.load(newFile);
5:
6:      protected void doGet(HttpServletRequestreq, HttpServletResponseresp) throws IOException{
7:              URL url = new URL(properties.getProperty("connectUrl"));
8:              HttpURLConnection conn = (HttpURLConnection) url.openConnection();
9:      }
10:}
```

다음 예제에서는 inURL 값이 추가적인 검증 없이 3번의 URL(url).openStream()에 전달된다. 이 때, 공격자가 inURL 입력 값을 악의적으로 조작하여 피해를 끼칠 수 있으므로 취약하다고 판정한다.

정탐코드의 예

```
1:private String getRemoteContent(String url) throws IOException {
2:      BufferedReader in = new BufferedReader(new InputStreamReader(
3:      new URL(url).openStream()));
4:      return FileCopyUtils.copyToString(in);
5:}
6:
7:public String getContent(String inUrl) throws IOException {
8:      try {
9:              String str = getRemoteContent(inUrl);
10:             str = str.replace("<head>", "<head><base href='" + inUrl
11:             + "' /><base target='_blank' /><script>top.studio.startPageIFrameLoaded();"
12:             + "</script>");
13:             return str;
14:             } catch (Exception e) {
15:                     return "";
16:             }
17:     }
```

다음 예제는 공격자가 host 파라미터를 조작하여 내부 서버의 정보를 조회할 수 있음을 보여준다. 이때, host 값을 별도의 검증 없이 전달하여 내부 서버의 데이터(secrets.txt) 파일이 외부로 유출될 수 있으므로 취약하다고 판정한다.

삽입 코드의 예 : /getFavicon?host=192.168.176.1:8080/secrets.txt?

정탐코드의 예

```
1:public void doGet(HttpServletRequest request, HttpServletResponse response) {
2:        String host = request.getParameter("host");
3:
4:        byte[] bytes = getImage(host, defaultBytes);
5:        if (bytes != null) {
6:                writeBytesToStream(bytes, response);
7:        }
8:}
9:
10:private byte[] getImage(String host, byte[] defaultImage) {
11:        byte[] bytes = getImage("http://" + host + "/favicon.ico");
12:        ...
13:}
```

다음 예제는 11라인에서 Public Cloud의 메타데이터 서비스 주소인 169.254.169.254를 블랙리스트 방식으로 필터링하여 에러를 리턴하여 메타서버의 크레덴셜 조회를 차단할 수 있으므로 취약하지 않다고 판정한다.

오탐코드의 예

```
1:<?php
2:        require_once('./htmlpurifier/library/HTMLPurifier.includes.php');
3:        $purifier = new HTMLPurifier();
4:
5:        $ch = curl_init();
6:        $url = $_GET['url'];
7:        $urlinfo = parse_url($url); // URL 파싱
8:        $scheme = $urlinfo['scheme'];
```

오탐코드의 예

```
9:      $host = $urlinfo['host'];
10:     $ip = gethostbyname($host);
11:     if ($ip === "169.254.169.254") { // 퍼블릭클라우드 메타데이터 서비스 IP를 검증
12:             die("Invalid host");
13:     } elseif ($scheme !== 'http' && $scheme !== 'https') { // HTTP(S) 스킴 체크
14:             die("Invalid scheme");
15:     }
16:     curl_setopt($ch, CURLOPT_URL, $url);
17:     curl_setopt($ch, CURLOPT_FOLLOWLOCATION, false);
18:     curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
19:
20:     $html = curl_exec($ch);
21:     echo $purifier->purify($html);
```

다음 예제는 내부 대역의 IP들에 대한 접근 시도를 블랙리스트 방식으로 필터링한다. 2라인에서 isPrivateIP() 메소드를 호출하여 입력 값을 블랙리스트 목록과 비교하여 허용되지 않은 접근에 대하여 오류를 리턴한다. 이 경우, 블랙리스트를 우회할 수 있는 다양한 공격을 감안해서 필터링하고 있는지 검토 후 취약 여부를 판단할 필요가 있다.

오탐코드의 예

```
1:private static String getRemoteContent(String url) throws IOException {
2:      if ( isPrivateIP(url) == true ) {
3:              return "invalid url";
4:      }
5:      BufferedReader in = new BufferedReader(new InputStreamReader(
6:      new URL(url).openStream()));
7:      return FileCopyUtils.copyToString(in);
8:}
9:
```

오탐코드의 예

```
10:public static boolean isPrivateIP(String r){
11:     //블랙리스트 목록(lookback 주소, IPv4 및 IPv6 주소)
12:     String private_ip[] = {
13:             //loopback addresses
14:             "127.", "0.",
15:             //IP V4 prefix for private addresses
16:             "10.",
17:             "172.16.", "172.17.", "172.18.", "172.19.", "172.20.", "172.21.",
18:             "172.22.", "172.23.", "172.24.", "172.25.", "172.26.", "172.27.",
19:             "172.28.", "172.29.", "172.30.", "172.31.", "192.168.", "169.254.",
20:             //IP V6 prefix for private addresses
21:             "fc", "fd", "fe", "ff", "::1"
22:     };
23:
24:     for(int i = 0; i<private_ip.length; i++){
25:              if(r.toLowerCase().trim().startsWith(private_ip[i])){
26:                      return true;
27:                      }
28:             }
29:     return false;
30:}
```

[참고 : SSRF 예시]

*** 문자열 인코딩 후 SSRF 예시**

URL	설명
/ssrf.php?url=http://425.510.425.510/	오버플로우 된 형식의 점이 포함된 IP
/ssrf.php?url=http://2852039166/	점이 포함되지 않은 10진수 형식
/ssrf.php?url=http://7147006462/	점이 포함되지 않은 오버플로된 형식
/ssrf.php?url=http://0xA9.0xFE.0xA9.0xFE/	점이 포함된 16진수 형식
/ssrf.php?url=http://0xA9FEA9FE/	점이 포함되지 않은 16진수 형식
/ssrf.php?url=http://0x41414141A9FEA9FE/	점이 포함되지 않은 16진수의 오버플로된 형식
/ssrf.php?url=http://0251.0376.0251.0376/	점이 포함된 8진수 형식
/ssrf.php?url=http://0251.00376.000251.0000376/	패딩이 포함된 점 포함 8진수 형식

- **IDNA2003 변환을 이용한 SSRF 예시**

URL	설명
/ssrf.php?url=http://ⓖⓞⓞⓖⓛⓔ.com	http://google.com (원문자는 영문으로 인식됨)
/ssrf.php?url=http://wordpreß.com	http://wordpress.com (독일어로 ß는 'ss'가 됨)

- **Broken Parser를 이용한 SSRF 예시**

URL	설명
/ssrf.php?url=https://evil-host#expected-host	#을 추가하여 탐지룰 회피 시도
/ssrf.php?url=https://expected-host@evil-host	@을 추가하여 탐지룰 회피 시도

- **SSRF를 이용한 LFI 예시**

URL	설명
/ssrf.php?url=file:///etc/passwd	로컬 파일을 읽음
/ssrf.php?url=ldap://localhost:1337/%0astats%0aquit	로컬 LDAP을 읽음

마. 참고자료

① CWE-918: Server-Side Request Forgery (SSRF), MITRE, https://cwe.mitre.org/data/definitions/918.html

② Server Side Request Forgery, OWASP https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

13. HTTP 응답분할

가. 개요

HTTP 요청에 들어 있는 파라미터(Parameter)가 HTTP 응답헤더에 포함되어 사용자에게 다시 전달될 때, 입력값에 CR(Carriage Return)이나 LF(Line Feed)와 같은 개행문자가 존재하면 HTTP 응답이 2개 이상으로 분리될 수 있다. 이 경우 공격자는 개행문자를 이용하여 첫 번째 응답을 종료시키고, 두 번째 응답에 악의적인 코드를 주입하여 XSS 및 캐시 훼손(Cache Poisoning) 공격 등을 수행 할 수 있다.

나. 보안대책

요청 파라미터의 값을 HTTP응답헤더(예를 들어, Set-Cookie 등)에 포함시킬 경우 CR, LF와 같은 개행문자를 제거한다.

다. 코드예제

외부입력값을 사용하여 반환되는 쿠키의 값을 설정하고 있다. 그런데, 공격자가 Wiley Hack er₩r₩nHTTP/1.1 200 OK₩r₩n를 lastLogin의 값으로 설정할 경우, 응답이 분리되어 전달되며 분리된 응답 본문의 내용을 공격자가 마음대로 수정할 수 있다.

안전하지 않은 코드의 예 JAVA

```
//외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
1: String lastLogin = request.getParameter("last_login");
2: if (lastLogin == null || "".equals(lastLogin)) {
3:     return;
4: }
// 쿠키는 Set-Cookie 응답헤더로 전달되므로 개행문자열 포함 여부 검증이 필요
5: Cookie c = new Cookie("LASTLOGIN", lastLogin);
6: c.setMaxAge(1000);
7: c.setSecure(true);
8: response.addCookie(c);
9: response.setContentType("text/html");
```

외부에서 입력되는 값에 대하여 null 여부를 체크하고, 응답이 여러 개로 나눠지는 것을 방지하기 위해 개행문자를 제거하고 응답헤더의 값으로 사용한다.

안전한 코드의 예 JAVA

```
1: String lastLogin = request.getParameter("last_login");
2: if (lastLogin == null || "".equals(lastLogin)) {
3:     return;
4: }
// 외부 입력값에서 개행문자(\r\n)를 제거한 후 쿠키의 값으로 설정
5: lastLogin = lastLogin.replaceAll("[\\r\\n]", "");
6: Cookie c = new Cookie("LASTLOGIN", lastLogin);
7: c.setMaxAge(1000);
8: c.setSecure(true);
9: response.addCookie(c);
```

외부 입력값으로 반환되는 쿠키값을 설정하는 C#코드이다. 이는 응답이 분리되어 전달될 수 있으며 분리된 응답 본문의 내용을 공격자가 마음대로 수정할 수 있다.

안전하지 않은 코드의 예 C#

```
//외부 입력값을 검증 없이 사용하는 것은 안전하지 않다.
1: string usrInput = Request.QueryString["ID"];
2: Response.AddHeader("foo", "bar" + usrInput);
```

개행문자를 모두 제거한 이후에 사용해야 한다.

안전한 코드의 예 C#

```
1:string usrInput = Request.QueryString["ID"];
//개행문자를 제거 한 이후에 사용해야 합니다.
2:string validatedInput = usrInput.Replace("\n", "").Replace("\r","");
3:Response.AddHeader("foo", "bar" + validatedInput);
```

라. 진단방법

Response 헤더에 변수가 사용되는 것을 확인하고(①), 변수가 외부 입력값인지 확인한 후(②), 개행문자(\r, \n) 제거를 위한 필터링 또는 검증절차가 있는지 확인한다. 외부 입력값에 대한 필터링 절차가 없다면 취약하다.

일반적인 진단의 예

```
1:  protected void common(HttpServletRequest request, HttpServletResponse
    response) throws Exception {
2:    String fileid = request.getParameter("fileid");  ····· ②
3:
4:    BufferedInputStream in = null;
5:    String uploadPath = properties.getString(propertyName + "UPLOAD");
6:
7:    try {
8:     String fileName = uploadPath + "/" + fileid;
9:
10:    response.setContentType(mimetype + ";charset=utf-8");
11:    String fileName = java.net.URLEncoder.encode(fileid, "UTF-8");
12:
13:    response.setHeader("Content-Disposition",
14:    "attachment; filename=\"" + fileName + "\"");  ····· ①
15:    response.setHeader("Content-Transfer-Encoding", "binary");
16:
17:    response.getOutputStream().flush();
18:    response.getOutputStream().close();
19:  }
20:  catch(Exception e) {
21:   if(in!=null) {
22:    in.close();
23:   }
24:   }
25:   finally{ }
26: }
```

HTTP 응답분할은 공격자 보낸 조작된 입력값이 헤더에 쓰이고 이로 인해 HTTP 응답이 나누어지게 되는 취약점이다. response.sendRedirect 함수는 URL을 다른 곳으로 보내는 함수인데 그 방식은 헤더에서 상태 값을 301로 바꾸고 가고자 하는 URL을 헤더에 입력하는 것이다. 이때 url에 공격자의 입력값이 쓰이게 되면 조작된 입력값으로 인해 응답이 변경된다.

다음의 예제를 살펴보면, 5번째 라인에서 request의 파라미터에서 filename 값을 가져오고 7번째 라인에서 filename값을 이용하여 헤더에 값을 설정하고 있어 취약하다.

정탐코드의 예

```
1:  public voidservice(HttpServletRequest request, HttpServletResponse res) throws
    Serv- letException, IOException {
2:  String contextRealPath = request.getSession().getServletContext().getRealPath("/");
3:  String savePath = contextRealPath + "upfolder"
4:  res.setContentType("utf-8");
5:  String filename = request.getParameter("file");
6:  res.setContentType("application/octet;charset=utf-8");
7:  res.setHeader("Content-Disposition", "attachment;filename=" + filename);
```

다음의 예제를 살펴보면, 2번째 라인에서 request의 last_login 파라미터에서 값을 가져오고 7번째 라인에서 lastLogin 값을 쿠키에 설정하고 있어 취약하다.

정탐코드의 예

```
1: ......
2:String lastLogin = request.getParameter("last_login");
3:if (lastLogin == null || "".equals(lastLogin)) {
4:return;
5:}
6:// 쿠키는 Set-Cookie 응답헤더로 전달되므로 개행문자열 포함 여부 검증이 필요
7:Cookie c = new Cookie("LASTLOGIN", lastLogin);
8:c.setMaxAge(1000);
9:c.setSecure(true);
10:response.addCookie(c);
11:response.setContentType("text/html");
12: ......
```

아래 코드를 살펴보면, 2번째 라인에서 request의 last_login 파라미터에서 가져온 값을 6번 라인에서 개행문자를 필터링한 후 7번째 라인에서 lastLogin값을 쿠키에 설정하고 있어 취약하지 않다고 판단한다.

오탐코드의 예

```
1: String lastLogin = request.getParameter("last_login");
2: if (lastLogin == null || "".equals(lastLogin)) {
3:     return;
4: }
5: // 외부 입력값에서 개행문자를 제거한 후 쿠키의 값으로 설정
6: lastLogin = lastLogin.replaceAll("[₩₩r ₩₩n]",
"");  7:Cookie c = new Cookie("LASTLOGIN",
lastLogin);  8: c.setMaxAge(1000);
9: c.setSecure(true);
10:response.addCookie(c);
11:response.setContentType("text/html");
12:          .....
```

아래코드의 HttpRequest.getContextPath() 함수는 내장함수로 context path를 리턴 하므로 HTTP응답 분할이 이루어지지 않는다.

오탐코드의 예

```
1:   1: response.sendRedirect(request.getContextPath() + "/login.do");
```

Long, Integer 값 등 Numeric 값은 문자열로 치환 했을때 \r\n 등의 문자열이 포함될 수 없으므로 취약 하지 않다.

오탐코드의 예

```
1:   1: 1: response.setHeader("Content-Length",Long.toString(file.length()));
```

마. 참고자료

[1] CWE-113 HTTP Response Splitting, MITRE,
http://cwe.mitre.org/data/definitions/113.html

[2] HTTP Response Splitting, OWASP
https://www.owasp.org/index.php/HTTP_Response_Splitting

14. 정수형 오버플로우

가. 개요

정수형 오버플로우는 정수값이 증가하면서 허용된 가장 큰 값보다 커져서 실제 저장되는 값이 의도치 않게 아주 작은 수이거나 음수가 되어 발생한다. 특히 반복문 제어, 메모리 할당, 메모리 복사 등을 위한 조건으로 사용자가 제공하는 입력값을 사용하고 그 과정에서 정수형 오버플로우가 발생하는 경우 보안상 문제를 유발할 수 있다.

나. 보안대책

언어·플랫폼별 정수타입의 범위를 확인하여 사용한다. 정수형 변수를 연산에 사용하는 경우, 결과값의 범위를 체크하는 모듈을 사용한다. 외부입력 값을 동적 메모리 할당에 사용하는 경우, 변수값이 적절한 범위 내에 존재하는 값인지 확인한다.

다. 코드예제

다음의 예제는 외부의 입력(slf_msg_param_num)을 이용하여 동적으로 계산한 값을 배열의 크기(size)를 결정하는데 사용하고 있다. 만일 외부 입력으로부터 계산된 값(param_ct)이 오버플로우에 의해 음수값이 되면, 배열의 크기가 음수가 되어 시스템에 문제가 발생할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: String msg_str = "";
2: String tmp = request.getParameter("slf_msg_param_num");
3: tmp = StringUtil.isNullTrim(tmp);
4: if (tmp.equals("0")) {
5:     msg_str = PropertyUtil.getValue(msg_id);
6: } else {
// 외부 입력값을 정수형으로 사용할 때 입력값의 크기를 검증하지 않고 사용
7:     int param_ct = Integer.parseInt(tmp);
8:     String[] strArr = new String[param_ct];
```

동적 메모리 할당을 위해 외부 입력값을 배열의 크기로 사용하는 경우 그 값이 음수가 아닌지 검사 하는 작업이 필요하다.

안전한 코드의 예 JAVA

```
1: String msg_str = "";
2: String tmp = request.getParameter("slf_msg_param_num");
3: tmp = StringUtil.isNullTrim(tmp);
4: if (tmp.equals("0")) {
5:     msg_str = PropertyUtil.getValue(msg_id);
6: } else {
// 외부 입력값을 정수형으로 사용할 때 입력값의 크기를 검증하고 사용
7:     try {
8:       int param_ct = Integer.parseInt(tmp);
9:       if (param_ct < 0) {
10:          throw new Exception();
11:      }
12:      String[] strArr = new String[param_ct];
13:   } catch(Exception e) {
14:      msg_str = "잘못된 입력(접근) 입니다.";
15:   }
```

외부 입력값으로 배열의 접근 할 경우, 입력값이 너무 클 때 음수가 되어 시스템에 문제가 발생 할 수 있다.

안전하지 않은 코드의 예 C#

```
1: public static void Main(string[] args)
2: {
   // 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
3:     int usrNum = Int32.Parse(args[0]);
4:     string[] array = {"one", "two", "three", "four"};
5:     string num = array[usrNum];
6: }
```

checked 구문을 이용하여 오버플로우 발생 확인 및 처리를 해야 한다.

안전한 코드의 예 C#

```
1: public static void Main(string[] args)
2: {
3:     // checked 구문을 사용하여 오버플로우의 발생 여부 및 크기 확인
4:       try{
5:       int usrNum = checked(Int32.Parse(args[0]));
6:       string[] array = {"one", "two", "three", "four"};
7:       if(usrNum < 3)string num = array[usrNum];
8:     }
9:     catch (System.OverflowException e) { … }
10: }
```

안전하지 않은 코드의 예 C

```
1: void main(int argc, char* argv[])
2: {
   // 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
3:       int usr_num = 0;
4:       char* num_array[] = {“one”, “two”, “three”, “four”};
5:       char* num = NULL;
6:       usr_num = atoi(argv[1]);
7:       num = num_array[usr_num];
8: }
```

안전한 코드의 예 C

```
1: void main(int argc, char* argv[])
2: {
   // 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
3:     int usr_num = 0;
4:     char* num_array[] = {“one”, “two”, “three”, “four”};
5:     char* num = NULL;
```

안전한 코드의 예 C

```
6:    usr_num = atoi(argv[1]);
7:    if (usr_num >= 0 && usr_num < 4) {
8:      num = num_array[usr_num];
9:    }
10:}
```

라. 진단방법

변수를 사용하여 배열의 크기를 동적으로 결정하고 있는 경우(①), 변수가 외부 입력값인지 확인하고(②), 해당 변수가 의도한 범위내에 존재하는지 확인하는 절차가 있는지 확인한다. 외부 입력값에 대한 검증 절차가 없다면 취약하다.

일반적인 진단의 예

```
1:    …
2:    String cnt = request.getParameter("cnt");  ····· ②
3:    if( cnt == null ) {
4:    cnt = "0";
5:    }
6:    int cntI = Integer.parseInt(cnt);
7:    String[] arr = new String[cntI];  ····· ①
8:    for( int i = 0 ; i <cntI ; i++ ) {
9:    arr[i] = request.getParameter("r"+i);
10:   }
11:   …
```

정수형 변수의 값이 증가하면서 정수형 한계값 보다 더 커지는 경우 아주 작은 값이 되거나 음수가 될 수 있다. 이러한 상황에 대한 검사가 이루어지지 않고 진행하는 경우 취약하다.

정탐코드의 예

```
1:  public static Vector parsFileBySize(String parFile, int[] parLen, int parLine) throws
    Exception {
```

정탐코드의 예

```
2:  // 파싱결과 구조체
3:  Vector parResult = new Vector();
4:  // 파일 오픈
5:  String parFile1 = parFile.replace('\\', FILE_SEPARATOR).replace('/', FILE_SEPARATOR);
6:  File file = new File(parFile1);
7:  BufferedReader br = null;
8:  try {
9:  // 파일이며, 존재하면 파싱 시작
10: if (file.exists() && file.isFile()) {
11:  // 1. 입력된 라인수만큼 파일 텍스트 내용을 읽어서 String[]에 쌓는다.
12:  br = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
13:  String [] strArr = new String [parLine];
14:  String line = "";
15:  int readCnt = 0;
16:  while ((line = br.readLine()) != null && readCnt <
parLine) { 17:          if (line.length() <= MAX_STR_LEN)
strArr[readCnt++] = line; 18:     }
```

다음의 예제는 2번라인에서 외부의 입력(slf_msg_param_num)으로 받은 값을 9번라인에서 배열의 크기(size)를 결정하는데 사용하고 있다. 만일 외부에서 입력받은 값(param_ct)이 음수값인 경우, 배열의 크기가 음수가 되어 시스템에 문제가 발생할 수 있으므로 취약하다고 판단한다.

정탐코드의 예

```
1:  String msg_str = "";
2:  String tmp = request.getParameter("slf_msg_param_num");
3:  tmp = StringUtil.isNullTrim(tmp);
4:  if (tmp.equals("0")) {
5:      msg_str = PropertyUtil.getValue(msg_id);
6:  } else {
7:      // 외부 입력값을 정수형으로 사용할 때 입력값의 크기를 검증하지 않고 사용
8:      int param_ct = Integer.parseInt(tmp);
9:      String[] strArr = new String[param_ct];
10:         ......
11:}
```

다음의 예제에서 pubKey.available() 함수는 int 값을 리턴 한다. 그러므로 정수 값 한계를 벗어난 값은 발생하지 않는다.

오탐코드의 예

```
31:  try {
32:    URL url = Util.class.getClassLoader().getResource(publicKeyFilepath);
33:    pubKey = url.openStream();
34:  } catch (MalformedURLException e) {
35:    pubKey = new FileInputStream(publicKeyFilepath);
36:  }
37:  byte[] bytes = new byte[pubKey.available()];
```

다음의 예제에서 입력 값은 HTTP 헤더의 한계값이 있으므로 정수의 한계를 초과할 가능성이 없다.

오탐코드의 예

```
1:   String[] referer_split = Util.split(request.getHeader("Referer").toString(),"/");
2:   String refererValue = referer_split[referer_split.length-1];
```

다음의 예제에서 인자의 개수가 정수의 한계 이상으로 입력된다는 것은 실현 가능성이 희박하므로 취약하지 않은 것으로 판단한다.

오탐코드의 예

```
1:   public static void main(String[] args) {
2:     String[] jsargs = {"-j="+args[0]};
3:     String[] allArgs = new String[jsargs.length + args.length];
```

사용자 입력 값 등 외부 값이 아닌 경우 취약하지 않다.

마. 참고자료

[1] CWE-190 Integer Overflow, MITRE,
http://cwe.mitre.org/data/definitions/190.html

[2] Enforce limits on integer values originating from tainted sources, CERT,
http://www.securecoding.cert.org/confluence/display/c/INT04-C.+Enforce+limits+on+integer+values+originating+from+tainted+sources

[3] Verify that all integer values are in range, CERT,
http://www.securecoding.cert.org/confluence/display/c/INT08-C.+Verify+that+all+integer+values+are+in+range

[4] Integer overflow, OWASP,
https://www.owasp.org/index.php/OWASP_Periodic_Table_of_Vulnerabilities_-_Integer_Overflow/Underflow

15. 보안기능 결정에 사용되는 부적절한 입력값

가. 개요

응용프로그램이 외부입력값에 대한 신뢰를 전제로 보호메커니즘을 사용하는 경우 공격자가 입력값을 조작할 수 있다면 보호메커니즘을 우회할 수 있게 된다.

개발자들이 흔히 쿠키, 환경변수 또는 히든필드와 같은 입력값이 조작될 수 없다고 가정하지만 공격자는 다양한 방법으로 이러한 입력값들을 변경할 수 있고 조작된 내용은 탐지되지 않을 수 있다. 인증이나 인가와 같은 보안결정이 이런 입력값(쿠키, 환경변수, 히든필드 등)에 기반해 수행되는 경우 공격자는 이런 입력값을 조작하여 응용프로그램의 보안을 우회할 수 있으므로 충분한 암호화, 무결성 체크를 수행하고 이와 같은 메커니즘이 없는 경우엔 외부사용자에 의한 입력값을 신뢰해서는 안된다.

나. 보안대책

상태정보나 민감한 데이터 특히 사용자 세션정보와 같은 중요한 정보는 서버에 저장하고 보안확인 절차도 서버에서 실행한다. 보안설계관점에서 신뢰할 수 없는 입력값이 응용프로그램 내부로 들어올 수 있는 지점과 보안결정에 사용되는 입력값을 식별하고 제공되는 입력값에 의존할 필요가 없는 구조로 변경할 수 있는지 검토한다.

다. 코드예제

구입품목의 가격을 사용자 웹브라우저에서 처리하고 있어 이 값이 사용자에 의해 변경되는 경우 가격(단가)정보가 의도하지 않은 값으로 할당될 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: 〈input type="hidden" name="price" value="1000"/〉
2: 〈br/〉품목 : HDTV
3: 〈br/〉수량 : 〈input type="hidden" name="quantity" /〉개
4: 〈br/〉〈input type="submit" value="구입" /〉
5: .....
6: try {
   // 서버가 보유하고 있는 가격(단가) 정보를 사용자 화면에서 받아서 처리
```

안전하지 않은 코드의 예 JAVA

```
7:      price = request.getParameter("price");
8:      quantity = request.getParameter("quantity");
9:      total = Integer.parseInt(quantity) * Float.parseFloat(price);
10:} catch (Exception e) {
11:.....
```

사용자 권한, 인증 여부 등 보안결정에 사용하는 값은 사용자 입력값을 사용하지 않고 서버 내부의 값을 활용한다. 또한 사용자 입력에 의존해야 하는 값을 제외하고는 반드시 서버가 보유하고 있는 정보를 이용하여 처리한다.

안전한 코드의 예 JAVA

```
1: <input type="hidden" name="price" value="1000"/>
2: <br/>품목 : HDTV
3: <br/>수량 : <input type="hidden" name="quantity" />개
4: <br/><input type="submit" value="구입" />
5: .....
6: try {
7:      item = request.getParameter("item");
    // 가격이 아니라 item 항목을 가져와서 서버가 보유하고 있는 가격 정보를
    // 이용하여 전체 가격을 계산
8:      price = productService.getPrice(item);
9:      quantity = request.getParameter("quantity");
10:     total = Integer.parseInt(quantity) * price;
11:} catch (Exception e) {
12:   .....
13:}
14:   .....
```

평문으로 사용자의 인증정보를 쿠키에 저장하고 있는 C# 예제코드이다.

안전하지 않은 코드의 예 C#

```
1: HttpCookie cookie = new HttpCookie("Authentificated", "1");
//평문으로 사용자의 인증정보를 쿠키에 저장한다.
2: Response.Cookies.Add(cookie);
```

중요한 정보를 쿠키에 저장 시에는 암호화해서 사용하고, 되도록이면 해당 정보는 서버의 세션에 저장하도록 한다.

안전한 코드의 예 C#

```
// 사용자의 인증정보를 세션에 저장한다.
1: Session["Authentificated"] = "1";
```

아래 C 코드는 외부에서 가져온 서버 정보를 기반으로 연결을 진행한다. 사용자가 환경 변수를 조작하면 의도하지 않은 곳으로 연결을 진행할 수 있다. 예를 들어 온라인으로 제품 라이선스를 검증하는 경우, 인증 서버의 주소를 사용자가 변경하여 임의로 라이선스 검증을 통과할 수 있다.

안전하지 않은 코드의 예 C

```
1: void SecurityDecision() {
2:   int sockfd = socket(PF_INET, SOCK_STREAM, 0);
3:   char* server_info = getenv("server_addr");
4:   // 외부에서 가져온 서버 정보를 그대로 사용한다.
5:   if( connect( sockfd, (struct sockaddr *)server_addr, sizeof(struct socketaddr) ) < 0 ) {
6: return;
7:    }
   /* 라이선스 검증 코드 */
8: }
```

인증 서버의 정보를 환경 변수가 아닌 고정된 정보를 이용하여 연결을 진행한다.

안전한 코드의 예 C

```
SecurityDecision() {
2:   int sockfd = socket(PF_INET, SOCK_STREAM, 0);
3:   struct sockaddr_in server_addr;
4:   memset( &server_ info, 0, sizeof(server_info));
5:   server_info.sin_family = AF_INET;
6:   server_info.sin_port = htons(5555);
7:   server_info.sin_addr.s_addr = inet_addr("127.0.0.1");
   // 고정된 서버 주소를 사용하여 연결을 진행한다.
8:    if( connect( sockfd, (struct sockaddr *)server_addr, sizeof(struct socketaddr) ) < 0 ) {
9:         return;
10:  }
   /* 라이선스 검증 코드 */
11:}
```

라. 진단방법

인증여부를 확인하기 위해 사용하는 변수를 확인하고(①), 변수가 세션정보 등 서버내부에서 검증된 값인지 확인한다(②). 인증결정의 기준으로 외부 입력값을 그대로 사용하는 경우 취약하다.

일반적인 진단의 예

```
1:  …
2:  Cookie[] cookies = request.getCookies() ;  ····· ②
3:  for(int i=0 ; i<cookies.length ; i++) {
4:    Cookie c = cookies[i] ;
5:    if(c.getName().equals("authecticated")&&Boolean.TRUE.equals(c.getValue())){  ····· ①
6:    authenticated = true ;
7:    }
8:  }
9:  …
```

다음의 예제에서는 평문으로 사용자의 인증정보 및 “authenticated”를 쿠키에 저장하고 있다. 공격 자는 쿠키정보를 변경 가능하기 때문에 중요한 정보를 쿠키에 저장 시에는 암호화해서 사용하고, 가급적 해당정보는 WAS(Web Application Server) 서버의 세션에 저장한다.

정탐코드의 예

```
10: <%
11: String username = request.getParameter(“username”);
12: String password = request.getParameter(“password”);
13: if (username==nill || password==null || !isAuthenticatedUser(usename, password)) {
14:     throw new MyException(“인증 에러”);
15: }
16: Cookie userCookie = new    Cookie(“user”,username);17:
Cookie authCookie =
new Cookie(“authenticated”,”1”);18:
19: response.addCookie(userCookie);
20: response.addCookie(authCookie);
21: %>
```

다음의 예제에서는 HIDDEN 필드로 전송된 price를 가격으로 사용하지 않고 8번 라인에서 정해진 가격을 사용하고 있어 취약하지 않다.

오탐코드의 예

```
1:  <input type=”hidden” name=”price” value=”1000”/>
2:  <br/>품목 : HDTV
3:  <br/>수량 : <input type=”hidden” name=”quantity” />개
4:  <br/><input type=”submit” value=”구입” />
5:          .....
6:  try {
7:     // 서버가 보유하고 있는 정보를 이용하여 가격을 처리
8:     price = 1000;
9:     quantity = request.getParameter(“quantity”);
10:    total = quantity * Float.parseFloat(price);
11:}catch (Exception e) {
12:         .....
```

오탐코드의 예

```
13:}
14:          ......
```

마. 참고자료

[1] CWE-807 Reliance on Untrusted Inputs in a Security Decision, MITRE, http://cwe.mitre.org/data/definitions/807.html

[2] Do not trust the values of environment variables, CERT, http://www.securecoding.cert.org/confluence/display/java/ENV02-J.+Do+not+trust+the+values+of+environment+variables

[3] Session Management, OWASP https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

16. 메모리 버퍼 오버플로우

가. 개요

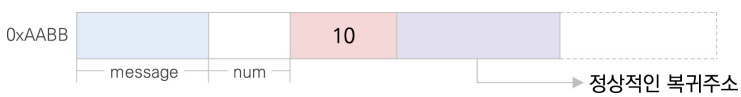
메모리 버퍼 오버플로우 보안약점은 연속된 메모리 공간을 사용하는 프로그램에서 할당된 메모리의 범위를 넘어선 위치에 자료를 읽거나 쓰려고 할 때 발생한다. 메모리 버퍼 오버플로우는 프로그램의 오동작을 유발시키거나, 악의적인 코드를 실행시킴으로써 공격자 프로그램을 통제할 수 있는 권한을 획득하게 한다.

메모리 버퍼 오버플로우에는 스택 메모리 버퍼 오버플로우와 힙 메모리 버퍼 오버플로우가 있다. 다음은 스택 메모리 버퍼 오버플로우를 발생시키는 코드이다.

```
1:  void foo(){
2:      int num = 10;
3:      char message[40];
4:      gets(message);
5:  }
```

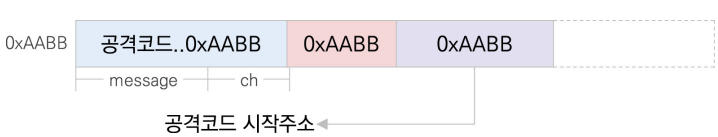
정상적인 프로그램의 실행은 다음과 같은 메모리 구조를 가지며, 함수 foo()의 스택 공간 끝에는 복귀주소가 보관된다.

[정상 수행인 경우]



gets()와 같은 함수는 문자열을 크기와 상관없이 연속된 기억공간에 저장시키는 함수이므로, 공격자는 정상적인 문자열 대신 공격코드를 입력하고 스택의 시작주소 0xAABB를 반복 입력한다. 이 경우 다음과 같은 메모리 구조에 의해 프로그램은 정상 주소로 복귀하는 대신 공격코드의 시작주소로 복귀하여 공격코드를 수행하게 된다.

[버퍼 오버플로우 공격인 경우]



나. 보안대책

프로그램 상에서 메모리 버퍼를 사용할 경우 적절한 버퍼의 크기를 설정하고, 설정된 범위의 메모리 내에서 올바르게 읽거나 쓸 수 있게 통제하여야 한다. 특히, 문자열 저장시 널(Null) 문자로 종료하지 않으면 의도하지 않은 결과를 가져오게 되므로 널(Null) 문자를 버퍼 범위내에 삽입하여 널(Null) 문자로 종료되도록 해야 한다.

다. 코드예제

다음 코드는 포인터 구조체의 개별 필드에 특정 문자열을 복사하는 프로그램이다. 잘못 계산된 데이터 크기 sizeof(cv_struct)로 인해 프로그램은 연속된 메모리 공간인 포인터 y를 덮어쓰는 버퍼 오버플로우를 발생시킨다. 또한 프로그램은 복사된 문자열에 대해 종료 문자를 첨가시키지 않았기 때문 에 문자열의 참조시 잘못된 결과를 가져올 수 있다.

안전하지 않은 코드의 예 C

```
1: typedef struct _charvoid {
2:     char x[16];
3:     void * y;
4:     void * z;
5: } charvoid
6: void badCode() {
7:     charvoid cv_struct
8:     cv_struct.y = (void *) SRC_STR;
9:     printLine((char *) cv_struct.y);

 /* sizeof(cv_struct)의 사용으로 포인터 y에 덮어쓰기 발생 */
10:    memcpy(cv_struct.x, SRC_STR, sizeof(cv_struct));
11:    printLine((char *) cv_struct.x);
12:    printLine((char *) cv_struct.y);
13:}
```

안전한 코드가 되기 위해서는 첫째, 문자열 복사는 구조체 내의 필드값 x에 한정되는 것이므로 정확한 문자열 계산인 sizeof(cv_struct.x)으로 허용된 범위의 인덱스만을 사용하도록 수정한다. 둘째, 복사된 문자열은 올바른 널(Null) 정보를 가져야 하므로 복사된 값을 가진 cv_struct.x 배열의 가장 마지막 인덱스를 계산하여 널(Null) 문자를 패딩해야 한다.

안전한 코드의 예 C

```
1: typedef struct _charvoid {
2:     char x[16];
3:     void * y;
4:     void * z;
5: } charvoid

6: static void goodCode() {
7:     charvoid cv_struct
8:     cv_struct.y = (void *) SRC_STR;
9:     printLine((char *) cv_struct.y);

 /* sizeof(cv_struct.x)로 변경하여 포인터 y의 덮어쓰기를 방지함 */
10:    memcpy(cv_struct.x, SRC_STR, sizeof(cv_struct.x));

 /* 문자열 종료를 위해 널 문자를 삽입함 */
11:    cv_struct.x[(sizeof(cv_struct.x)/sizeof(char))-1] = '\0';
12:    printLine((char *) cv_struct.x);
13:    printLine((char *) cv_struct.y);
14:}
```

라. 진단방법

버퍼에 값을 기록하는 경우, 값의 크기가 대상 버퍼보다 작은지 확인한다. 버퍼의 크기나 데이터의 크기가 외부 입력 값에 의해 결정되는 경우 입력 값의 크기가 대상 데이터를 충분히 포함할 수 있는지 확인한다.

버퍼의 크기를 비교하거나 인덱싱으로 접근할 경우에는 데이터의 크기 비교 외에도 음수값이 포함 되지 않도록 0보다 큰지 반드시 확인한다. 메모리 버퍼에 접근할 때 상수로 바로 접근하는 경우 해당 상수값을 확인해야하며, 상수를 이용하는 것보다 버퍼의 범위를 고려하여 접근을 하도록 코드의 수정이 이루어져야한다. 특히, 반복문으로 버퍼에 접근 할 때 반드시 경계값에 대한 확인이 필요하다.

또한 문자열을 처리하기 위해 버퍼를 사용할 경우 문자열의 마지막에 널(Null) 문자가 포함되는지 반드시 확인한다.

다음 예제는 매개변수로 입력받은 값을 버퍼에 복사하는 코드이다. 버퍼의 크기는 16 byte로 한정되어 있기 때문에 매개변수의 크기가 이보다 클 경우 버퍼의 영역을 벗어나 데이터가 기록되지만, 입력값의 크기에 대한 어떠한 검사도 이루어 지지 않기 때문에 아래와 같은 코드는 보안약점이 존재하는 코드로 진단할 수 있다.

정탐코드의 예

```
1: void foo(char* string){
2:   char buf[16];
3:   strcpy(buf, string);
4:   …
5: }
```

다음 예제는 외부 입력값을 호스트 이름으로 사용하는 코드로 hostname의 값을 64 byte로 한정 하여 설정하였다. 하지만 외부 입력값을 호스트 이름으로 사용하고 있기 때문에 이름값이 꼭 64byte 보다 작음을 보장 할 수 없으며, 공격자가 매우 긴 호스트 이름값을 입력하는 경우 버퍼 오버플로우 공격이 가능해진다.

정탐코드의 예

```
1: void host_lookup(char *user_supplied_addr)
2: {
3:   struct hostent *hp;
4:   in_addr_t *addr;
5:   char hostname[64];
6:   in_addr_t inet_addr(const char *cp);
7:   validate_addr_form(user_supplied_addr);
```

정탐코드의 예

```
8:    addr = inet_addr(user_supplied_addr);
9:    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
10:   strcpy(hostname, hp->h_name);
11:}
```

다음 예제는 버퍼에 메시지를 저장하고 메시지의 뒷부분 공백을 제거하는 함수이다. 먼저 4번 라인에서 버퍼의 생성은 널(Null)문자 저장을 위해 대상 데이터 보다 큰 공간을 할당하고 있으며, 9~11 라인으로 인덱스 값을 버퍼의 범위에 맞추어 변경해가면서 값을 할당하고 있다. 12번 라인의 경우 널(Null) 문자를 할당하여 올바른 방식으로 코딩이 이루어 졌다. 그러나 16~19번 라인의 공백문자를 제거하기 위한 코드에서 버퍼 인덱스 len의 값이 반복문 안에서 감소되고 있지만 0보다 작아질 경우 를 검사하고 있지 않다. 따라서 루프의 조건값에 따라 len 값이 0보다 작아질 수 있으며, 아래의 예제는 버퍼의 범위를 벗어난 값을 참조하게 되므로 보안약점이 존재하는 코드로 진단할 수 있다.

정탐코드의 예

```
1:    char* trimTrailingWhitespace(char *strMessage, int length)
2:    {
3:      char *retMessage;
4:      char *message = malloc(sizeof(char)*(length+1));
5:
6:      // copy input string to a temporary string
7:      char message[length+1];
8:      int index;
9:      for (index = 0; index < length; index++) {
10:       message[index] = strMessage[index];
11:     }
12:     message[index] = '\0';
13:
14:     // trim trailing whitespace
15:     int len = index-1;
16:     while (isspace(message[len])) {
17:       message[len] = '\0';
18:       len--;
19:     }
```

정탐코드의 예

```
20:
21:    // return string without trailing whitespace
22:    retMessage = message;
23:    return retMessage;
24:  }
```

마. 참고자료

[1] CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer, MITRE,
http://cwe.mitre.org/data/definitions/119.html

[2] Buffer overflow attack, OWASP,
https://www.owasp.org/index.php/Buffer_overflow_attack

17. 포맷 스트링 삽입

가. 개요

외부로부터 입력된 값을 검증하지 않고 입·출력 함수의 포맷 문자열로 그대로 사용하는 경우 발생 할 수 있는 보안약점이다. 공격자는 포맷 문자열을 이용하여 취약한 프로세스를 공격하거나 메모리 내용을 읽거나 쓸 수 있다. 그 결과, 공격자는 취약한 프로세스의 권한을 취득하여 임의의 코드를 실행할 수 있다.

나. 보안대책

printf(), snprintf() 등 포맷 문자열을 사용하는 함수를 사용할 때는 사용자 입력값을 직접적으로 포맷 문자열로 사용하거나 포맷 문자열 생성에 포함시키지 않는다. 포맷 문자열을 사용하는 함수에 사용자 입력값을 사용할 때는 사용자가 포맷 스트링을 변경할 수 있는 구조로 쓰지 않는다. 특히, %n, %hn은 공격자가 이를 이용해 특정 메모리 위치에 특정값을 변경할 수 있으므로 포맷 스트링 매개변수로 사용하지 않는다. 사용자 입력값을 포맷 문자열을 사용하는 함수에 사용할 때는 가능하면 %s 포맷 문자열을 지정하고, 사용자 입력값은 2번째 이후의 파라미터로 사용한다.

다. 코드예제

포맷 스트링 보안약점은 C 언어에 국한된 것은 아니다. 아래 예제 코드는 입력 자료의 유효성을 검증하지 않은 Java 프로그램에서도 발생할 수 있음을 보여준다. 이 프로그램에서 공격자는 %1$tm, %1$te, 또는 %1$tY과 같은 문자열을 입력하여 포맷 문자열에 포함시킴으로써, 실제 유효기간 valid- Date가 출력되도록 할 수 있다.

안전하지 않은 코드의 예 JAVA

```
// 외부 입력값에 포맷 문자열 포함 여부를 확인하지 않고 포맷 문자열 출력에 값으로 사용
// args[0]의 값으로 "%1$tY-%1$tm-%1$te"를 전달하면 시스템에서 가지고 있는 날짜(2014-10-14) 정보가 노출
1: import java.util.Calendar
2: .....
3: public static void main(String[] args) {
4:     Calendar validDate = Calendar.getInstance();
```

안전하지 않은 코드의 예 JAVA

```
5:    validDate.set(2014, Calendar.OCTOBER, 14);

6:    System.out.printf( args[0] + " did not match! HINT: It was issued on %1$terd of
  some month", validate);
7: }
```

사용자로부터 입력받은 문자열을 포맷 문자열에 직접 포함시키지 않고, %s 포맷 문자열을 사용함으로써 정보유출을 방지한다.

안전한 코드의 예 JAVA

```
// 외부 입력값이 포맷 문자열 출력에 사용되지 않도록 수정
1: import java.util.Calendar
2:       :
3: public static void main(String[] args) {
4:    Calendar validDate = Calendar.getInstance();
5:    validDate.set(2014, Calendar.OCTOBER, 14);
6:       System.out.printf("%s did not match! HINT: It was issued on %2$terd of some
  month", args[0], validate);
7: }
```

이 예제의 msg는 신뢰할 수 없는 사용자 입력을 포함하고 있고 fprintf() 호출에서 포맷 문자열 인자로 전달되기 때문에 포맷 스트링 삽입에 취약하다.

안전하지 않은 코드의 예 C

```
1: void incorrect_password(const char *user) {
2:    static const char msg_format[] = "%s cannot be authenticated.\n";
3:    size_t len = strlen(user) + sizeof(msg_format);
4:    char *msg = (char *)malloc(len);
5:    if (msg == NULL) {
```

안전하지 않은 코드의 예 C

```
    /* 오류 처리 */
6:   }
7:   int ret = snprintf(msg, len, msg_format, user);
8:     if (ret < 0 || ret >= len) {
    /* 오류 처리 */
9:   }
//
10:    fprintf(stderr, msg);
11:    free(msg);
12:   msg = NULL;
13:}
```

이 예제는 fprintf() 대신에 fputs()를 사용하여, msg를 포맷 문자열처럼 취급하지 않고 그대로 stderr로 출력한다.

안전한 코드의 예 C

```
1: void incorrect_password(const char *user) {
2:     static const char msg_format[] = "%s cannot be authenticated.\n";
3:     size_t len = strlen(user) + sizeof(msg_format);
4:     char *msg = (char *)malloc(len);
5:     if (msg == NULL) {
       /* 오류 처리 */
6:     }
7:     int ret = snprintf(msg, len, msg_format, user);
8:     if (ret < 0 || ret >= len) {
  /* 오류 처리 */
9:  }
//
10:  if (fputs(msg, stderr) == EOF) {
  /* 오류 처리 */
11: }
```

안전한 코드의 예 C

```
12:   free(msg);
13:   msg = NULL;
14:}
```

라. 진단방법

포맷 스트링을 이용하는 함수를 사용하는 경우 인자로 포맷 스트링이 포함되어 있는지 반드시 확인하고, 외부 입력 값이 포맷 스트링에 생성에 사용되는지 확인한다. 포맷 스트링이 함수의 인자로 포함 되어 있고, 외부 입력 값이 포맷 스트링 생성에 사용되지 않으며, 포맷 스트링에서 사용하는 인자의 개수와 매개변수로 포함된 인자의 개수가 일치하는 경우 안전하다고 판단한다. 또한 포맷에 출력할 데이터의 길이를 반드시 한정하도록 한다. 그 외에는 모두 취약하다고 판단한다.

다음 예제는 포맷 스트링이 인자로 존재하지만, 사용자 입력값을 포맷 스트링 생성에 사용하기 때문에 보안약점이 존재하는 코드로 진단할 수 있다. 공격자가 “%1$tm, %1$te, %1$tY”와 같은 입력을 하게 된다면, 예제코드는 문제를 일으키게 된다.

정탐코드의 예

```
1:  class Format {
2:    static Calendar c = new GregorianCalendar(1995, GregorianCalendar.MAY, 23);
3:    public static void main(String[] args) {
4:      System.out.printf(args[0] + " did not match! HINT: It was issued on %1$terd of
    some month", c);
5:   }
6: }
```

마. 참고자료

[1] CWE-134 Uncontrolled Format String, MITRE,
http://cwe.mitre.org/data/definitions/134.html

[2] Exclude user input from format strings, CERT,
http://www.securecoding.cert.org/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+stringsb

[3] Use valid format strings, CERT,
http://www.securecoding.cert.org/confluence/display/c/FIO47-C.+Use+valid+format+strings

[4] Format string attack, OWASP,
https://www.owasp.org/index.php/Format_string_attack

| 제 2 절 | 보안기능

보안기능(인증, 접근제어, 기밀성, 암호화, 권한관리 등)을 부적절하게 구현시 발생할 수 있는 보안약점으로 적절한 인증 없는 중요기능 허용, 부적절한 인가 등이 포함된다.

1. 적절한 인증 없는 중요기능 허용

가. 개요

적절한 인증과정이 없이 중요정보(계좌이체 정보, 개인정보 등)를 열람(또는 변경)할 때 발생하는 보안약점이다.

나. 보안대책

클라이언트의 보안검사를 우회하여 서버에 접근하지 못하도록 설계하고 중요한 정보가 있는 페이지는 재인증을 적용(은행 계좌이체 등)한다. 또한 안전하다고 검증된 라이브러리나 프레임워크(OpenSSL이나 ESAPI의 보안기능 등)를 사용하는 것이 중요하다.

다. 코드예제

회원정보 수정시 수정을 요청한 사용자와 로그인한 사용자의 일치 여부를 확인하지 않고 처리하고 있다.

안전하지 않은 코드의 예 JAVA

```
1: @RequestMapping(value = "/modify.do", method = RequestMethod.POST)
2: public ModelAndView memberModifyProcess(@ModelAttribute("MemberModel")
   MemberModel memberModel, BindingResult result, HttpServletRequest request,
   HttpSession session) {
3:    ModelAndView mav = new ModelAndView();
  //1. 로그인한 사용자를 불러온다.
4:    String userId = (String) session.getAttribute("userId");
```

안전하지 않은 코드의 예 JAVA

```
5:     String passwd = request.getParameter("oldUserPw");
6: ...
   //2. 실제 수정하는 사용자와 일치 여부를 확인하지 않고, 회원정보를 수정하여 안전하지 않다.
7:     if (service.modifyMember(memberModel)) {
8:         mav.setViewName("redirect:/board/list.do");
9:         session.setAttribute("userName", memberModel.getUserName());
10:        return mav;
11:    } else {
12:        mav.addObject("errCode", 2);
13:        mav.setViewName("/board/member_modify");
14:        return mav;
15:    }
16:}
```

로그인한 사용자와 요청한 사용자의 일치 여부를 확인한 후 회원정보를 수정하도록 한다.

안전한 코드의 예 JAVA

```
1: @RequestMapping(value = "/modify.do", method = RequestMethod.POST)
2: public ModelAndView memberModifyProcess(@ModelAttribute("MemberModel")
   MemberModel memberModel, BindingResult result, HttpServletRequest request,
   HttpSession session) {
3:    ModelAndView mav = new ModelAndView();

   //1. 로그인한 사용자를 불러온다.
4:    String userId = (String) session.getAttribute("userId");
5:    String passwd = request.getParameter("oldUserPw");

//2. 회원정보를 실제 수정하는 사용자와 로그인 사용자와 동일한지 확인한다.
6:    String requestUser = memberModel.getUserId();
7:    if (userId != null && requestUser != null && !userId.equals(requestUser)) {
8:        mav.addObject("errCode", 1);
```

안전한 코드의 예 JAVA

```
9:          mav.addObject("member",  memberModel);
10:         mav.setViewName("/board/member_modify");
11:         return  mav;
12:   }
13: ...
   //3. 동일한 경우에만 회원정보를 수정해야 안전하다.
14:   if  (service.modifyMember(memberModel)) {
...
```

사용자 자격인증 없이 로그인 기능을 수행하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
1: protected void LoginButton_Click(object sender, EventArgs e) {
// 사용자의 자격인증 과정이 없이 로그인 기능을 수행합니다.
2:    FormsAuthentication.RedirectFromLoginPage(UserName.Text,
RememberMe.Checked);
3: }
```

사용자의 자격인증 후 로그인 기능을 수행해야 한다.

안전한 코드의 예 C#

```
1: protected void LoginButton_Click(object sender, EventArgs e) {
// 사용자의 자격인증 과정을 수행합니다.
2:    if(Membership.ValidateUser(UserName.Text,  Password.Text)) {
      FormsAuthentication.RedirectFromLoginPage(UserName.Text,  RememberMe.Checked);
3:    }
4:}
```

라. 진단방법

해당 취약점은 보안특성 중 개인정보와 관련된 취약점으로 정적도구를 사용하여 이를 판단하는 것은 쉽지 않다. 이에 따라 진단원이 직접 코드를 보며 해당 정보가 중요정보인지 파악해야한다. 비밀번호, 개인정보(주민등록번호, 여권번호, 외국인 식별번호 등), 금융정보(카드번호, 계좌정보 등) 게시글 수정, 삭제 등을 포함하여 접근 및 사용이 제한되어야 하는 중요정보 및 기능을 정의하였는지 확인 하고 사용여부를 확인한다(①). 중요정보를 사용하는 경우 접근 변경 시 적절한 인증 여부를 확인하여 허용하는 기능이 구현되었는지 확인(②)한다. 만약 적절한 인증여부를 확인하는 경우 안전 하다고 판단하고, 인증여부를 확인하지 않는 경우 취약하다고 판단한다.

일반적인 진단의 예

```
1:  …
2:  @Controller
3:  @RequestMapping("bbs")
4:  public class BbsController {
5:  @Autowired
6:  private BoardDao boardDao;
7:  @RequestMapping("delete")
8:  public String delete(@Requestparam("contentid") int contentId, HttpServletRequest
    reques, Map model) {
9:    if(request.getSession().getAttribute("isLogin") == "logined") {  ……②
10:     boardDao.delete(contenId);  ……①
11:  }
12:  …
```

다음의 예제에서는 재인증을 거치지 않고 계좌이체를 하고 있으므로 취약하다고 판정한다.

정탐코드의 예

```
1: public void sendBankAccount(String accountNumber,double balance) {
2:  …
3:  BankAccount account = new BankAccount();
4:  account.setAccountNumber(accountNumber);
5:  account.setToPerson(toPerson);
6:  account.setBalance(balance);
7:  AccountManager.send(account);
8:  …
9: }
```

다음의 예제에서는 클라이언트 측에서 인증을 수행 하고 있어 우회가 가능하므로 취약하다고 판정 한다.

정탐코드의 예

```
1:  〈script  type='text/javascript'〉
2:    if (${command.allowedIp} == false) {
3:      document.write(""Check  Administrator  IP …"");
4:      alert(""권한이 없습니다."")
5:        history.back(-1);
6:      }
```

마. 참고자료

[1] CWE-306 Missing Authentication for Critical Function, MITRE, http://cwe.mitre.org/data/definitions/306.html

[2] Access Control, OWASP, https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

2. 부적절한 인가

가. 개요

프로그램이 모든 가능한 실행경로에 대해서 접근제어를 검사하지 않거나 불완전하게 검사하는 경우, 공격자는 접근 가능한 실행경로 정보를 유출할 수 있다.

나. 보안대책

응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격노출면[15](Attack Surface)을 최소화하고 사용자의 권한에 따른 ACL(Access Control List)을 관리한다. JAAS Authorization Framework나 OWASP ESAPI Access Control 등의 프레임워크를 사용해서 취약점을 제거할 수도 있다.

다. 코드예제

아래의 코드는 사용자 입력값에 따라 삭제작업을 수행하고 있으며, 사용자의 권한확인을 위한 별도의 통제가 적용되지 않고 있다.

안전하지 않은 코드의 예 JAVA

```
1: private BoardDao boardDao;
2: String action = request.getParameter("action");
3: String contentId = request.getParameter("contentId");
//요청을 하는 사용자의 delete 작원 권한 확인 없이 수행하고 있어 안전하지 않다.
4:    if (action != null && action.equals("delete")) {
5:        boardDao.delete(contentId);
6:    }
```

15) OSSTMM 3 Defines Attack Surface as “The lack of specific separations and functional controls that exist for that vector”

아래의 코드처럼 세션에 저장된 사용자 정보로 해당 사용자가 삭제작업을 수행할 권한이 있는지 확인한 뒤 권한이 있는 경우에만 수행하도록 해야 한다.

안전한 코드의 예 JAVA

```
1: private BoardDao boardDao;
2: String action = request.getParameter("action");
3: String contentId = request.getParameter("contentId");
   // 세션에 저장된 사용자 정보를 얻어온다.
4:    User user= (User)session.getAttribute("user");
   //사용자정보에서 해당 사용자가 delete작업의 권한이 있는지 확인한 뒤 삭제 작업을 수행한다.
5:    if (action != null && action.equals("delete") && checkAccessControlList(user,action)) {
6:        boardDao.delete(contenId);
7:    }
8: }
```

운영자 권한 검사 없이 컨트롤러와 내부의 개별액션에 접근이 가능한 C# 코드이다.

안전하지 않은 코드의 예 C#

```
// 운영자 권한 검사 없이 컨트롤러와 내부의 개별 액션에 접근 가능
1: public class AdministrationController : Controller
2: {
3:    ...
4: }
```

운영자 권한 검사 후에 개별 액션에 접근해야 한다.

안전한 코드의 예 C#

```
// 운영자 권한 검사 후 개별 액션에 접근
1: [Authorize(Roles = "Administrator")]
2: public class AdministrationController : Controller
3: {
```

안전한 코드의 예 C#

```
4:  …
5: }
```

사용자 자격인증 없이 LDAP 검색을 시도하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
1: #define FIND_DN "uid=han,ou=staff,dc=example,dc=com"
2: int searchData2LDAP(LDAP *ld, char *username) {
3:  unsigned long rc;
4:  char filter[20];
5:  LDAPMessage *result;
6:  snprintf(filter, sizeof(filter),"(name=%s)", username);
   // 사용자의 인증 없이 LDAP 검색을 시도합니다.
7:     rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0,
   NULL,NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
8:  return rc;
9: }
```

사용자의 자격인증 및 로그인 정보와 일치하는지 검사 후 LDAP 검색을 진행해야 한다.

안전한 코드의 예 C

```
1: #define FIND_DN "uid=han,ou=staff,dc=example,dc=com"
2:  int searchData2LDAP(LDAP *ld, char *username, char *password) {
3:  unsigned long rc;
4: char filter[20];
5:  LDAPMessage *result
   //username을 인증합니다.
6:  if ( ldap_simple_bind_s(ld, username, password) != LDAP_SUCCESS ) {
7:   printf("authorization error");
8:   return(FAIL);
```

안전한 코드의 예 C

```
9:  }
   // username 이 로그인 정보와 일치하는지 검사합니다.
10:  if ( strcmp(username,getLoginName()) != 0 ) {
11:    printf("Login error");
12:    return(FAIL);
13: }
14: snprintf(filter, sizeof(filter), "(name=%s)", username);
15:  rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0, NULL,
     NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
16:  return rc;
17: }
```

라. 진단방법

해당 취약점은 보안특성 중 외부 시스템(웹서버, DB서버, LADP 등)의 권한 설정과 관련된 취약점으로 정적도구를 사용하여 이를 판단하는 것은 쉽지 않다.

먼저 중요정보를 저장할 수 있는 외부시스템이 존재하는지 식별하고 해당시스템에 접근(열람) 및 변경(생성·삭제·수정 등)에 대한 권한을 사전에 적절하게 정의하였는지 확인한다. 해당 정보 또는 기능(접근·변경)을 호출하는 함수에서, 사전에 정의한 권한 소유여부를 검사하는지 확인(①)한다. 또한 접근제어를 서버측이 아닌 JavaScript 등과 같이 클라이언트측에서 제어 하여 우회가 가능한지 확인한다.

일반적인 진단의 예

```
1:  public void f(String sSingleId, int iFlag, String sServiceProvider, String sUid, String
    sPwd) {
2:  …
3:  env.put(Context.INITIAL_CONTEXT_FACTORY, CommonMySingleConst.INITCTX);
4:  env.put(Context.PROVIDER_URL, sServiceProvider);
5:  // 익명으로 LDAP 인증을 사용
6:  env.put(Context.SECURITY_AUTHENTICATION, “none”); ……①
7:  env.put(Context.SECURITY_PRINCIPAL, sUid);
```

일반적인 진단의 예

```
8:    env.put(Context.SECURITY_CREDENTIALS, sPwd);
9:    …
```

외부의 입력인 name 값이 필터가 아닌 동적인 LDAP 쿼리문 에서 사용자명으로 사용되었으며, 사용자 인증을 위한 별도의 접근제어 방법이 사용되지 않고 있다. 이는 anonymous binding을 허용 하는 것으로 볼 수 있다. 따라서 임의 사용자의 정보를 외부에서 접근할 수 있게 된다.

정탐코드의 예

```
1: public void f(String sSingleId, int iFlag, String sServiceProvider, String sUid, String
   sPwd) {
2: …
3: env.put(Context.INITIAL_CONTEXT_FACTORY, CommonMySingleConst.INITCTX);
4: env.put(Context.PROVIDER_URL, sServiceProvider);
5: // 익명으로 LDAP 인증을 사용
6: env.put(Context.SECURITY_AUTHENTICATION, “none”);
7: env.put(Context.SECURITY_PRINCIPAL, sUid);
8: env.put(Context.SECURITY_CREDENTIALS, sPwd);
9: …
10:}
```

아래의 코드처럼 세션에 저장된 사용자 정보로 해당 사용자가 삭제작업을 수행할 권한이 있는지 확인한 뒤 권한이 있는 경우에만 수행하도록 해야 한다.

오탐코드의 예

```
1: public String doSometing(HttpServletRequest request, HttpServletResponse response,
   HttpSession session) {
2:    String action= request.getParameter(“action”);
3:    // 세션에 저장된 사용자 정보를 얻어온다.
4:    User user= (User)session.getAttribute(“user”);
5:
6:    // 사용자정보에서 해당 사용자가 delete작업의 권한이 있는지 확인한 뒤 삭제 작업을 수행한다.
7:    if (action != null && action.equals(“delete”) && checkAccessControlList(user,action))
```

오탐코드의 예

```
    {
8:          // 삭제작업을 수행한다.
9:      }
10:}
```

마. 참고자료

[1] CWE-285 Improper Authorization, MITRE,
http://cwe.mitre.org/data/definitions/285.html

[2] Access Control, OWASP,
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

3. 중요한 자원에 대한 잘못된 권한 설정

가. 개요

SW가 중요한 보안관련 자원에 대하여 읽기 또는 수정하기 권한을 의도하지 않게 허가할 경우, 권한을 갖지 않은 사용자가 해당자원을 사용하게 된다.

나. 보안대책

설정파일, 실행파일, 라이브러리 등은 SW 관리자에 의해서만 읽고 쓰기가 가능하도록 설정하고 설정파일과 같이 중요한 자원을 사용하는 경우, 허가받지 않은 사용자가 중요한 자원에 접근 가능한지 검사한다.

다. 코드예제

"/home/setup/system.ini" 파일에 대해 모든 사용자가 읽고, 쓰고, 실행할 수 있도록 권한을 부여하고 있다.

- setExecutable(p1, p2) : 첫 번째 파라미터의 true/false 값에 따라 실행가능 여부를 결정한다. 두 번째 파라미터가 true일 경우 소유자만 실행권한을 가지며, false일 경우 모든 사용자가 실행 권한을 가진다.
- setReadable(p1, p2) : 첫 번째 파라미터의 true/false 값에 따라 읽기가능 여부를 결정한다. 두 번째 파라미터가 true일 경우 소유자만 읽기권한을 가지며, false일 경우 모든 사용자가 읽기 권한을 가진다.
- setWritable(p1, p2) : 첫 번째 파라미터의 true/false 값에 따라 쓰기가능 여부를 결정한다. 두 번째 파라미터가 true일 경우 소유자만 쓰기권한을 가지며, false일 경우 모든 사용자가 쓰기 권한을 가진다.

안전하지 않은 코드의 예 JAVA

```
1:    File file = new File("/home/setup/system.ini");
      //모든 사용자에게 실행 권한을 허용하여 안전하지 않다.
2:    file.setExecutable(true, false);
```

안전하지 않은 코드의 예 JAVA

```
        //모든 사용자에게 읽기 권한을 허용하여 안전하지 않다.
3:      file.setReadable(true, false);
        //모든 사용자에게 쓰기 권한을 허용하여 안전하지 않다.
4:      file.setWritable(true, false);
```

파일에 대해서는 최소권한을 할당해야 한다. 즉 해당 파일의 소유자에게만 읽기 권한을 부여한다.

- setExecutable(p1) : 파라미터의 true/false 값에 따라 소유자의 실행권한 여부를 결정한다.
- setReadable(p1) : 파라미터의 true/false 값에 따라 소유자의 읽기권한 여부를 결정한다.
- setWritable(p1) : 파라미터의 true/false 값에 따라 소유자의 쓰기권한 여부를 결정한다.

안전한 코드의 예 JAVA

```
1:      File file = new File("/home/setup/system.ini");
        //소유자에게 실행 권한을 금지하였다.
2:      file.setExecutable(false);
        //소유자에게 읽기 권한을 허용하였다.
3:      file.setReadable(true);
        //소유자에게 쓰기 권한을 금지하였다.
4:      file.setWritable(false);
```

아래의 C# 코드는 모든 사용자에게 파일의 권한을 부여하고 있다.

안전하지 않은 코드의 예 C#

```
1: public static void AddDirectorySecurity(string FileName)
2: {
        // 디렉토리 정보 객체 생성
3:      DirectoryInfo dInfo = new DirectoryInfo(FileName);
4:      DirectorySecurity dSecurity = dInfo.GetAccessControl();
```

안전하지 않은 코드의 예 C#

```
// 모든 사용자에게 권한 부여.
5:      dSecurity.AddAccessRule(new FileSystemAccessRule("everyone",
        FileSystemRights.FullControl,
        InheritanceFlags.ObjectInherit | InheritanceFlags.ContainerInherit,
        PropagationFlags.NoPropagateInherit, AccessControlType.Allow));

6:      dInfo.SetAccessControl(dSecurity);

7: }
```

적절한 권한을 파일에 설정해야 한다.

안전한 코드의 예 C#

```
1: public static void AddDirectorySecurity(string FileName, string Account,
   FileSystemRights Rights, AccessControlType ControlType)
2: {
      // 디렉토리 정보 객체 생성
3:    DirectoryInfo dInfo = new DirectoryInfo(FileName);
4:    DirectorySecurity dSecurity = dInfo.GetAccessControl();

      // FileSystemAccessRule 에 권한 설정
5:    dSecurity.AddAccessRule(new FileSystemAccessRule(Account,
                                                       Rights,
                                                       ControlType));
6:    dInfo.SetAccessControl(dSecurity);

6:

7:}
```

모든 사용자에게 권한을 부여하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
//모든 사용자가 읽기/쓰기 권한을 갖게 됩니다.
1: umask(0);
2: FILE *out = fopen("file_name", "w");
3: if(out){
4:   fprintf(out, "secure code\n");
5:   fclose(out);
```

umask()를 사용하여 권한 설정을 할 때, 올바른 권한을 설정해야 합니다.

안전한 코드의 예 C

```
//유저 외에는 아무런 권한을 주지 않습니다.
1: umask(077);
2: FILE *out = fopen("file_name", "w");
3: if(out){
4:   fprintf(out, "secure code\n");
5:   fclose(out);
```

라. 진단방법

해당 취약점은 보안특성 중 중요자원에 대한 접근 권한 설정과 관련된 취약점으로 정적도구를 사용하여 중요자원이 무엇인지 판단하는 것은 어려운 작업이다. 이에 따라 진단원이 직접 중요자원을 식별하고 이에 대한 취약성을 판단하는 것이 필요하다.

SW에서 생성하는 중요자원(파일 등)이 존재하는지 식별한다. 사용자 업로드 파일, 프로그램이 사용하는 설정파일 등 중요자원에 대해 읽기, 쓰기, 실행 등의 권한을 사전에 정의하였는지 확인하고 사전에 정의한 권한대로 중요자원에 접근권한을 허용하는지 확인한다. 설정파일, 문서파일의 경우 실행 권한이 설정되지 않았는지 확인하여야 한다.

SW가 중요한 보안관련 자원에 대하여 읽기 또는 수정하기 권한을 의도하지 않게 허가할 경우, 권한을 갖지 않은 사용자가 해당자원을 사용하게 된다.

일반적인 진단의 예

```
1:// 파일 권한 : rw-rw-rw-, 디렉토리 권한 : rwxrwxrwx
2:String cmd = "umask 0";
3:File file = new File("/home/report/report.txt");
4:…
5:Runtime.getRuntime().exec(cmd);
```

아래 코드는 4 ~ 6번 라인에서 실행권한과 쓰기권한은 제외하고 읽기 권한만 부여하고 있어 취약하지 않다.

오탐코드의 예

```
1:   try {
2:          ……
3:     File file = new File("/home/setup/system.ini");
4:     file.setExecutable(false);
5:     file.setReadable(true);
6:     file.setWritable(false);
7:
8:     if (file.createNewFile()) {
9:          System.out.println("File is created!");
10:    } else {
11:         System.out.println("File already exists.");
12:    }
13:} catch (IOException e) {
14:         ……
15:}
```

마. 참고자료

[1] CWE-732 Incorrect Permission Assignment for Critical Resource, MITRE, http://cwe.mitre.org/data/definitions/732.html

[2] Create files with appropriate access permissions, CERT, http://www.securecoding.cert.org/confluence/display/c/FIO06-C.+Create+files+with+appropriate+access+permissions

4. 취약한 암호화 알고리즘 사용

가. 개요

SW 개발자들은 환경설정 파일에 저장된 비밀번호를 보호하기 위하여 간단한 인코딩 함수를 이용하여 비밀번호를 감추는 방법을 사용하기도 한다. 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수로는 비밀번호를 제대로 보호할 수 없다.

정보보호 측면에서 취약하거나 위험한 암호화 알고리즘을 사용해서는 안 된다. 표준화되지 않은 암호화 알고리즘을 사용하는 것은 공격자가 알고리즘을 분석하여 무력화시킬 수 있는 가능성을 높일 수도 있다. 몇몇 오래된 암호화 알고리즘의 경우는 컴퓨터의 성능이 향상됨에 따라 취약해지기도 해서, 예전에는 해독하는데 오랜 시간이 걸릴 것이라고 예상되던 알고리즘이 며칠이나 몇 시간 내에 해독되기도 한다. RC2, RC4, RC5, RC6, MD4, MD5, SHA1, DES 알고리즘이 여기에 해당된다.

나. 보안대책

자신만의 암호화 알고리즘을 개발하는 것은 위험하며, 학계 및 업계에서 이미 검증된 표준화된 알고리즘을 사용한다. 기존에 취약하다고 알려진 DES, RC5 등의 암호알고리즘을 대신하여, 3DES, AES, SEED 등의 안전한 암호알고리즘으로 대체하여 사용한다. 또한, 업무관련 내용, 개인정보 등에 대한 암호 알고리즘 적용시, IT보안인증 사무국이 안전성을 확인한 검증필 암호모듈을 사용해야한다.

[참고 : 안전한 암호알고리즘 및 키 길이]

분류		보호함수 목록
최소 안전성 수준		112비트
블록암호		ARIA(키 길이 : 128/192/256), SEED(키 길이 : 128)
블록암호 운영모드	기밀성	ECD, CBC, CFB, OFB, CTR
	기밀성/인증	CCM, GCM
해시함수		SHA-224/256/384/512
메시지 인증코드	해시함수기반	HMAC
	블록기반	CMAC, GMAC
난수 발생기	해시함수 /HMAC 기반	HASH_DRBG, HMAC_DRBG
	블록기반	CTR_DRBG

분류	보호함수 목록
공개키 암호	RSAES - (공개키 길이) 2048, 3072 - RSA-OAEP에서 사용되는 해시함수 : SHA-224/256
전자서명	RSA-PSS, KCDSA, ECDSA, EC-KCDSA
키 설정 방식	DH, ECDH

보호함수		보호함수 목록
시스템 파라미터	RSA-PSS	(공개키 길이) 2048, 3072
	KCDSA, DH	(공개키 길이, 개인키 길이) (2048, 224), (2048, 256)
	ECDSA, EC-KCDSA, ECDH	(FIPS) B-233, B-283 (FIPS) K-233, K-283 (FIPS) P-224, P-256

[출처 : 암호알고리즘 검증기준 Ver 2.0(2012.3), 암호모듈시험기관]

다. 코드예제

다음 예제는 취약한 DES 알고리즘으로 암호화하고 있다.

안전하지 않은 코드의 예 JAVA

```
1: import java.security.*;
2: import javax.crypto.Cipher;
3: import javax.crypto.NoSuchPaddingException;

4: public class CryptoUtils {
5:     public byte[] encrypt(byte[] msg, Key k) {
6:         byte[] rslt = null;
7:         try {
//키 길이가 짧아 취약함 암호와 알고리즘인 DES를 사용하여 안전하지 않다.
8:             Cipher c = Cipher.getInstance("DES");
9:             c.init(Cipher.ENCRYPT_MODE, k);
10:             rslt = c.update(msg);
11:         }
```

아래 코드처럼 안전하다고 알려진 AES 알고리즘 등을 적용해야 한다.

안전한 코드의 예 JAVA

```
1: import java.security.*;
2: import javax.crypto.Cipher;
3: import javax.crypto.NoSuchPaddingException;

4: public class CryptoUtils {
5:     public byte[] encrypt(byte[] msg, Key k) {
6:         byte[] rslt = null;
7:         try {
          //키 길이가 길어 강력한 알고리즘인 AES를 사용하여 안전하다.
8:             Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
9:             c.init(Cipher.ENCRYPT_MODE, k);
10:             rslt = c.update(msg);
11:     }
```

다음 예제는 취약한 DES 알고리즘을 사용하고 있다.

안전하지 않은 코드의 예 C#

```
1: static string Enc(string input) {
//키 길이가 짧아 취약함 암호와 알고리즘인 DES를 사용하여 안전하지 않다.
2:     var des = new DESCryptoServiceProvider();
3:     ...
4: }
```

아래 코드처럼 안전하다고 알려진 AES 알고리즘 등을 적용해야 한다.

안전한 코드의 예 C#

```
1: static string Enc(string input) {
//키 길이가 길어 강력한 알고리즘인 AES를 사용하여 안전하다.
```

안전한 코드의 예 C#

```
2:    var des = new AesCryptoServiceProvider();
3:    ...
4:  }
```

다음 예제는 취약한 DES 알고리즘을 사용하고 있는 C 코드이다.

안전하지 않은 코드의 예 C

```
1: EVP_CIPHER_CTX ctx;
2: EVP_CIPHER_CTX_init(&ctx);
   // 취약한 DES 알고리즘을 사용한다.
3: EVP_EncryptInit(&ctx, EVP_des_ecb(), NULL, NULL);
```

안전하다고 알려진 AES 알고리즘을 사용하도록 한다.

안전한 코드의 예 C

```
1: EVP_CIPHER_CTX ctx;
2: EVP_CIPHER_CTX_init(&ctx);
   // 안전한 AES 알고리즘을 사용한다.
3: EVP_EncryptInit(&ctx, EVP_aes_128_cbc(), key, iv);
```

라. 진단방법

정적분석 도구 및 진단원이 각 언어에서 제공하는 암호화 함수가 취약한 암호화 알고리즘 사용 여부는 확인가능하나, 자체 암호화 알고리즘을 사용하는 경우 알고리즘의 적절성 여부를 판단하는 것은 어려우며, 수동진단이 필요하다.

자체 구현한 암호화 관련 알고리즘을 사용하는지 확인하고 자체 구현한 암호알고리즘을 사용할 경우 취약함으로 판단한다. 만약, 암호전문가가 존재할 경우 해당 암호함수의 암호알고리즘의 적절성을 판단하여 보증하는 것은 가능하다.

특정언어에서 제공하는 암호관련 함수를 호출하는지 식별한다.

- 취약한 알고리즘 : RC2, RC4, RC5, RC6, MD4, MD5, SHA1, DES 등
- 안전한 알고리즘 : SHA-256, AES, SEED, ARIA 등

일반적인 진단의 예

```
1: try {
2: MessageDigest md = MessageDigest.getInstance("SHA1");
3: md.update(msg.getBytes());
4: return md.digest();
5: } catch (NoSuchAlgorithmException e) {
```

비밀번호를 Base64로 인코딩하여 configuration 파일에 저장한 경우이다. Base64 인코딩 기법 자체가 가지는 취약점 때문에 이 경우 비밀번호를 안전하게 보호할 수 없다.

Base64 encoding의 경우는 암호화가 아닌 인코딩이다. 개발 시 Base64 encoding을 사용하는 경우는 암호화 목적이 아닌 가시성과 보관성을 위해서이다. 그러므로 Base64 encoding을 사용하였을 때 취약한 암호화 알고리즘이라 하기 어렵다. 하지만 개발자가 암호화를 위해 Base64 encoding을 사용하였다면 취약하다고 판정한다.

정탐코드의 예

```
1: public boolean DBConnect() {
2:   String url = "DBServer";
3:   String usr = "Scott";
4:   Connection con = null;
5:
6:   FileInputStream fis = null;
7:   try {
8:    Properties prop = new Properties();
9:    fis = new FileInputStream("config.properties");
10:
11:  // 비밀번호를 64bit로 decoding한다.
12:  byte password[] = Base64.decode(prop.getProperty("password"));
13:
```

정탐코드의 예

```
14:  // 유효성 점검없이 패스워드를 문자열로 읽는다.
15:  con = DriverManager.getConnection(url, usr, password.toString());
16: } catch (FileNotFoundException e) {
17:  …
18: } catch (SQLException e) {
19:  …
20: }
```

MD5 등의 낮은 보안 수준의 알고리즘을 사용하는 경우는 안전하지 않다.

정탐코드의 예

```
1:   public static EgovFormBasedUUID nameUUIDFromBytes(byte[] name) {
2:     MessageDigest md = null;
3:     try {
4:       md = MessageDigest.getInstance("MD5");
5:     } catch (NoSuchAlgorithmException e) {
6:       throw new InternalError("MD5 not supported");
7:     }
8:     byte[] md5Bytes = md.digest(name);
9:     md5Bytes[6] &= 0x0f; /* clear version */
10:    md5Bytes[6] |= 0x30; /* set to version 3 */
11:    md5Bytes[8] &= 0x3f; /* clear variant */
12: md5Bytes[8] |= 0x80; /* set to IETF variant */
13:return new EgovFormBasedUUID(md5Bytes);
14:}
```

마. 참고자료

[1] CWE-327 Use of a Broken or Riscky Cryptographic Algorithm, MITRE, http://cwe.mitre.org/data/definitions/327.html

[2] Do not use insecure or weak cryptographic algorithms, CERT, http://www.securecoding.cert.org/confluence/display/java/MSC61-J.+Do+not+use+insecure+or+weak+cryptographic+algorithms?focusedCommentId=186843241#comment-186843241

[3] Cryptanalysis, OWASP, https://www.owasp.org/index.php/Cryptanalysis

5. 암호화되지 않은 중요정보

가. 개요

많은 응용프로그램은 메모리나 디스크에서 중요한 정보(개인정보, 인증정보, 금융정보 등)를 처리한다. 이러한 중요정보가 제대로 보호되지 않을 경우, 보안이나 데이터의 무결성을 잃을 수 있다. 특히 사용자 또는 시스템의 중요정보가 포함된 데이터를 평문으로 송·수신 또는 저장할 때 인가되지 않은 사용자에게 민감한 정보가 노출될 수 있다.

나. 보안대책

개인정보(주민등록번호, 여권번호 등), 금융정보(카드번호, 계좌번호 등), 비밀번호 등 중요정보를 저장하거나 통신채널로 전송할 때는 반드시 암호화 과정을 거쳐야 하며 중요정보를 읽거나 쓸 경우에 권한인증 등으로 적합한 사용자가 중요정보에 접근하도록 해야 한다.

필요한 경우 SSL 또는 HTTPS 등과 같은 보안 채널을 사용해야 하며, HTTPS와 같은 보안 채널을 사용하거나 브라우저 쿠키에 중요 데이터를 저장하는 경우, 쿠키객체에 보안속성을 설정하여(Ex. setSecure(true)메소드 사용 등) 중요정보의 노출을 방지한다.

다. 코드예제

· **중요정보 평문저장**

아래 예제는 인증을 통과한 사용자의 비밀번호 정보가 평문으로 DB에 저장된다.

안전하지 않은 코드의 예 JAVA

```
1: String id = request.getParameter("id");
   // 외부값에 의해 비밀번호 정보를 얻고 있다.
2: String pwd = request.getParameter("pwd");
3: ......
4: String sql = " insert into customer(id, pwd, name, ssn, zipcode, addr)"
      + " values (?, ?, ?, ?, ?, ?)";
5: PreparedStatement stmt = con.prepareStatement(sql);
6: stmt.setString(1, id);
7: stmt.setString(2, pwd);
   ......
   // 입력받은 비밀번호가 평문으로 DB에 저장되어 안전하지 않다.
8: stmt.executeUpdate();
```

다음 예제는 비밀번호 등 중요 데이터를 해시값으로 변환하여 저장하고 있다.

안전한 코드의 예 JAVA

```
1: String id = request.getParameter("id");
   // 외부값에 의해 비밀번호 정보를 얻고 있다.
2: String pwd = request.getParameter("pwd");
// 비밀번호를 솔트값을 포함하여 SHA-256 해시로 변경하여 안전하게 저장한다.
3: MessageDigest md = MessageDigest.getInstance("SHA-256");
4: md.reset();
5: md.update(salt);
6: byte[] hashInBytes = md.digest(pwd.getBytes());
7: StringBuilder sb = new StringBuilder();
8:       for (byte b : hashInBytes) {
9:          sb.append(String.format("%02x", b));
10:     }
11:pwd = sb.toString();
12: ......
13:String sql = " insert into customer(id, pwd, name, ssn, zipcode, addr)“
   + " values (?, ?, ?, ?, ?, ?)";
14:PreparedStatement stmt = con.prepareStatement(sql);
15:stmt.setString(1, id);
16:stmt.setString(2, pwd);
17: ......
18:stmt.executeUpdate();
```

다음은 사용자의 비밀번호를 평문으로 저장해 놓고 출력하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
1: namespace Security
2: {
3:    public class FindPassword : System.Web.UI.Page
4:    {
5:       protected void Page_Load(object sender, EventArgs e)
```

안전하지 않은 코드의 예 C#

```
6:    {
7:       var userId = "tmp";
8:       MembershipUser user = Membership.GetUser(userId);
9:       if (user != null)
10:      {
11:         var password = user.GetPassword();
12:         Response.Write(password);
13:      }
14:      else
15:      {
16:         Response.Write("the given userId is not valid");
17:      }
18:   }
19: }
20:}
```

비밀번호는 암호화 하여 저장해야 하며, 출력하지 않도록 한다.

안전한 코드의 예 C#

```
1: namespace Security
2: {
3:    public class FindPassword : System.Web.UI.Page
4:    {
5:       protected void Page_Load(object sender, EventArgs e)
6:       {
7:          var userId = "tmp";
8:          MembershipUser user = Membership.GetUser(userId);
9:          if (user != null)
10:         {
11:            var encrypetedPassword = user.GetPassword();
12:            SecureFindPasswordFunction();
```

안전한 코드의 예 C#

```
13:         }
14:         else
15:         {
16:           Response.Write("the given userId is not valid");
17:         }
18:      }
19:   }
20:}
```

· **중요정보 평문전송**

아래 예제는 비밀번호를 암호화하지 않고 네트워크로 전송하고 있다. 이 경우 패킷 스니핑으로 비밀번호가 노출될 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: try {
2:    Socket s = new Socket("taranis", 4444);
3:    PrintWriter o = new PrintWriter(s.getOutputStream(), true);
  //비밀번호를 평문으로 전송하여 안전하지 않다.
4:    String password = getPassword();
5:    o.write(password);
6: } catch (FileNotFoundException e) {
7:    ……
```

아래 예제는 비밀번호를 네트워크로 서버로 전송하기 전에 AES 등의 안전한 암호알고리즘으로 암호화한 안전한 프로그램이다.

안전한 코드의 예 JAVA

```
// 비밀번호를 암호화 하여 전송
1: try {
2:     Socket s = new Socket("taranis", 4444);
3:     PrintStream o = new PrintStream(s.getOutputStream(), true);
//비밀번호를 강력한 AES암호화 알고리즘으로 전송하여 사용한다.
4:     Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");

5:     String password = getPassword();
6:     byte[] encPassword = c.update(password.getBytes());
7:   o.write(encPassword, 0, encPassword.length);
8: } catch (FileNotFoundException e) {
9:       ……
```

아래 C# 예제 또한 비밀번호를 암호화하지 않고 네트워크로 전송하고 있다. 이 경우 패킷 스니핑으로 비밀번호가 노출될 수 있다.

안전하지 않은 코드의 예 C#

```
1: public void EmailPassword_OnClick(object sender, EventArgs args)
2: {
3:   MembershipUser u = Membership.GetUser(UsernameTextBox.Text, false);
4:   String password;

5:   if (u != null)
6:   {
7:     try
8:     {
9:       password = u.GetPassword(); // sensitive data created
10:   }
11:   catch (Exception e)
```

안전하지 않은 코드의 예 C#

```
12:  {
13:            Msg.Text = "An exception occurred retrieving your password: " +
   Server.HtmlEncode(e.Message);
14:    return;
15:  }
16:  MailMessage Message = new MailMessage();
17:  Message.Body = "Your password is: " + Server.HtmlEncode(password);
     //비밀번호가 포함된 메시지를 네트워크로 전송하고 있다.
18:  SmtpMail.Send(Message);
19:  Msg.Text = "Password sent via e-mail.";
20: }
21: else
22: {
23:  Msg.Text = "User name is not valid. Please check the value and try again.";
24: }
25:}
```

비밀번호를 네트워크로 전송할 때에는 암호화 하는 것이 바람직하다.

안전한 코드의 예 C#

```
1: public void EmailPassword_OnClick(object sender, EventArgs args)
2: {
3:  MembershipUser u = Membership.GetUser(UsernameTextBox.Text, false);
4:  String password;

5:  if (u != null)
6:  {
7:   try
8:   {
9:    password = u.GetPassword();
10:   byte[] data = System.Text.Encoding.ASCII.GetBytes(password);
11:   data = new
```

안전한 코드의 예 C#

```
        System.Security.Cryptography.SHA256Managed().ComputeHash(data); String
        hashedPassword = System.Text.Encoding.ASCII.GetString(data);
12:   }
13:   catch (Exception e)
14:   {
15:                 Msg.Text = "An exception occurred retrieving your password: " +
   Server.HtmlEncode(e.Message);
16:     return;
17:   }
18:   MailMessage Message = new MailMessage();
19:   Message.Body ="Your password is: "+Server.HtmlEncode(hasedPassword);
20:   SmtpMail.Send(Message);
21:   Msg.Text = "Password sent via e-mail.";
22: }
23: else
24: {
25:   Msg.Text = "User name is not valid. Please check the value and try again.";
26: }
27:}
```

파일에서 읽어온 비밀번호를 바로 사용하는 C예제 코드이다.

안전하지 않은 코드의 예 C

```
1: int dbaccess(){
2:   FILE *fp; char *server = "DBserver";
3:   char passwd[20];
4:   char user[20];
5:   SQLHENV henv;
6:   SQLHDBC hdbc;
7:   fp = fopen("config", "r");
8:   fgets(user, sizeof(user), fp);
```

안전하지 않은 코드의 예 C

```
    // 비밀번호를 파일에서 읽어 옵니다.
9:  fgets(passwd, sizeof(passwd), fp);
10: fclose(fp);
11: SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
12: SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
13: SQLConnect(hdbc,
14:             (SQLCHAR*) server,
15:             (SQLSMALLINT) strlen(server),
16:             (SQLCHAR*) user,
17:             (SQLSMALLINT) strlen(user),
18:             //비밀번호의 암호화 없이 직접 연결합니다.
19:             (SQLCHAR*) passwd,
20:             (SQLSMALLINT) strlen(passwd) );
21: return 0;
```

외부에서 입력된 비밀번호는 검증의 과정을 거쳐서 사용해야 한다.

안전한 코드의 예 C

```
1: int dbaccess(){
2:   FILE *fp; char *server = "DBserver";
3:   char passwd[20];
4:   char user[20];
5:   char *encPasswd;
6:   char *key;
7:   SQLHENV henv;
8:   SQLHDBC hdbc;
   // AES-CBC로 암호화 모드를 설정합니다.
9:   HCkCrypt2 crypt = CkCrypt2_putCryptAlgorithm(crypt,"aes");
10:   CkCrypt2_putCipherMode(crypt,"cbc");
   // 외부에서 암호화 키를 불러와 설정합니다.
11:   key = getenv("encrypt_key");
12:   CkCrypt2_SetEncodedKey(crypt,key,"hex");
```

안전한 코드의 예 C

```
13: fp = fopen("config", "r");
14: fgets(user, sizeof(user), fp);
  // 비밀번호를 파일에서 읽어옵니다.
15: fgets(passwd, sizeof(passwd), fp);
16: fclose(fp);
  // 비밀번호 암호화를 진행합니다.
17: ncPasswd = CkCrypt2_encryptStringENC(crypt, passwd);
18: SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
19: SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
20: SQLConnect(hdbc,
21:                (SQLCHAR*) server,
22:                (SQLSMALLINT) strlen(server),
23:                (SQLCHAR*) user,
24:                (SQLSMALLINT) strlen(user),
25:                // 암호화된 비밀번호를 사용합니다.
26:                (SQLCHAR*) encPasswd,
27:                (SQLSMALLINT) strlen(verifiedPwd) );
28: return 0;
29:}
```

라. 진단방법

· 중요정보 평문저장

중요정보란 일반적으로 설계과정에서 결정되기 때문에, 중요정보의 평문저장을 일반적으로 검사할 수 있는 기법은 존재하지 않는다. 따라서 이러한 정보를 저장하고 사용하는 경우 반드시는 암호화 및 복호화 과정을 거쳐야 한다. 다만, 로그인이나 암호의 사용과 같은 특정한 경우 해당 메소드의 인자 값 추적으로 평문 유무를 판단할 수 있다.

중요정보라고 판단된 데이터 경우에는 불필요한 참조가 존재하지 않도록 제거해야하며, 임의의 변수에 임시 저장할 경우에도 암호화 유무를 확인해야하며 사용이 완료된 임시변수의 값은 반드시 초기화 해주어야 한다. 또한, 쿠키에 값을 저장하는 경우에는 중요정보를 저장하지 않거나 또는 저장을 하더라도 암호화로 저장해야 한다.

다음 예제는 소켓 연결이 수립되고 난 후, 전달 받은 암호 값을 password_buffer에 저장하고 이를 다시 파일에 직접 기록하는 코드이다. 이 때, 전달받은 값을 암호화하지 않고 저장하기 때문에 해당 파일에 접근할 수 있는 사용자라면, 암호 값을 획득할 수 있다.

정탐코드의 예

```
1: if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0)
2: error("Connecting");
3: …
4: while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
5: … // buffer값을 password_buffer에 추가
6: write(passFileD,password_buffer,n);
7: …
```

중요정보를 암호화 작업 없이 password_buffer에 그대로 유지하고 있으며, 해당 내용을 파일에 저장하기 때문에 위 예제 코드는 보안약점이 존재한다고 진단할 수 있으며, 임시 변수인 buffer를 사용완료하고 난 후 값의 초기화 여부 또한 확인해야 한다.

다음 예제는 데이터베이스에 접근하기 위한 정보를 설정파일로 작성한 경우이다. 다른 소스 파일에서 해당 파일을 포함(Include)하여 사용하지만, 설정파일에 데이터베이스를 접근하는 ID와 비밀번호가 평문으로 저장되어 있기 때문에 이러한 정보가 외부로 노출될 수 있는 위험성이 있다.

정탐코드의 예

```
1: …
2: 〈connectionStrings〉
3: 〈add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=-
   password; dbalias=uDB;" providerName="System.Data.Odbc" /〉
4: 〈/connectionStrings〉
5: …
```

· 중요정보 평문전송

해당 보안약점은 보안특정 중 평문전송과 관련된 내용으로 정적도구를 사용하여 중요정보의 기준을 판단하는 것은 어렵다.

개인정보(주민등록번호, 여권번호 등), 금융정보(카드·계좌번호), 비밀번호 등 민감한 정보를 다루는지 확인하고, 해당 정보가 네트워크 등으로 전송될 때 암호화 여부를 확인하고, 보안 채널을 이용하도록 한다. 민감한 정보가 있는 URL을 전송할 경우에도 마찬가지로 암호화 작업을 수행 하도록 한다. 또한 중요정보를 쿠키에 저장하여 전송할 경우 암호화하여 전송해야 하며, 특히 setSecure(true)와 같은 함수를 이용하여 암호화를 해야 한다. 이러한 절차가 생략되어 있는 경우 취약하다고 판단한다.

다음 예제는 외부에서 읽어 들인 비밀번호를 암호화하지 않고 네트워크로 서버에 전송하고 있다. 이 경우 패킷 스니핑으로 비밀번호가 노출될 수 있기 때문에 보안약점이 존재하는 코드이다. 또한 임시 변수 password는 값의 사용 후 반드시 초기화 해주어야 한다.

정탐코드의 예

```
1: void foo()
2: {
3:   try {
4:     Socket socket = new Socket("taranis", 4444);
5:     PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
6:     String password = getPassword();
7:     out.write(password);
8:   } catch (FileNotFoundException e) {
9: …
```

다음 예제는 중요정보에 접근 가능한 URL을 일반 채널을 이용하여 연결하는 예제이다. 따라서 해당 채널에서 주고받는 데이터는 외부에서 쉽게 확인이 가능하고 공격에 활용될 수 있다. 보안 채널을 이용하고 있지 않기 때문에 보안약점을 가지고 있는 코드라고 진단할 수 있다.

정탐코드의 예

```
1: try {
2:     URL u = new URL("http://www.secret.example.org/");
3:     HttpURLConnection hu = (HttpURLConnection) u.openConnection();
4:     hu.setRequestMethod("PUT");
5:     hu.connect();
6:     OutputStream os = hu.getOutputStream();
7:     hu.disconnect();
8: } catch (IOException e) {
9:     //…
10:}
```

마. 참고자료

[1] CWE-312, Cleartext Storage of Sensitive Information, MITRE, http://cwe.mitre.org/data/definitions/312.html

[2] Clear sensitive information stored in reusable resources, CERT, http://www.securecoding.cert.org/confluence/display/c/MEM03-C.+Clear+sensitive+information+stored+in+reusable+resources

[3] Be careful while handling sensitive data, such as passwords, in program code, CERT, http://www.securecoding.cert.org/confluence/display/c/MSC18-C.+Be+careful+while+handling+sensitive+data%2C+such+as+passwords%2C+in+program+code

[4] Never hard code sensitive information, CERT, http://www.securecoding.cert.org/confluence/display/java/MSC03-J.+Never+hard+code+sensitive+information

[5] Do not store unencrypted sensitive information on the client side, CERT, http://www.securecoding.cert.org/confluence/display/java/FIO52-J.+Do+not+store+unencrypted+sensitive+information+on+the+client+side

[6] Password Plaintext Storage, OWASP
https://www.owasp.org/index.php/Password_Plaintext_Storage

[7] CWE-319, Cleartext Transmission of Sensitive Information, MITRE,
http://cwe.mitre.org/data/definitions/319.html

[8] nsecure Transport, OWASP,
https://www.owasp.org/index.php/Insecure_Transport

6. 하드코드된 중요정보

가. 개요

프로그램 코드 내부에 하드코드된 비밀번호 또는 암호화키를 포함하여 내부 인증에 사용하거나 암호화를 수행하면 중요정보(관리자 정보, 암호화된 정보 등)가 유출될 수 있는 보안약점이다.

나. 보안대책

비밀번호는 암호화 하여 별도의 파일에 저장하여 사용한다. 또한 중요정보를 암호화하면, 상수가 아닌 암호화 키를 사용하도록 하며 소스코드 내부에 상수형태의 암호화 키를 저장해서 사용하지 않도록 한다.

다. 코드예제

· 하드코드된 비밀번호

데이터베이스 연결을 위한 비밀번호를 소스코드 내부에 상수 형태로 하드코딩 하는 경우, 접속 정보가 노출될 수 있어 위험하다.

안전하지 않은 코드의 예 JAVA

```
1: public class MemberDAO {
2:     private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
3:     private static final String URL = "jdbc:oracle:thin:@192.168.0.3:1521:ORCL";
4:     private static final String USER = "SCOTT"; // DB ID;
       //DB 비밀번호가 소스코드에 평문으로 저장되어 있다.
5:     private static final String PASS = "SCOTT"; // DB PW;
6: ……
7:     public Connection getConn() {
8:         Connection con = null;
9:         try {
10:            Class.forName(DRIVER);
11:            con = DriverManager.getConnection(URL, USER, PASS);
12:        ……
```

비밀번호는 안전한 암호화 방식으로 암호화하여 별도의 분리된 공간(파일)에 저장해야 하며, 암호화된 비밀번호를 사용하기 위해서는 복호화 과정을 거쳐야 한다.

안전한 코드의 예 JAVA

```
1: public class MemberDAO {
2:     private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
3:     private static final String URL = "jdbc:oracle:thin:@192.168.0.3:1521:ORCL";
4:     private static final String USER = "SCOTT"; // DB ID
5: ……
6:     public Connection getConn() {
7:         Connection con = null;
8:         try {
9:             Class.forName(DRIVER);
               //암호화된 비밀번호를 프로퍼티에서 읽어들여 복호화해서 사용해야한다.
10:            String PASS = props.getProperty("EncryptedPswd");
11:            byte[] decryptedPswd = cipher.doFinal(PASS.getBytes());
12:            PASS = new String(decryptedPswd);
13:            con = DriverManager.getConnection(URL, USER, PASS);
14:            ……
```

Credential 객체를 생성하기 위한 비밀번호를 소스코드 내부에 상수 형태로 하드코딩 하는 경우, 접속 정보가 노출될 수 있어 위험하다.

안전하지 않은 코드의 예 C#

```
1: string UserName = "username";
2: string Password = "password";
   // 평문으로 저장된 비밀번호를 이용하여 NetworkCredential 생성
3: NetworkCredential myCred = new NetworkCredential(UserName, Password);
```

암호화된 비밀번호로 Credential 객체를 생성한다.

안전한 코드의 예 C#

```
1: string UserName = "username";
2: string Password = "password";
3: SecureString SecurelyStoredPassword = new SecureString();

4: foreach (char c in Password)
5: {
6:     SecurelyStoredPassword.AppendChar(c);
7: }

    // 암호화된 비밀번호를 사용하여 NetworkCredential 생성
8:     NetworkCredential secure_myCred = new NetworkCredential(UserName,
       SecurelyStoredPassword);
```

하드코드된 비밀번호를 바로 사용하는 C예제 코드이다.

안전하지 않은 코드의 예 C

```
1: int dbaccess(char *server, char *user){
2:   SQLHENV henv;
3:   SQLHDBC hdbc;
4:   char *password = "password";
5:   SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
6:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
     // 하드코드된 비밀번호를 사용
7:   SQLConnect(hdbc,(SQLCHAR*)server,strlen(server),user,strlen(user),
   password, strlen(password));
8:   return 0;
```

비밀번호는 코드 상에서 보이지 않게 사용해야 한다. 아래의 예시는 환경 변수를 이용하여 비밀번호 를 사용한다.

안전한 코드의 예 C

```
1: int dbaccess(char *server, char *user, char *passwd){
2:   SQLHENV henv;
3:   SQLHDBC hdbc;
4:   // 비밀번호를 외부에서 불러와서 사용
5:   char *password = getenv("password");
6:   SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
7:   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
8:   SQLConnect(hdbc, (SQLCHAR*) server, strlen(server), user, strlen(user),
  password, strlen(password));
9:   SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
10: SQLFreeHandle(SQL_HANDLE_ENV, henv);
11: return 0;
12:}
```

· **하드코드된 암호화 키**

소스 코드 내부에 암호화 키를 상수 형태로 하드코딩하여 사용하면 악의적인 공격자에게 암호화 키가 노출될 위협이 있다.

안전하지 않은 코드의 예 JAVA

```
1: import javax.crypto.KeyGenerator;
2: import javax.crypto.spec.SecretKeySpec;
3: import javax.crypto.Cipher;
4:    ……
5: public String encriptString(String usr) {
   //암호화 키를 소스코드 내부에 사용하는 것은 안전하지 않다.
6:    String key = "22df3023sf~2;asn!@#/>as";
7:    if (key != null) {
8:        byte[] bToEncrypt = usr.getBytes("UTF-8");
9:        SecretKeySpec sKeySpec = new SecretKeySpec(key.getBytes(), "AES");
```

암호화 과정에 사용하는 암호화 키는 외부 공간(파일)에 안전한 방식으로 암호화하여 보관해야 하며, 암호화된 암호화 키는 복호화하여 사용한다.

안전한 코드의 예 JAVA

```
1: import javax.crypto.KeyGenerator;
2: import javax.crypto.spec.SecretKeySpec;
3: import javax.crypto.Cipher;
4:  ……
5: public String encriptString(String usr) {
   //암호화 키는 외부 파일에서 암호화 된 형태로 저장하고, 사용시 복호화 한다.
6:    String key = getPassword("./password.ini");
7:    key = decrypt(key);
8:    if (key != null) {
9:        byte[] bToEncrypt = usr.getBytes("UTF-8");
10:       SecretKeySpec sKeySpec = new SecretKeySpec(key.getBytes(), "AES");
```

소스코드 내부에 암호화 키를 상수 형태로 하드코딩하여 사용하면 악의적인 공격자에게 암호화 키가 노출될 위협이 있다.

안전하지 않은 코드의 예 C#

```
//암호화 키를 소스코드 내부에 사용하는 것은 안전하지 않다.
1: byte[] key = new byte[] { 0x43, 0x87, 0x23, 0x72 };
2: byte[] iv = new byte[] { 0x43, 0x87, 0x23, 0x72 };
3: FileStream fStream = File.Open(fileName, FileMode.OpenOrCreate);

4: CryptoStream cStream = new CryptoStream(fStream,
   new TripleDESCryptoServiceProvider().CreateEncryptor(key, iv),
   CryptoStreamMode.Write);
```

암호화 과정에 사용하는 암호화 키는 외부 공간(파일)에 안전한 방식으로 암호화하여 보관해야 하며, 암호화된 암호화 키는 복호화하여 사용한다.

안전한 코드의 예 C#

```
//암호화 키는 외부 파일에서 암호화 된 형태로 저장하고, 사용시 복호화 한다.

1:  byte[] key = GetKey(./password.ini);
2:  byte[] iv = GetIV(./password.ini);
3:  FileStream fStream = File.Open(fileName, FileMode.OpenOrCreate);

4:  CryptoStream cStream = new CryptoStream(fStream, new
    TripleDESCryptoServiceProvider().CreateEncryptor(Decrypt(key),
    Decrypt(iv)),
     CryptoStreamMode.Write);
```

하드코드된 비밀번호를 사용할 경우, 코드에 접근 권한이 있는 사용자가 비밀번호를 알 수 있습니다.

안전하지 않은 코드의 예 C

```
1: typedef int SQLSMALLINT;
2: int dbaccess(char *user, char *passwd){
3:  char *server = "DBserver";
4:  char *cpasswd;
5:  SQLHENV henv;
6:  SQLHDBC hdbc;
7:  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
8:  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
  //암호화된 비밀번호와 솔트를 사용합니다. 코드에 접근 권한이 있는 사용자는 해당 비밀번호와 솔트를
    획득할 수 있습니다.
9:  cpasswd = crypt(passwd, "salt");
10: if (strcmp(cpasswd, "68af404b513073582b6c63e6b") != 0) {
11:  // USE_OF_HARDCODED_CRYPTOGRAPHIC_KEY
12:  printf("Incorrect password\n");
13:  return -1;
```

안전하지 않은 코드의 예 C

```
14:   }
15: }
```

암호화되어 저장된 암호를 외부에서 불러오고 이를 비교하는 코드가 작성되어야 한다.

안전한 코드의 예 C

```
1: extern char *salt;
2: typedef int SQLSMALLINT;
3: int dbaccess(char *user, char *passwd){
4:   char *server = "DBserver";
5:   char *cpasswd;
     // 외부에 있는 암호화된 비밀번호와 솔트를 불러옵니다.
6:   char* storedpasswd = getenv("password");
7:   char* salt = getenv("salt");
8:   SQLHENV henv;
9:   SQLHDBC hdbc;
10: SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
11: SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    // 외부에서 불러온 솔트 값을 사용해 비밀번호를 암호화합니다.
12: cpasswd = crypt(passwd, salt);
  // 암호화된 비밀번호와 외부에서 불러온 값을 비교합니다.
13: if (strcmp(cpasswd, storedpasswd) != 0){
14:   printf("Incorrect password\n");
15:   SQLFreeHandle(SQL_HANDLE_DBC, &hdbc);
16:   SQLFreeHandle(SQL_HANDLE_ENV, &henv);
17:   return -1;
18: }
19:}
```

라. 진단방법

· 하드코드된 비밀번호

사용자·관리자 로그인페이지(식별·인증)를 요청하는 모듈·함수를 확인하여 사용자·관리자 비밀번호가 소스코드 안에 직접 코딩되어 있는지 확인하고 내·외부 서버(업무서버, DB 등)에 접속을 관리하는 모듈 · 함수를 확인하여 서버 접속 비밀번호가 소스코드 안에 직접 코딩되어 있는지 확인한다.

비밀번호를 별도의 파일에 저장하여 사용하지 않고 소스코드 안에 하드코딩 되어 있는 경우엔 취약하다고 판정한다.

다음의 예제에서 데이터베이스 접속 정보인 비밀번호를 4번 라인에서 입력하고 있다.

일반적인 진단의 예

```
1:  private static final String JDBC_DRIVER = "org.gjt.mm.mysql.Driver";
2:  private static final String JDBC_URL = "jdbc:mysql://192.168.200.24:1621/com";
3:  private static final String JDBC_USER = "com";
4:  private static final String JDBC_PASSWORD = "com01";
5:  ...
6:  public Connection DBConnect(String url, String id) {
7:    try {
8:      conn = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);
9:    } catch (SQLException +
      e)
      { System.err.println("
      ...");
10:  }
11:  return conn;
12: }
```

· 하드코드된 암호화 키

DB접속 등 비밀번호 사용이 필요한 로직을 확인하고(①), 사용된 비밀번호의 암호화 여부 확인한다(②). 암호화된 비밀번호가 소스내에 존재하는지 여부 확인한다(③). 소스코드내에 암호화된 비밀번호를 저장하는 경우 취약하다.

일반적인 진단의 예

```
1:  …public Connection DBConnect(String url, String usr) {
2:    String password = "68af404b513073584c4b6f22b6c63e6b";  ……②
3:    try {
4:     // 비밀번호로 상수를 사용하고있다.
5:     con = DriverManager.getConnection(url, usr, password);  ……①
6:    } catch (SQLException e) {
7:     System.err.println("…");
8:    }
9:    return con;
10: }
11: …
```

암호화를 하는데 사용되는 암호화 키를 소스에 하드코딩해서 사용하는 경우 취약하다.

정탐코드의 예

```
1: public class U321 {
2:   public void foo(String url, String usr) {
3:    try {
4:     String encryptedPwd = getpassword();
5:     // private key를 상수로 정의
6:    byte[] privateKey = { '6', '8', 'a', 'f', '4', '0', '4', 'b', '5', '1', '3', '0', '7' };
7:    javax.crypto.Cipher cipher = javax.crypto.Cipher.getInstance("RSA");
8:    // 상수 key를 사용하여 보안키 생성
9:    javax.crypto.SecretKey myDesKey = new javax.crypto.spec.SecretKeySpec(privateKey,
10:   "DES");
11:   cipher.init(javax.crypto.Cipher.DECRYPT_MODE, myDesKey);
12:   // 암호화된 password를 복호화
13:   byte[] plainTextPwdBytes = cipher.doFinal(encryptedPwd.getBytes());
14:
15:   // DB 연결
16:   java.sql.DriverManager.getConnection(url, usr, new String(plainTextPwdBytes));
17:   } catch (SQLException e) {
18:   …
```

암호화를 위한 암호화 키가 아닌 알고리즘명을 지정하는 경우 취약하지 않다.

오탐코드의 예

```
1:  // 키를 설정한다.
2:  sKey = new SecretKeySpec( key, "DESede" );
```

마. 참고자료

[1] CWE-259 Use of Hard-coded Password, MITRE, http://cwe.mitre.org/data/definitions/259.html

[2] Be careful while handling sensitive data, such as passwords, in program code, CERT http://www.securecoding.cert.org/confluence/display/c/MSC18-C.+Be+careful+while+handling+sensitive+data,+such+as+passwords,+in+program+code

[3] Password Management: Hardcoded Password, OWASP https://www.owasp.org/index.php/Password_Management:_Hardcoded_Password

[4] CWE-321 Use of Hard-coded Cryptographic Key, MITRE, http://cwe.mitre.org/data/definitions/321.html

[5] Be careful while handling sensitive data, such as passwords, in program code, CERT, http://www.securecoding.cert.org/confluence/display/c/MSC18-C.+Be+careful+while+handling+sensitive+data,+such+as+passwords,+in+program+code

[6] Use of hard-coded password, OWASP, https://www.owasp.org/index.php/Use_of_hard-coded_password

7. 충분하지 않은 키 길이 사용

가. 개요

길이가 짧은 키를 사용하는 것은 암호화 알고리즘을 취약하게 만들 수 있다. 키는 암호화 및 복호화에 사용되는데, 검증된 암호화 알고리즘을 사용하더라도 키 길이가 충분히 길지 않으면 짧은 시간 안에 키를 찾아낼 수 있고 이를 이용해 공격자가 암호화된 데이터나 비밀번호를 복호화 할 수 있게 된다.

나. 보안대책

RSA 알고리즘은 적어도 2,048 비트 이상의 길이를 가진 키와 함께 사용해야 하고, 대칭암호화 알고리즘(Symmetric Encryption Algorithm)의 경우에는 적어도 128비트 이상의 키를 사용한다.

다. 코드예제

다음의 예제는 보안성이 강한 RSA 알고리즘을 사용함에도 불구하고, 키 사이즈를 작게 설정함으로써 프로그램의 보안약점을 야기한 경우이다.

안전하지 않은 코드의 예 JAVA

```
1: public static final String ALGORITHM = "RSA";
2: public static final String PRIVATE_KEY_FILE = "C:/keys/private.key";
3: public static final String PUBLIC_KEY_FILE = "C:/keys/public.key";
4: public static void generateKey() {
5:     try {
6:         final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
           //RSA 키 길이를 1024 비트로 짧게 설정하는 경우 안전하지 않다.
7:         keyGen.initialize(1024);
8:         final KeyPair key = keyGen.generateKeyPair();
9:         File privateKeyFile = new File(PRIVATE_KEY_FILE);
10:        File publicKeyFile = new File(PUBLIC_KEY_FILE);
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 JAVA

```
1: public static final String ALGORITHM = "RSA";
2: public static final String PRIVATE_KEY_FILE = "C:/keys/private.key";
3: public static final String PUBLIC_KEY_FILE = "C:/keys/public.key";
4: public static void generateKey() {
5:     try {
6:         final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
7:         keyGen.initialize(2048);
8:         final KeyPair key = keyGen.generateKeyPair();
9:         File privateKeyFile = new File(PRIVATE_KEY_FILE);
10:        File publicKeyFile = new File(PUBLIC_KEY_FILE);
```

다음의 예제는 보안성이 강한 RSA 알고리즘을 사용함에도 불구하고, 키 사이즈를 작게 설정함으로써 프로그램의 보안약점을 야기한 경우이다.

안전하지 않은 코드의 예 C#

```
1: static string UseRSA(string input) {
   //RSA 키 길이를 1024 비트로 짧게 설정하는 경우 안전하지 않다.
2:     var rsa = new RSACryptoServiceProvider(1024);
3:     ...
4: }
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 C#

```
1: static string UseRSA(string input) {
   //RSA 키 길이를 2048 비트 이상으로 길게 설정한다
2:     var rsa = new RSACryptoServiceProvider(2048);
3:     ...
4: }
```

다음의 예제는 보안성이 강한 RSA 알고리즘을 사용함에도 불구하고, 키 사이즈를 작게 설정함으로써 프로그램의 보안약점을 야기한 경우이다.

안전하지 않은 코드의 예 C

```
1: EVP_PKEY *RSAKey(){
2:   EVP_PKEY *pkey;
3:   RSA *rsa;
     //RSA 키 길이를 512 비트로 짧게 설정하는 경우 안전하지 않다.
4:   rsa = RSA_generate_key(512, 35, NULL, NULL);
5:   if(rsa == NULL){
6:     printf("Error\n");
7:     return NULL;
8:   }
9:   pkey = EVP_PKEY_new();
10: EVP_PKEY_assign_RSA(pkey, rsa);
11: return pkey;
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 C

```
1: EVP_PKEY *RSAKey(){
2:   EVP_PKEY *pkey;
3:   RSA *rsa;
     //2048비트 이상으로 설정한 후에 사용해야 한다.
4:   rsa = RSA_generate_key(2048, 35, NULL, NULL);
5:   if(rsa == NULL){
6:     printf("Error\n");
7:     return NULL;
8:   }
9:   pkey = EVP_PKEY_new();
10: EVP_PKEY_assign_RSA(pkey, rsa);
11: return pkey;
12:}
```

라. 진단방법

대칭키 암호알고리즘(예,AES, ARID, SEED 등)의 경우 128bit 이상, 해시 함수(예, SHA 등)의 경우 128bit 이상, 공개키 암호알고리즘(예, RSA, DSA 등) 2,048bit 이상(ECC의 경우, 256 이상)의 키를 사용하는지 확인한다.

다음의 예제는 보안성이 강한 RSA 알고리즘을 사용함에도 불구하고, 키 사이즈를 작게 설정함으로써 프로그램의 취약점을 야기한 경우이다.

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

일반적인 진단의 예

```
1:  …
2:  public void target() throws NoSuchAlgorithmException {
3:  KeyPairGenerator keyGen = KeyPairGenerator.getInstance(“RSA”);
4:  // Key generator의 불충분한 키 크기
5:  keyGen.initialize(512);
6:  KeyPair myKeys = keyGen.generateKeyPair();
7:  }
```

마. 참고자료

[1] CWE-326 Inadequate Encryption Strength, MITRE,
http://cwe.mitre.org/data/definitions/326.html

8. 적절하지 않은 난수 값 사용

가. 개요

예측 가능한 난수를 사용하는 것은 시스템에 보안약점을 유발한다. 예측 불가능한 숫자가 필요한 상황에서 예측 가능한 난수를 사용한다면, 공격자는 SW에서 생성되는 다음 숫자를 예상하여 시스템을 공격하는 것이 가능하다.

나. 보안대책

컴퓨터의 난수발생기는 난수 값을 결정하는 시드(Seed)값이 고정될 경우, 매번 동일한 난수값이 발생한다. 이를 최대한 피하기 위해 Java에서는 Random()과 Math.random() 사용 시 java.util.Random 클래스에서 기본값으로 현재시간을 기반으로 조합하여 매번 변경 되는 시드(Seed)값을 사용하며, C에서는 rand()함수 사용 시 매번 변경되는 기본 시드(Seed)값이 없으므로, srand()로 매번 변경되는 현재시간 기반 등으로 시드(Seed)값을 설정 하여야 한다.

그러나 세션 ID, 암호화키 등 보안결정을 위한 값을 생성하고 보안결정을 수행하는 경우에는, Java에서 Random()과 Math.random()을 사용하지 말아야 하며, 예측이 거의 불가능하게 암호학적으로 보호된 java.security.SecureRandom 클래스를 사용하는 것이 안전하다.

다. 코드예제

java.util.Random 클래스의 random() 메소드 사용시, 고정된 seed를 설정하면 동일한 난수 값이 생성되어 안전하지 않다. 매번 변경되는 seed를 설정하더라도 보안결정을 위한 난수 이용시에는 안전하지 않다.

안전하지 않은 코드의 예 JAVA

```
1: import java.util.Random;
2: ...
3: public Static int getRandomValue(int maxValue) {
   //고정된 시드값을 사용하여 동일한 난수값이 생성되어 안전하지 않다.
4:    Random random = new Random(100);
5:    return random.nextInt(maxValue);
```

안전하지 않은 코드의 예 JAVA

```
6: }
7: public Static String getAuthKey() {
   //매번 변경되는 시드값을 사용하여 다른 난수값이 생성되나 보안결정을 위한 난수로는
   안전하지 않다.
8:     Random random = new Random();
9:     String authKey = Integer.toString(random.nextInt());
```

java.util.Random 클래스는 setSeed로 매번 변경되는 시드값을 설정 하거나, 현재 시간 기반으로 매번 변경되는 별도 seed를 설정하지 않는 기본값을 사용한다. 보안결정을 위해 난수 사용시에는 java.security.SecureRandom 클래스를 사용하는 것이 보다 안전하다.

안전한 코드의 예 JAVA

```
1: import java.util.Random;
2: import java.security.SecureRandom;
3: ...
4: public Static int getRandomValue(int maxValue) {
   //setSeed로 매번 변경되는 시드값을 설정 하거나, 기본값인 현재 시간 기반으로 매번
   변경되는 시드값을 사용하도록 한다.
5:     Random random = new Random();
6:     return random.nextInt(maxValue);
7: }
8: public Static String getAuthKey() {
   //보안결정을 위한 난수로는 예측이 거의 불가능하게 암호학적으로 보호된 SecureRandom을
   사용한다.
9: try{
10:   SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");
11:   MessageDigest digest = MessageDigest.getInstance("SHA-256");
12:   secureRandom.setSeed(secureRandom.generateSeed(128));
13:   String authKey = new String(digest.digest((secureRandom.nextLong() + "").getBytes()));
14:   ...
15: } catch (NoSuchAlgorithmException e) {
```

보안결정을 위한 난수로는 안전하지 않은 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
1: static int GenerateDigit()
2: {
   //매번 변경되는 시드값을 사용하여 다른 난수값이 생성되나 보안결정을 위한
   난수로는 안전하지 않다.
3:  Random rng = new Random();
4:  return rng.Next(10);
5: }
```

암호학적으로 보호된 SecureRandom 을 사용한다.

안전한 코드의 예 C#

```
1: static int GenerateDigitGood()
2: {
   //보안결정을 위한 난수로는 예측이 거의 불가능하게 암호학적으로 보호된
   SecureRandom을 사용한다.
3:  byte[] b = new byte[4];
4:  new
   System.Security.Cryptography.RNGCryptoServiceProvider().GetBytes(b);
5:  return (b[0] & 0x7f) << 24 | b[1] << 16 | b[2] << 8 | b[3];
6: }
```

Seeding 없이 사용하는 rand() 함수는 암호화에 사용되기 힘듭니다.

안전하지 않은 코드의 예 C

```
1: void foo(){
2:  int i;
3:  for(i=0; i<20; i++)
```

안전하지 않은 코드의 예 C

```
    // 프로그램을 여러 번 실행 했을 때 얻는 결과 값이 같다. 범위가 작기 때문에
    암호화에 사용되기 힘들다.
4:  printf("%d", rand());
```

srandom(), random() 함수를 사용하면 보안적으로 안전한 난수를 얻을수 있다.

안전한 코드의 예 C

```
1: void foo(){
2:   srandom(time(NULL));
3:   int i;
4:   for(i=0; i<20; i++)
5:     printf("%ld", random());
6: }
```

라. 진단방법

Math.random() 메소드를 사용하는지 확인한다.(①), Math.random()을 사용하는 경우 취약하다. 난수값을 세션ID로 설정하여 보안결정에 사용하는지 확인한다(②). 보안 결정인 경우 java.security.SecureRandom을 사용하면 안전하다.

일반적인 진단의 예

```
1:  …
2:  public void setSessionID() {  ····· ②
3:  if (!Get_Cookie('SessionID'))
4:  Set_Cookie('SessionID',Math.random());  ····· ①
5:  }
```

Math.random() 함수를 사용하여 난수 값을 발생시켰을 경우 취약하다.

정탐코드의 예

```
1: public static int[] insertRandom(int[] Cnt, int i ,int scope) {
2: int ran = (int) (Math.random() * scope) - 1;
3: if (checkDigit(ran,Cnt)) {
4: Cnt[i] = ran;
5: } else {
6: insertRandom(Cnt,i,scope);
7: }
8: return Cnt;
9: }
```

자체 랜덤 키 생성으로 취약점 보완하는 경우, 즉 다음의 예제와 같이 Date 요소들로 취약점을 보완하는 경우는 안전하다고 판정한다.

오탐코드의 예

```
1:  1: long rand = ((r.nextLong()>>>1)%(endDate.getTimeInMillis()-beginDate.getTi-
    meInMillis() + 1)) + beginDate.getTimeInMillis();
```

Java.util.Random, java.security.SecurityRandom 사용시에는 복잡도가 상대적으로 높으므로 취약 하지 않다.

오탐코드의 예

```
1: import java.security.SecureRandom;
2: ….
3: SecureRandom r = new SecureRandom();
4: long rand = ((r.nextLong()>>>1)%(endDate.getTimeInMillis()-
   beginDate.getTimeInMil- lis() + 1)) + beginDate.getTimeInMillis();
```

마. 참고자료

[1] CWE-330 Use of Insufficiently Random Values, MITRE,
http://cwe.mitre.org/data/definitions/330.html

[2] Generate strong random numbers, CERT,
http://www.securecoding.cert.org/confluence/display/java/MSC02-J.+Generate+strong+random+numbers

[3] Do not use the rand() function for generating pseudorandom numbers, CERT,
http://www.securecoding.cert.org/confluence/display/c/MSC30-C.+Do+not+use+the+rand%28%29+function+for+generating+pseudorandom+numbers

[4] Insecure Randomness, OWASP,
https://www.owasp.org/index.php/Insecure_Randomness

9. 취약한 비밀번호 허용

가. 개요

사용자에게 강한 비밀번호 조합규칙을 요구하지 않으면, 사용자 계정이 취약하게 된다. 안전한 비밀번호를 생성하기 위해서는 한국인터넷진흥원 「암호이용안내서」의 비밀번호 설정규칙을 적용해야 한다.

나. 보안대책

비밀번호 생성 시 강한 조건 검증을 수행한다. 비밀번호(비밀번호)는 숫자와 영문자, 특수문자 등을 혼합하여 사용하고, 주기적으로 변경하여 사용하도록 해야 한다.

다. 코드예제

가입자가 입력한 비밀번호에 대한 복잡도 검증 없이 가입 승인 처리를 수행하고 있다.

안전하지 않은 코드의 예 JAVA

```
1: String id = request.getParameter("id");
2: String pass = request.getParameter("pass");
3: UserVo userVO = new UserVo(id, pass);
4: ……
   // 비밀번호의 자릿수, 특수문자 포함 여부 등 복잡도를 체크하지 않고 등록
5: String result = registerDAO.register(userVO);
```

사용자 계정을 보호하기 위해 가입 시, 비밀번호 복잡도 검증 후 가입 승인처리를 수행한다.

안전한 코드의 예 JAVA

```
1: String id = request.getParameter("id");
2: String pass = request.getParameter("pass");
   //비밀번호에 자릿수, 특수문자 포함여부등의 복잡도를 체크하고 등록하게 한다.
3: Pattern pattern = Pattern.compile("((?=.*[a-zA-Z])(?=.*[0-9@#$%]). {9, })");
```

안전한 코드의 예 JAVA

```
4: Matcher matcher = pattern.matcher(pass);
5: if (!matcher.matches()) {
6:     return "비밀번호 조합규칙 오류";
7: }
8: UserVo userVO = new UserVo(id, pass);
9: ……
10:String result = registerDAO.register(userVO);
```

빈 비밀번호를 허용하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
// 빈 비밀번호를 허용
1: NetworkCredential myCred = new NetworkCredential(UserName, "");
```

빈 비밀번호를 허용하지 않도록 한다.

안전한 코드의 예 C#

```
// 빈 비밀번호를 사용하지 않음
1: NetworkCredential secure_myCred = new NetworkCredential(UserName,
   Password);
```

비밀번호에 대한 검증 없이 사용하는 C코드 예제이다.

안전하지 않은 코드의 예 C

```
1: bool authentication(char* id, char* pwd)
2: {
3: MYSQL *connectInstance;
4: connectInstance = mysql_init(NULL);
```

안전하지 않은 코드의 예 C

```
    //비밀번호 값에 대한 검증 없이 사용합니다.
5: mysql_real_connect(connectInstance, "192.168.100.211", id, pwd,
   "database", 0, NULL, 0);
6: ...
```

비밀번호에 대한 적절한 검증이 필요하다.

안전한 코드의 예 C

```
1: bool authentication(char* id, char* pwd)
2: {
3:   MYSQL *connectInstance;
4:   connectInstance = mysql_init(NULL);
     //비밀번호에 대한 적절한 검증을 수행해야 한다.
5:   if(checkValidationId( id ) == true && checkValidationPwd( pwd ) ==
   true )
6:   {
7:   mysql_real_connect(connectInstance, "192.168.100.211", id, pwd,
   "database", 0, NULL, 0);
8: }
9: ...
10:}
```

라. 진단방법

비밀번호 생성 또는 변경 때 입력값을 다음과 같은 규칙으로 검사하는 모듈이 존재하는지 확인한다.

※ (참고) 안전한 비밀번호 조합규칙

- 영문자(대·소 구별), 숫자, 특수문자 조합하여, 비밀번호 길이는 8자리~10자리 이상
- 특정 패턴 및 사용자 ID 등을 비밀번호로 사용 금지 등

다음의 예제와 같이 사용자 계정 등록 등 비밀번호를 요구하는 정보 입력에서 널(Null) 체크, 비밀번호의 자릿수, 특수문자 포함 등 비밀번호 요구 조건이 없거나 약하면 취약하다고 진단한다.

정탐코드의 예

```
1: <%
2:   String id = request.getParameter("id");
3:   String pass = request.getParameter("pass");
4:   UserVo userVO = new UserVo(id,pass);
5:   …….
6:   // 사용자를 등록한다.
7:   String result = registDAO.regist(userVO);
8:   …….
```

다음의 예제와 같이 가입자가 입력한 비밀번호에 대한 복잡도 검증 없이 가입 승인 처리를 수행하게 되면 사용자 계정을 보호하기 힘들다.

정탐코드의 예

```
1:  …
2:  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws
    IOException, ServletException {
3:    try {
4:      String id = request.getParameter("id");
5:      String passwd = request.getParameter("passwd");
6:      // 비밀번호 복잡도 검증 없이 가입 승인 처리
```

정탐코드의 예

```
7:      ….
8:    } catch (SQLException e) { … }
9: }
```

다음의 예제는 로그인시에 비밀번호를 확인하는 것은 비밀번호를 저장하는 것이 아니므로 취약하지 않다.

오탐코드의 예

```
1: <% //로그인시 입력한 값받음
2: String id = request.getParameter("id");
3: String pass = request.getParameter("pass");
```

다음의 예제와 같이 비밀번호 저장을 위한 목적이 아닌 확인의 목적일 경우 취약하지 않다고 판정 한다.

오탐코드의 예

```
1: fis = new FileInputStream(PROPERTIES_FILE);
2: props.load(new java.io.BufferedInputStream(fis));
3: userName = (String)props.getProperty("id");
4: password = (String)props.getProperty("password");
5: if (authenticate(userName, password)) { … }
```

마. 참고자료

[1] CWE-521 Weak Password Requirements, MITRE, http://cwe.mitre.org/data/definitions/521.html

[2] Password Complexity, OWASP https://www.owasp.org/index.php/Authentication_Cheat_Sheet#Implement_Proper_ Password_Strength_Controls

10. 부적절한 전자서명 확인

가. 개요

전자서명이란 서명자의 신원을 확인하고 서명된 파일의 무결성을 보장할 수 있는 디지털 정보이다.

전자서명이 사용된 경우, 전자서명을 검증하지 않거나 검증절차가 부적절하면 위변조된 파일으로 악성코드에 감염될 수 있으므로 전자서명을 확인하여 위변조 여부를 판별하고 사용해야 한다.

나. 보안대책

전자서명을 포함하는 파일을 사용할 때는 항상 전자서명을 확인하여야 한다. 이 경우, 전자서명 파일의 출처 등을 확인하여 신뢰할 수 없는 곳에서 생성된 파일을 사용하지 않도록 한다.

다. 코드예제

다음 예제는 신뢰할 수 없는 곳에서 다운로드 한 JAR 파일의 서명을 확인하지 않고 사용한다. 이 경우, 악성코드가 삽입되어 실행될 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: File f = new File(downloadedFilePath);
2: JarFile jf = new JarFile(f);
```

아래 예제는 JarFile 생성자에 boolean형 파라미터를 사용하여 전자서명을 확인한다. 전자서명 여부를 확인한 후, JarEntry.getCodeSigners() 메소드를 사용하여 JAR 객체에 대한 전자서명 주체를 신뢰할 수 있는지 확인하여야 한다.

쿠키의 만료시간은 해당 기능에 맞춰 최소로 설정하고 영속적인 쿠키에는 중요 정보가 포함되지 않도록 한다.

안전한 코드의 예 JAVA

```
1:File f = new File(downloadedFilePath);
2:JarFile jf = new JarFile(f, true);
3:Enumeration<JarEntry> ens = jf.entries();
4:while (ens.hasMoreElements()) {
5:      JarEntry en = ens.nextElement();
6:      if (!en.isDirectory()) {
7:              if (en.toString().equals(path)) {
8:              byte[] data = readAll(jar.getInputStream(en), en.getSize());
9:              CoeSigner[] signers = en.getCodeSigners();
10:             ...
11:             }
12:     }
13:}
14:jf.close();
```

라. 진단방법

JarFile을 생성할 때 전자서명 여부를 확인하는 Boolean형 파라미터가 있는 생성자를 사용하여 전자서명 여부를 검증하는지 확인한다. 또한 JarEntry.getCodeSigners() 메소드로 전자서명 주체 검증 여부를 확인한다.

일반적인 진단의 예

```
1:List<FileItem> fileItemsList = uploader.parseRequest(request);
2:Iterator<FileItem> fileItemsIterator = fileItemsList.iterator();
3:while (fileItemsIterator.hasNext()) {
4:      FileItem fileItem = fileItemsIterator.next();
5:      File file = new File(request.getServletContext().getAttribute("FILES_DIR") +
File.separator + fileItem.getName());
```

일반적인 진단의 예

```
6:      fileItem.write(file);
7:      // 입력받은 FileItem 을 이용해서 JarFile 을 생성한다.
8:      JarFile driver = new JarFile(file);
9:      // 생성된 JarFile을 이용한다.
10:     useDriver(driver);
11:}
```

다음 예제는 사용자가 업로드한 데이터를 이용해 JarFile 객체를 생성 시, 전자서명 여부를 확인하는 생성자를 사용하고 있으나 전자서명 주체를 검증하는 코드를 사용하지 않으므로 취약하다고 판정한다.

정탐코드의 예

```
1:@RequestMapping(value = "/upload", method = RequestMethod.POST)
2:public Student upload(@RequestParam("file") MultipartFile multipartFile)
3:throws IllegalStateException, IOException {
4:      File f = new File(multipartFile.getOriginalFilename());
5:      multipartFile.transferTo(f);
6:      JarFile jf = new JarFile(f, true);
7:      ...
8:}
```

다음 예제는 전자서명 여부를 확인하는 생성자를 사용하고, 전자서명 주체를 검증하는 코드를 사용하므로 취약하지 않다고 판정한다.

오탐코드의 예

```
1:@RequestMapping(value = "/upload", method = RequestMethod.POST)
2:public Student upload(@RequestParam("file") MultipartFile multipartFile)
3:throws IllegalStateException, IOException {
4:      File f = new File(multipartFile.getOriginalFilename());
5:      multipartFile.transferTo(f);
6:      JarFile jf = new JarFile(f, true);
```

오탐코드의 예

```
7:      Enumeration<JarEntry> ens = jf.entries();
8:      while (ens.hasMoreElements()) {
9:              JarEntry en = ens.nextElement();
10:             if (!en.isDirectory()) {
11:                     if (en.toString().equals(path)) {
12:                             byte[] data = readAll(jar.getInputStream(en), en.getSize());
13:                             CodeSigner[] signers = en.getCodeSigners();
14:                             if (signers != null && signers.length != 0){
15:                                     ...
16:                             }
17:                     }
18:             }
19:     }
20:     return null;
21:}
```

마. 참고자료

① CWE-347 : Improper Verification of Cryptographic Signature, MITRE, http://cwe.mitre.org/data/definitions/347.html

11. 부적절한 인증서 유효성 검증

가. 개요

인증서를 확인하지 않거나 인증서 확인 절차를 적절하게 수행하지 않아, 악의적인 호스트에 연결되거나 신뢰할 수 없는 호스트에서 생성된 데이터를 수신하게 되는 보안약점이다.

나. 보안대책

쿠키의 만료시간은 세션이 지속되는 시간을 고려하여 최소한으로 설정하고 영속적인 쿠키에는 사용자 권한 등급, 세션ID 등 중요정보가 포함되지 않도록 한다.

인증서를 사용하기 전에 인증서의 유효성을 확인한다. 인증서의 Common Name과 실제 호스트가 일치하는지, 신뢰된 발급기관(CA, RootCA)의 서명 여부, 인증서의 유효기간, 인증서의 해지여부, 안전한 암호화 알고리즘 사용 여부 확인 등으로 유효한 인증서인지 검증하는 절차를 구현하여야 한다.

다. 코드예제

아래 예제는 SSL_get_verify_result의 결과값이 X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN인 경우에 자체 서명된 인증서이다. 이 경우, 해당 어플리케이션이 악의적인 행위를 할 수 있다.

안전하지 않은 코드의 예 C

```
1:if ((cert = SSL_get_peer_certificate(ssl)) && host)
2:foo=SSL_get_verify_result(ssl);
3:if ((X509_V_OK==foo) ||X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
// 자체 서명된 인증서일 수 있다.
```

아래 예제는 인증서 검증결과 X509_V_OK로 반환되더라도 호스트가 Common Name과 일치하는지 확인하지 않으므로 인증서가 허가된 호스트용이라는 것을 확신할 수 없다.

안전하지 않은 코드의 예 C

```
1:cert = SSL_get_peer_certificate(ssl);
2:if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
3:/* CN 을 확인하지 않았지만 신뢰하고 진행한다. 이럴 경우, 공격자가 Common Name을
www.attack.com으로 설정하여 중간자 공격에 사용할 경우 데이터가 중간에서 복호화 되고 있음을
탐지하지 못한다. */
4: }
```

아래 예제는 인증서 DN 일치여부와 유효기간 등을 검증한다.

안전한 코드의 예 JAVA

```
1:private boolean verifySignature(X509Certificate toVerify, X509Certificate signingCert) {
2:      /* 검증하려는 호스트 인증서(toVerify)와 CA인증서(signing Cert)의 DN(Distinguished
Name)이 일치하는지 여부를 확인한다.*/
3:      if (!toVerify.getIssuerDN().equals(signingCert.getSubjectDN())) return false;
4:      try {
5:              // 호스트 인증서가 CA인증서로 서명 되었는지 확인한다.
6:              toVerify.verify(signingCert.getPublicKey());
7:              // 호스트 인증서가 유효기간이 만료되었는지 확인한다.
8:              toVerify.checkValidity();
9:              return true;
10:     } catch (GeneralSecurityException verifyFailed) {
11:             return false;
12:     }
13:}
```

또한, 유효기간이 남아 있는 인증서의 해지여부를 확인하기 위해서는 CRL(Certificate revocation lists)로 인증서 해지목록 또는 OCSP(Online Certificate Status Protocol)로 실시간 인증서 상태확인이 필요하다. CRL 리스트는 해당 인증서를 참조하여 인증기관에서 다운받을 수 있으며, 다운받은 CRL으로 해지된 인증서를 확인할 수 있다.

라. 진단방법

인증서를 확인하여(①), 검증결과 값이 X509_V_OK 이외의 값을 허용하는지 확인한다(②). 신뢰되지 않은 발급기관으로 받은 인증서를 허용하거나 유효기간이 만료된 인증서를 허용하면 위험하다. 또한 인증서의 Common Name 확인한다.(③)

일반적인 진단의 예

```
1:cert = SSL_get_peer_certificate(ssl); ····· ①
2:if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) { ····· ②
3:      SSL_set_hostflags(ssl, X509_CHECK_FLAG_NO_PARTIAL_WILDCARDS);
4:      if (!SSL_set1_host(ssl, "www.securecoding_example.com")) { ····· ③
5:              error("Invalid Common Name");
6:              return -1;
7:      }
8:// 서버와 클라이언트간에 피어 확인을 사용하며 콜백함수는 지정하지 않는다.
9:      SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);
10:     ...
11:}
```

다음 예제는 유효한 인증서인지 확인하지 않고, 인증서를 사용하고 있기 때문에 취약하다고 판정한다.

정탐코드의 예

```
1:cert = SSL_get_peer_certificate(ssl);
2:if (cert) {
3:      // 인증서를 확인하지 않고 작업 수행
4:      ...
5:}
```

마. 참고자료

① CWE-295: Improper Certificate Validation, MITRE, http://cwe.mitre.org/data/definitions/295.html

12. 사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출

가. 개요

대부분의 웹 응용프로그램에서 쿠키는 메모리에 상주하며, 브라우저의 실행이 종료되면 사라진다. 프로그래머가 원하는 경우, 쿠키를 디스크에 저장할 수 있으며, 다음 브라우저 세션이 시작되었을 때 메모리에 로드된다. 개인정보, 인증정보 등이 이와 같은 영속적인 쿠키(Persistent Cookie)에 저장된다면, 공격자는 쿠키에 접근할 수 있는 보다 많은 기회를 가지게 되어 시스템을 취약하게 만든다.

나. 보안대책

쿠키의 만료시간은 세션이 지속되는 시간을 고려하여 최소한으로 설정하고 영속적인 쿠키에는 사용자 권한 등급, 세션ID 등 중요정보가 포함되지 않도록 한다.

다. 코드예제

쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있다. 쿠키의 유효기간이 긴 경우 사용자 하드디스크에 쿠키가 저장되며 저장된 쿠키는 쉽게 도용될 수 있으므로 취약하다.

안전하지 않은 코드의 예 JAVA

```
1: Cookie loginCookie = new Cookie("rememberme", "YES");
//쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있어 안전하지 않다.
2: loginCookie.setMaxAge(60*60*24*365);
3: response.addCookie(loginCookie);
```

쿠키의 만료시간은 해당 기능에 맞춰 최소로 설정하고 영속적인 쿠키에는 중요 정보가 포함되지 않도록 한다.

안전한 코드의 예 JAVA

```
1: Cookie loginCookie = new Cookie("rememberme", "YES");
//쿠키의 만료시간은 해당 기능에 맞춰 최소로 사용한다.
```

안전한 코드의 예 JAVA

```
2: loginCookie.setMaxAge(60*60*24);
3: response.addCookie(loginCookie);
```

다음은 쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있는 C# 코드의 예제이다. 쿠키의 유효기간이 긴 경우 사용자 하드디스크에 쿠키가 저장되며 저장된 쿠키는 쉽게 도용될 수 있으므로 취약하다.

안전하지 않은 코드의 예 C#

```
1: HttpCookie cookie = Request.Cookies.Get("ExampleCookie");
//쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있어 안전하지 않다.
2: cookie.Expires = DateTime.Now.AddMinutes(60.0*24.0*365.0);
3: Response.Cookies.Add(cookie);
```

쿠키의 만료 시간을 10분으로 설정하고 있는 C# 코드의 예제이다.

안전한 코드의 예 C#

```
1: HttpCookie cookie = Request.Cookies.Get("ExampleCookie");
//쿠키의 만료시간은 해당 기능에 맞춰 최소로 사용한다.
2: cookie.Expires = DateTime.Now.AddMinutes(10d);
3: Response.Cookies.Add(cookie);
```

라. 진단방법

사용자 브라우저로 쿠키를 전송하는지 확인하고(①), 쿠키 유효기간 확인한다(②). 쿠키 설정 값에 id 정보 등 중요 정보 포함 여부를 확인한다(③)

일반적인 진단의 예

```
1:  …
2:  Cookie ck = new Cookie(“id”, “test123”) ; ……③
3:  ck.setMaxAge(60*60*24*365*10) ; ……②
4:  response.addCookie(ck) ; ……①
5:  …
```

다음의 예제는 쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있다. 쿠키의 유효기간이 긴 경우 사용자 하드디스크에 쿠키가 저장되며 저장된 쿠키는 쉽게 도용될 수 있으므로 취약하다.

정탐코드의 예

```
1: protected void doPost(HttpServletRequest request, HttpServletResponse response) {
2:              ……
3:    String username = request.getParameter(“username”);
4:    char[] password = request.getParameter(“password”).toCharArray();
5:    boolean rememberMe = Boolean.valueOf(request.getParameter(“rememberme”));
6:
7:    LoginService loginService = new LoginServiceImpl();
8:    if (rememberMe) {
9:        Cookie loginCookie = new Cookie(“rememberme”, “YES”);
10:       loginCookie.setMaxAge(60*60*24*30*12);
11:       response.addCookie(loginCookie);
12:           ……
13:   } else {
14:           ……
```

아래 예제에서는 쿠키의 만료시간을 해당 기능에 맞춰 최소로 설정하고 있어 안전하다.

오탐코드의 예

```
1: protected void doPost(HttpServletRequest request, HttpServletResponse response) {
2:       ……
3:    String username = request.getParameter("username");
4:    char[] password = request.getParameter("password").toCharArray();
5:    boolean rememberMe = Boolean.valueOf(request.getParameter("rememberme"));
6:
7:    LoginService loginService = new LoginServiceImpl();
8:    if (rememberMe) {
9:        Cookie loginCookie = new Cookie("rememberme", "YES");
10:       loginCookie.setMaxAge(60*60*24);
11:       response.addCookie(loginCookie);
12:       ……
13:   } else {
14:       ……
15:   }
16:   Arrays.fill(password, ' ');
17:}
```

마. 참고자료

[1] CWE-539 Information Exposure Through Persistent Cookies, MITRE, http://cwe.mitre.org/data/definitions/539.html

[2] Do not store unencrypted sensitive information on the client side, CERT, http://www.securecoding.cert.org/confluence/display/java/FIO52-J.+Do+not+store+unencrypted+sensitive+information+on+the+client+side

[3] Expire and Max-Age Attributes, OWASP, https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Expire_and _Max-Age_Attributes

13. 주석문 안에 포함된 시스템 주요정보

가. 개요

비밀번호를 주석문에 넣어두면 시스템 보안이 훼손될 수 있다. 소프트웨어 개발자가 편의를 위해서 주석문에 비밀번호를 적어둔 경우, 소프트웨어가 완성된 후에는 제거하는 것이 어렵게 된다. 또한, 공격자가 소스코드에 접근할 수 있다면,아주 쉽게 시스템에 침입할 수 있다.

나. 보안대책

주석에는 ID, 비밀번호 등 보안과 관련된 내용을 기입하지 않는다.

다. 코드예제

다음 예제는 개발자의 이해를 돕기 위한 목적 등 편리성을 위해 비밀번호를 주석문 안에 서술하고 제대로 지우지 않아서 보안약점이 발생한 경우이다.

안전하지 않은 코드의 예 JAVA

```
//주석문으로 DB연결 ID, 비밀번호의 중요한 정보를 노출시켜 안전하지 않다.
// DB연결 root / a1q2w3r3f2!@
1: con = DriverManager.getConnection(URL, USER, PASS);
```

프로그램 개발 시에 주석문 등에 남겨놓은 사용자 계정이나 비밀번호 등의 정보는 개발 완료 시에 확실하게 삭제하여야 한다.

안전한 코드의 예 JAVA

```
// ID, 비밀번호등의 중요 정보는 주석에 포함해서는 안된다.
1: con = DriverManager.getConnection(URL, USER, PASS);
```

주석에 비밀번호를 포함하고 있는 C# 코드이다.

안전하지 않은 코드의 예 C#

```
//주석문으로 DB연결 ID, 비밀번호의 중요한 정보를 노출시켜 안전하지 않다.
//DB연결 root / a1q2w3r3f2!@
1: conn = customGetConnection(USER, PASS);
```

프로그램 개발 시에 주석문 등에 남겨놓은 사용자 계정이나 비밀번호 등의 정보는 개발 완료 시에 확실하게 삭제하여야 한다.

안전한 코드의 예 C#

```
// ID, 비밀번호등의 중요 정보는 주석에 포함해서는 안된다.
1: conn = customGetConnection(USER, PASS);
```

주석에 비밀번호를 포함하고 있는 C 예제 코드이다.

안전하지 않은 코드의 예 C

```
/* password is "admin" */
/* passwd is "admin" */
1: int verfiyAuth(char *ipasswd, char *orgpasswd){
2:   char *admin = "admin";
3:   if(strncmp(ipasswd, oprgpasswd, sizeof(ipasswd)) != 0){
4:     printf("Authentication Fail!\n");
5:   }
6:   return admin;
```

불필요한 주석은 삭제해야 한다.

안전한 코드의 예 C

```
1: int verfiyAuth(char *ipasswd, char *orgpasswd){
2:   char *admin = "admin";
3:   if(strncmp(ipasswd, oprgpasswd, sizeof(ipasswd)) != 0){
4:     printf("Authentication Fail!\n");
5:   }
6:   return admin;
7: }
```

라. 진단방법

DB접속, 관리자 로그인 등이 구현된 코드를 확인하고(①), 주석을 확인하여 암호 포함여부 확인한다(②).

일반적인 진단의 예

```
1:   …/*
2:   * Password for administrator is “tiger.” ····· ②
3:   */
4:   public Connection DBConnect(String id, String password) {
5:     String url = “DBServer”;
6:     Connection conn = null;
7:     try {
8:       String CONNECT_STRING = url + “:” + id + “:” + password;
9:       InitialContext ctx = new InitialContext();
10:      DataSource datasource = (DataSource) ctx.lookup(CONNECT_STRING); ····· ①
11:      conn = datasource.getConnection();
12:    } catch (SQLException e) {
13:      System.err.printf(“….”);
14:    }
15: …
16: }
```

다음의 예제에서는 주석문 안에 개발자의 이해를 돕기 위한 목적 등으로 비밀번호를 적어 놓고 있으므로 취약하다고 판정한다.

정탐코드의 예

```
1: public void daoTest() throws Exception {
2:   // db sample : 84d5d0a08a3ec5e2d91a
3:   // 암호화 전, 후 : 1365ADMIN_01, aa84c40031d808196537ad3dcf81f9af
4:   String pwd= "46c165a343fd6841273ae04655af24dd";
5:   String pwd1= ARIAEngine.decARIA(pwd);
6:   System.out.println(pwd1);
7: }
```

주석문안에 "password", "passwd", "비밀번호"등의 텍스트가 존재하지만 실제로 그 내용이 비밀번호가 아닌 경우에는 취약하지 않다고 판정한다.

오탐코드의 예

```
1: // 암호화
2: String Sid = getSessionValue(session, "ihidnum");
3: Sid = AEScryptWithSaltKey.encode(StrTool.sNN(Sid));
4: String sIhidnum = Sid;
```

마. 참고자료

[1] CWE-615 Information Exposure Through Comments, MITRE, http://cwe.mitre.org/data/definitions/615.html

14. 솔트 없이 일방향 해쉬 함수 사용

가. 개요

비밀번호를 저장시 일방향 해시함수의 성질을 이용하여 비밀번호의 해시값을 저장한다. 만약 비밀번호를 솔트(Salt)없이 해시하여 저장한다면, 공격자는 레인보우 테이블과 같이 해시값을 미리 계산 하여 비밀번호를 찾을 수 있게 된다.

나. 보안대책

비밀번호를 저장시 비밀번호와 솔트를 해시함수에 함께 입력하여 얻은 해시값을 저장한다.

다. 코드예제

다음의 예제는 비밀번호 저장 시 솔트 없이 비밀번호에 대한 해시값을 얻는 과정을 보여준다.

안전하지 않은 코드의 예 JAVA

```
1: public String getPasswordHash(String password) throws Exception {
2:     MessageDigest md = MessageDigest.getInstance("SHA-256");
       //해시에 솔트를 적용하지 않아 안전하지 않다.
3:     md.update(password.getBytes());
4:     byte byteData[] = md.digest();
5:     StringBuffer hexString = new StringBuffer();
6:     for (int i=0; i<byteData.length i++) {
7:         String hex=Integer.toHexString(0xff & byteData[i]);
8:         if (hex.length() == 1) {
9:             hexString.append('0');
10:        }
11:        hexString.append(hex);
12:    }
13:    return hexString.toString();
14:}
```

비밀번호만을 해시함수의 입력으로 사용하기에 레인보우 테이블을 이용한 사전 공격이 가능하며, 이를 방지하기 위해 비밀번호와 솔트를 함께 해시함수에 적용하여 사용한다.

안전한 코드의 예 JAVA

```
1: public String getPasswordHash(String password, byte[] salt) throws +Exception {
2:     MessageDigest md = MessageDigest.getInstance("SHA-256");
3:     md.update(password.getBytes());
//해시사용시에는 원문을 찾을 수 없도록 솔트를 사용하여야한다.
4:     md.update(salt);
5:     byte byteData[] = md.digest();
6:     StringBuffer hexString = new StringBuffer();
7:     for (int i=0; i<byteData.length i++) {
8:         String hex=Integer.toHexString(0xff & byteData[i]);
9:         if (hex.length() == 1) {
10:             hexString.append('0');
11:     }
12:     hexString.append(hex);
13:   }
14:   return hexString.toString();
15:}
```

다음의 예제는 비밀번호 저장 시 솔트 없이 비밀번호에 대한 해시값을 얻는 과정을 보여준다.

안전하지 않은 코드의 예 C#

```
1: static void HashWithoutSalt()
2: {
       //해시에 솔트를 적용하지 않아 안전하지 않다.
3:     var bytes = new byte[100];
4:     (new Random()).NextBytes(bytes);
5:     var source = bytes;
6:     var sha256 = new SHA256CryptoServiceProvider();
7:     sha256.ComputeHash(source);
8: }
```

비밀번호만을 해시함수의 입력으로 사용하기에 레인보우 테이블을 이용한 사전 공격이 가능하며, 이를 방지하기 위해 비밀번호와 솔트를 함께 해시함수에 적용하여 사용한다.

안전한 코드의 예 C#

```
1: static void HashWithSalt(int saltLength)
2: {
   //해시에 솔트를 적용하여 원문을 찾을 수 없게 한다.
3:    var bytes = new byte[100];
4:    (new Random()).NextBytes(bytes);
5:    var source = bytes;
6:    var sha256 = new SHA256CryptoServiceProvider();
7:    byte[] saltBytes = GenerateRandomCryptographicBytes(saltLength);
8:    List<byte> sourceWithSaltBytes = new List<byte>();
9:    sourceWithSaltBytes.AddRange(source);
10:   sourceWithSaltBytes.AddRange(sourceWithSaltBytes);
11:   sha256.ComputeHash(sourceWithSaltBytes.ToArray());
12:}
```

솔트 값없이 해시를 생성하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
1: void GenerateHash(char* data)
2: {
3: char[512] hashedData = {0};
   //솔트 값 부분이 NULL 로 되어있어 들어가지 않는다.
4: MD5HashAlgorithm( data, hashedData, NULL );
5: ...
```

솔트 값을 인자로 넘겨줘야 한다.

안전한 코드의 예 C

```
1: void GenerateHash(char* data, char* salt)
2: {
```

안전한 코드의 예 C

```
3: char hashedData[512] = {0};
4: MD5HashAlgorithm( data, hashedData, salt );
5: ...
6: }
```

라. 진단방법

getInstance 함수로 안전한 해시 알고리즘을 인자로 하여 MessageDigest 객체를 생성하는지 확인 하고(①), 해시 값을 반환(②) 하기 전에 update 함수에 솔트를 사용하여 데이터를 해시하는지 확인한다.

다음의 예제에서는 MessageDigest 객체를 생성하고 솔트 없이 해시 값을 반환하고 있으므로 취약하다고 판정한다.

일반적인 진단의 예

```
1: MessageDigest digest = MessageDigest.getInstance("SHA-512"); ····· ①
2: digest.reset();
3: return digest.digest(password.getBytes("UTF-8")); ····· ②
```

마. 참고자료

[1] CWE-759, Use of a One-Way Hash without a Salt, MITRE, http://cwe.mitre.org/data/definitions/759.html

[2] Store passwords using a hash function, CERT, http://www.securecoding.cert.org/confluence/display/java/MSC62-J.+Store+passwords+using+a+hash+function

[3] Use_a_cryptographically_strong_credential-specific_salt, OWASP, https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Use_a_cryptog raphically_strong_credential-specific_salt

15. 무결성 검사 없는 코드 다운로드

가. 개요

원격으로부터 소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받고, 이를 실행하는 제품들이 종종 존재한다. 이는 호스트 서버의 변조, DNS 스푸핑 (Spoofing) 또는 전송시의 코드 변조 등의 방법을 이용하여 공격자가 악의적인 코드를 실행할 수 있도록 한다.

나. 보안대책

DNS 스푸핑(Spoofing)을 방어할 수 있는 DNS lookup을 수행하고 코드 전송시 신뢰할 수 있는 암호 기법을 이용하여 코드를 암호화한다. 또한 다운로드한 코드는 작업 수행을 위해 필요한 최소한의 권한으로 실행하도록 한다.

다. 코드예제

이 예제는 URLClassLoader()로 원격에서 파일을 다운로드한 뒤 로드하면서, 대상 파일에 대한 무결성 검사를 수행하지 않아 파일변조 등으로 인한 피해가 발생할 수 있는 경우이다. 이러한 경우 공격자는 악의적인 실행코드로 클래스의 내용을 수정할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: URL[] classURLs = new URL[] { new URL("file:subdir/") };
2: URLClassLoader loader = new URLClassLoader(classURLs);
3: Class loadedClass = Class.forName("LoadMe", true, loader);
```

이를 안전한 코드로 변환하면 다음과 같다. 클래스를 로드하기 전 클래스의 체크섬(Checksum)을 실행하여 로드하는 코드가 변조되지 않았음을 확인한다.

안전한 코드의 예 JAVA

```
// 공개키 방식의 암호화 알고리즘과 메커니즘을 이용하여 전송파일에 대한 시그니처를 생성하고 파일의
   변조유무를 판단한다. 서버에서는 Private Key를 가지고 MyClass를 암호화한다.
```

안전한 코드의 예 JAVA

```
1: String jarFile = "./download/util.jar";
2: byte[] loadFile = FileManager.getBytes(jarFile);
3: loadFile = encrypt(loadFile, privateKey);
// jarFileName으로 암호화된 파일을 생성한다.
4: FileManager.createFile(loadFile, jarFileName);

// 클라이언트에서는 파일을 다운로드 받을 경우 Public Key로 복호화한다.
5: URL[] classURLs = new URL[] { new URL("http://filesave.com/download/util.jar") };
6: URLConnection conn = classURLs.openConnection();
7: InputStream is = conn.getInputStream();
// 입력 스트림을 jarFile명으로 파일을 출력한다.
8: FileOutputStream fos = new FileOutputStream(new File(jarFile));
9: While (is.read(buf) != -1) {
10: .....
11:}
12:byte[] loadFile = FileManager.getBytes(jarFile);
13:loadFile = decrypt(loadFile, publicKey);
// 복호화된 파일을 생성한다.
14:FileManager.createFile(loadFile, jarFile);
15:URLClassLoader loader = new URLClassLoader(classURLs);
16:Class loadedClass = Class.forName("MyClass", true, loader);
```

파일 무결성 검사를 하지 않고 파일을 다운로드하는 C#코드 예제이다.

안전하지 않은 코드의 예 C#

```
1: public override bool DownloadFile()
2:     {
3:         var url = "https://www.somewhere.untrusted.com";
4:         var desDir = "D:/DestinationPath";
5:         string fileName = Path.GetFileName(url);
6:         string descFilePath = Path.Combine(desDir, fileName);
7:         try
```

안전하지 않은 코드의 예 C#

```
8:      {
9:          WebRequest myre = WebRequest.Create(url);
10:     }
11:     catch (Exception ex)
12:     {
13:         throw new Exception(ex.Message);
14:     }
15:     try
16:     {
17:         byte[] fileData;
        //파일 무결성 검사 없이 다운로드
18:       using (WebClient client = new WebClient())
19:       {
20:           fileData = client.DownloadData(url);
21:       }
22:       using (FileStream fs = new FileStream(descFilePath,
  FileMode.OpenOrCreate))
23:       {
24:           fs.Write(fileData, 0, fileData.Length);
25:       }
26:       return true;
27:     }
28:     catch (Exception ex)
29:     {
30:         throw new Exception(ex.Message);
31:     }
32: }
```

해시값등을 이용하여 파일 무결성 검사 후 다운로드를 해야 한다.

안전한 코드의 예 C#

```
1: public override bool DownloadFile()
```

안전한 코드의 예 C#

```
2:    {
3:      var url = "https://www.somewhere.untrusted.com";
4:      var desDir = "D:/DestinationPath";
5:      string fileName = Path.GetFileName(url);
6:      string descFilePath = Path.Combine(desDir, fileName);
7:      try
8:      {
9:        WebRequest myre = WebRequest.Create(url);
10:     }
11:     catch (Exception ex)
12:     {
13:       throw new Exception(ex.Message);
14:     }
15:     try
16:     {
17:       byte[] fileData;
18:       using (WebClient client = new WebClient())
19:       {
20:         fileData = client.DownloadData(url);
21:       }
        //해시 값 등을 사용하여 다운로드 받은 파일 무결성 검사
22:       CheckIntegrity(fileData);
23:       using (FileStream fs = new FileStream(descFilePath,
  FileMode.OpenOrCreate))
24:       {
25:         fs.Write(fileData, 0, fileData.Length);
26:       }
27:       return true;
28:     }
29:     catch (Exception ex)
30:     {
31:       throw new Exception(ex.Message);
32:     }
33:   }
```

리턴값을 이용하여 무결성 검사를 하지 않은 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
1: void foo(){
2: /* ... */
3:    hFile = CreateFile((LPCWSTR)data,GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
  FILE_ATTRIBUTE_NORMAL, NULL);
4:   InternetQueryDataAvailable(m_hURL, &dwSize,0,0);
5:   InternetReadFile(m_hURL, lpBuffer, dwSize, &dwRead);
6:   WriteFile(hFile, lpBuffer, dwRead, &dwWritten, NULL);
7:   /* ... */
```

리턴 값을 이용하여 무결성을 확인한 후 사용해야 한다.

안전한 코드의 예 C

```
1: void foo(){
2:   /* ... */
3:   hFile = CreateFile((LPCWSTR)data,GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
  FILE_ATTRIBUTE_NORMAL, NULL);
4:   InternetQueryDataAvailable(m_hURL, &dwSize,0,0);
5:   bool result = InternetReadFile(m_hURL, lpBuffer, dwSize, &dwRead);
6:   if( result == true){
7: WriteFile(hFile, lpBuffer, dwRead, &dwWritten, NULL);
8:   }
9:   /* ... */
10:}
```

라. 진단방법

클래스를 로드하기 위한 코드를 확인(①)하고 클래스를 로드하기 전에 체크섬을 실행 및 비교하여 로드하려는 코드가 변조되지 않았음을 확인하다. 다음의 예제에서는 로드하려는 클래스의 체크섬을 비교하여 무결성을 확인하고 있지 않으므로 취약하다고 판정한다.

일반적인 진단의 예

```
1: URL[] classURLs = new URL[]{
2: new URL("file:subdir/")
3: };
4: URLClassLoader loader = new URLClassLoader(classURLs);
5:  Class loadedClass = Class.forName("LoadMe", true, loader);  ····· ①
```

마. 참고자료

[1] CWE-494 Download of Code Without Integrity Check, MITRE, http://cwe.mitre.org/data/definitions/494.html

[2] Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar, CERT, http://www.securecoding.cert.org/confluence/display/java/SEC06-J.+Do+not+rely+on+the+default+automatic+signature+verification+provided+by+URLClassLoader+and+java.util.jar

16. 반복된 인증시도 제한 기능 부재

가. 개요

일정 시간 내에 여러 번의 인증을 시도하여도 계정잠금 또는 추가 인증 방법 등의 충분한 조치가 수행되지 않는 경우, 공격자는 예상 ID와 비밀번호들을 사전(Dictionary)으로 만들고 무차별 대입(brute force)하여 로그인 성공 및 권한획득이 가능하다.

나. 보안대책

인증시도 횟수를 적절한 횟수로 제한하고 설정된 인증실패 횟수를 초과했을 경우 계정을 잠금하거나 추가적인 인증과정을 거쳐서 시스템에 접근이 가능하도록 한다.

다. 코드예제

다음 예제는 로그인 정보를 잘못 입력하였을 경우 다시 입력을 시도하는데 있어 제한이 없다. 따라서 공격자는 여러 가지 비밀번호로 인증을 재시도하여 올바른 비밀번호를 알아내고 로그인에 성공할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: private static final String SERVER_IP = "127.0.0.1";
2: private static final int SERVER_PORT = 8080;
3: private static final int FAIL = -1;
4: public void login() {
5:   String username = null;
6:   String password = null;
7: Socket socket = null;
8:   in t result = FAIL;
9:   try {
10:      socket = new Socket(SERVER_IP, SERVER_PORT);
         //인증 실패에 대해 제한을 두지 않아 안전하지 않다.
11:      while (result == FAIL) {
12:      ...
```

안전하지 않은 코드의 예 JAVA

```
13:     result = verifyUser(username, password);
14:     }
15: }
```

다음 예제는 사용자 인증시도 횟수를 기록하는 MAX_ATTEMPTS 변수를 정의하고, 이를 인증시도 횟수를 제한하는 카운터로 사용함으로써 무차별 공격에 대응하는 코드이다.

안전한 코드의 예 JAVA

```
1: private static final String SERVER_IP = "127.0.0.1";
2: private static final int SERVER_PORT = 8080;
3: private static final int FAIL = -1;
4: private static final int MAX_ATTEMPTS = 5;
5: public void login() {
6:   String username = null;
7:   String password = null;
8:   Socket socket = null;
9:   int result = FAIL;
10: int count = 0;
11: try {
12:     socket = new Socket(SERVER_IP, SERVER_PORT);
        //인증 실패 및 시도 횟수에 제한을 두어 안전하다.
13:     while (result == FAIL && count < MAX_ATTEMPTS) {
14:     ...
15:     result = verifyUser(username, password);
16:     count++;
17:     }
18: }
```

다음 예제는 로그인 정보를 잘못 입력하였을 경우 다시 입력을 시도하는데 있어 제한이 없다. 따라서 공격자는 여러 가지 비밀번호로 인증을 재시도하여 올바른 비밀번호를 알아내고 로그인에 성공할 수 있다.

안전하지 않은 코드의 예 C#

```
//로그인 실패 시 아무런 제약이 없음
1: override protected void OnLoginError(EventArgs e)
2: {
3:     //do nothing
4: }
```

로그인 시도에 대한 횟수를 제한한다.

안전한 코드의 예 C#

```
1: override protected void OnLoginError(EventArgs e)
2:     {
       // 연속적인 사용자 인증 시도에 대한 횟수를 제한
3:         if(ViewState["LoginErrors"] == null)
4:             ViewState["LoginErrors"] = 0;
5:         int ErrorCount = (int)ViewState["LoginErrors"] + 1;
6:         ViewState["LoginErrors"] = ErrorCount;

7:         if((ErrorCount > 3) && Login1.PasswordRecoveryUrl !=
string.Empty)
8:             Response.Redirect(Login1.PasswordRecoveryUrl);
9:     }
```

안전하지 않은 코드의 예 C

```
1: int validateUser(char *host, int port) {
2:     int socket = openSocketConnection(host, port);
3:     if (socket < 0) {
```

안전하지 않은 코드의 예 C

```
4:      printf("Unable to open socket connection");
5:      return(FAIL);
6:   }
7:   int isValidUser = 0;
8:   char nm[NAME_SIZE];
9:   char pw[PSWD_SIZE];
// 인증시도 횟수를 제한하고 있지 않음
10:  while (isValidUser==0) {
11:     if (getNextMsg(socket, nm, NAME_SIZE) > 0) {
12:        if (getNextMsg(socket, pw, PSWD_SIZE) > 0) {
13:           isValidUser = AuthenticateUser(nm, pw);
14:        }
15:     }
16:  }
17:  return(SUCCESS);
18:}
```

안전한 코드의 예 C

```
1: #define MAX_ATTEMPTS 5

2: int validateUser(char *host, int port) {
3:      .....
  // 연속적인 사용자 인증 시도에 대한 횟수를 제한
4:     int count = 0;
5:     while ((isValidUser==0) && (count<MAX_ATTEMPTS)) {
6:        if (getNextMsg(socket, nm, NAME_SIZE) > 0) {
7:           if (getNextMsg(socket, pw, PSWD_SIZE) > 0) {
8:              isValidUser = AuthenticateUser(nm, pw);
9:           }
10:       }
11:       count++;
12:   }
```

안전한 코드의 예 C

```
13:  if (isValidUser) {
14:      return(SUCCESS);
15:  } else {
16:      return(FAIL);
17:  }
18:}
```

라. 진단방법

인증을 위한 함수를 호출하는 경우, 이 함수의 호출 횟수를 확인하고 함수의 호출을 제한하는 코드가 존재하는지 확인한다. 그렇지 않은 경우는 취약하다고 판단한다.

다음 예제는 인증을 시도를 할 때, 인증 시도에 대한 제한 없이 반복문 안에서 계속 인증 시도를 하는 코드이다. 인증 시도 회수에 대한 검사 루틴이 존재하지 않으므로 보안약점이 존재하는 코드라고 진단할 수 있다.

정탐코드의 예

```
1:  int validateUser(char *host, int port)
2:  {
3:    int socket = openSocketConnection(host, port);
4:    if (socket < 0)
5:    {
6:      printf("Unable to open socket connection");
7:      return(FAIL);
8:    }
9:
10: int isValidUser = 0;
11: char username[USERNAME_SIZE];
12: char password[PASSWORD_SIZE];
13:   while (isValidUser == 0)
14:   {
15:     if (getNextMessage(socket, username, USERNAME_SIZE) > 0)
```

정탐코드의 예

```
16:    {
17:      if (getNextMessage(socket, password, PASSWORD_SIZE) > 0)
18:      {
19:        isValidUser = AuthenticateUser(username, password);
20:       }
21:      }
22:    }
23:   return(SUCCESS);
24:  }
```

마. 참고자료

[1] CWE-307, Improper Restriction of Excessive Authentication Attempts, MITRE, http://cwe.mitre.org/data/definitions/307.html

[2] Blocking Brute Force Attacks, OWASP, https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

| 제 3 절 | 시간 및 상태

동시 또는 거의 동시 수행을 지원하는 병렬 시스템이나 하나 이상의 프로세스가 동작되는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안약점이다.

1. 경쟁조건: 검사 시점과 사용 시점(TOCTOU)

가. 개요

병렬시스템(멀티프로세스로 구현한 응용프로그램)에서는 자원(파일, 소켓 등)을 사용하기에 앞서 자원의 상태를 검사한다. 하지만, 자원을 사용하는 시점과 검사하는 시점이 다르기 때문에, 검사하는 시점(Time Of Check)에 존재하던 자원이 사용하던 시점(Time Of Use)에 사라지는 등 자원의 상태가 변하는 경우가 발생한다.

예를 들어, 프로세스 A와 B가 존재하는 병렬시스템 환경에서 프로세스 A는 자원사용(파일 읽기)에 앞서 해당 자원(파일)의 존재 여부를 검사(TOC) 한다. 이때는 프로세스 B가 해당 자원(파일)을 아직 사용(삭제)하지 않았기 때문에, 프로세스 A는 해당 자원(파일)이 존재한다고 판단한다. 그러나 프로세스 A가 자원 사용(파일읽기)을 시도하는 시점(TOU)에 해당 자원(파일)은 사용불가능 상태이기 때문에 오류 등이 발생할 수 있다.

이와 같이 하나의 자원에 대하여 동시에 검사시점과 사용시점이 달라 생기는 보안약점으로 인해 동기화 오류뿐만 아니라 교착상태 등과 같은 문제점이 발생할 수 있다.

나. 보안대책

공유자원(예: 파일)을 여러 프로세스가 접근하여 사용할 경우, 동기화 구문을 사용하여 한 번에 하나의 프로세스만 접근 가능하도록(synchronized, mutex 등) 하는 한편, 성능에 미치는 영향을 최소화하기 위해 임계코드 주변만 동기화 구문을 사용한다.

다. 코드예제

다음의 예제는 파일을 대한 읽기와 삭제가 두 개의 스레드에 동작하게 되므로 이미 삭제된 파일을 읽으려고 하는 레이스컨디션[16]이 발생할 수 있다.

16) 레이스컨디션(Race Condition): Race Condition은 두 개 이상의 프로세스가 공용 자원을 병행적으로(concurrently) 읽거나 쓸 때, 공용 데이터에 대한 접근이 어떤 순서에 따라 이루어졌는지에 따라 그 실행 결과가 달라지는 상황을 말한다.

안전하지 않은 코드의 예 JAVA

```
1: class FileMgmtThread extends Thread {
2:   private String manageType = "";
3:   public FileMgmtThread (String type) {
4:     manageType = type;
5:   }
     //멀티쓰레드 환경에서 공유자원에 여러프로세스가 사용하여 동시에 접근할 가능성이 있어 안전하지
     않다.
6:   public void run() {
7:     try {
8:       if (manageType.equals("READ")) {
9:         File f = new File("Test_367.txt");
10:        if (f.exists()) {
11:          BufferedReader br = new BufferedReader(new FileReader(f));
12:          br.close();
13:        }
14:      } else if (manageType.equals("DELETE")) {
15:        File f = new File("Test_367.txt");
16:        if (f.exists()) {
17:          f.delete();
18:      } else { … }
19:    }
20:  } catch (IOException e) { … }
21: }
22:}

23:public class CWE367 {
24: public static void main (String[] args) {
25:   FileMgmtThread fileAccessThread = new FileMgmtThread("READ");
26:   FileMgmtThread fileDeleteThread = new FileMgmtThread("DELETE");
    //파일의 읽기와 삭제가 동시에 수행되어 안전하지 않다.
27:   fileAccessThread.start();
28:   fileDeleteThread.start();
29: }
30:}
```

따라서 다음 예제와 같이 동기화 구문인 synchronized를 사용하여 공유자원 (Test_367.txt)에 대한 안전한 읽기/쓰기를 수행할 수 있도록 한다.

안전한 코드의 예 JAVA

```
1: class FileMgmtThread extends Thread {
2:   private static final String SYNC = "SYNC";
3:   private String manageType = "";
4:   public FileMgmtThread (String type) {
5:     manageType = type;
6: }
7: public void run() {
   //멀티쓰레드 환경에서 synchronized를 사용하여 동시에 접근할 수 없도록 사용해야한다.
8:   synchronized(SYNC) {
9:     try {
10:      if (manageType.equals("READ")) {
11:        File f = new File("Test_367.txt");
12:        if (f.exists()) {
13:          BufferedReader br
14:            = new BufferedReader(new FileReader(f));
15:          br.close();
16:        }
17:      } else if (manageType.equals("DELETE")) {
18:          File f = new File("Test_367.txt");
19:          if (f.exists()) {
20:            f.delete();
21:          } else { … }
22:        }
23:      } catch (IOException e) { … }
24:    }
25: }
26:}

27:public class CWE367 {
28: public static void main (String[] args) {
29:   FileMgmtThread fileAccessThread = new FileMgmtThread("READ");
```

안전한 코드의 예 JAVA

```
30:   FileMgmtThread fileDeleteThread = new FileMgmtThread("DELETE");
31:   fileAccessThread.start();
32:   fileDeleteThread.start();
33: }
34:}
```

다음의 C# 코드도 파일에 동시에 접근하는 레이스 컨디션이 발생할 수 있다.

안전하지 않은 코드의 예 C#

```
//멀티쓰레드 환경에서 동시에 접근할 수 없도록 사용해야한다.
1: public void ReadFile(String f)
2: {
3:    if(File.Exists(f))
4:    {
5:       File.ReadAllLines(f);
6:    }
7: }
```

아래와 같은 코드를 추가하여 레이스 컨디션을 방지해야 한다.

안전한 코드의 예 C#

```
//멀티쓰레드 환경에서 동시에 접근할 수 없도록 사용해야한다.
1: [MethodImpl(MethodImplOptions.Synchronized)]
2: public void ReadFile(String f)
3: {
4:    if(File.Exists(f))
5:    {
6:      File.ReadAllLines(f);
7:    }
8: }
```

아래 C 코드는 공유 자원 account에 대해 lock을 설정하지 않아 경쟁 조건이 발생할 수 있다. 입금과 출금이 빈번하게 발생하는 상황에서 경쟁 조건이 발생하면 account의 값이 달라진다. 아래 ex1은 정상적으로 deposit과 withdraw가 호출되는 상황이고 ex2는 경쟁 조건이 발생하는 상황이다. 최종 account의 값이 0과 -100으로 다른 것을 확인할 수 있다.

ex1) deposit(100) 호출 : (account: 0)
deposit(100) 종료 : (account: 100)
withdraw(100) 호출: (account: 100)
withdraw(100) 종료: (account: 0)

ex2) deposit(100) 호출 : (account: 0)
withdraw(100) 호출: (account: 0)
deposit(100) 종료 : (account: 100)
withdraw(100) 종료: (account: -100)

안전하지 않은 코드의 예 C

```
1: static volatile double account;

2: void deposit(int amount) {
   // lock 없이 공유 자원에 접근
3:   account += amount;
4: }

5: void withdraw(int amount) {
6:   account -= amount;
7: }
```

아래 C 코드는 mutex_lock()로 공유 자원에 대한 동시 접근을 제한한 것이다.

안전한 코드의 예 C

```
1: static volatile double account;
2: static mtx_t account_lock;

3: void deposit(int amount) {
```

안전한 코드의 예 C

```
// mutex_lock, mutex_unlock을 이용해 공유 자원 접근을 제한한다.
4:   mutex_lock(&account_lock);
5:   account += amount;
6:   mutex_unlock(&account_lock);
7: }

8: void withdraw(int amount) {

9:   mutex_lock(&account_lock);
10:   account -= amount;
11:   mutex_unlock(&account_lock);
12:}
```

라. 진단방법

소스코드 상에 다음과 같이 공유자원(파일/폴더, 소켓, 드라이버 등)을 여러 프로세스가 사용하는지 확인한다(①). 이 때, 하나의 공유자원을 동시에 접근할 가능성이 존재한다면 취약하다고 판단하며, 동기화 구문 등을 사용할 경우 안전하다고 판단한다. 그 외 공유자원 억세스를 관리하는 pool 형태의 자체 모듈을 제작할 경우에도 안전하다고 판단한다.

일반적인 진단의 예

```
1:   import java.io.*;
2:
3:   class FileAccessThread extends Thread {
4:     public void run() {
5:       try {
6:         File f = new File("Test_367.txt");  ····· ①
7:         if(f.exists()) {
8:           // 만약 파일이 존재하면 파일 내용을 읽음
9:           BufferedReader br = new BufferedReader(new FileReader(f));
10:          br.close();
11:       }
```

일반적인 진단의 예

```
12:  } catch(IOException e) {
13:    System.err.println("IOException occured");}
14:  }
15: }
16:
17: class FileDeleteThread extends Thread {
18:  public void run() {
19:    File f = new File("Test_367.txt");  ····· ①
20:    if (f.exists()) {
21:    // 만약 파일이 존재하면 파일을 삭제함
22:      f.delete();
23:    }
24:  }
25: }
26:
27: public class U367 {
28:  public static void main(String[] args) {
29:    // 파일의 읽기와 파일을 삭제하는 것을 동시에 수행한다.
30:    FileAccessThread fileAccessThread = new FileAccessThread(); ②
31:    FileDeleteThread fileDeleteThread = new FileDeleteThread();
32:    fileAccessThread.start();
33:    fileDeleteThread.start();
34:  }
35: }
```

다음의 예제와 같이 검사시점과 사용시점이 달라 발생하는 경쟁조건을 가정해 볼 수 있다.

일반적인 진단의 예

```
1:  File f = new File("toctou.txt");
2:  if (!f.exists()) {
3:    FileOutputStream fos = null;
4:    try {
5:      fos = new FileOutputStream("toctou.txt")
```

일반적인 진단의 예

```
6:    // 파일에 대한 처리를 한다.
7:  } catch (IOException e) {
8:    // TODO 에러 처리를 한다.
9:  } finally {
10:   // 자원을 해제한다.
11: }
```

하지만 자바에서는 스트림을 생성할 때 IOExceptin을 발생시키며 이에 따라 예외에 대한 적절한 처리를 하면 되므로 문제가 되지 않는다.

TOCTOU는 C 프로그래밍을 할 때 다음과 같은 경우에 발생한다.

정탐코드의 예

```
1: if (!access(file,W_OK))
2: {
3:    f = fopen(file,"w+");
4:    operate(f);
5:    …
6: }
7: else
8: {
9:  fprintf(stderr,"Unable to open file %s.\n",file);
10:}
```

실제로 경쟁 조건이 이루어지지 않는 경우 취약하지 않다.

오탐코드의 예

```
1:  file = new File(filename);
2:  File[] fList = file.listFiles();
3:  for (int i = 0; i < fList.length; i++) {
4:    currentList.add(fList[i].getAbsolutePath()
5:       + "$" + getLastModifiedTime(fList[i]) + "$"
```

오탐코드의 예

```
6:      + ((fList[i].length() / 1024) > 0 ? (fList[i].length() / 1024) : 1)
7:      + "KB");
8: }
```

마. 참고자료

[1] CWE-367 Time-of-check Time-of-use(TOCTOU) Race Condition, MITRE
http://cwe.mitre.org/data/definitions/367.html

[2] Avoid TOCTOU race conditions while accessing files, CERT
http://www.securecoding.cert.org/confluence/display/c/FIO45-C.+Avoid+TOCTOU+race+conditions+while+accessing+files

2. 종료되지 않는 반복문 또는 재귀 함수

가. 개요

재귀의 순환횟수를 제어하지 못하여 할당된 메모리나 프로그램 스택 등의 자원을 과다하게 사용하면 위험하다. 대부분의 경우, 귀납 조건(Base Case)이 없는 재귀함수는 무한 루프에 빠져 들게 되고 자원고갈을 유발함으로써 시스템의 정상적인 서비스를 제공할 수 없게 한다.

나. 보안대책

모든 재귀 호출시, 재귀 호출 횟수를 제한하거나, 초기값을 설정(상수)하여 재귀 호출을 제한해야 한다.

다. 코드예제

factorial 함수는 함수 내부에서 자신을 호출하는 재귀함수로, 재귀문을 빠져 나오는 조건을 정의하고 있지 않아 무한 재귀에 빠져 시스템 장애를 유발할 수 있다.

안전하지 않은 코드의 예 C

```
1: #include <stdio.h>

2: int factorial(int i)
3: {
   //재귀함수 탈출 조건을 설정하지 않아 무한루프가 된다.
4:    return i * factorial(i - 1);
5: }

6: int main()
7: {
8:   int num = 5;
9:   int result = factorial(num);
10:   printf("%d! : %d\n", num, result);
11:   return 0;
12:}
```

재귀 함수를 구현할 때는 아래와 같이 재귀문을 빠져 나오는 조건인 귀납조건(Base case)을 반드시 구현해야 한다.

안전한 코드의 예 C

```
1: #include <stdio.h>

2: int factorial(int I)
3: {
   //재귀함수 사용시에는 아래와 같이 탈출 조건을 사용해야 한다.
4:    if (i <= 1) {
5:       return 1;
6:    }
7:    return i * factorial(i - 1);
8: }

9: int main()
10:{
11:   int num = 5;
12:   int result = factorial(num);
13:   printf("%d! : %d\n", num, result);
14:    return 0;
15:}
```

라. 진단방법

자신을 호출하는 재귀함수나 메소드가 존재하는지 식별하고, 해당 함수 내에 제어문으로 해당 함수에서 리턴될 수 있는지 확인한다. 제어문으로 재귀함수를 빠져나올 수 있으면 안전하다.

일반적인 진단의 예

```
1: …
2: public int factorial(int n) {
3:                 return n * factorial(n - 1);  ……①
4: }
5: …
```

재귀함수는 함수내부에서 자신을 호출하는 경우이다. 함수명과 파라미터의 개수, 파라미터의 자료형이 일치하는 경우 취약하다.

정탐코드의 예

```
1: public class RecursiveCall {
2:   private int a;
3:
4:   public void func(int a) {
5:     this.a = a;
6:   }
7:
8:   public void func(String a) {
9:     func(a);
10: }
11:}
```

다음의 예제와 같이 같은 이름에 파라미터가 다른 함수가 있는 경우는 취약하지 않다. 다음의 예제는 func(String a)에서 func(int a)를 호출하고 있는데 recursive call 이 아닌 다른 함수의 호출이므로 안전하다.

오탐코드의 예

```
1: public class RecursiveCall {
2:   private int a
3:
4:   public void func(int a) {
5:     this.a = a;
6:   }
7:
8:   public void func(String a) {
9:     func(Integer.parseInt(a));
10: }
11:}
```

제어조건이 존재하여 제어가 되는 재귀인 경우 취약하지 않다.

오탐코드의 예

```
1: public MenuVO getFirstLeafChildMenu(int menuSeq) {
2:   List<MenuVO> childMenuList = menuMap.get(menuSeq).getChildMenuList();
3:   if (CollectionUtils.isEmpty(childMenuList)) {
4:     return menuMap.get(menuSeq);
5:   }
6:   return getFirstLeafChildMenu(childMenuList.get(0).getMenuSeq());
7: }
```

마. 참고자료

[1] CWE-674 Uncontrolled Recursion, MITRE,
http://cwe.mitre.org/data/definitions/674.html

[2] CWE-835, Loop with Unreachable Exit Condition ('Infinite Loop'), MITRE,
http://cwe.mitre.org/data/definitions/835.html

| 제 4 절 | 에러처리

에러를 처리하지 않거나, 불충분하게 처리하여 에러 정보에 중요정보(시스템 내부정보 등)가 포함될 때, 발생할 수 있는 취약점으로 에러를 부적절하게 처리하여 발생하는 보안약점이다.

1. 오류 메시지 정보노출

가. 개요

응용프로그램이 실행환경, 사용자 등 관련 데이터에 대한 민감한 정보를 포함하는 오류 메시지를 생성하여 외부에 제공하는 경우, 공격자의 악성 행위를 도울 수 있다. 예외발생시 예외이름이나 스택 트레이스를 출력하는 경우, 프로그램 내부구조를 쉽게 파악할 수 있기 때문이다.

나. 보안대책

오류 메시지는 정해진 사용자에게 유용한 최소한의 정보만 포함하도록 한다. 소스코드에서 예외상황은 내부적으로 처리하고 사용자에게 민감한 정보를 포함하는 오류를 출력하지 않도록 미리 정의된 메시지를 제공하도록 설정한다.

다. 코드예제

다음 예제는 오류 메시지에 예외 이름이나 오류추적 정보를 출력하여 프로그램 내부 정보가 유출되는 경우이다.

안전하지 않은 코드의 예 JAVA

```
1: try {
2:     rd = new BufferedReader(new FileReader(new File(filename)));
3: } catch(IOException e) {
   // 에러 메시지로 스택 정보가 노출됨
4:   e.printStackTrace();
5: }
```

안전하지 않은 코드의 예 JAVA

```
1: } catch(IOException e) {
   // 오류발생시 화면에 출력된 시스템 정보로 다른 공격의 빌미를 제공한다.
2:    System.err.print(e.getMessage());
3: }
```

아래 코드와 같이 예외 이름이나 오류추적 정보를 출력하지 않도록 한다.

안전한 코드의 예 JAVA

```
1: try {
2:     rd = new BufferedReader(new FileReader(new File(filename)));
3: } catch(IOException e) {
   // 에러 코드와 정보를 별도로 정의하고 최소 정보만 로깅
4:    logger.error("ERROR-01: 파일 열기 에러");
5: }
```

다음 예제는 오류 메시지에 예외 이름이나 오류추적 정보를 출력하여 프로그램 내부 정보가 유출되는 C#코드이다.

안전하지 않은 코드의 예 C#

```
1: try
2: {
3:    //do something
4: }
5: catch (CustomException e)
6: {
7:    Console.WriteLine(e);
8: }
```

예외 관련한 최소한의 정보만 출력하도록 한다.

안전한 코드의 예 C#

```
1: try
2: {
3:    //do something
4: }
5: catch (CustomException e)
6: {
7:    _log.Debug("ERROR-01 : error information");
8: }
```

라. 진단방법

해당 취약점에서 시스템 환경, 유저정보, 민감한 정보 등에 대한 기준을 정적도구가 판단하기 어려움에 따라 진단원이 출력함수 등으로 외부에 출력되는 값 중 민감한 정보 등을 판단할 필요가 있다. 오류메시지를 출력하는 경우(①) 해당오류에 시스템 환경, 유저정보, 데이터 등 민감한 정보가 포함되어 있는지 확인한다.

일반적인 진단의 예

```
1: public static void main(String[] args) {
2:   String urlString = args[0];
3:   try {
4:     URL url = new URL(urlString);
5:     URLConnection cmx =
6:     url.openConnection();
7:     cmx.connect();
8:   } catch (Exception e) {
9:     e.printStackTrace(); ····· ①
10: }
11:}
```

오류 메시지로 환경, 사용자, 관련 데이터 등의 내부 정보가 유출될 경우 취약하다.

정탐코드의 예

```
1: <%@ page language="java" contentType="text/html; charset=UTF-8" buffer="-
2: none"%>
3: <%@page import="egovframework.com.utl.fda.ucc.service.EgovUnitCalcUtil" %>
4: <%
5:   String sCmd = request.getParameter("cmd") == null ? "" : (String)
     request.getParam-
6: eter("cmd");
7:   double nResult=0.0;
8:   try {
9:    if(!sCmd.equals("")) {
10:    EgovUnitCalcUtil egovUnitCalcUtil= new EgovUnitCalcUtil();
```

정탐코드의 예

```
11:   ….
12: }
13:   ……
14: }
15:} catch(Exception e) {
16: e.printStackTrace();
17:}
18:%>
```

오류 메시지로 환경, 사용자, 관련 데이터 등의 내부 정보가 유출될 경우 취약하다.

오탐코드의 예

```
1: BufferedReader br = null;
2: try {
3:   br = new BufferedReader(new InputStreamReader(new FileInputStream(file), "UTF
     8"));
4:   String line = null;
5:   while ((line = br.readLine()) != null) {
6:     // …
7:     }
8:   } catch (IOException e) {
9:     logger.error(e, e);
10: } finally {
11:     if (br != null) {
12:       try {
13:         br.close();
14:       } catch (IOException e) {
15:           logger.error(e, e);
16:       }
17:     }
18: }
```

마. 참고자료

[1] CWE-209 Information Exposure Through an Error Message, MITRE, http://cwe.mitre.org/data/definitions/209.html

[2] Do not allow exceptions to expose sensitive information, CERT, http://www.securecoding.cert.org/confluence/display/java/ERR01-J.+Do+not+allow+exceptions+to+expose+sensitive+information?focusedCommentId=61702253#comment-61702253

[3] Error Handling, OWASP https://www.owasp.org/index.php/Error_Handling

2. 오류상황 대응 부재

가. 개요

오류가 발생할 수 있는 부분을 확인하였으나, 이러한 오류에 대하여 예외 처리를 하지 않을 경우, 공격자는 오류 상황을 악용하여 개발자가 의도하지 않은 방향으로 프로그램이 동작하도록 할 수 있다.

나. 보안대책

오류가 발생할 수 있는 부분에 대하여 제어문을 사용하여 적절하게 예외 처리(C/C++에서 if와 switch, Java에서 try-catch 등)를 한다.

다. 코드예제

다음 예제는 try 블록에서 발생하는 오류를 포착(catch)하고 있지만, 그 오류에 대해서 아무 조치를 하고 있지 않음을 보여준다. 아무 조치가 없으므로 프로그램이 계속 실행되기 때문에 프로그램에서 어떤 일이 일어났는지 전혀 알 수 없게 된다.

안전하지 않은 코드의 예 JAVA

```
1: protected Element createContent(WebSession s) {
2:           ……
3:    try {
4:       username = s.getParser().getRawParameter(USERNAME);
5:       password = s.getParser().getRawParameter(PASSWORD);
6:       if (!"webgoat".equals(username) || !password.equals("webgoat")) {
7:           s.setMessage("Invalid username and password entered.");
8:           return (makeLogin(s));
9:       }
10:   } catch (NullPointerException e) {
   //요청 파라미터에 PASSWORD가 존재하지 않을 경우 Null Pointer Exception이 발생하고 해당
     오류에 대한 대응이 존재하지 않아 인증이 된 것으로 처리
11: }
```

예외를 포착(catch)한 후, 각각의 예외 사항(Exception)에 대하여 적절하게 처리해야 한다.

안전한 코드의 예 JAVA

```
1: protected Element createContent(WebSession s) {
2:         ……
3:    try {
4:       username = s.getParser().getRawParameter(USERNAME);
5:       password = s.getParser().getRawParameter(PASSWORD);
6:       if (!"webgoat".equals(username) || !password.equals("webgoat")) {
7:          s.setMessage("Invalid username and password entered.");
8:          return (makeLogin(s));
9:       }
10:   } catch (NullPointerException e) {
      //예외 사항에 대해 적절한 조치를 수행하여야 한다.
11:      s.setMessage(e.getMessage());
12:      return (makeLogin(s));
13:   }
```

다음의 C# 코드도 예외상황에 대한 조치가 없다.

안전하지 않은 코드의 예 C#

```
1: try {
2:    InvokeMtd();
3:  } catch (CustomException e) {
4:    //예외 상황에 대한 대응 부재
5: }
```

각각의 예외 상황에 대해 적절한 조치를 수행해야 한다.

안전한 코드의 예 C#

```
1: try {
2:    InvokeMtd();
```

안전한 코드의 예 C#

```
3: } catch (CustomException e) {
      //예외 상황에 대해 적절한 조치를 수행하여야 한다.
4:   logger.Debug("log message");
5: }
```

라. 진단방법

오류가 발생할 수 있는 부분에 대하여 예외처리를 수행했는지 확인하고 제어문으로 예외 처리하는 루틴이 비어있는지 확인한다.

다음의 예제는 try 블록에서 발생하는 오류를 포착(catch)하고 있지만 그 오류에 대해서 아무 조치를 하고 있지 않다. 따라서 프로그램이 계속 실행되기 때문에 프로그램에서 어떤 일이 일어났는지 전혀 알 수 없게 된다.

일반적인 진단의 예

```
1: …
2: private Connection conn;
3:
4: public Connection DBConnect(String url, String id, String password) {
5:   try {
6:     String CONNECT_STRING = url + ":" + id + ":" + password;
7:     InitialContext ctx = new InitialContext();
8:     DataSource datasource = (DataSource) ctx.lookup(CONNECT_STRING);
9:     conn = datasource.getConnection();
10: } catch (SQLException e) {
11:   // catch 블록이 비어있음
12: } catch (NamingException e) {
13:   // catch 블록이 비어있음
14: }
15: return conn;
16:}
```

마. 참고자료

[1] CWE-390 Detection of Error Condition Without Action, MITRE, http://cwe.mitre.org/data/definitions/390.html

[2] Do not suppress or ignore checked exceptions, CERT, http://www.securecoding.cert.org/confluence/display/java/ERR00-J.+Do+not+suppress+or+ignore+checked+exceptions

3. 부적절한 예외 처리

가. 개요

프로그램 수행 중에 함수의 결과값에 대한 적절한 처리 또는 예외상황에 대한 조건을 적절하게 검사하지 않을 경우, 예기치 않은 문제를 야기할 수 있다.

나. 보안대책

값을 반환하는 모든 함수의 결과값을 검사하여, 그 값이 의도했던 값인지 검사하고, 예외 처리를 사용하는 경우에 광범위한 예외 처리 대신 구체적인 예외 처리를 수행한다.

다. 코드예제

다음 예제는 try 블록에서 다양한 예외가 발생할 수 있음에도 불구하고 예외를 세분화하지 않고 광범위한 예외 클래스인 Exception을 사용하여 예외를 처리하고 있다.

안전하지 않은 코드의 예 JAVA

```
1: try {
2:     ...
3:   reader = new BufferedReader(new InputStreamReader(url.openStream()));
4:   String line = reader.readLine();
5:   SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
6:   Date date = format.parse(line);
     //예외처리를 세분화 할 수 있음에도 광범위하게 사용하여 예기치 않은 문제가 발생 할 수 있다.
7: } catch (Exception e) {
8:     System.err.println("Exception : " + e.getMessage());
9: }
```

발생 가능한 예외를 세분화하고 발생 가능한 순서에 따라 예외를 처리하고 있다.

안전한 코드의 예 JAVA

```
1: try {
2:     ...
3:    reader = new BufferedReader(new InputStreamReader(url.openStream()));
4:    String line = reader.readLine();
5:    SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
6:    Date date = format.parse(line);
     // 발생할 수 있는 오류의 종류와 순서에 맞춰서 예외 처리 한다.
7:  } catch (MalformedURLException e) {
8:     System.err.println("MalformedURLException : " + e.getMessage());
9:  } catch (IOException e) {
10:    System.err.println("IOException : " + e.getMessage());
11: } catch (ParseException e) {
12:    System.err.println("ParseException : " + e.getMessage());
13: }
```

다음 예제는 try 블록에서 다양한 예외가 발생할 수 있음에도 불구하고 예외를 세분화하지 않고 광범위한 예외 클래스인 Exception을 사용하여 예외를 처리하고 있다.

안전하지 않은 코드의 예 C#

```
1: try {
2:     InvokeMtd();
//예외처리를 세분화 할 수 있음에도 광범위하게 사용하여 예기치 않은 문제가 발생 할 수 있다.
3:  } catch (Exception e) {

4: }
```

발생 가능한 예외를 세분화하고 발생 가능한 순서에 따라 예외를 처리하고 있다.

안전한 코드의 예 C#

```
1: try {
2:     InvokeMtd();
// 발생할 수 있는 오류의 종류와 순서에 맞춰서 예외 처리 한다.
3:  } catch (IOException e) {
4:  logger.Debug("IOException log here");
5: } catch (SQLException e){
6:  logger.Debug("SQLException log here");
7: }
```

라. 진단방법

함수 또는 메소드에 대하여 반환값을 검사하고 예외를 발생시키는 경우(①) 구체적인 예외처리를 수행하는지 확인한다.

일반적인 진단의 예

```
1: public void readFromFile(String fileName) {
2:  try {
3:   …
4:            File myFile = new File(fileName);
5:            FileReader fr = new FileReader(myFile);
6:            …
7:   } catch(Exception ex){…}  ····· ①
8:  }
```

마. 참고자료

[1] CWE-754 Improper Check for Unusual or Exceptional Conditions, MITRE
http://cwe.mitre.org/data/definitions/754.html

[2] Do not complete abruptly from a finally block, CERT,
http://www.securecoding.cert.org/confluence/display/java/ERR04-J.+Do+not+complete+abruptly+from+a+finally+block

[3] Exception Handling in Spring MVC, Spring,
http://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc

| 제 5 절 | 코드오류

타입 변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩오류로 인해 유발되는 보안약점이다.

1. Null Pointer 역참조

가. 개요

널 포인터(Null Pointer) 역참조는 '일반적으로 그 객체가 널(Null)이 될 수 없다'라고 하는 가정을 위반했을 때 발생한다. 공격자가 의도적으로 널 포인터 역참조를 발생시키는 경우, 그 결과 발생하는 예외 상황을 이용하여 추후의 공격을 계획하는 데 사용될 수 있다.

나. 보안대책

널이 될 수 있는 레퍼런스(Reference)는 참조하기 전에 널 값인지를 검사하여 안전한 경우에만 사용한다.

다. 코드예제

다음의 예제의 경우 obj가 null이고, elt가 null이 아닌 경우 널(Null) 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 JAVA

```
1: public static int cardinality (Object obj, final Collection col) {
2:     int count = 0;
3:     if (col == null) {
4:         return count;
5:     }
6:     Iterator it = col.iterator();
7:     while (it.hasNext()) {
8:         Object elt = it.next();
```

안전하지 않은 코드의 예 JAVA

```
        //obj가 null이고 elt가 null이 아닐 경우, Null.equals 가 되어 널(Null) 포인터 역참조가 발생한다.
9:        if ((null == obj && null == elt) || obj.equals(elt)) {
10:          count++;
11:       }
12:    }
13:    return count;
14: }
```

obj가 null인지 검사 후 참조해야 한다.

안전한 코드의 예 JAVA

```
1: public static int cardinality (Object obj, final Collection col) {
2:    int count = 0;
3:    if (col == null) {
4:        return count;
5:    }
6:    Iterator it = col.iterator();
7:    while (it.hasNext()) {
8:       Object elt = it.next();
         //obj가 null이 아닌 경우에만 obj.equal를 실행한다.
9:       if ((null == obj && null == elt) || (null != obj && obj.equals(elt))) {
10:          count++;
11:      }
12:   }
13:   return count;
14:}
```

다음 예제의 경우 request.getParameter에 의해 null이 들어오게 되면 널(Null) 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 JAVA

```
1: String url = reuqest.getParamter("url");
2: //url 에 null이 들어오면 널(Null) 포인터 역참조가 발생한다.
3: if (url.equals(""))
```

null을 가질 수 있는 참조 변수를 사용해 객체의 속성이나 메소드를 사용하는 경우 null 검사를 수행하고 사용한다.

안전한 코드의 예 JAVA

```
1: String url = reuqest.getParamter("url");
2: //null값을 가지는 참조 변수를 사용할 경우, null 검사를 수행하고 사용한다.
3: if ( url != null || url.equals("") )
```

다음 예제의 경우 Request 객체에서 QeuryString을 사용하여 url의 파라미터 중 name에 해당하는 값을 가져오는 코드이다. url의 파라미터에 name이 없으면 QueryString[“name”]은 null을 리턴하게 되고, username은 null 값을 가지게 되어 널(Null) 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 C#

```
1: protected void Page_Load(object sender, EventArgs e) {
2:   // url 파라미터에 name이 없으면 username은 null 값을 가지게 된다.
3:   string username = Request.QueryString[“name”];
4:   // null 값을 가지는 username을 참조하여 널(Null) 포인터 역참조가 발생한다.
5:   if (username.Length 〉 20) {
6:     // length error
7:   }
8: }
```

null을 가질 수 있는 참조 변수를 사용해 객체의 속성이나 메소드를 사용하는 경우 null 검사를 수행하고 사용한다.

안전한 코드의 예 C#

```
1: protected void Page_Load(object sender, EventArgs e) {
2:   // url 파라미터에 name 이 없으면 username은 null 값을 가지게 된다.
3:   string username = Request.QueryString[“name”];
4:   // null 값을 가지는 username을 참조하기 전에 null 검사를 수행하므로 안전하다.
5:   if ( username != null && username 〉 20) {
6:     // length error
7:   }
8: }
```

아래 C 코드는 null 값을 반환할 수 있는 함수 IntegerAddressReturn()을 호출한다. P가 null인 상태에서 p 값을 참조하면 널 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 C

```
1: void NullPointerDereference(int count) {
// IntegerAddressReturn()이 0을 return 하면 p는 null 값을 가지게 된다.
2: int *p = IntegerAddressReturn();
// null 값을 가지는 p 값을 참조하여 널(Null) 포인터 역참조가 발생한다.
3:  *p = count;
4: }
```

아래 C 코드는 null 값을 가질 수 있는 p를 참조하기 전에 null 검사를 진행하므로 안전한다.

안전한 코드의 예 C

```
1: void NullPointerDereference(int count) {
// IntegerAddressReturn()이 0을 return 하면 p는 null 값을 가지게 된다.
2:  int *p = IntegerAddressReturn();
// 참조하기전에 null 검사를 수행하므로 안전하다.
3:  If(p != 0) *p = count;
```

라. 진단방법

표현된 객체가 널(Null) 값이 될 수 있는지 확인한다. 만약 널(Null) 값이 될 수 있는지 체크하여 예외처리를 한 경우 안전으로 판단하고 널(Null) 체크를 하지 않은 경우 취약한 것으로 판단한다. (①)

일반적인 진단의 예

```
1: …
2: public void f() {
3:  String cmd = System.getProperty("cmd");
4:  // cmd가 널(null)인지 체크하지 않았다.
5:  cmd = cmd.trim();
6:  System.out.println(cmd);  ····· ①
7: …
```

※ (참고) 널(Null) 값이 될 가능성은 다음의 기준으로 확인

- 초기값이 널(Null)로 할당된 값은 널(Null)이 될 수 있음
- 선언이 된 후 객체가 생성되지 않은 값은 널(Null)이 될 수 있음
- 널(Null)인 객체로 필드 접근을 하거나/널(Null)인 객체에 메소드 호출을 할 경우 결과 값은 널(Null)이 될 수 있음
- 문자열 등 '널(Null)이 될 수 있는(Nullable)' 객체들의 연산 결과 값은 널(Null)이 될 수 있음

변수를 널(Null)로 초기화 한 후 값을 대입하지 못하는 경우에 해당 변수를 참조하면 널(Null) 포인터 역참조가 발생한다.

다음의 예제는 3번째 라인에서 널(Null)로 초기화 한 이후에 14번째 라인에서 널(Null)값인 vmrs 값을 참조하고 있다. 이 경우는 조건문 등에 의해 분기가 이루어지면서 널(Null)로 초기화된 변수가 초기화 되지 않는 경우이다.

정탐코드의 예

```
1: public Hashtable deleteAll(Connection con, GenericModel model) throws Exception {
2:   Hashtable<String, Object> m = new Hashtable<String, Object>();
3:   VMResultSet vmrs = null;
4: int rValue = 0;
5: try {
6: ArrayList grid =null ;
7: ….
8: con.commit();
9: con.setAutoCommit(true);
10:if(rValue > 0) {
11:vmrs = dao.listGoodKnowQuestion(con, model);
12:vmrs.setMessage(UtilMsg.getInstance().getMsg("SUC003"));
13:} else {
14:vmrs.setMessage(UtilMsg.getInstance().getMsg("ERR000"));
15:}
16:m.put("result", vmrs); 17: } catch (Exception e) { 18: ….
19:} finally {
20:….
21:}
22:return m;
23:}
```

다른 유형으로 자주 발생하는 경우는 database의 Connection, Statement 등의 자원을 사용한 후에 해제하는 과정에서 해당 자원이 널(Null) 인지 확인하지 않는 경우이다.

다음의 예제에서는 4번째 라인에서 Statement를 널(Null)로 초기화하고 있고 85번째 라인에서 예외가 발생하면 10번 라인의 Statement의 값을 대입하지 못하고 20번의 finally 문으로 분기한 후 25번 라인이 실행되면서 널(Null) 값을 참조하게 된다.

정탐코드의 예

```
1: private static void makeList() {
2:   Connection con = null;
3:   StringBuffer sql = new StringBuffer();
4:     PreparedStatement statement = null;
5:     ResultSet rs = null;
6:     list.clear();
7:     try {
8:       con = GenericDAO.getDataSource().getConnection();
9:     ……..
10:    statement = con.prepareStatement(sql.toString());
11:    rs = statement.executeQuery();
12:    while (rs.next()) {
13:      String authGrp = rs.getString("AUTHORITY_GROUP");
14:      …….
15:    }
16:  } catch (Exception e) {
17:    if (verbose) {
18:      ……..
19:    }
20:  } finally {
21:    try {
22:      rs.close();
23:    } catch (Exception e1) {}
24:    try {
25:      statement.close();
26:    } catch (Exception e1) {}
27:      if (con != null) {
```

정탐코드의 예

```
28:      try {
29:        con.close();
30:      } catch (SQLException e) {
31:        e.printStackTrace();
32:     }
33:   }
34: }
35:}
```

enter2br()에서 data가 널(Null)일 경우 널(Null)을 리턴하기 때문에 strContent가 널(Null) 값을 가질 가능성이 존재한다. 따라서 5번 라인에서 널(Null) Pointer 역참조가 발생할 수 있기 때문에 취약하다고 판정한다.

정탐코드의 예

```
1: [program_view.jsp]
2: …
3: String strContent = UTIL.enter2br((String)hsROW.get("content"));
4: String linkPage= "";
5: if (!strContent.equals("")) {
6: linkPage = strContent.replaceAll("상세보기", "<img src='
7: …

1: [Util.java]
2: public static String enter2br(String data) {
3: if(data == null)
4: return null;
5: StringTokenizer st = new StringTokenizer(data, "\n");
6: …
7: }
```

널(Null) 검사를 수행하였으나, 조건문에서 널(Null) 객체를 참조하는 구문을 먼저 비교하여 널(Null) Pointer 역참조가 발생하여 취약하다고 판정한다.

정탐코드의 예

```
1:  1: if ( str.length() == 0 || str == null )
```

Interface Iteration은 널(Null)을 리턴하지 않는데 널(Null) 포인터 역참조를 탐지 해 내는 경우나, 함수의 설계상 널(Null)을 리턴 하지 않도록 정의되어 있는 함수의 리턴 값을 참조할 경우에는 널(Null) 포이터 역참조가 발생하지 않는다. 이를 확인하기 위해서는 함수의 Java document를 유의 깊게 살펴보아야 한다. 함수의 리턴 값에 대한 기술되어 있는 Javadoc을 참조해서 코드를 검토하여야 한다.

오탐코드의 예

```
1:  String Tokenizer st = new String Tokenizer(info_url,"?");
2:  int sti=0;
3:  while (st.hasMoreTokens()) {
4:   if(sti==0)
5:   jsonUrl=st.nextToken();
6:   if(sti==1)
7:   jsparam=st.nextToken();
8:   sti++;
9: }
10:String [] jsonParams = jsparam.split("&");
11:jsparam="";
12:for (int i=0;i<jsonParams.length;i++) {
13: jsparam=jsparam+jsonParams[i].split("=")[0]+": '"+jsonParams[i].split("=")[1]+"'";
14: if(i!=jsonParams.length-1)jsparam=jsparam+",";
15:}
```

try ~ catch 문에서 throw를 하는 경우 catch 절 이후는 실행되지 않음으로 널(Null) 포인터 역참조가 발생하지 않는다.

다음의 예제에서 System.out.println(“After Throw 1”);은 실행되지 않는다.

오탐코드의 예

```
1: public static void testFunction(String dateStr1) throws Exception {
2:   SimpleDateFormat sdf = new SimpleDateFormat(“yyyyMMdd”,Locale.getDefault());
3:   Date date1 = null;
4:   try {
5:     date1 = sdf.parse(dateStr1);
6:   } catch (ParseException e) {
7:     System.err.println(“Inner Function”);
8:     e.printStackTrace();
9:     thrownew Exception(e);
10:  }
11:  System.out.println(“After Throw 1”);
12:  int days1 = (int)((date1.getTime()/100000)/24);
13:  System.out.println(“After Throw 2”);
14:  }
```

다음의 예제에서 intdays1 = (int)((date1.getTime()/100000)/24);에서 data1이 널(Null)일 경우, 널 (Null) 포인터 역참조가 발생하고 다음 단계로 진행되지 않게 된다. 이러한 이유로 intdays2 = (int)((date1.getTime()/100000)/24); 문은 실행되지 않으며 널(Null)을 참조하지 않게 된다.

오탐코드의 예

```
1:SimpleDateFormat sdf = new SimpleDateFormat(“yyyyMMdd”,Locale.getDefault());
2:Date date1 = null;
3:try {
4:date1 = sdf.parse(dateStr1);
5:} catch (ParseException e) {
6:System.err.println(“Inner Function”);
7:e.printStackTrace();
8:}
9:  System.out.println(“After Throw 1”);
10:intdays1 =
(int)((date1.getTime()/100000)/24);
11: intdays2
= (int)((date1.getTime()/100000)/24); 12:
System.out.println(“After Throw 2”);
```

다음과 같은 조건은 앞의 널(Null) 체크 조건이 참이면 뒤의 OR 조건은 판단하지 않는다. 그러므로 뒤의 문장에서 널(Null) 포이터 역참조는 발생하지 않는다.

오탐코드의 예

```
1: if (a == null || a.length() == 0) {
2:                                    // 처리한다.
3: }
```

다음의 예제에서는 catch 절안에서 return 하고 있으므로 예외가 발생하여 f에 값을 넣지 못하더라도 f.write0는 실행되지 않는다.

오탐코드의 예

```
1:  FileOutputStream f = null;
2:  try {
3:   f = new FileOutputStream("toctou");
4:   f.write();
5: } catch (FileNotFoundException e) {
6:   // TODO Auto-generated catch block
7:   e.printStackTrace();
8: }
```

다음의 예제에서는 JSP 내장 객체를 참조하고 있다. 이 경우에는 널(Null) 포인터 역참조가 발생하지 않는다.

오탐코드의 예

```
1:  〈td align="center" colspan='2'〉〈input type='button' value='다시로그인'
    onclick="loca-
    tion.href='${pageContext.request.contextPath}/utl/sec/certLogin.do'"〉〈/td〉
```

다음과 같은 Integer 클래스의 static 함수호출은 Integer가 객체가 아니므로 널(Null)값을 참조하지 않는다.

오탐코드의 예

```
1:  1: Integer.parseInt(stringValue);
```

다음과 같이 Data Flow 상이 아닌 단일 함수에서 파라미터가 널(Null)일 경우는 취약하지 않은 것으로 판단한다. 해당 함수를 호출할 때 널(Null) 체크를 하도록 해야 한다.

오탐코드의 예

```
1: …
2:    public SmsConnection sendRequsest(SmsConnection smsConn) throws
      Exception
3:    // SMS 전송 요청
4:    SmsSender sender = null;
5:    SmsConnection result = null;
6:   try {
7:      sender = new SmsSender(smeConfigPath);
8:      sender.open();
9:      result = sendRequsest(smsConn, sender);
10:   } finally {
11:      if (sender != null) {
12:      sender.close();
13:      }
14:    }
15:   }
16:   smsConn.setResult(result.getResult());
17:   smsConn.setResultMessage(result.getResultMessage());
18:   return smsConn;
19:}
```

JRE 기본 패키지 클래스의 생성자의 경우, 널(Null) 을 반환 할 수 있다고 명시되어 있지 않은한 널(Null)을 리턴 한다고 판단하지 않는다.

오탐코드의 예

```
1:  if(infoList == null) infoList = new ArrayList();
2:  for(int i=0; containers != null && i < containers.length; i++) {
3:  hm = new HashMap();
4:  for(int j=0; j < paramRecvData.length; j++) {
5:   hm.put(paramRecvData[j], containers[i].getField(paramRecvData[j]).getValueAs-
    String());
6:  }
7:  infoList.add(hm);
8:  }
```

마. 참고자료

[1] CWE-476 NULL Pointer Dereference, MITRE, http://cwe.mitre.org/data/definitions/476.html

[2] Do not dereference null pointers, CERT, http://www.securecoding.cert.org/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers

[3] Null Dereference, OWASP, https://www.owasp.org/index.php/Null_Dereference

2. 부적절한 자원 해제

가. 개요

프로그램의 자원, 예를 들면 열린 파일디스크립터(Open File Descriptor), 힙 메모리(Heap Memory), 소켓(Socket) 등은 유한한 자원이다. 이러한 자원을 할당받아 사용한 후, 더 이상 사용하지 않는 경우에는 적절히 반환하여야 하는데, 프로그램 오류 또는 에러로 사용이 끝난 자원을 반환하지 못하는 경우이다.

나. 보안대책

자원을 획득하여 사용한 다음에는 반드시 자원을 해제하여 반환한다.

다. 코드예제

try구문 내 처리 중 오류가 발생할 경우, close()메서드가 실행되지 않아 사용한 자원이 반환되지 않을 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: InputStream in = null;
2: OutputStream out = null;
3:  try {
4:  in = new FileInputStream(inputFile);
5:  out = new FileOutputStream(outputFile);
6:  ...
7:  FileCopyUtils.copy(fis, os);
8:  //자원반환 실행 전에 오류가 발생할 경우 자원이 반환되지 않으며, 할당된 모든 자원을 반환해야
    한다.
9:  in.close();
10:  out.close();
11:} catch (IOException e) {
12:    logger.error(e);
13:}
```

예외상황이 발생하여 함수가 종료될 때, 예외의 발생 여부와 상관없이 항상 실행되는 finally 블록에서 할당받은 모든 자원을 반드시 반환하도록 한다.

안전한 코드의 예 JAVA

```
1: InputStream in = null;
2: OutputStream out = null;
3:  try {
4:    in = new FileInputStream(inputFile);
5:    out = new FileOutputStream(outputFile);
6:    ...
7:    FileCopyUtils.copy(fis, os);
8:  } catch (IOException e) {
9:     logger.error(e);
10: //항상 수행되는 finally 블록에서 할당받은 모든 자원에 대해 각각 null검사를 수행 후 예외처리를
    하여 자원을 해제하여야 한다.
11: } finally {
12:  if (in != null) {
13:   try {
14:      in.close();
15:   } catch (IOException e) {
16:      logger.error(e);
17:   }
18:  }
19:  if (out != null) {
20:      try {
21:         out.close();
22:    } catch (IOException e) {
23:       logger.error(e);
24:    }
25:  }
26:}
```

파일스트림이 해제되지 않는 C# 코드 예제이다.

안전하지 않은 코드의 예 C#

```
1: public void FileStreamTest()
2: {
3: // fsSource에 자원이 할당되었으나 해제되지 않습니다.
4:   FileStream fsSource = new FileStream(pathSource, FileMode.Open, FileAccess.Read);

5:   byte[] bytes = new byte[fsSource.Length];
6:   int numBytesToRead = (int)fsSource.Length;
7:   int numBytesRead = 0;

8:   while(numBytesToRead > 0)
9:   {
10:   int n = fsSource.Read(bytes, numBytesRead, numBytesToRead);
11:   if(n==0) break;
12:      numBytesToRead += n;
13:      numBytesToRead -= n;
14: }

15:   using(FileStream fsNew = new FileStream(pathNew, FileMode.Create,
    FileAccess.Write)) { /* OK */
16:    fsNew.Write(bytes, 0, numBytesToRead);
17:   }
18:}
```

using 구문을 이용하여 쉽게 자원을 해제할 수 있다.

안전한 코드의 예 C#

```
1: public void FileStreamTest()
2: {
```

안전한 코드의 예 C#

```
    //using 구문으로 자원을 할당하면 구문이 끝나는 지점에서 자동으로 자원이
    해제됩니다.
3:   using(FileStream fsSource = new FileStream(pathSource, FileMode.Open,
    FileAccess.Read)){

4:      byte[] bytes = new byte[fsSource.Length];
5:      int numBytesToRead = (int)fsSource.Length;
6:      int numBytesRead = 0;

7:      while(numBytesToRead > 0)
8:      {
9:        int n = fsSource.Read(bytes, numBytesRead, numBytesToRead);
10:         if(n==0)
11:           break;
12:           numBytesToRead += n;
13:           numBytesToRead -= n;
14:     }
15:   }
16:   using(FileStream fsNew = new FileStream(pathNew, FileMode.Create,
    FileAccess.Write)) { /* OK */
17:     fsNew.Write(bytes, 0, numBytesToRead);
18:   }
19:}
```

아래 C 코드는 파일을 연 상태에서 오류가 발생했을 때 자원 누수가 발생한다.

안전하지 않은 코드의 예 C

```
1: void ImproperResourceRelease(char* filename) {
2:   char buf[BUF_SIZE];
3:   FILE *f = fopen(filename, "r");
```

안전하지 않은 코드의 예 C

```
4:   if(!checkSomething()) {
5:     printf("Something is wrong");
6:     return;
7:   }
8:  // checkSomething에서 false를 반환하는 경우, 파일 핸들러를 종료할 수
    없습니다.
9:   fclose(f);
10:}
```

오류가 발생한 상황에서도 정상적으로 파일 핸들러를 종료하도록 수정한다.

안전한 코드의 예 C#

```
1: void ImproperResourceRelease(char* filename) {
2:   char buf[BUF_SIZE];
3:   FILE *f = fopen(filename, "r");
4:   if(!checkSomething()) {
5:       printf("Something is wrong");
6:       // checkSomthing에서 false를 반환해도 파일 핸들러를 종료하도록 수정
7:     fclose(f);
8:     return;
9:   }
10:  fclose(f);
11:}
```

라. 진단방법

자원(파일기술자, 힙메모리, 소켓)이 선언되고, 선언된 자원이 할당된 경우 해제되는지 확인한다. 진단자는 제어문, 예외처리문 등의 분기 등에 따라 모든 흐름(control flow)을 판단하여 자원해제 여부를 체크해야 한다. 할당된 자원이 해제될 경우 안전하다고 판단하고 자원이 해제되지 않는 분기가 존재할 경우 취약하다.

일반적인 진단의 예

```
1: …
2: try {
3:   Class.forName(“com.mysql.jdbc.Driver”);
4:   conn = DriverManager.getConnection(url);
5:   …
6: } catch (ClassNotFoundException e) {
7:    conn.rollback();
8: } finally {
9:     if(conn != null) conn.close();
10:}
```

다음과 같은 예제는 rs.close(), pstmt.close() 등 자원 해제가 try catch 문안에서 함께 이루어지고 있다. 이 경우 rs.close()문에서 예외가 발생하면 pstmt.close() 이하의 자원 해제는 이루어지지 않으므로 자원유출이 일어난다.

정탐코드의 예

```
1:  try {
2:   String sqlSelect = “SELECT USER_ID, PWD FROM USER”
3:   String sqlUpdate = “UPDATE USER SET PASSWD = ? WHERE USER_ID = ?”
4:   Class.forName(“oracle.jdbc.driver.OracleDriver”);
5:    conn =
    DriverManager.getConnection(“jdbc:oracle:thin:@192.168.1.1:1521:db”, “aaa”, “AAA”);
6:    pstmt = conn.prepareStatement(sqlSelect);
7:    pstmt1 = conn.prepareStatement(sqlUpdate);
8:    rs = pstmt.executeQuery();
9:    while( rs.next()) {
10:      System.out.println(“#################### user_id : “ +
```

정탐코드의 예

```
      rs.getString("US- ER_ID"));
11:      pstmt1.setString(1, CryptUtils.encrypt(rs.getString("PWD")));
12:      pstmt1.setString(2, rs.getString("USER_ID"));
13:      pstmt1.executeUpdate();
14:      pstmt1.clearParameters();
15:      System.out.println("#################### 01 : ");
16:   }
17:   conn.commit();
18:} catch(Exception e) {
19:   e.printStackTrace();
20:   try {
21:      conn.rollback();
22:   } catch(Exception ex) {}
23:} finally {
24:   try {
25:     rs.close();
26:     pstmt.close();
27:     pstmt1.close();
28:     conn.close();
29:     } catch(Exception e) { }
30:   }
```

다음의 예제와 같이 finally 절에서 connection을 close 해주고 있으므로 자원의 부적절한 반환이 이루어지지 않는다.

오탐코드의 예

```
1: try {
2:   String sqlSelect = "SELECT USER_ID, PWD FROM USER"
3:   String sqlUpdate = "UPDATE USER SET PASSWD = ? WHERE USER_ID = ?
4:   Class.forName("oracle.jdbc.driver.OracleDriver");
5:    conn =
   DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.1:1521:db",
   "aaa", "AAA");
```

오탐코드의 예

```
6:    pstmt = conn.prepareStatement(sqlSelect);
7:    pstmt1 = conn.prepareStatement(sqlUpdate);
8:    rs = pstmt.executeQuery();
9:    while( rs.next()) {
10:      System.out.println("#################### user_id : " + rs.getString("US
      ER_ID"));
11:      pstmt1.setString(1, CryptUtils.encrypt(rs.getString("PWD")));
12:      pstmt1.setString(2, rs.getString("USER_ID"));
13:      pstmt1.executeUpdate();
14:      pstmt1.clearParameters();
15:      System.out.println("#################### 01 : ");
16: }
17:  conn.commit();
18:} catch(Exception e) {
19:  e.printStackTrace();
20:  try {
21:   conn.rollback();
22: } catch (Exception ex) { }
23:} finally {
24: try {
25:  rs.close();
26: } catch (Exception e) { }
27: try {
28:  pstmt.close();
29: } catch(Exception e) { }
30: try {
31:  pstmt1.close();
32: } catch (Exception e) { }
33: try {
34:  conn.close();
35: } catch (Exception e) { }
36:}
```

함수의 역할이 자원을 리턴하는 경우에는 자원의 해제는 함수를 호출한 쪽에서 담당하므로 취약하지 않다.

오탐코드의 예

```
1: public class DBUtil {
2:   public static Connection getConnection() {
3:     Connection conn = null;
4:     try {
5:       Class.forName("oracle.jdbc.driver.OracleDriver");
6:       conn = DriverManager.getConnection(DbInfo.url,DbInfo.id, DbInfo.password);
7:     } catch (SQLException e) {
8:       // TODO Auto-generated catch block
9:       e.printStackTrace();
10:   } catch (ClassNotFoundException e) {
11:     // TODO Auto-generated catch block
12:     e.printStackTrace();
13:   }
14:   return conn;
15: }
16:}
```

마. 참고자료

[1] CWE-404 Improper Resource Shutdown or Release, MITRE, http://cwe.mitre.org/data/definitions/404.html

[2] Release resources when they are no longer needed, CERT, http://www.securecoding.cert.org/confluence/display/java/FIO04-J.+Release+resources+when+they+are+no+longer+needed

[3] Unreleased Resource, OWASP, https://www.owasp.org/index.php/Unreleased_Resource

3. 해제된 자원 사용

가. 개요

C언어에서 동적 메모리 관리는 보안 취약점을 유발하는 대표적인 프로그램 결함의 원인이다. 해제한 메모리를 참조하게 되면 예상치 못한 값 또는 코드를 실행하게 되어 의도하지 않은 결과가 발생하게 된다.

나. 보안대책

동적으로 할당된 메모리를 해제한 후 그 메모리를 참조하고 있던 포인터를 참조 추적이나 형 변환, 수식에서의 피연산자 등으로 사용하여 해제된 메모리에 접근하도록 해서는 안된다. 또한, 메모리 해제 후, 포인터에 널(Null)값을 저장하거나 다른 적절한 값을 저장하면 의도하지 않은 코드의 실행을 막을 수 있다.

다. 코드예제

다음 예제는 동적 변수 temp에 할당된 동적 메모리를 해제 후 다시 사용하고 있다. 이 경우 예상치 못한 임의의 프로그램이 수행되는 취약점을 유발할 수 있다.

안전하지 않은 코드의 예 C

```
1: int main(int argc, const char *argv[]) {
2:     char *temp;
3:     temp = (char *)malloc(BUFFER_SIZE);
4:     ……
5:     free(temp);
6:     //해제한 자원을 사용하고 있어 의도하지 않은 결과가 발생하게 된다.
7:     stmcpy(temp, argv[1], BUFFER_SIZE-1);
8: }
```

다음 예제와 같이 메모리를 해제하기 전에 할당한 메모리를 사용하는 작업을 수행하고 최종적으로 메모리를 해제한다.

안전한 코드의 예 C

```
1: int main(int argc,const char *argv[]) {
2:     char *temp;
3:     temp = (char *)malloc(BUFFER_SIZE);
4:     ……
5:     //할당된 자원을 최종적으로 사용하고 해제하여야 한다.
6:     stmcpy(temp,argv[1], BUFFER_SIZE-1);
7:     free(temp);
8: }
```

다음은 해제 후 사용과 관련된 안전하지 않은 코드의 예이다. 프로그램에서는 문자형으로 동적 할당된 메모리를 참조하는 포인터와 정수형 변수 data_type을 사용한다. 만일 data_type값이 val_1과 동일하면서 동시에 val_2와도 동일한 값이 된다면, 두 번째 조건문에서 이중 해제 문제가 발생한다.

안전하지 않은 코드의 예 C

```
1: char *data;
2: int data_type
3: if (data_type==val_1) { free(data); }
4:         ……
5: // 이미 해제된 자원을 이중 해제하여 문제가 발생한다.
6: if (data_type==val_2) { free(data); }
```

동적 할당된 포인터를 해제한 후에 NULL값으로 설정함으로써 동일한 메모리 할당에 대해서는 한번만 해제하도록 하여 이중 해제 문제를 방지한다.

안전한 코드의 예 C

```
1: char *data;
```

안전한 코드의 예 C

```
2: int data_type
3: if (data_type==val_1) {
4:     free(data);
5:     // 메모리를 해제한 후 항상 포인터에 NULL을 할당하여 이중 해제하더라도 무시되게 한다.
6:     data = NULL;
7: }
8:  ……
9:  if (data_type==val_2) {
10:  free(data);
11:   // 메모리를 해제한 후 항상 포인터에 NULL을 할당하여 이중 해제하더라도 무시되게 한다.
12:  data = NULL;
13: }
```

라. 진단방법

소스코드 상에 메모리나 파일과 같은 자원을 사용하는 코드를 사용하는지 확인한다. 이 때, 자원을 사용하는 코드의 앞에서 자원의 해제가 발생하면 취약하다고 판단하며, 자원의 해제 후 모든 경우의 제어 흐름이 자원을 사용하는 코드로 도달하지 않을 경우 안전하다고 판단한다. 또한 자원을 참조 하는 포인터변수에 연산 작업을 수행하는 경우 보다 정밀하게 올바른 값을 참조하는지 검사해야 하며, 사용이 완료된 포인터 변수의 초기화가 이루어지는지 확인한다.

다음 예제는 할당된 메모리를 해제한 후 다시 접근하는 예제로, 21번 라인에서 실행되는 if문에서 messageBody에 할당된 메모리를 해지하였지만, 32번 라인에서 다시 해당 변수에 접근하고 있으므로, 보안약점이 존재하는 코드라고 판단한다. 또한 사용이 완료된 변수는 반드시 널(Null) 값으로 초기화 해주도록 한다.

정탐코드의 예

```
1: #define FAIL 0
2: #define SUCCESS 1
3: #define ERROR -1
4: #define MAX_MESSAGE_SIZE 32
```

정탐코드의 예

```
5:
6: int processMessage(char **message)
7: {
8:    int result = SUCCESS;
9:    int length = getMessageLength(message[0]);
10:   char *messageBody;
11:
12:   if ((length > 0) && (length < MAX_MESSAGE_SIZE))
13:   {
14:     messageBody = (char*)malloc(length*sizeof(char));
15:     messageBody = &message[1][0];
16:     int success = processMessageBody(messageBody);
17:
18:     if (success == ERROR)
19:     {
20:       result = ERROR;
21:       free(messageBody);
22:     }
23: }
24: else
25: {
26:   printf("Unable to process message; invalid message length");
27:   result = FAIL;
28: }
29:
30: if (result == ERROR)
31: {
32:   logError("Error processing message", messageBody);
33: }
34: return result;
35: }
```

마. 참고자료

[1] CWE-416, Use After Free, MITRE,
http://cwe.mitre.org/data/definitions/416.html

[2] Do not access freed memory, CERT,
http://www.securecoding.cert.org/confluence/display/c/MEM30-C.+Do+not+access +freed+memory

[3] Using freed memory, OWASP,
https://www.owasp.org/index.php/Using_freed_memory

4. 초기화되지 않은 변수 사용

가. 개요

C 언어의 경우 스택 메모리에 저장되는 지역변수는 생성될 때 자동으로 초기화되지 않는다. 초기화되지 않은 변수를 사용하게 될 경우 임의 값을 사용하게 되어 의도하지 않은 결과를 출력하거나 예상치 못한 동작을 수행할 수 있다.

나. 보안대책

초기화되지 않은 스택 메모리 영역의 변수는 임의값이라 생각해서 대수롭지 않게 생각할 수 있으나 사실은 이전 함수에서 사용되었던 내용을 포함하고 있다. 공격자는 이러한 약점을 사용하여 메모리 에 저장되어 있는 값을 읽거나 특정 코드를 실행할 수 있다. 모든 변수를 사용 전에 반드시 올바른 초기값을 할당함으로서 이러한 문제를 예방한다.

다. 코드예제

다음 코드는 커서의 위치를 정하는 프로그램이다. switch문 안에서 초기화를 수행하도록 구현이 되어 있으나, default 부분에서 변수 x만 초기화하고 변수 y는 초기화되지 않았으므로 이 함수가 수행되기 전에 공격자가 이 변수에 원하는 값을 저장해 놓는다면 서비스 거부 공격도 가능하다.

안전하지 않은 코드의 예 C

```
1: //변수의 초기값을 지정하지 않을 경우 공격에 사용 될 수 있어 안전하지 않다.
2: int x, y;
3: switch(position) {
4:     case 0: x = base_position y = base_position beak;
5:     case 1: x = base_position + i y = base_position - i break;
6:     default: x=1; break;
7: }
8: setCursorPosition(x,y);
```

아래의 예제는 switch 문 안에 case 항목으로 존재하던 초기화 구문을 switch문 밖으로 꺼내어 변수를 올바르게 초기화 하고 있으므로 안전하다.

안전한 코드의 예 C

```
1: //변수의 초기값은 항상 지정하여야 한다.
2: int x=1, y=1;
3: switch(position) {
4:     case 0: x = base_position y = base_position beak;
5:     case 1: x = base_position + i y = base_position - i break;
6:     default: x=1; break;
7: }
8: setCursorPosition(x,y);
```

라. 진단방법

변수의 선언과 동시에 초기화가 이루어지는지 확인한다. C++ 언어와 같이 객체의 필드가 초기 값을 가지지 않는 경우에는 생성자로 필드의 초기화가 이루어지는지 확인한다. 이와 같은 작업이 이루어지지 않을 경우에는 기본적으로 취약하다고 판단한다.

다음 예제는 C++ 코드로, 디폴트 생성자에서 필드의 초기화가 이루어지지 않기 때문에 디폴트 생성자를 이용할 경우 필드는 임의의 값을 가진다. 따라서 다음 예는 초기화되지 않은 필드에 접근하여 값을 가져오게 되므로 보안약점이 존재하는 코드라고 진단할 수 있다.

정탐코드의 예

```
1: class Foo
2: {
3:   public:
4:     Foo() {} // default constructor, doesn't initialize a_
5:     Foo(int a) : a_(a) {} // constructor
6:     int get_a() const {return a_;}
7:   private:
8:     int a_;
9:   };
```

정탐코드의 예

```
10:
11:int main(void)
12:{
13: Foo foo1; // calls default
constructor 14:std::cout << foo1.get_a() <<
 std::endl; 15: return 0;
16:}
```

마. 참고자료

[1] CWE-457, Use of Uninitialized Variable, MITRE,
http://cwe.mitre.org/data/definitions/457.html

[2] Do not read uninitialized memory, CERT,
http://www.securecoding.cert.org/confluence/display/c/EXP33-C.+Do+not+read+uninitialized+memory

[3] Uninitialized Variable, OWASP,
https://www.owasp.org/index.php/Uninitialized_variable

5. 신뢰할 수 없는 데이터의 역직렬화

가. 개요

직렬화(Serialization)는 프로그램에서 특정 클래스의 현재 인스턴스 상태를 다른 서버로 전달하기 위해 클래스의 인스턴스 정보를 바이트 스트림으로 복사하는 작업으로, 메모리 상에서 실행되고 있는 객체의 상태를 그대로 복제하여 파일로 저장하거나 수신측에 전달하게 된다. 역직렬화(Deserialization)는 반대 연산으로 바이너리 파일이나 바이트 스트림으로부터 객체 구조로 복원하게 된다.

이 때, 송신자가 네트워크를 이용하여 직렬화된 정보를 수신자에게 전달하는 과정에서 공격자가 전송 또는 저장된 스트림을 조작할 수 있는 경우에는 신뢰할 수 없는 역직렬화를 이용하여 무결성 침해, 원격 코드 실행, 서비스 거부 공격 등이 발생 할 수 있는 보안약점이다.

나. 보안대책

초기화되지 않은 스택 메모리 영역의 변수는 임의값이라 생각해서 대수롭지 않게 생각할 수 있으나 사실은 이전 함수에서 사용되었던 내용을 포함하고 있다. 공격자는 이러한 약점을 사용하여 메모리에 저장되어 있는 값을 읽거나 특정 코드를 실행할 수 있다. 모든 변수를 사용 전에 반드시 올바른 초기값을 할당함으로서 이러한 문제를 예방한다.

신뢰할 수 없는 데이터를 역직렬화 하지 않도록 응용프로그램을 구성한다. 민감정보 또는 중요정보를 전송 시 암호화 통신을 적용하지 못하는 경우, 송신 측에서 서명을 추가하고 수신 측에서 서명을 확인하여 데이터의 무결성을 검증한다.

또는, 신뢰할 수 있는 데이터의 식별을 위해 역직렬화 대상의 데이터가 사전에 검증된 클래스만을 포함하는지 검증하거나, 제한된 실행 권한을 구성하여 역직렬화 코드를 실행한다.

다. 코드예제

다음 예제는 맵(map)을 직렬화하고 역직렬화 하는 코드이다. 데이터를 전송할 경우 공격자가 바이트 스트림을 조작하여 역직렬화 공격이 가능한 객체를 생성할 수 있는 예제이다.

안전하지 않은 코드의 예 JAVA

```
1:public static void main(String[] args) throws
2:IOException, GeneralSecurityException, ClassNotFoundException {
3:      ....
4:      // map을 역직렬화 한다.
5:      ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
6:      sealedMap = (SealedObject) in.readObject();
7:      in.close();
8:
9:      // 객체를 추출한다.
10:     cipher = Cipher.getInstance("AES");
11:     cipher.init(Cipher.DECRYPT_MODE, key);
12:     signedMap = (SignedObject) sealedMap.getObject(cipher);
13:
14:     // 서명값 검증 과정에서 불일치 시 예외를 리턴하고, 일치 시 map 값을 읽는다.
15:     if (!signedMap.verify(kp.getPublic(), sig)) {
16:             throw new GeneralSecurityException("Map failed verification");
17:   }
18:   map = (SerializableMap<String, Integer>) signedMap.getObject();
19:}
```

다음 예제는 서명 값을 검증하여 메시지 위변조를 방지할 수 있는 코드이다.

안전한 코드의 예 JAVA

```
1:public static void main(String[] args) throws
2:IOException, GeneralSecurityException, ClassNotFoundException {
3:      ....
4:      // map을 역직렬화 한다.
5:      ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
6:      sealedMap = (SealedObject) in.readObject();
```

안전한 코드의 예 JAVA

```
7:      in.close();
8:
9:      // 객체를 추출한다.
10:     cipher = Cipher.getInstance("AES");
11:     cipher.init(Cipher.DECRYPT_MODE, key);
12:     signedMap = (SignedObject) sealedMap.getObject(cipher);
13:
14:     // 서명값 검증 과정에서 불일치 시 예외를 리턴하고, 일치 시 map 값을 읽는다.
15:     if (!signedMap.verify(kp.getPublic(), sig)) {
16:             throw new GeneralSecurityException("Map failed verification");
17:     }
18:     map = (SerializableMap<String, Integer>) signedMap.getObject();
19:}
```

다음 예제는 바이트 배열의 입력 값을 검증 없이 readObject()로 역직렬화하여 악의적인 코드가 실행될 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: class DeserializeExample {
2:      public static Object deserialize(byte[] buffer)
3:      throws IOException, ClassNotFoundException {
4:              Object ret = null;
5:              try (ByteArrayInputStream bais = new ByteArrayInputStream(buffer)) {
6:                      try (ObjectInputStream ois = new ObjectInputStream(bais)) {
7:                              ret = ois.readObject();
8:                      }
9:              }
10:     return ret;
11:     }
12:}
```

이를 안전한 코드로 변환하기 위해서는 ObjectInputStream을 상속 받아 WhitelistedObjectInputStream 객체를 구현하여 사용한다. WhitelistedObjectInputStream에서는 readObject()를 실행 시, resolveClass 함수를 호출하여 설정한 화이트리스트와 비교하여 리스트에 없는 데이터일 경우 예외를 발생시킨다.

안전한 코드의 예 JAVA

```
1:public class WhitelistedObjectInputStream extends ObjectInputStream {
2:      public Set<String> whitelist;
3:      // WhilelistedObjectInputStream을 생성할 때 화이트리스트를 입력받는다.
4:      public WhitelistedObjectInputStream(InputStream inputStream, Set<String> wl)
5:      throws IOException {
6:              super(inputStream);
7:              whitelist = wl;
8:      }
9:
10:@Override
11:protected Class<?> resolveClass(ObjectStreamClass cls) throws IOException,
ClassNotFoundException {
12:     // ObjectStreamClass의 클래스명이 화이트리스트에 있는지 확인한다.
13:     if (!whitelist.contains(cls.getName())) {
14:     throw new InvalidClassException("Unexpected serialized class", cls.getName());
15:             }
16:             return super.resolveClass(cls);
17:     }
18:}
19:
20:@RequestMapping(value = "/upload", method = RequestMethod.POST)
21:public Student upload(@RequestParam("file") MultipartFile multipartFile)
 throws ClassNotFoundException, IOException {
22:     Student student = null;
```

안전한 코드의 예 JAVA

```
23:     File targetFile = new File("/temp/" + multipartFile.getOriginalFilename());
24:     // 역직렬화 대상 클래스 이름의 화이트리스트 생성한다.
25:     Set<String> whitelist = new HashSet<String>(Arrays.asList(
26:     new String[] {
27:             "Student"
28:     }));
29:     try (InputStream fileStream = multipartFile.getInputStream()) {
30:             try (WhitelistedObjectInputStream ois =
31:             new WhitelistedObjectInputStream(fileStream, whitelist)) {
32:                     // 화이트리스트에 없는 역직렬화 데이터의 경우 예외 발생시킨다.
33:                     student = (Student) ois.readObject();
34:             }
35:     }
36:     return student;
37:}
```

라. 진단방법

java.io.ObjectInputStream.readObject()와 같이 각 언어에서 제공하는 역직렬화 함수를 확인하고(①), 해당 역직렬화 함수에서 사용하는 데이터가 신뢰할 수 있는 값인지 확인한다(②).

만약 사용자 입력 값, 소켓으로 입력받은 값 등 출처를 신뢰할 수 없는 데이터를 검증하는 절차가 없으면 취약하다고 판정한다.

일반적인 진단의 예

```
1:public class G02 extends HttpServlet {
2:      protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
3:      ServletException, IOException {
4:              ObjectInputStream ois = null;
5:              ImageInformation retImgInfo = null;
6:
7:              try {
8:                      // HttpServletRequest의 InputStream은 사용자 입력값이다.
9:                      ois = new ObjectInputStream(req.getInputStream());  ····· ②
10:                     retImgInfo = (ImageInformation) ois.readObject();  ····· ①
11:                     retImgInfo.setImgUrl("SERVERIMGURL");
12:             } catch (Exception e) {
13:             ...
14:     } finally
15:     ...
16:}
```

다음 예제에서 XMLEncoder와 XMLDecoder는 JDK에 포함된 컴포넌트로, javaBeans Persistence 기능을 제공하기 위해 XML 직렬화를 사용한다. 아래 코드와 같이 4라인에서 XMLDecoder를 사용하여 사용자가 제공한 데이터를 8라인에서 readObject()를 실행하면서 역직렬화하는 경우 공격자의 명령이 실행될 수 있으므로 취약하다고 판정한다.

정탐코드의 예

```
1:private static void byXmlString( String xml ) {
2:      XMLDecoder xd = null;
3:      try {
4:              xd = new XMLDecoder(new ByteArrayInputStream(xml.getBytes()));
5:      } catch (Exception e) {
6:              e.printStackTrace();
7:      }
8:      Object s2 = xd.readObject();
9:      xd.close();
10: }
```

다음 예제는 서버가 특정 포트를 열어 클라이언트가 데이터를 보낼 때까지 기다렸다가 데이터를 받으면 역직렬화 코드가 실행된다. 이 경우 신뢰할 수 없는 데이터 값에 대한 검증 과정이 없으므로 아래 코드는 취약하다고 판정한다.

정탐코드의 예

```
1:conn,addr = self.receiver_socket.accept()
2:data = conn.recv(1024)
3:return Pickle.loads(data)
```

다음 예제는 HMAC을 사용하여 데이터 무결성을 검증하므로 취약하지 않다고 판정한다.

· **클라이언트 코드**

오탐코드의 예

```
1:pickled_data = pickle.dumps(data)
2:digest = hmac.new('shared-key', pickled_data, hashlib.sha1).hexdigest()
3:header = '%s' % (digest)
4:conn.send(header + ' ' + pickled_data)
```

· **서버 코드**

오탐코드의 예

```
1:conn,addr = self.receiver_socket.accept()
2:data = conn.recv(1024)
3:recvd_digest, pickled_data = data.split(' ')
4:new_digest = hmac.new('shared-key', pickled_data, hashlib.sha1).hexdigest()
5:if recvd_digest != new_digest:
6:print 'Integrity check failed'
7:else:
8:unpickled_data = pickle.loads(pickled_data)
```

마. 참고자료

① CWE-918: Server-Side Request Forgery (SSRF), MITRE, https://cwe.mitre.org/data/definitions/918.html

② Server Side Request Forgery, OWASP, https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

| 제 6 절 | 캡슐화

중요한 데이터 또는 기능성을 불충분하게 캡슐화하거나 잘못 사용함으로써 발생하는 보안약점으로 정보노출, 권한문제 등이 발생할 수 있다.

1. 잘못된 세션에 의한 데이터 정보 노출

가. 개요

다중 스레드 환경에서는 싱글톤(Singleton)[17] 객체 필드에 경쟁조건(Race Condition) 발생할 수 있다. 따라서, 다중 스레드 환경인 Java의 서블릿(Servlet) 등에서는 정보를 저장하는 멤버변수가 포함되지 않도록 하여, 서로 다른 세션에서 데이터를 공유하지 않도록 해야 한다.

나. 보안대책

싱글톤 패턴을 사용하는 경우, 변수 범위(Scope)에 주의를 기울여야 한다. 특히 Java에서는 HttpServ-let 클래스의 하위클래스에서 멤버 필드를 선언하지 않도록 하고, 필요한 경우 지역 변수를 선언하여 사용한다.

다. 코드예제

JSP 선언부에 선언한 변수는 해당 JSP에 접근하는 모든 사용자에게 공유된다. 먼저 호출한 사용자가 값을 설정하고 사용하기 전에 다른 사용자의 호출이 발생하게 되면, 뒤에 호출한 사용자가 설정한 값이 모든 사용자에게 적용되게 된다.

안전하지 않은 코드의 예 JAVA

```
1: 〈%@page import="javax.xml.namespace.*"%〉
2: 〈%@page import="gov.mogaha.ntis.web.frs.gis.cmm.util.*" %〉
3: 〈%
```

17) 싱글톤 패턴 : 하나의 프로그램 내에서 하나의 인스턴스만을 생성해야만 하는 패턴. Connection Pool, Thread Pool과 같이 Pool 형태로 관리되는 클래스의 경우 프로그램 내에서 단하나의 인스턴트로 관리해야 하는 경우를 말함. java에서는 객체로 제공됨

안전하지 않은 코드의 예 JAVA

```
4:    // JSP에서 String 필드들이 멤버 변수로 선언됨
5:    String username = "/";
6:    String imagePath = commonPath + "img/";
7:    String imagePath_gis = imagePath + "gis/cmm/btn/";
8:    ……
9: %〉
```

JSP의 서블릿(〈% 소스코드 %〉)에 정의한 변수는 _jspService 메소드의 지역변수로 선언되므로 공유가 발생하지 않아 안전하다.

안전한 코드의 예 JAVA

```
1: 〈%@page import="javax.xml.namespace.*"%〉
2: 〈%@page import="gov.mogaha.ntis.web.frs.gis.cmm.util.*" %〉
3: 〈%
4:    // JSP에서 String 필드들이 로컬 변수로 선언됨
5:    String commonPath = "/";
6:    String imagePath = commonPath + "img/";
7:    String imagePath_gis = imagePath + "gis/cmm/btn/";
8: ……
9: %〉
```

Controller에 멤버 변수를 사용하면 공유가 발생하여 동기화 오류가 발생할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: @Controller
2: public class TrendForecastController {
3:    // Controller에서 int 필드가 멤버 변수로 선언되어 스레드간에 공유됨
4:    private int currentPage = 1;
5:    public void doSomething(HttpServletRequest request) {
```

안전하지 않은 코드의 예 JAVA

```
6:        currentPage = Integer.parseInt(request.getParameter("page"));
7:    }
8: ……
```

Controller에 멤버 변수를 사용하지 않고 지역 변수로 사용한다.

안전한 코드의 예 JAVA

```
1: @Controller
2: public class TrendForecastController {
3:    public void doSomething(HttpServletRequest request) {
4:    // 지역변수로 사용하여 스레드간 공유되지 못하도록 한다.
5:    int currentPage = Integer.parseInt(request.getParameter("page"));
6:    }
7: ……
```

다중 스레드 환경에서 IHttpHandler 클래스에 정보를 저장하는 필드가 포함되서는 안된다.

안전하지 않은 코드의 예 C#

```
1: class DataLeakBetweenSessions : IHttpHandler
2:    {
3:            //다중 스레드 환경에서 IHttpHandler 를 구현하는 클래스에 정보를 저장하는 필드가
   포함되어서는 안됩니다.
4:       private String id;

5:       public void ProcessRequest(HttpContext ctx)
6:       {
7:          var json = new JSONResonse()
8:          {
9:             Success = ctx.Request.QueryString["name"] != null,
```

안전하지 않은 코드의 예 C#

```
10:          Name = ctx.Request.QueryString["name"]
11:        };
12:        ctx.Response.ContentType = "application/json";
13:        ctx.Response.Write(JsonConvert.SerializeObject(json));
14:     }

15:     public bool IsReusable
16:     {
17:        get { return false; }
18:     }
19:  }
```

해당 내용을 지역 변수나 세션변수를 이용하여 처리하여야 한다.

안전한 코드의 예 C#

```
1: class DataLeakBetweenSessions : IHttpHandler
2:    {
3:       public void ProcessRequest(HttpContext ctx)
4:       {
 //지역 변수나 세션변수를 선언해서 사용해야 합니다.
5:                         ctx.Session["id"] = ctx.Request.QueryString["id"];

6:             var json = new JSONResonse()
7:             {
8:                Success = ctx.Request.QueryString["name"] != null,
9:                Name = ctx.Request.QueryString["name"]
10:            };

11:           ctx.Response.ContentType = "application/json";
12:           ctx.Response.Write(JsonConvert.SerializeObject(json));
```

안전한 코드의 예 C#

```
13:     }

14:     public bool IsReusable
15:     {
16:         get { return false; }
17:     }
18: }
```

라. 진단방법

HttpServlet의 하위클래스에 멤버필드가 선언되어 있고 final이 아닌 경우는 취약한 것으로 판단한다.

일반적인 진단의 예

```
1: public class U488 extends HttpServlet
2: {
3: private String name;
4: protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
5: {
6: name = request.getParameter("name");
7: …
8: out.println(name + ", thanks for visiting!");
9: }
10:}
```

서블릿(JSP 포함)에서 상수로 사용하지 않는 멤버 변수를 사용하면 취약하다.

정탐코드의 예

```
1: <%@ page import="javax.xml.namespace.*" %>
2: <%@ page import="maha.ns.web.fta.gaa.comm.util.*" %>
```

정탐코드의 예

```
3:  〈%@ page import="maha.ns.web.fta.gaa.comm.util.SSOSessionUtil"%〉
4:  〈%!
5:  String commonPath = "/";
6:  String imagePath= commonPath +"img/";
7:  String imagePath_gis = imagePath +"gis/cmm/btn/";
```

JSP 페이지나 서블릿 내에서의 내부 클래스 사용은 멤버필드가 아니므로 취약하지 않다.

오탐코드의 예

```
1:  〈%@ page language="java" import="java.io.*, java.text.* ,java.util.*, java.net.*,
    com.ins.
    system.config.*, com.ins.system.exception.*"%〉
2:  〈%!
3:          private class CacheEntity {
4:                  String name;
5:                  String lastModified;
6:                  String expires;
7:                  String eTag;
8:          }
```

Final 필드의 경우 클래스 내부에서 상수로 사용되는 값이므로 취약하지 않다.

오탐코드의 예

```
1:  〈%@ page import="java.util.*" %〉
2:  〈%@ page import="maha.ns.comm.util.StringUtil" %〉
3:  〈%@ page import="maha.ns.comm.util.Page" %〉
4:  〈%@ taglib uri="comm.tld" prefix="comm" %〉
5:  〈%!
6:  final String imagePath = "/img/";
7:  String treeImagePath = imagePath+"/port/tree.gif";
8:  %〉
```

Spring 프레임워크 기반의 eGov 프레임워크는 IoC를 사용하고 있다. 이는 프레임워크에서 인스턴스를 관리하기 때문에 안전하다.

오탐코드의 예

```
1:  @Resource(name = "fileMngService")
2:  private FileMngService fileService;
```

마. 참고자료

[1] CWE-488 Exposure of Data Element to Wrong Session, MITRE, http://cwe.mitre.org/data/definitions/488.html

[2] CWE-543 Use of Singleton Pattern Without Synchronization in a Mulitithreaded Context, MITRE, http://cwe.mitre.org/data/definitions/543.html

[3] Do not let session information leak within a servlet, CERT, http://www.securecoding.cert.org/confluence/display/java/MSC11-J.+Do+not+let+session+information+leak+within+a+servlet

2. 제거되지 않고 남은 디버그 코드

가. 개요

디버깅 목적으로 삽입된 코드는 개발이 완료되면 제거해야 한다. 디버그 코드는 설정 등의 민감한 정보를 담거나 시스템을 제어하게 허용하는 부분을 담고 있을 수 있다. 만일, 남겨진 채로 배포될 경우, 공격자가 식별 과정을 우회하거나 의도하지 않은 정보와 제어 정보가 노출될 수 있다.

나. 보안대책

소프트웨어 배포 전, 반드시 디버그 코드를 확인 및 삭제한다. 일반적으로 Java 개발자의 경우 웹 응용 프로그램을 제작할 때 디버그 용도의 코드를 main()에 개발한 후 이를 삭제하지 않는 경우가 많다. 디버깅이 끝나면 main() 메서드를 삭제해야 한다.

다. 코드예제

다음의 예제는 main() 메서드 내에 화면에 출력하는 디버깅 코드를 포함하고 있다. J2EE의 경우 main() 메서드 사용이 필요 없으며, 개발자들이 콘솔 응용프로그램으로 화면에 디버깅코드를 사용하는 경우가 일반적이다.

안전하지 않은 코드의 예 JAVA

```
1: class Base64 {
2:     public static void main(String[] args) {
3:         if (debug) {
4:             byte[] a = { (byte) 0xfc, (byte) 0x0f, (byte) 0xc0 };
5:             byte[] b = { (byte) 0x03, (byte) 0xf0, (byte) 0x3f };
6:                     ……
7:         }
8:     }
9:     public void otherMethod() { … }
10:}
```

이에 따라 J2EE와 같은 응용프로그램에서 main() 메소드는 삭제한다. J2EE의 main() 메소드의 경우 디버깅 코드인 경우가 일반적이다.

안전한 코드의 예 JAVA

```
1: class Base64 {
2:     public void otherMethod() { … }
3: }
```

디버그용 코드가 남아있는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
1: class Example {
2: public void Log() {
3: //Console.WriteLine 등의 메소드를 사용한 디버그용 코드가 남아있습니다.
4:     Console.WriteLine("sensitive info");
5:   }
6: }
```

디버그용 코드를 삭제 하여야 한다.

안전한 코드의 예 C#

```
1: class Example {
2: public void Log() {
3: //디버그용 코드를 삭제해야 합니다.
4: //Console.WriteLine("sensitive info");
5: }
6: }
```

아래는 디버그용 코드가 남아있는 C 코드의 예제이다. 공격자가 출력되는 콜 스택으로 프로그램 구조를 유추할 수 있다.

안전하지 않은 코드의 예 C

```
1: void LeftoverDebugCode() {
2:   int i, ntprs;
3:   char **strings;
4:   nptrs = backtrace(buffer, 100);
5:   strings = backtrace_symbols(buffer, nptrs);
6:   …
7:   // 디버그 모드일 시 콜스택을 출력한다
8:   if(debug) {
9:     for(i=0; i < nptr; i++) printf("%s\n", strings[j]);
10:  }
11:}
```

릴리즈 시에는 디버그용 코드를 삭제 하여야 한다.

안전한 코드의 예 C

```
1: void LeftoverDebugCode() {
2: … // 디버그 코드를 삭제하고 동작 코드만 남긴다.
3: }
```

라. 진단방법

J2EE를 제외하고 디버그 코드를 정적도구만으로 판단하기는 쉽지 않다.

개발 중 테스트 목적으로 남아 있는 디버그 코드가 존재하는지 확인한다(①). J2EE 환경(J2EE Standard)에서는 main 메소드 작성을 금하고 있으며, 일반적으로 개발자들은 디버그 코드를 main으로 작성하므로 main 메소드가 사용되는 경우 디버그코드 인지 확인하여야 한다.

일반적인 진단의 예

```
1: public class U489 extends HttpServlet {
2:   protected void doGet(HttpServletRequest request, … ) throws … { … }
3:   protected void doPost(HttpServletRequest request, … ) throws … { … }
4:   // 테스트를 위한 main()함수나 디버깅용 로그 출력문 등이 남아 있다.
5:   public static void main(String args[]) { ……①
6:     System.err.printf(“Print debug code”);
7:   }
8: …
```

J2EE와 같은 응용프로그램에서 디버깅용으로 사용되는 main() 메소드는 삭제한다.

정탐코드의 예

```
1: public class U489 extends HttpServlet {
2: protected void doGet(HttpServletRequest request, …) throws …{ …}
3: protected void doPost(HttpServletRequest request, …) throws …{ …}
4: // 테스트를 위한 main()함수나 디버깅용 로그 출력문 등이 남아 있다.
5: public static void main(String args[]) {
6: System.err.printf(“Print debug code”);
7: }
8: }
```

마. 참고자료

[1] CWE-489 Leftover Debug Code, MITRE,
http://cwe.mitre.org/data/definitions/489.html

[2] Production code must not contain debugging entry points, CERT,
http://www.securecoding.cert.org/confluence/display/java/ENV06-J.+Production+code+must+not+contain+debugging+entry+points

3. Public 메소드부터 반환된 Private 배열

가. 개요

private로 선언된 배열을 public으로 선언된 메소드로 반환(return)하면, 그 배열의 레퍼런스가 외부에 공개되어 외부에서 배열수정과 객체 속성변경이 가능해진다.

나. 보안대책

private로 선언된 배열을 public으로 선언된 메소드로 반환하지 않도록 해야 한다. private 배열에 대한 복사본을 반환하도록 하고 배열의 원소에 대해서는 clone() 메소드로 복사된 원소를 저장하도록 하여 private 선언된 배열과 객체속성에 대한 의도하지 않게 수정되는 것을 방지한다. 만약 배열의 원소가 String 타입 등과 같이 변경이 되지 않는 경우에는 Private 배열의 복사본을 만들고 이를 반환하도록 작성한다.

다. 코드예제

멤버 변수 colors는 private로 선언되었지만 public으로 선언된 getColors() 메소드로 참조를 얻을 수 있다. 이 경우 의도하지 않은 수정이 발생할 수 있다.

아래의 코드는 멤버 변수 colors는 private로 선언되었지만 public으로 선언된 getUserColors 메소드로 private 배열에 대한 reference를 얻을 수 있다. 이 경우 의도하지 않은 수정이 발생할 수 있다.

안전하지 않은 코드의 예 JAVA (배열의 원소가 일반객체일 경우)

```
1: // private 인 배열을 public인 메소드가 return한다.
2: private Color[] colors;
3: public Color[] getUserColors(Color[] userColors) { return colors; }
```

private배열에 대한 복사본을 만들고, 복사된 배열의 원소로는 clone() 메소드로 private 배열의 원소의 복사본을 만들어 저장하여 반환하도록 작성하면, private선언된 배열과 원소에 대한 의도하지 않은 수정을 방지할 수 있다.

안전한 코드의 예 JAVA (배열의 원소가 일반객체일 경우)

```
1: private Color[] colors;
2: //메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.
3: public void onCreate(Bundle savedInstanceState) {
4:   super.onCreate(savedInstanceState);
5:   Color[] newColors = getUserColors();
6:    .....
7: }
8: public Color[] getUserColors(Color[] userColors) {
9:   //배열을 복사한다.
10: Color[] colors = new Color [userColors.length];
11: for (int i = 0; i < colors.length; i++)
12: //clone()메소드를 이용하여 배열의 원소도 복사한다.
13: colors[i] = this.colors[i].clone();
14: return colors;
15:}
```

아래의 코드는 멤버 변수 colors는 private로 선언되었지만, public으로 선언된 getColors() 메소드로 reference를 얻을 수 있다. 이 경우 의도하지 않은 배열의 수정이 발생할 수 있다.

안전하지 않은 코드의 예 JAVA (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
1: // private 인 배열을 public인 메소드가 return한다.
2: private String[] colors;
3: public String[] getColors() { return colors; }
```

private배열의 복사본을 만들고, 이를 반환하도록 작성하면, private 선언된 배열에 대한 의도하지 않은 수정을 방지할 수 있다.

안전한 코드의 예 JAVA (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
1: private String[] colors;
2: // 메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.
3: public void onCreate(Bundle savedInstanceState) {
4: super.onCreate(savedInstanceState);
5: String[] newColors = getColors();
6: ......
7: }
8: public String[] getColors() {
9: String[] ret = null;
10: if ( this.colors != null ) {
11:       ret = new String[colors.length];
12:     for (int i = 0; i < colors.length; i++) { ret[i] = this.colors[i]; }
13: }
14: return ret;
15: }
```

아래의 코드는 멤버 변수 colors는 private로 선언되었지만 public으로 선언된 getUserColors 메소드로 private 배열에 대한 reference를 얻을 수 있다. 이 경우 의도하지 않은 수정이 발생할 수 있다.

안전하지 않은 코드의 예 C#

```
1: // private 인 collection을 public인 메소드가 return한다.
2: private List<Color> colors;
3: public List<Color> getUserColors() { return colors; }
```

메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만들어야 한다.

안전한 코드의 예 C#

```
1: private List<Color> colors;
2: //메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.
3: public List<Color> getUserColors() {
4:   //배열을 복사한다.
5:   List< ICloneable> newList = new List< ICloneable>(colors.Count);
6:   //Clone()메소드를 이용하여 collection의 원소도 복사한다.
7:   colors.ForEach((item) =>
8:   {
9:     newList.Add((ICloneable)item.Clone());
10: });
11: return newList;
12:}
```

라. 진단방법

private 배열이 선언되어 있는지 확인하고(①), 해당 배열이 public 메소드에서 반환될 경우 취약하다고 판단한다(②). 이때 선언된 private 배열의 원소가 일반 객체일 경우 각 원소별로 객체를 생성하고 객체 내부의 값을 복사하는지 확인한다.

일반적인 진단의 예

```
1: public class U495 {
2:    //Private 인 배열을 public 인 메소드가 return한다.
3:    private String[] colors; ····· ①
4:
5:    public String[] getColors() {
6:          return colors; ····· ②
7:    }
8: }
```

함수 안에서 멤버필드의 배열을 리턴 하면 해당 배열을 리턴 받아 그 값을 바꿀 수 있다. Private 배열을 리턴 받아 그 내용을 수정하게 되면 Private의 의도에 벗어나므로 취약하다.

정탐코드의 예

```
1: public class AddManager extends BaseManager {
2:   private String[] checkb;
3:   private String[] prsn_info_sn;
4:   private String login_dept_id;
5:   private String login_user_id;
6:   private String[] prsn_group_se;
7:   private String[] isInterested;
8:   private String act;
9:   public String[] getChkb() {
10:   return this.checkb;
11: }
12:}
```

아래 코드는 retColors 배열을 새로 생성하고 for문을 이용하여 복사하였으나, 배열의 원소가 일반 객체이기 때문에 원본(myColors)의 주소값만 복사되어 취약하다.

정탐코드의 예

```
1: // private 인 배열을 public인 메소드가 return한다
2: private Color[] myColors;
3: …
4: public Color[] getColors() {
5:   Color[] retColors = new Color[myColors.length];
6:   for(int i =0; i < myColors.length; i++) {
7:     retColors[i] = this.myColors[i]; // 일반 객체일 경우 주소값만 복사된다.
8:   }
9:   return retColors;
10:}
```

다음과 같은 배열은 private도 아니고 final 이므로 취약하지 않다.

오탐코드의 예

```
1:   1: public static final String[] XML_TAG = { "<?xml version=\"1.0\" encoding=\"", "\"?>"
     }
```

마. 참고자료

[1] CWE-495 Private Array-Typed Field Returned From A Public Method, MITRE, http://cwe.mitre.org/data/definitions/495.html

[2] Do not return references to private mutable class members, CERT, http://www.securecoding.cert.org/confluence/display/java/OBJ05-J.+Do+not+return+references+to+private+mutable+class+members

4. Private 배열에 Public 데이터 할당

가. 개요

public으로 선언된 메소드의 인자가 private선언된 배열에 저장되면, private배열을 외부에서 접근하여 배열수정과 객체 속성변경이 가능해진다.

나. 보안대책

public으로 선언된 메서드의 인자를 private선언된 배열로 저장되지 않도록 한다. 인자로 들어온 배열의 복사본을 생성하고 clone() 메소드로 복사된 원소를 저장하도록 하여 private변수에 할당하여 private선언된 배열과 객체속성에 대한 의도하지 않게 수정되는 것을 방지한다. 만약 배열 객체의 원소가 String 타입 등과 같이 변경이 되지 않는 경우에는 인자로 들어온 배열의 복사본을 생성하여 할당한다.

다. 코드예제

아래의 코드는 멤버 변수 userRoles는 private로 선언되었지만 public으로 선언된 setUserRoles 메소드로 인자가 할당되어 배열의 원소를 외부에서 변경할 수 있다. 이 경우 의도하지 않은 배열과 원소에 대한 객체속성 수정이 발생할 수 있다.

안전하지 않은 코드의 예 JAVA (배열의 원소가 일반객체일 경우)

```
1: //userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면, 사실상
   public 필드가 된다.
2: private UserRole[] userRoles;
3: public void setUserRoles(UserRole[] userRoles) {
4:     this.userRoles = userRoles;
5: }
```

인자로 들어온 배열의 복사본을 생성하고 clone() 메소드로 복사된 원소를 저장하도록 하여 private 변수에 할당하면 private으로 할당된 배열과 원소에 대한 의도하지 않은 수정을 방지할 수 있다.

안전한 코드의 예 JAVA (배열의 원소가 일반객체일 경우)

```
1: //객체가 클래스의 private member를 수정하지 않도록 한다.
2: private UserRole[] userRoles;
3: public void setUserRoles(UserRole[] userRoles) {
4: this.userRoles = new UserRole[userRoles.length];
5: for (int i = 0; i < userRoles.length; ++i)
6:    this.userRoles[i] = userRoles[i].clone();
7: }
```

아래의 코드는 멤버 변수 userRoles는 private로 선언되었지만 public으로 선언된 setUserRoles 메소드로 인자가 할당되어 배열의 원소를 외부에서 변경할 수 있다. 이 경우 의도하지 않은 배열에 대한 수정이 발생할 수 있다.

안전하지 않은 코드의 예 (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
1: //userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면, 사실상
   public 필드가 된다.
2: private String[] userRoles;
3: public void setUserRoles(String[] userRoles) {
4:  this.userRoles = userRoles;
5: }
```

인자로 들어온 배열의 복사본을 생성하여 private변수에 할당하면 private으로 할당된 배열에 대한 의도하지 않은 수정을 수 있다.

안전한 코드의 예 (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
1: //객체가 클래스의 private member를 수정하지 않도록 한다.
2: private String[] userRoles;
```

안전한 코드의 예 (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
3: public void setUserRoles(String[] userRoles) {
4:   this.userRoles = new String[userRoles.length];
5:    for (int i = 0; i < userRoles.length; ++i)
6:     this.userRoles[i] = userRoles[i];
7: }
```

아래의 코드는 멤버 변수 userRoles는 private로 선언되었지만 public으로 선언된 setUserRoles 메소드로 인자가 할당되어 배열의 원소를 외부에서 변경할 수 있다. 이 경우 의도하지 않은 배열과 원소에 대한 객체속성 수정이 발생할 수 있다.

안전하지 않은 코드의 예 C#

```
1: class Program
2:    {
3: //userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면, 사실상 public 필드가 된다.
4:       private String[] userRoles;
5:       public void SetUserRoles(String[] userRoles)
6:       {
7:         this.userRoles = userRoles;
8:       }
9: }
```

객체가 클래스의 private member를 수정하지 않도록 하여야 한다.

안전한 코드의 예 C#

```
1: class Program
2:    {
3: //객체가 클래스의 private member를 수정하지 않도록 한다.
4:       private String[] userRoles;
5:       public void SetUserRoles(String[] userRoles)
```

안전한 코드의 예 C#

```
6:        {
7:           int length = userRoles.Length;
8:           this.userRoles = new String[length];
9:           for(int i = 0; i < length; i++) {\
10:           this.userRoles[i] = userRoles[i];
11:          }
12:       }
13:    }
```

라. 진단방법

private 배열이 선언되어 있는지 확인하고(①), public 메소드의 인자로 받은 배열을 private 배열 필드에 직접 할당할 경우 취약하다고 판단한다(②). 이때 선언된 private 배열의 원소가 일반 객체일 경우 각 원소별로 객체를 생성하고 객체 내부의 값을 복사하는지 확인한다.

userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면, 사실상 public 필드가 된다.

일반적인 진단의 예

```
1: public class U496 {
2:   private String[] userRoles; ····· ①
3:   public void setUserRoles(String[] userRoles) {
4:     this.userRoles = userRoles; ····· ②
5:   }
6:
7:   public void print() {
8:     for (int i = 0; i < userRoles.length; i++)
9:       System.out.println(userRoles[i]);
10:  }
11: }
```

아래 코드는 새로운 배열을 새로 생성하고 for문을 이용하여 복사하였으나, 배열의 원소가 일반객체이기 때문에 원본(userRoles)의 주소값만 복사되어 취약하다.

정탐코드의 예

```
1: private UserRole[] userRoles;
2: …
3: public void setUserRoles(UserRole[] userRoles) {
4:   this.userRoles = new UserRole[userRoles.length];
5:   for(int i =0; i 〈 userRoles.length; i++) {
6:     this.userRoles[i] = userRoles[i]; // 일반 객체일 경우 주소값만 복사된다.
7:   }
8: }
```

마. 참고자료

[1] CWE-496 Public Data Assigned to Private Array-Typed Field, MITRE, http://cwe.mitre.org/data/definitions/496.html

| 제 7 절 | API 오용

의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안약점이다.

1. DNS lookup에 의존한 보안결정

가. 개요

공격자가 DNS 엔트리를 속일 수 있으므로 도메인명에 의존에서 보안결정(인증 및 접근통제 등)을 하지 않아야 한다. 만약, 로컬 DNS 서버의 캐시가 공격자에 의해 오염된 상황이라면, 사용자와 특정 서버 간의 네트워크 트래픽이 공격자를 경유하도록 할 수도 있다. 또한, 공격자가 마치 동일 도메인에 속한 서버인 것처럼 위장할 수도 있다.

나. 보안대책

보안결정에서 도메인명을 이용한 DNS lookup을 하지 않도록 한다.

다. 코드예제

다음의 예제는 도메인명으로 해당 요청을 신뢰할 수 있는지를 검사한다. 그러나 공격자는 DNS 캐시 등을 조작해서 쉽게 이러한 보안 설정을 우회할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1:  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
2:    boolean trusted = false;
3:    String ip = req.getRemoteAddr();
4:    InetAddress addr = InetAddress.getByName(ip);
5:  //도메인은 공격자에 의해 실행되는 서버의 DNS가 변경될 수 있으므로 안전하지 않다.
6:    if (addr.getCanonicalHostName().endsWith("trustme.com")) {
7:      do_something_for_Trust_System();
8:  }
```

그러므로, 다음의 예제와 같이 DNS lookup에 의한 호스트 이름 비교를 하지 않고, IP 주소를 직접 비교하도록 수정한다.

안전한 코드의 예 JAVA

```
1: public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
2:    String ip = req.getRemoteAddr();
3:    if (ip == null || "".equals(ip)) return ;
4: //이용하려는 실제 서버의 IP 주소를 사용하여 DNS변조에 방어한다.
5:    String trustedAddr = "127.0.0.1";
6:    if (ip.equals(trustedAddr)) {
7:        do_something_for_Trust_System();
8:    }
```

다음의 예제는 도메인명으로 해당 요청을 신뢰할 수 있는지를 검사한다. 그러나 공격자는 DNS 캐시 등을 조작해서 쉽게 이러한 보안 설정을 우회할 수 있다.

안전하지 않은 코드의 예 C#

```
1: bool trusted;
2: string remoteIpAddress = Request.ServerVariables["REMOTE_HOST"];
3: IPAddress hostIPAddress = IPAddress.Parse(remoteIpAddress);
4: IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
5: string hostName = hostInfo.HostName;

6: if (hostName.EndsWith("trust.com"))
7: {
8: trusted = true;
9: }
```

IP주소를 직접 비교하도록 한다.

안전한 코드의 예 C#

```
1: bool trusted;
2: string remoteIpAddress = Request.ServerVariables["REMOTE_HOST"];
3: if (remoteIpAddress.Equals(trustedAddr))
4: {
5:   trusted = true;
6:   Do_something_for_Trust_System();
7: }
```

아래 C 코드는 요청의 신뢰성을 호스트의 이름으로 판별하고 있다. 공격자는 DNS 캐시를 조작하여 보안 정책을 우회할 수 있다.

안전하지 않은 코드의 예 C

```
1: struct hostent *hp;struct in_addr myaddr;
2: char* tHost = "trustme.example.com";
3: myaddr.s_addr=inet_addr(ip_addr_string);

4: hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
5: // 요청의 신뢰성을 호스트의 이름으로 판별하고 있다.
6: if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
7:   trusted = true;
8: } else {
9:   trusted = false;
10:}
```

다음의 C 코드는 IP 주소를 직접 비교하여 DNS 캐시 조작에 대해 안전하다.

안전한 코드의 예 C

```
1:  struct hostent *hp;struct in_addr myaddr;
2:  char* tHost = "127.0.0.1";
3:  myaddr.s_addr=inet_addr(ip_addr_string);
4:  hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
5:  // 호스트의 이름이 아니라 IP로 직접 비교한다.
6:  if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
7:    trusted = true;
8:  } else {
9:    trusted = false;
10: }
```

라. 진단방법

DNS lookup을 하는 모듈이 존재하는지 확인하고(①), 보안결정을 하는 부분이 존재하는지 확인한다(②). DNS 이름으로 해당 요청이 신뢰할 수 있는지를 검사한다. 그러나 공격자가 DNS 캐시 등을 조작하면 잘못된 신뢰 상태 정보를 얻을 수 있다.

일반적인 진단의 예

```
1: public class U247 extends HttpServlet {
2:  public void doGet(HttpServletRequest req, HttpServletResponse res) throws Servlet
    Exception, IOException {
3:    boolean trusted = false;
4:    String ip = req.getRemoteAddr();
5:    // 호스트의 IP 주소를 얻어온다.
6:    InetAddress addr = InetAddress.getByName(ip);  ····· ①
7:    // 호스트의 DNS이름이 신뢰할 수 있는 도메인(trustme.com)에 속하는지 검사한다.
8:    if (addr.getCanonicalHostName().endsWith("trustme.com") ) {  ····· ②
9:      trusted = true;
10:   }
11:   if (trusted) {
```

일반적인 진단의 예

```
12:        …
13:     } else {
14:        …
15:     }
16:  }
17:}
```

마. 참고자료

[1] CWE-247 Reliance on DNS Lookups in a Security Decision, MITRE, http://cwe.mitre.org/data/definitions/247.html

2. 취약한 API 사용

가. 개요

취약한 API는 보안상 금지된(banned) 함수이거나, 부주의하게 사용될 가능성이 많은 API를 의미한다. 이들 범주의 API에 대해 확인하지 않고 사용할 때 보안 문제를 발생시킬 수 있다. 금지된 API의 대표적인 예로는 스트링 자료와 관련된 strcat(), strcpy(), strncat(), strncpy(), sprintf() 등이 있다. 또한 보안상 문제가 없다 하더라도 잘못된 방식으로 함수를 사용할 때도 역시 보안 문제를 발생 시킬 수 있다.

나. 보안대책

보안 문제로 인해 금지된 함수는 이를 대체할 수 있는 안전한 함수를 사용한다. 그 예로, 위에 언급된 API 대신에 strcat_s(), strcpy_s(), strncat_s(), strncpy_s(), sprintf_s()과 같은 안전한 함수를 사용하는 것이 권장된다. 또한 금지된 API는 아니지만 취약한 API의 예시로, 문자열을 정수로 변환할 때 사용하는 strtol()과 같은 함수는 작은 크기의 부호 있는 정수인 int, short, char와 같은 자료형 변환에 사용하면 범위 제한 없이 값을 평가할 수 있다.

취약한 API의 분류는 일반적인 것은 아니지만 개발 조직에 따라 이를 명시한 경우가 있다면1) 반드시 준수한다.

다. 코드예제

아래 예제에서 gets() 함수는 크기와 상관없이 입력값을 버퍼에 저장하기 때문에 버퍼 오버플로우를 유발할 수 있다.

안전하지 않은 코드의 예 C

```
1:#include 〈stdio.h〉

2:void requestString()
3:{
4: char str[100];
5://gets() 함수는 문자열 길이를 제한 할 수 없어 안전하지 않다.
6: gets(str);
7:}
```

아래 예제와 같이 gets_s() 또는 fgets()함수를 사용하여 입력값의 크기를 제한하여 사용해야 한다.

안전한 코드의 예 C

```
1: #include <stdio.h>

2: void requestString()
3: {
4:   char str[100];
5:   //gets_s() 함수는 문자열 길이 제한이 가능하다.
6:   gets_s(str, sizeof(str));
7: }
```

다음 예제는 J2EE 응용 프로그램에서 프레임워크 메소드 호출 대신에 소켓을 직접 사용하고 있어, 프레임워크에서 제공하는 보안기능을 제공받지 못한다.

안전하지 않은 코드의 예 JAVA

```
1: public class S246 extends javax.servlet.http.HttpServlet {
2:    private Socket socket
3:    protected void doGet(HttpServletRequest request,
4:       HttpServletResponse response) throws ServletException {
5:       try {
6: //프레임워크의 메소드 호출 대신 소켓을 직접 사용하고 있어 프레임워크에서 제공하는 보안기능을
   제공받지 못해 안전하지 않다.
7:          socket = new Socket("kisa.or.kr", 8080);
8:       } catch (UnknownHostException e) {
9:                  .....
```

타겟이 WAS로 작성될 경우 아래의 코드처럼 보안기능을 제공하는 프레임워크 메소드인 URL Connection 을 이용하여야한다.

안전한 코드의 예 JAVA

```
1: public class S246 extends javax.servlet.http.HttpServlet {
2:    protected void doGet(HttpServletRequest request,
```

안전한 코드의 예 JAVA

```
3:    HttpServletResponse response) throws ServletException {
4:    ObjectOutputStream oos = null;
5:    ObjectInputStream ois = null;
6:    try {
7:     URL url = new URL("http://127.0.0.1:8080/DataServlet");
8:    //보안기능을 제공하는 프레임워크의 메소드를 사용하여야한다.
9:     URLConnection urlConn = url.openConnection();
10:     urlConn.setDoOutput(true);
11:           .....
```

아래 예제는 J2EE 프로그램에서 System.exit()를 사용하고 있다. System.exit() 메소드를 호출하는 경우 웹 애플리케이션을 실행하고 있는 컨테이너를 종료할 수 있다.

안전하지 않은 코드의 예 JAVA

```
1: public class U382 extends HttpServlet {
2:    public void doPost(HttpServletRequest request, HttpServletResponse response)
3:       throws ServletException, IOException {
4:       try {
5:          do_something(logger);
6:       } catch (IOException ase) {
7:          logger.info("ERROR");
8:    // J2EE 프로그램에서 System.exit()을 사용하여 서비스가 종료 될 수 있다.
9:          System.exit(1);
10:      }
```

서비스 종료를 막기 위해, J2EE 프로그램에서는 System.exit() 메소드를 사용하지 않는다.

안전한 코드의 예 JAVA

```
1: public class U382 extends HttpServlet {
2:    public void doPost(HttpServletRequest request, HttpServletResponse response)
```

안전한 코드의 예 JAVA

```
3:     throws ServletException, IOException {
4:     try {
5:        do_something(logger);
6:     } catch (IOException ase) {
7:        logger.info("ERROR");
8:     //서비스 종료를 막기 위해 J2EE에서는 System.exit()를 사용하지 않는다.
9:     }
```

C# 코드에서 Application.Exit() 을 사용하면 일부 이벤트가 처리되지 않습니다.

안전하지 않은 코드의 예 C#

```
1: try{
2:   ...
3: } catch (Exception e){
4:   ...
5:   //Application.Exit() 은 즉시 프로그램을 종료하기 때문에, Form.Closed 혹은 Form.Closing
     이벤트가 처리되지 않습니다.
6:    Application.Exit();
7: }
```

이벤트 처리를 위해 Application.Exit()을 사용하지 않습니다.

안전한 코드의 예 C#

```
1: try{
2: ...
3: } catch (Exception e){
4:   ...
5:   // Application.Exit() 을 사용하지 않으면, 이벤트를 처리하지 못하고 프로그램이 종료되는 것을
     방지할 수 있습니다.
6:    this.Close();
7: }
```

라. 진단방법

취약한 API의 리스트를 작성하고 검사하고자 하는 프로그램에서 해당 API를 사용하는지 확인한다. 해당 API를 사용하는 경우 프로그램에 예기치 않은 문제가 발생할 수 있으므로 취약한 것으로 판단한다. 만약 취약한 API를 대체할 수 있는 API가 없을 경우, 해당 API의 인자와 반환 값에 대한 검사가 이루어지는지 확인한다.

다음 예제는 함수의 매개변수를 버퍼에 복사하는 코드이다. 예제에서 버퍼에 값을 복사하기 위해 사용한 strcpy() 함수는 보안에서 취약한 대표적인 API중 하나로, 만약 매개 변수의 길이가 지정된 버퍼의 크기보다 클 경우 예기치 못한 문제가 발생 할 수가 있다. 따라서 해당 코드는 보안약점이 존재한다고 진단할 수 있다.

정탐코드의 예

```
1: void manipulate_string(char * string)
2: {
3:   char buf[24];
4:   strcpy(buf, string);
5:   …
6: }
```

아래 예제는 J2EE 프로그램에서 System.exit()를 사용하고 있다. System.exit() 메소드를 호출 하는 경우 웹 애플리케이션을 실행하고 있는 컨테이너를 종료할 수 있다.

정탐코드의 예

```
1: public class U382 extends HttpServlet {
2:   public void doPost(HttpServletRequest request, HttpServletResponse response)
3:     throws ServletException, IOException {
4:     try {
5:       do_something(logger);
6:     } catch (IOException ase) {
7:       System.exit(1); /* J2EE 프로그램에서 System.exit()을 사용하고 있음 */
8:     }
9:             .....
```

마. 참고자료

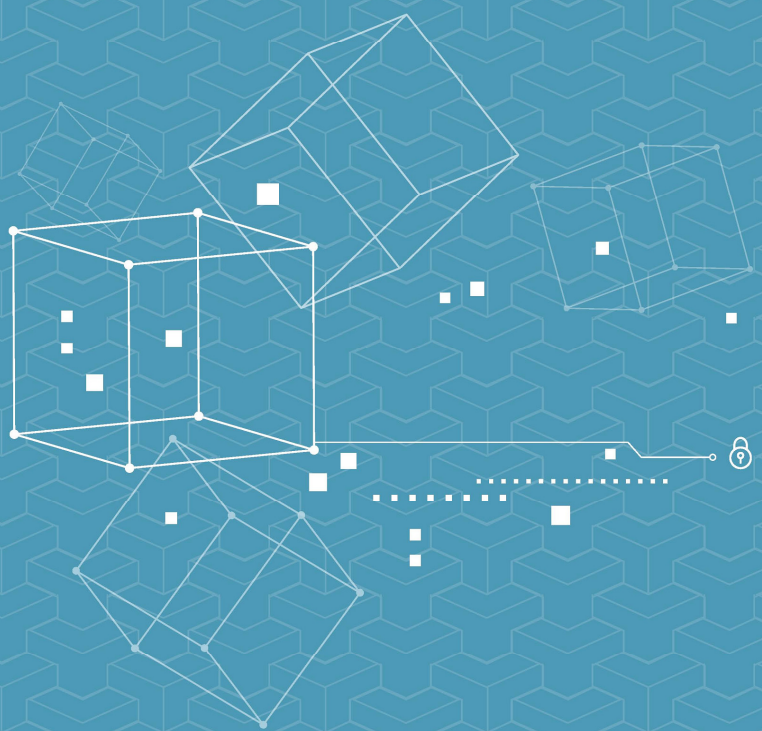
[1] CWE-676 Use of Potentially Dangerous Function, MITRE,
http://cwe.mitre.org/data/definitions/676.html

[2] CWE-242 Use of Inherently Dangerous Function, MITRE,
http://cwe.mitre.org/data/definitions/242.html

[3] CWE-246 J2EE Bad Practices: Direct Use of Sockets, MITRE,
http://cwe.mitre.org/data/definitions/246.html

[4] CWE-382 J2EE Bad Practices: Use of System.exit(), MITRE,
http://cwe.mitre.org/data/definitions/382.html

[5] Do not use deprecated or obsolescent functions, CERT,
http://www.securecoding.cert.org/confluence/display/c/MSC24-C.+Do+not+use+deprecated+or+obsolescent+functions



소프트웨어 보안약점 진단가이드

KOREA INTERNET & SECURITY AGENCY

PART 5

제5장 부록

PART 5

제5장 부록

| 제 1 절 | 설계단계 보안설계 기준

1. 입력데이터 검증 및 표현

사용자, 프로그램 입력 데이터에 대한 유효성 검증체계를 갖추고, 유효하지 않은 값에 대한 처리방법 설계

SR1-1. DBMS 조회 및 결과 검증	
설명	DBMS 조회시 질의문(SQL) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다. ② 외부입력값이 삽입되는 SQL쿼리문을 동적으로 생성해서 실행하지 않도록 해야 한다. ③ 외부입력값을 이용해 동적으로 SQL쿼리문을 생성해야 하는 경우, 입력값에 대한 검증을 수행한 뒤 사용해야 한다.

SR1-2. XML조회 및 결과 검증	
설명	XML 조회시 질의문(XPath, XQuery 등) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① XML문서를 조회하는 기능을 구현해야 하는 경우 XML쿼리에 사용되는 파라미터는 반드시 XML 쿼리를 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 쿼리문에 입력값을 자료형에 따라 바인딩해서 사용해야 한다.

SR1-3. 디렉토리 서비스 조회 및 결과 검증	
설명	디렉토리 서비스(LDAP 등)를 조회할 때 입력값과 그 조회결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용되는 외부입력값은 LDAP 삽입 취약점을 가지지 않도록 필터링해서 사용해야 한다.

SR1-4. 시스템 자원 접근 및 명령어 수행 입력값 검증	
설명	시스템 자원접근 및 명령어를 수행할 때 입력값에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 외부입력값을 이용하여 시스템자원(IP, PORT번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다. ② 서버 프로그램 안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

SR1-5. 웹 서비스 요청 및 결과 검증	
설명	웹 서비스(게시판 등) 요청(스크립트 게시 등)과 응답결과(스크립트를 포함한 웹 페이지)에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 사용자로부터 입력 받은 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다. ② DB조회결과를 동적으로 생성되는 응답페이지에 사용하는 경우 HTML인코딩 또는 크로스 사이트 스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

SR1-6. 웹 기반 중요 기능 수행 요청 유효성 검증	
설명	비밀번호 변경, 결제 등 사용자 권한 확인이 필요한 중요기능을 수행할 때 웹 서비스 요청에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별 할 수 있도록 해야 한다.

SR1-7. HTTP 프로토콜 유효성 검증	
설명	비정상적인 HTTP 헤더, 자동연결 URL 링크 등 사용자가 원하지 않은 결과를 생성하는 HTTP 헤더·응답결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 외부입력값을 쿠키 및 HTTP 헤더정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다. ② 외부입력값이 페이지이동(리다이렉트 또는 포워드)을 위한 URL로 사용되어야 하는 경우, 해당 값은 시스템에서 허용된 URL목록의 선택자로 사용되도록 해야 한다.

SR1-8. 허용된 범위내 메모리 접근	
설명	설명 해당 프로세스에 허용된 범위의 메모리 버퍼에만 접근하여 읽기 또는 쓰기 기능을 하도록 검증방법 설계 및 메모리 접근요청이 허용범위를 벗어났을 때 처리방법 설계
요구 사항 내용	① C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계 값을 넘어서 메모리를 읽거나 저장하지 않도록 경계설정 또는 검사를 반드시 수행해야 한다. ② 개발 시, 메모리 버퍼오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 통제해야 한다.

SR1-9. 보안기능 입력값 검증	
설명	설명 보안기능(인증, 권한부여 등) 입력 값과 함수(또는 메소드)의 외부입력 값 및 수행결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계
요구 사항 내용	① 사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다. ② 쿠키값, 환경변수, 파라미터값 등 외부입력값이 보안기능을 수행하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다. ③ 중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송해야 하는 경우에는 해당 정보를 암호화해서 전송해야 한다.

SR1-10. 업로드·다운로드 파일 검증	
설명	설명 업로드·다운로드 파일의 무결성, 실행권한 등에 관한 유효성 검증방법 설계 및 부적합한 파일에 대한 처리방법 설계
요구 사항 내용	① 업로드되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다. ② 업로드되어 저장되는 파일은 외부에서 식별되지 않아야 한다. ③ 파일 다운로드 요청 시, 요청파일명에 대한 검증작업을 수행해야 한다. ④ 다운로드 받은 소스코드나 실행파일은 무결성 검사를 실행해야 한다.

2. 보안기능

인증, 접근통제, 권한관리, 비밀번호 등의 정책이 적절하게 반영될 수 있도록 설계

SR2-1. 인증 대상 및 방식	
설명	중요정보·기능의 특성에 따라 인증방식을 정의하고 정의된 인증방식을 우회하지 못하게 설계
요구 사항 내용	① 중요기능이나 리소스에 대해서는 인증 후 사용 정책이 적용되어야 한다. ② 안전한 인증방식을 사용하여 인증우회나 권한상승이 발생하지 않도록 해야 한다. ③ 중요기능에 대해 2단계(2-factor)인증을 고려해야 한다.

SR2-2. 인증 수행 제한	
설명	반복된 인증 시도를 제한하고 인증 실패한 이력을 추적하도록 설계
요구 사항 내용	① 로그인 기능 구현 시, 인증시도 횟수를 제한하고 초과된 인증시도에 대해 인증제한 정책을 적용해야 한다. ② 실패한 인증시도에 대한 정보를 로깅하여 인증시도 실패가 추적될 수 있게 해야 한다.

SR2-3 비밀번호 관리	
설명	생성규칙, 저장방법, 변경주기 등 비밀번호 관리정책별 안전한 적용방법 설계
요구 사항 내용	① 비밀번호를 설정할 때 한국인터넷진흥원의 『암호이용안내서』의 비밀번호 생성규칙을 적용해야 한다. ② 네트워크로 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화하거나 암호화된 통신 채널을 이용해야 한다. ③ 비밀번호 저장 시, 솔트가 적용된 안전한 해쉬함수를 사용해야 하며, 해쉬함수 실행은 서버에서 해야 한다. ④ 비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다. ⑤ 비밀번호 관리 규칙을 정의해서 적용해야 한다.

SR2-4. 중요자원 접근통제	
설명	중요자원(프로그램 설정, 민감한 사용자 데이터 등)을 정의하고, 정의된 중요자원에 대한 신뢰할 수 있는 접근통제 방법(권한관리 포함) 설계 및 접근통제 실패 시 처리방법 설계
요구 사항 내용	① 중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다. ② 중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다. ③ 관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.

SR2-5. 암호키 관리	
설명	암호키 생성, 분배, 접근, 파기 등 암호키 생명주기별 암호키 관리방법을 안전하게 설계
요구 사항 내용	① DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다. ② 설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉터리에 보관해야 한다.

SR2-6. 암호연산	
설명	국제표준 또는 검증필 암호모듈로 등재된 안전한 암호 알고리즘을 선정하고 충분한 암호키 길이, 솔트, 충분한 난수 값을 적용한 안전한 암호연산 수행방법 설계
요구 사항 내용	① 대칭키 또는 비대칭키를 이용해서 암호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호이용 안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다. ② 복호화되지 않는 암호화를 수행하기 위해 해쉬함수를 사용하는 경우 안전한 해쉬 알고리즘과 솔트 값을 적용하여 암호화해야 한다. ③ 난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.

SR2-7. 중요정보 저장	
설명	중요정보(비밀번호, 개인정보 등)를 저장·보관하는 방법이 안전하도록 설계
요구 사항 내용	① 중요정보 또는 개인정보는 암호화해서 저장해야 한다. ② 불필요하거나 사용하지 않는 중요정보가 메모리에 남지 않도록 해야 한다.

SR2-8. 중요정보 전송	
설명	중요정보(비밀번호, 개인정보, 쿠키 등)를 전송하는 방법이 안전하도록 설계
요구 사항 내용	① 인증정보와 같은 민감한 정보 전송 시 안전하게 암호화해서 전송해야 한다. ② 쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.

3. 에러처리

에러 또는 오류상황을 처리하지 않거나 불충분하게 처리되어 중요정보 유출 등 보안약점이 발생하지 않도록 설계

SR3-1. 예외처리	
설명	오류메시지에 중요정보(개인정보, 시스템 정보, 민감 정보 등)가 노출되거나, 부적절한 에러·오류 처리로 의도치 않은 상황이 발생하지 않도록 설계
요구 사항 내용	① 명시적인 예외의 경우 예외처리 블럭을 이용하여 예외발생시 수행해야 하는 기능이 구현되도록 해야 한다. ② 런타임 예외의 경우 입력값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다. ③ 에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.

4. 세션통제

다른 세션간 데이터 공유 금지 등 세션을 안전하게 관리할 수 있도록 설계

SR4-1. 세션통제	
설명	설명 다른 세션 간 데이터 공유금지, 세션ID 노출금지, (재)로그인시 기존 세션ID 재사용금지 등 안전한 세션 관리방안 설계
요구 사항 내용	① 세션간 데이터가 공유되지 않도록 설계해야 한다. ② 세션이 안전하게 관리되도록 해야 한다. ③ 세션ID가 안전하게 관리되도록 해야 한다.

| 제 2 절 | 구현단계 보안약점 제거 기준

1. 입력데이터 검증 및 표현

번호	보안약점	설명
1	SQL 삽입	SQL 질의문을 생성할 때 검증되지 않은 외부 입력 값을 허용하여 악의적인 질의문이 실행가능한 보안약점
2	코드 삽입	프로세스가 외부 입력 값을 코드(명령어)로 해석·실행할 수 있고 프로세스에 검증되지 않은 외부 입력 값을 허용한 경우 악의적인 코드가 실행 가능한 보안약점
3	경로 조작 및 자원 삽입	시스템 자원 접근경로 또는 자원제어 명령어에 검증되지 않은 외부 입력값을 허용하여 시스템 자원에 무단 접근 및 악의적인 행위가 가능한 보안약점
4	크로스사이트 스크립트	사용자 브라우저에 검증되지 않은 외부 입력값을 허용하여 악의적인 스크립트가 실행 가능한 보안약점
5	운영체제 명령어 삽입	운영체제 명령어를 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 명령어가 실행 가능한 보안약점
6	위험한 형식 파일 업로드	파일의 확장자 등 파일형식에 대한 검증없이 파일 업로드를 허용하여 공격이 가능한 보안약점
7	신뢰되지 않는 URL 주소로 자동접속 연결	URL 링크 생성에 검증되지 않은 외부 입력값을 허용하여 악의적인 사이트로 자동 접속 가능한 보안약점
8	부적절한 XML 외부 개체 참조	임의로 조작된 XML 외부개체에 대한 적절한 검증 없이 참조를 허용하여 공격이 가능한 보안약점
9	XML 삽입	XQuery, XPath 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점
10	LDAP 삽입	LDAP 명령문을 생성할 때 검증되지 않은 외부 입력 값을 허용하여 악의적인 명령어가 실행가능한 보안약점
11	크로스사이트 요청 위조	사용자 브라우저에 검증되지 않은 외부 입력 값을 허용하여 사용자 본인의 의지와는 무관하게 공격자가 의도한 행위가 실행 가능한 보안약점
12	서버사이드 요청 위조	서버 간 처리되는 요청에 검증되지 않은 외부 입력값을 허용하여 공격자가 의도 한 서버로 전송하거나 변조하는 보안약점
13	HTTP 응답분할	HTTP 응답헤더에 개행문자(CR이나 LF)가 포함된 검증되지 않은 외부 입력값을 허용하여 악의적인 코드가 실행 가능한 보안약점
14	정수형 오버플로우	정수형 변수에 저장된 값이 허용된 정수 값 범위를 벗어나 프로그램이 예기치 않게 동작 가능한 보안약점
15	보안기능 결정에 사용되는 부적절한 입력값	보안기능(인증, 권한부여 등) 결정에 검증되지 않은 외부 입력값을 허용하여 보안기능을 우회하는 보안약점
16	메모리 버퍼 오버플로우	메모리 버퍼의 경계값을 넘어서 메모리값을 읽거나 저장하여 예기치 않은 결과 가 발생하는 보안약점
17	포맷 스트링 삽입	printf 등 포맷 스트링 제어함수에 검증되지 않은 외부 입력값을 허용하여 발생하는 보안약점 * 포맷 스트링: 입·출력에서 형식이나 형태를 지정해주는 문자열

2. 보안기능

번호	보안약점	설명
1	적절한 인증 없는 중요 기능 허용	중요정보(금융정보, 개인정보, 인증정보 등)를 적절한 인증없이 열람(또는 변경) 가능한 보안약점
2	부적절한 인가	중요자원에 접근할 때 적절한 제어가 없어 비인가자의 접근이 가능한 보안약점
3	중요한 자원에 대한 잘못된 권한 설정	중요자원에 적절한 접근 권한을 부여하지 않아 중요정보가 노출·수정 가능한 보안약점
4	취약한 암호화 알고리즘 사용	중요정보(금융정보, 개인정보, 인증정보 등)의 기밀성을 보장할 수 없는 취약한 암호화 알고리즘을 사용하여 정보가 노출 가능한 보안약점
5	암호화되지 않은 중요정보	중요정보(비밀번호, 개인정보 등) 전송 시 암호화 또는 안전한 통신채널을 이용하지 않거나, 저장 시 암호화하지 않아 정보가 노출 가능한 보안약점
6	하드코드된 중요정보	소스코드에 중요정보(비밀번호, 암호화키 등)를 직접 코딩하여 소스코드 유출 시 중요정보가 노출되고 주기적 변경이 어려운 보안약점
7	충분하지 않은 키 길이 사용	암호화 등에 사용되는 키의 길이가 충분하지 않아 데이터의 기밀성·무결성을 보장할 수 없는 보안약점
8	적절하지 않은 난수 값 사용	사용한 난수가 예측 가능하여, 공격자가 다음 난수를 예상해서 시스템을 공격 가능한 보안약점
9	취약한 비밀번호 허용	비밀번호 조합규칙(영문, 숫자, 특수문자 등) 미흡 및 길이가 충분하지 않아 비밀번호가 노출 가능한 보안약점
10	부적절한 전자서명 확인	프로그램, 라이브러리, 코드의 전자서명에 대한 유효성 검증이 적절하지 않아 공격자의 악의적인 코드가 실행 가능한 보안약점
11	부적절한 인증서 유효성 검증	인증서에 대한 유효성 검증이 적절하지 않아 발생하는 보안약점
12	사용자 하드디스크에 저장되는 쿠키를 통한 정보노출	쿠키(세션 ID, 사용자 권한정보 등 중요정보)를 사용자 하드디스크에 저장되어 중요정보가 노출 가능한 보안약점
13	주석문 안에 포함된 시스템 주요정보	소스코드 주석문에 인증정보 등 시스템 주요정보가 포함되어 소스코드 노출 시 주요정보도 노출 가능한 보안약점
14	솔트 없이 일방향 해쉬 함수 사용	솔트를 사용하지 않고 생성된 해쉬 값으로부터 공격자가 미리 계산된 레인보우 테이블을 이용하여 해쉬 적용 이전 원본 정보를 복원가능한 보안약점 *솔트 : 해쉬 적용하기 전 평문인 전송정보에 덧붙인 무의미한 데이터
15	무결성 검사 없는 코드 다운로드	소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받아 실행하는 경우, 공격자의 악의적인 코드가 실행 가능한 보안약점
16	반복된 인증시도 제한 기능 부재	인증 시도 수를 제한하지 않아 공격자가 반복적으로 임의 값을 입력하여 계정 권한을 획득 가능한 보안약점

3. 시간 및 상태

번호	보안약점	설명
1	경쟁조건 : 검사 시점과 사용 시점(TOCTOU)	멀티 프로세스 상에서 자원을 검사하는 시점과 사용하는 시점이 달라서 발생하는 보안약점
2	종료되지 않는 반복문 또는 재귀함수	종료조건 없는 제어문 사용으로 반복문 또는 재귀함수가 무한히 반복되어 발생 할 수 있는 보안약점

4. 에러처리

번호	보안약점	설명
1	오류 메시지 정보노출	오류메시지나 스택정보에 시스템 내부구조가 포함되어 민감한 정보, 디버깅 정보가 노출 가능한 보안약점
2	오류상황 대응 부재	시스템 오류상황을 처리하지 않아 프로그램 실행정지 등 의도하지 않은 상황이 발생 가능한 보안약점
3	부적절한 예외 처리	예외사항을 부적절하게 처리하여 의도하지 않은 상황이 발생 가능한 보안약점

5. 코드오류

번호	보안약점	설명
1	Null Pointer 역참조	변수의 주소 값이 Null인 객체를 참조하는 보안약점
2	부적절한 자원 해제	사용 완료된 자원을 해제하지 않아 자원이 고갈되어 새로운 입력을 처리할 수 없는 보안약점
3	해제된 자원 사용	메모리 등 해제된 자원을 참조하여 예기치 않은 오류가 발생하는 보안약점
4	초기화되지 않은 변수 사용	변수를 초기화하지 않고 사용하여 예기치 않은 오류가 발생하는 보안약점
5	신뢰할 수 없는 데이터의 역직렬화	악의적인 코드가 삽입·수정된 직렬화 데이터를 적절한 검증 없이 역질렬화 하여 발생하는 보안약점 * 직렬화: 객체를 전송 가능한 데이터형식으로 변환 * 역직렬화: 직렬화된 데이터를 원래 객체로 복원

6. 캡슐화

번호	보안약점	설명	페이지 번호
1	잘못된 세션에 의한 데이터 정보노출	잘못된 세션에 의해 인가되지 않은 사용자에게 중요 정보가 노출 가능한 보안약점	
2	제거되지 않고 남은 디버그 코드	디버깅을 위한 코드를 제거하지 않아 인가되지 않은 사용자에게 중 요정보가 노출 가능한 보안약점	
3	Public 메서드부터 반환된 Private 배열	Public으로 선언된 메소드에서 Private로 선언된 배열을 반환(return)하면 Private 배열의 주소 값이 외부에 노출되어 해당 Private 배열값을 외부에서 수정 가능한 보안약점	
4	Private 배열에 Public 데이터 할당	Public으로 선언된 데이터 또는 메소드의 인자가 Private으로 선언된 배열에 저장되면 이 Private 배열을 외부에서 접근하여 수정 가능한 보안약점	

7. API 오용

번호	보안약점	설명	페이지 번호
1	DNS lookup에 의존한 보안결정	도메인명 확인(DNS lookup)으로 보안결정을 수행할 때 악의적으로 변조된 DNS 정보로 예기치 않은 보안 위협에 노출되는 보안약점	
2	취약한 API 사용	취약한 함수를 사용해서 예기치 않은 보안위협에 노출되는 보안약점	

| 제 3 절 | 설계단계 보안설계 산출물 적용 예

1. 입력데이터 검증 및 표현

1-1. DBMS 조회 및 결과 검증

다음은 “DBMS 조회 및 결과 검증” 설계항목에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “DBMS 조회 및 결과 검증” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구 사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요도	해결방안	검수기준	비고
SR-0101101	DB 사용자 계정에 최소권한 부여	비기능	애플리케이션에서 사용하는 DB 사용자 계정에 대해 최소 권한이 사용될 수 있도록 해야 한다.	RFP 보안요구사항 1.1	없음	상	애플리케이션에서 사용하는 DB 사용자 계정은 애플리케이션에서 사용하는 테이블, 뷰, 프로시저에 대해서만 사용 권한을 부유여하다.	애플리케이션에서 사용 하는 DB 사용자 계정은 해당 애플리케이션 에서 사용하는 테이블 , 뷰, 프로시저에 대해서만 사용 권한이 적용 된다.	지속적인 DB 사용자 계정 관리가 필요
SR-010102	정적 SQL 사용	비기능	모든 쿼리는 정적으로 실행되도록 한다.	RFP 보안요구사항 1.1	없음	상	외부 입력값을 Query Map에 바인딩할 경우에는 #을 이용하도록 개발 가이드에 명시 하고, 개발보안 교육으로 개발자에게 전파하여 지켜질 수 있도록 한다.	동적 SQL 실행이 존재 하지 않는다.	정적분석으로 지속적인 검사가 필요
SR-010103	동적 SQL 사용 시 입력값 검증	비기능	동적으로 SQL문이 생성, 실행되어야 하는 경우, 반드시 입력값 검증 후 사용되어야 한다.	RFP 보안 요구 사항 1.1	없음	상	아키텍처 정의에 따라 모든 쿼리는 MyBatis 프레임워크의 쿼리맵으로 정의, 실행한다. 이때, 변수 바인딩은 SR 010102 에 따라 # 기호를 이용한다.	해당 없음	SR-010102 로 통합

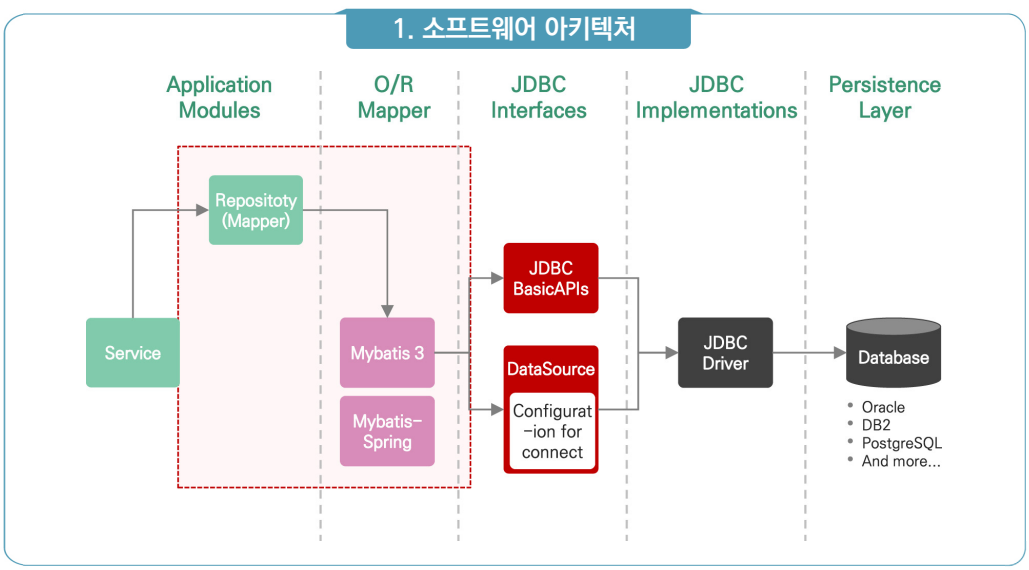
요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “DBMS 조회 및 결과 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		...	아키텍처정의서	개발 가이드	...	...	...
요구사항ID	요구사항 명						
SR-010101	DB 사용자 계정에 최소 권한 부여			1.1.1			
SR-010102	정적 SQL 사용		소프트웨어 아키텍처 아키텍처 요구사항 및 구현방안 SR-010102	1.1.2			
SR-010103	동적 SQL 사용시 입력값 검증		소프트웨어 아키텍처 아키텍처 요구사항 및 구현방안 SR-010102	1.1.3			

아키텍처 설계서의 소프트웨어 아키텍처에서는 MyBatis 프레임워크 사용을 명시하고 있으며, 아키텍처 요구사항 및 구현방안에서 MyBatis 프레임워크의 쿼리맵에 외부 입력값을 바인딩할 때 “#”기호를 이용해야 정적쿼리가 실행될 수 있도록 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010102
요구사항 내용	
동적으로 SQL문이 생성, 실행되지 않도록 해야 한다.	
구현방안	
SQL 삽입 취약점을 방어할 수 있도록 외부 또는 사용자 입력값을 MyBatis의 쿼리맵에 바인딩하는 경우, 반드시 "#" 기호를 이용하여 정의하도록 한다. 만약, "$" 기호를 사용하는 경우에는 파라미터로 전달되는 값이 해당 애플리케이션에서 정의한 상수 또는 고정된 값인 것을 보장해야 한다.	

개발가이드에는 안전한 DBMS 조회 및 결과검증에 필요한 데이터베이스 사용자 계정 및 권한 설정 방법, 정적 쿼리 실행 방법, 동적 쿼리 실행 시 입력값 검증 방법 등을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.1.1. 데이터베이스 사용자 계정 및 권한 설정

SQL 삽입 공격에 대응하기 위해서는 애플리케이션에서 사용하는 데이터베이스 사용자 계정은 필요한 만큼 최소한으로 설정하여야 한다.

특히, 관리자에게 제한된 권한이 부여되지 않도록 해야 하며, 해당 애플리케이션에서 필요한 테이블, 뷰, 프로시저 등에 대한 권한만 부여하여야 한다.

1.1.2. 정적 쿼리 실행

SQL 삽입 공격에 대응하기 위해서는 입력값의 형태와 관계없이 쿼리가 실행되는 정적 쿼리 실행을 구현해야 한다. Java의 경우, PreparedStatement 객체를 이용하여 정적 쿼리를 구현할 수 있으며, 적용 예는 아래와 같다.

```
String id = request.getParameter("id");
 :
String sql = "select * from users where id = ? ";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, Integer.parseInt(id));
rs = pstmt.executeQuery();
```

만약, MyBatis와 같은 ORM 프레임워크를 사용하는 경우에는 외부 입력값을 쿼리 맵에 바인딩할 때 "#" 기호를 이용하도록 한다.

```
〈select id="selectUser" parameterType="userVO"
 resultMap="userVO"〉 select * from users where id = #{userId}
〈/select〉
```

1.1.3. 동적 쿼리 실행 시 입력값 검증

정적 쿼리를 사용할 수 없는 경우에는 반드시 입력값 검증으로 쿼리 조작 문자열 포함 여부를 확인한 후 사용해야 한다. 해당 검증 로직은 공통 모듈로 개발하여 검증 규칙이 일관되게 적용될 수 있도록 한다.

```
st.creatStatement("select id from table where id='"+SQLFilter(id)+"'");
```

1-2. XML 조회 및 결과 검증

다음은 "XML 조회 및 결과 검증" 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 "XML 조회 및 결과 검증" 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010201	XML 조회 입력값 및 결과 검증	비기능	XML 문서 조회 기능 구현 시 XML 쿼리에 사용되는 파라미터는 반드시 XML 쿼리를 조작할 수 없도록 필터링해서 사용하거 나, 미리 작성된 쿼리 문에 입력값을 자료 형에 따라 바인딩 해서 사용해야 한다.	RFP 보안요구사항 1.2	없음	상	XQuery 표현식을 이용하여 피라미터화된 쿼리가 실행될 수 있도록 한다.	XPath 및 XQuery 삽입 공격을 방어한다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 조회 및 결과검증"보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		...	아키텍처정의서	개발 가이드	...	...	...
요구사항ID	요구사항 명						
SR-010201	XML 조회 입력값 및 결과 검증		아키텍처 요구사항 및 구현방안 SR- 010201	1.2.1			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 XML 문서에 대한 조회 기능을 파라미터화된 쿼리를 지원하는 XQuery 표현식을 사용하고, 외부 바인딩 함수를 이용하여 외부 입력값을 필터링하여 적용하도록 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010201
요구사항 내용	
XML 문서 조회 기능 구현 시 XML 쿼리에 사용되는 파라미터는 반드시 XML 쿼리를 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 쿼리문에 입력값을 자료형에 따라 바인딩해서 사용해야 한다.	
구현방안	
XML 문서에 대한 조회 기능은 XQuery 표현식을 이용하여 파라미터화된 쿼리를 만들고, 바인딩 함수를 이용하여 입력 값을 안전한 문자열로 필터링하여 적용, 실행될 수 있도록 한다.	

개발가이드에는 안전한 XML 조회 및 결과 검증에 필요한 XQuery 표현식의 사용 방법과 필터링 방법 등을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.2.1. XML 쿼리

XML 문서에 대한 조회 기능은 XQuery 표현식을 이용하여 파라미터화된 쿼리를 만들고, 바인딩 함수를 이용하여 입력 값을 안전한 문자열로 필터링하여 적용, 실행될 수 있도록 한다.

```
String es =
“doc(‘users.xml’)/userlist/user[uname=’$xname’]”;
XQPreparedExpression expr = conn.prepareExpression(es);
expr.bindString(new QName(“xname”), name, null);
XQResultSequence result = expr.executeQuery();
while (result.next()) {
 String str =
 result.getAtomicValue(); if
 (str.indexOf(‘>’) < 0) {
 System.out.println(str);
 }
}
```

1.2.1. XML 쿼리

XQuery 표현식을 사용할 수 없는 경우에는 입력값에 쿼리를 조작할 수 있는 특수기호 및 예약어가 포함되었는지를 확인하고 안전한 문자열로 필터링한 후 사용될 수 있도록 한다.

```
“ [ ] / = @
```

해당 필터링 로직은 공통 모듈로 개발하여 필터링 규칙이 일관되게 적용될 수 있도록 한다.

1-3. 디렉토리 서비스 조회 및 결과 검증

다음은 “디렉토리 서비스 조회 및 결과 검증” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “디렉토리 서비스 조회 및 결과 검증” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010301	LDAP 인증 시 입력값 검증	비기능	LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용 되는 외부 입력 값은 LDAP 삽입 취약점을 가지지 않도록 필터링 해서사용해야한다.	RFP 보안요구사항 1.3	없음	상	인증서로 인증 을 구현해야 한다.	LDAP 필터 규칙으로 인식가능한특수문자를 제거하는 기능을 공통 모듈로 개발하여 개발 자가 구현에 사용할 수 있도록한다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “디렉토리 서비스 조회 및 결과 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

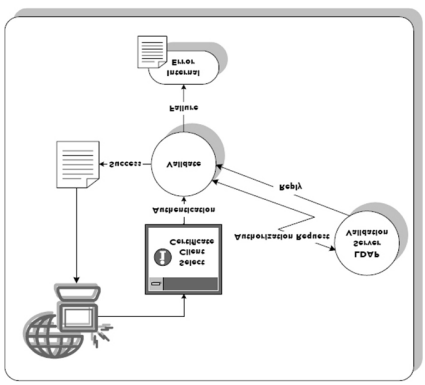
안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-010301	LDAP 인증 시 입력값 검증		아키텍처 요구사항 및 구현방안 SR- 010301	1.3.1			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 LDAP 삽입 취약점이 발생하지 않도록 LDAP 필터의 값으로 외부 입력값을 사용하는 경우 LDAP 필터 조작 문자열을 필터링하고 사용하도록 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

1. 소프트웨어 아키텍처



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010301
요구사항 내용	
LDAP 인증서버로 인증을 구현하는 경우 인증요청을 위해 사용되는 외부 입력값은 LDAP 삽입 취약점을 가지지 않도록 필터링해서 사용해야 한다.	
구현방안	
LDAP 인증서버로 인증처리 시 LDAP 삽입 취약점이 발생하지 않도록, LDAP 필터 규칙으로 인식 가능한 특수 문자를 제거하는 기능을 공통 모듈에 추가하여 인증처리에 사용될 수 있도록 한다.	

개발가이드에는 안전한 LDAP 인증을 위한 필터링 방법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.3.1. LDAP 인증

LDAP 조회 시 LDAP 삽입과 같은 취약점이 발생하지 않도록 하기 위해 DN(Distinguished Name)과 필터에 사용되는 사용자 입력값에 특수문자가 포함되지 않도록 특수문자를 제거한다. 만약 특수문자를 사용해야 하는 경우에는 특수 문자가 실행명령이 아닌 일반문자로 인식되도록 처리한다.

1.3.1. LDAP 인증

```
if (!userSN.matches("[₩₩w₩₩s]*")
|| !userPassword.matches("[₩₩w]*")) { throw new
IllegalArgumentException("Invalid input");
}
```

해당 검증 로직은 공통 모듈로 개발하여 검증 규칙이 일관되게 적용될 수 있도록 한다.

1-4. 시스템 자원 접근 및 명령어 수행 입력값 검증

다음은 "시스템 자원 접근 및 명령어 수행 입력값 검증" 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 "시스템자원 접근 및 명령어 수행 입력값 검증" 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010401	경로조작및자원삽입취약점방어	비기능	외부 입력값을 이용하여 시스템 자원(IP, PORT 번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다.	RFP 보안요구사항 1.4	시스템 자원을 식별, 참조하는 부분이 존재 해야 한다.	상	사용가능한 자원의 식별자를 미리 정의하고 정의된 범위 내에서 사용되도록 한다. 파일의 경우 경로조작문자열을 필터링한다.	경로조작 및 자원 삽입 공격을 방어한다.	
SR-010402	운영체제명령어삽입취약점방어	비기능	서버프로그램안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부 입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.	RFP 보안요구사항 1.4	쉘 명령어 실행 부분이 존재 해야 한다.	상	사용가능한 명령어를 미리 정의하고 정의된 범위내에서 사용되도록 한다.	운영체제 명령어 삽입 공격을 방어한다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “시스템자원 접근 및 명령어 수행 입력값 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-010401	경로조작 및 자원삽입 취약점 방어		아키텍처 요구사항 및 구현방안 SR- 010401	1.4.1			
SR-010402	운영체제 명령어 삽입 취약점 방어		아키텍처 요구사항 및 구현방안 SR- 010402	1.4.2			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 경로조작 및 자원삽입 취약점 방어를 위한 방법과 운영체제 명령어 삽입 취약점 방어를 위한 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010401

요구사항 내용

외부 입력값을 이용하여 시스템 자원(IP, PORT 번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야한다.

구현방안

애플리케이션에서 필요로 하는 자원 목록을 도출한다.

자원	식별자	용도	비교
IP	192.168.10.31	업로드 파일 참조	
PORT	8888	파일 업로드	안전한 보안설계의 예 : 아키텍처 설계서
PORT	9999	파일 다운로드	

자원참조가 필요한 경우, 입력받은 자원의 식별자가 허용목록 내의 값인지를 확인하는 공통모듈을 추가하고, 이를 이용하여 허용목록 내의 식별자만 사용될 수 있도록 한다.

파일의 경우, 입력받은 파일이름 또는 파일경로에 경로조작 문자열 포함 여부를 확인, 제거하는 공통모듈을 추가하고, 이를 이용하여 필터링한 값만 사용될 수 있도록 한다.

2. 아키텍처 요구사항 및 구현방안

요구사항ID	SR-010402

요구사항 내용

서버 프로그램 안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부 입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

구현방안

외부 입력에 따라 실행되어야 하는 운영체제 명령어의 목록을 도출한다.

명령어	식별자	비교
start.sh	웹 서버 인스턴스를 시작한다.	/usr/local/server/에 위치
stop.sh	웹 서버 인스턴스를 중지한다.	/usr/local/server/에 위치
restart.sh	웹 서버 인스턴스를 다시 시작한다.	/usr/local/server/에 위치

운영체제 명령어 실행이 필요한 경우, 입력받은 명령어가 허용목록 내의 값인지를 확인하는 공통모듈을 추가하고, 이를 이용하여 허용목록 내의 명령어만 실행되도록 한다.

해당 공통모듈에 추가 명령어 실행에 사용되는 문자열을 제거하는 기능을 추가한다.

개발가이드에는 안전한 시스템자원 접근 및 명령어 수행을 위한 입력값 검증 방법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.4.1. 서버 자원 참조

외부에서 입력받은 값을 자원의 식별자로 사용하는 경우, 애플리케이션에서 필요로 하는 자원을 미리 정의하고, 정의된 범위 내에서 선택하여 사용되도록 한다.

```
String service = props.getProperty("Service
No"); if ("".equals(service)) service = "3000";
int port = Integer.parseInt(service);
switch(port) {
 case 1 : port = 3001;
 break; case 2 : port =
 3002; break; case 3 :
 port = 3003; break;
 default: port = 3000;
}
serverSocket = new ServerSocket(port);
```

외부에서 입력받은 값을 파일의 경로 또는 이름으로 사용하는 경우, 경로조작 공격을 유발할 수 있는 문자열을 제거한 후 사용되도록 한다.

1.4.1. 서버 자원 참조

```
String helpFile =
args[0]; if (helpFile !=
null) {
helpFile = helpFile.replaceAll("\\.{2,}[/\\\\]", "");
}
BufferedReader br = new BufferedReader(new FileReader(safeDir + helpFile));
```

해당 검증 로직은 공통 모듈로 개발하여 검증 규칙이 일관되게 적용될 수 있도록 한다.

1.4.2. 운영체제 명령어 실행

가급적이면 웹 인터페이스로 서버 내부로 시스템 명령어를 전달시키지 않도록 한다.

외부 입력에 따라 명령어를 생성하거나 선택이 필요한 경우에는 명령어 생성에 필요한 값들을 미리 지정해 놓고 외부 입력에 따라 선택하여 사용하도록 한다.

```
List<String> allowedCommands = new ArrayList<String>();
allowedCommands.add("notepad");
allowedCommands.add("calc");
 :
String cmd =
request.getParameter("cmd"); if
(!allowedCommands.contains(cmd)) {
 throw new Exception("허용되지 않은 명령어입니다.");
}
 :
Runtime.getRuntime().exec(cmd);
 :
```

해당 검증 로직은 공통 모듈로 개발하여 검증 규칙이 일관되게 적용될 수 있도록 한다.

1-5. 웹 서비스 요청 및 결과 검증

다음은 “웹 서비스 요청 및 결과 검증” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “웹 서비스 요청 및 결과 검증” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010501	반사XSS 취약점방어	비기능	사용자로부터 입력받은 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스사이트 필터링을 수행한 뒤 사용해야 한다.	RFP 보안요구사항 1.5	없음	상	서블릿 필터 컴포넌트에 XSS Filter 모듈을 추가 한다.	반사 XSS 공격이 차단 된다.	
SR-010502	저장XSS 취약점방어		DB조회 결과를 동적으로 생성 되는 응답페이지에 사용하는 경우 HTML인코딩또는 크로스사이트 스크립트 필터링을 수행한 뒤 사용해야 한다.	RFP 보안요구사항 1.5	없음	상	<c:out>을 이용하여 출력한다.	저장 XSS 공격이 차단된다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “웹 서비스 요청 및 결과 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

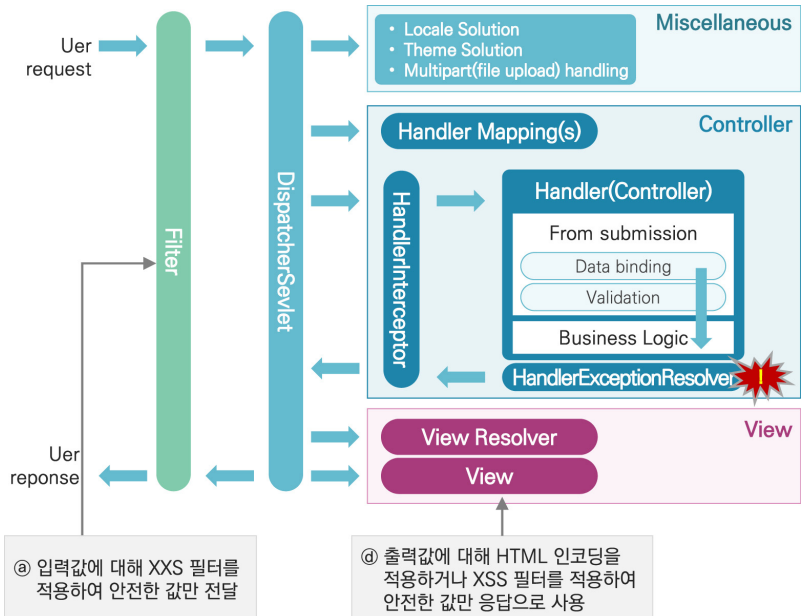
안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-010501	반사 XSS 취약점 방어		아키텍처 요구사항 및 구현방안 SR- 010501	1.5.1			
SR-010502	저장 XSS 취약점 방어		아키텍처 요구사항 및 구현방안 SR- 010502	1.5.2			

아키텍처 설계서의 소프트웨어 아키텍처에서는 입력값 검증과 출력값 필터링 위치를 명시하고 있으며, 아키텍처 요구사항 및 구현방안에서는 반사 XSS와 저장 XSS 방어 기법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

1. 소프트웨어 아키텍처



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010501
요구사항 내용	
사용자로부터 입력받은 값을 동적으로 생성되는 응답페이지에 사용하는 경우 크로스사이트 필터링을 수행한 뒤 사용해야 한다.	
구현방안	
사용자로부터 입력받은 값을 동적으로 생성되는 응답 페이지에 사용하는 경우, 크로스사이트 스크립트 필터링을 수행한 뒤 사용되도록 한다. 해당 필터링 기능은 공통 기능으로 서블릿 필터 컴포넌트로 작성하여 모든 애플리케이션에서 일괄되게 적용되도록 한다.	

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010502
요구사항 내용	
DB조회 결과를 동적으로 생성되는 응답 페이지에 사용하는 경우 HTML 인코딩 또는 크로스사이트스크립트 필터링을 수행한 뒤 사용해야 한다.	
구현방안	
DB로부터 조회한 값을 동적으로 생성되는 응답 페이지에 사용하는 경우, 크로스사이트 스크립트 필터링을 수행한 뒤 사 용되도록 한다. 필터링 기능은 안전성이 검증된 Lucy XSS Filter 라이브러리를 이용하여 구현하고, 필터링 정책이 일관성 있게 적용될 수 있도록 공통모듈로 등록하여 사용한다.	

개발가이드에는 XSS 공격을 방어에 필요한 XSS 필터링 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.5.1. XSS 필터링

외부에서 입력받은 값 또는 DB에서 조회한 값을 동적으로 생성되는 응답 페이지에 사용하거는 경우, 크로스사이트 스크립트 필터링을 수행한 후 사용되도록 한다.

```
public static String xssFilter(String s) { if (
 s != null) {
 s = s.replaceAll("<", "&lt;");
 s = s.replaceAll(">", "&gt;");
 }
 return s;
}
```

Lucy-XSS-Filter와 같은 검증된 필터 라이브러리를 사용하면 보다 안전하고 효율적으로 필터링을 수행할 수 있다.

```
XssFilter filter = XssFilter.getInstance("lucy-xss
superset.xml"); String out = filter.doFilter(in);
```

해당 필터 로직은 공통 모듈로 개발하여 필터링 규칙이 일관되게 적용될 수 있도록 한다.

별도의 필터링 모듈 구현이 어려운 경우에는 JSTL의 코어 태그 또는 함수를 이용하여 동적 페이지를 생성한다.

1.5.1. XSS 필터링

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn" %>
 :
<textarea name="content"><c:out value="${model.content}"/></textarea>
<textarea name="content">${fn:escapeXml(model.content)}</textarea>
```

1-6. 웹 기반 중요 기능 수행 요청 유효성 검증

다음은 "웹 기반 중요 기능 수행 요청 유효성 검증" 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 "웹 기반 중요기능 수행 요청 유효성 검증" 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010601	웹 기반 중요기능 수행 요청 유효성 검증	비기능	시스템으로전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별 할 수 있도록 해야 한다.	RFP 보안요구사항 1.6	없음	상	CAPTCHA를 이용하여 사용자에 의한 정상적인 절차에 따른 요청인지를 확인한다.	자동화된 요청이 처리되지 않는다.	

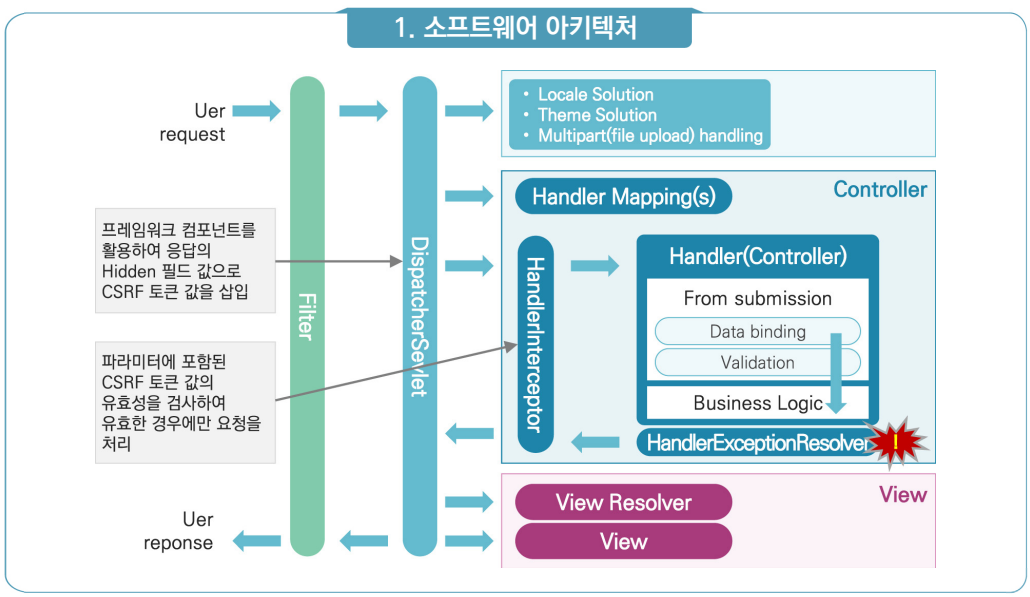
요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “웹 기반 중요기능 수행 요청 유효성 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계	
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이스	…
요구사항ID	요구사항 명							
SR-010601	웹 기반 중요기능 수행 요청 유효성 검증		아키텍처 요구사항 및 구현방안 SR-010601	1.6.1~ 1.6.3			UTC-010601	

아키텍처 설계서의 소프트웨어 아키텍처에서는 정상적인 절차에 따른 요청인지를 검증할 CSRF 토큰 생성 및 검증 위치를 명시하고 있으며, 아키텍처 요구사항 및 구현방안에서는 CSRF 방어 기법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010601

요구사항 내용

시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별할 수 있도록 해야 한다.

구현방안

① 정상적인 절차에 따른 요청인지 여부를 확인할 수 있도록 CSRF 토큰을 운영한다.
- 세션별로 CSRF 토큰을 생성하여 세션에 저장
- 사용자가 작업(입력) 페이지를 요청하면 토큰을 함께 전달
- 해당 클라이언트에서 데이터 처리를 요청하면 함께 전달된 토큰과 세션의 토큰을 비교
- 일치하는 경우 처리, 일치하지 않는 경우 오류 처리

② CSRF 토큰 검증은 Spring 프레임워크의 인터셉터(interceptor) 컴포넌트로 구현한다.

③ XSS 취약점을 이용한 자동화된 공격을 방어할 수 있도록 CAPTCHA를 이용하여 사용자와 상호 처리 가능하도록 한다.

④ 중요한 기능의 경우 재인증을 요구한다.
- 중요한 기능은 데이터가 생성, 수정, 삭제, 변경되는 경우
- 개인정보, 금융정보, 인증정보 등이 변경되는 경우
- 비밀번호 변경, 회원 정보 수정, 계좌 이체, ... 등

개발가이드에는 주요기능에 CSRF 토큰 운영, 주요 기능에 대해 재인증 처리와 같은 CSRF 공격 방어 기법과 프레임워크 및 라이브러리 정보를 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.6.1. 주요기능에 CSRF 토큰 운영

정상적인 절차에 따른 요청인지 여부를 확인하기 위해, 작업(입력) 페이지를 요청하면 서버에서 토큰을 생성하고 세션에 저장한다.

```
// 게시판 글쓰기 페이지 요청을 처리
@RequestMapping("/write.do")
public String boardWrite(@ModelAttribute("BoardModel")
BoardModel
boardModel, HttpSession session) {
 // 난수 형태의 토큰을 생성하여 세션에 저장
 session.setAttribute("SESSION_CSRF_TOKEN",
 UUID.randomUUID().toString()); return "/board/write";
}
```

```
생성한 토큰을 작업(입력) 페이지에 히든 필드의 값으로 전달한다.
 :
```

1.6.1. 주요기능에 CSRF 토큰 운영

```
〈input type="hidden" name="PARAM_CSRF_TOKEN"
value="${SESSION_CSRF_TOKEN}"/〉
 :
```

XSS 취약점을 이용한 자동화된 공격을 방어하기 위해서는 CAPTCHA를 이용하여 사용자와 상호 처리 가능하도록 해야 한다.

클라이언트에서 데이터 처리를 요청하면 세션에 저장된 토큰과 파라미터로 전달된 토큰을 비교하여 일치하는 경우에는 요청을 처리하고, 일치하지 않는 경우에는 오류 처리한다.

```
String sessionToken = (String)session.getAttribute("SESSION_CSRF_TOKEN");
String paramToken = request.getParameter("PARAM_CSRF_TOKEN");
if (paramToken != null && paramToken.equals(sessionToken)) {
 // 토큰 일치 = 정상적인 절차에 따른 요청 -〉 요청 처리
} else {
 // 토큰 불일치 = 비정상적인 절차에 따른 요청 -〉 오류 처리
}
:
```

토큰 검증 로직은 개별 컨트롤러 또는 인터셉터(interceptor)와 같은 공통 모듈에서 구현할 수 있다.

1.6.2. 주요 기능에 대해 재인증 처리

비밀번호 변경, 회원 정보 수정, 계좌 이체, ... 와 같은 주요 기능에 대해서는 요청을 처리하기 전에 재인증을 수행하며, 필요에 따라서는 추가 인증수단 도입을 검토한다.

1.6.3. CSRF 방어를 위한 프레임워크 및 라이브러리

개발환경	활용 가능한 프레임워크 및 라이브러리
Java	• Spring Security
	• Apache Struts
	• OWASP CSRFGuard
Python	• Django
PHP	• CSRF Protector

단위시험 케이스에는 정상적인 절차에 따른 요청인지 확인을 위해 발행한 CSRF 토큰을 검증하는 절차와 방법을 제시하고 있다.

안전한 보안설계의 예 : 단위 테스트 케이스

요구사항 ID	SR-010601					
설명	관리자 및 사용자가 게시판에 등록, 조회, 수정, 삭제한다. - 게시판은 공지사항, FAQ, Q&A, 자료실 모두를 포함한다. - 공지사항과 자료실은 관리자만 등록, 수정, 삭제할 수 있다.					
관련 컴포넌트 ID	CMT_010601			관련 프로그램 ID		
케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC_010601	게시판 글쓰기/ CSRF 토큰 검증	전체	게시물 데이터 (제목, 내용, 첨부 파일 등)	① 메뉴에서 게시판을 선택 ② 조회 화면 하단의 "등록" 버튼 클릭 ③ 제목, 내용, 첨부파일을 입력하고 화면 하단에 "저장" 버튼 클릭 ④ 조회 화면에 등록 여부 확인	① 브라우저의 소스 보기 기능을 이용하여, PARAM_CSRF_TOKEN 히든 필드가 존재하는지 확인한다. ② 브라우저의 개발자 도구를 이용하여, PARAM_CSRF_TOKEN 히든 필드의 값을 수정하거나 삭제할 경우 게시물이 저장되지 않는 것을 확인한다. (오류 메시지 출력)	

1-7. HTTP 프로토콜 유효성 검증

다음은 "HTTP 프로토콜 유효성 검증" 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 "HTTP 프로토콜 유효성 검증" 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구 사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요도	해결방안	검수기준	비고
SR-010701	HTTP 응답분할 취약점 대응	비 기능	외부입력값을 쿠키 및 HTTP 헤더 정보로 사용 하는 경우, HTTP 응답 분할 취약점을 가지지 않도 록 필터링해서 사용해 야 한다.	RFP 보안요구사항 1.7	없음	상	응답헤더 구성에 사용 되는 외부 입력값에 개행문자열 포함 여부를 확인하고 필터링 후 사용한다.	HTTP 응답분할 이 발생하지 않는다.	
SR-010702	오픈 리다이렉트 취약점 대응	비 기능	외부입력값이 페이지 이동 (리다이 렉트 또는 포워드)을 위한 URL로 사용 되어야 하는 경우, 해당 값은 시스템에서 허용된 URL 목록의 선택자로 사용되도록 해야 한다.	RFP 보안요구사항 1.7	리다이렉트 기능을 제공해야 한다.	상	리다이렉트 가능한 도 메인과 주소를 미리 정의하고, 정의된 범위 내에서만 리다이렉트 되도록 한다.	허용 목록 밖의 주소로 리다이렉트 되지 않는다.	

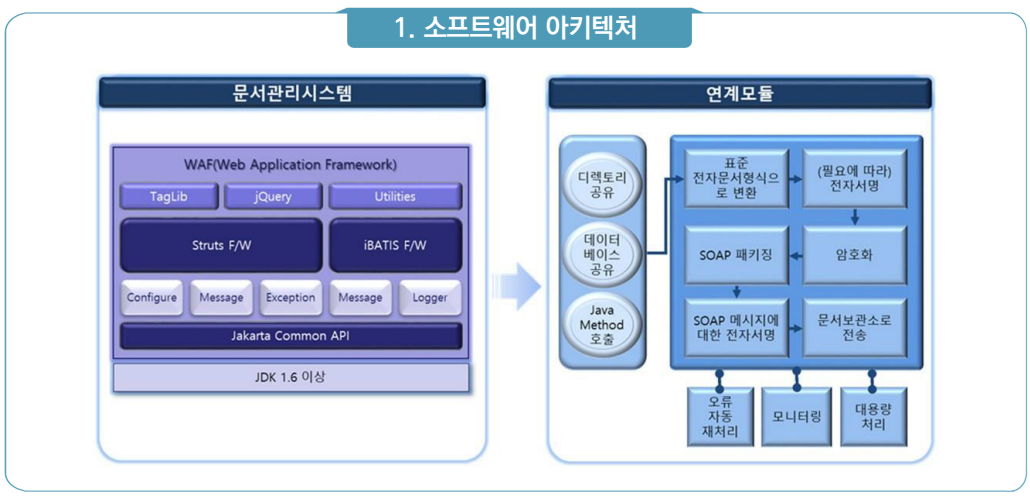
요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “HTTP 프로토콜 유효성 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-010701	HTTP 응답분할 취약점 대응		아키텍처 요구사항 및 구현방안 SR- 010701	1.7.1			
SR-010702	오픈 리다이렉트 취약점 대응		아키텍처 요구사항 및 구현방안 SR- 010702	1.7.2			

아키텍처 설계서의 소프트웨어 아키텍처에서는 HTTP 응답분할 방어를 위해 JDK 1.6 이상을 사용할 것을 명시하고 있으며, 아키텍처 요구사항 및 구현방안에서는 HTTP 응답분할과 오픈 리다이렉트 방어 기법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010701
요구사항 내용	
외부입력값을 쿠키 및 HTTP 헤더 정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다.	
구현방안	
① VM 레벨에서 방어할 수 있도록 JDK 1.6 이상의 버전을 사용한다. ② 공통 모듈(Utility)에 개행문자열을 제거하는 필터링 함수를 추가하고, 아래와 같은 경우에 필터링 함수의 반환 값을 이용하도록 한다. - 외부 입력값을 쿠키의 값으로 사용하는 경우 - 외부 입력값을 sendRedirect() 메소드의 인자값으로 사용하는 경우 - 외부 입력값을 setHeader() 메소드의 인자값으로 사용하는 경우 - 기타 외부 입력값을 HTTP 헤더의 값으로 사용하는 경우	

요구사항ID	SR-010702
요구사항 내용	
외부입력값이 페이지 이동(리다이렉트 또는 포워드)을 위한 URL로 사용되어야 하는 경우, 해당 값은 시스템에서 허용 된 URL 목록의 선택자로 사용되도록 해야 한다.	
구현방안	
① 관리자가 외부로 연결할 URL과 도메인을 관리(추가, 삭제, 수정)할 수 있도록 “외부 페이지 관리” 기능을 제공한다. ② 공통 예외(Exception) 모듈에 잘못된 페이지 이동 요청에 대한 예외를 추가한다. ③ 공통 모듈(Utility)에 response 객체의 sendRedirect() 메소드를 재정의(overriding)한 메소드를 생성한다. 해당 메소드는 리다이렉트할 주소가 외부 페이지 관리에 등록된 주소 범위 여부를 확인한 후 범위 내인 경우에는 페이지 이동을, 범위 밖인 경우에는 공통 예외 모듈에 정의한 잘못된 페이지 이동 요청 예외를 반환한다.	

개발가이드에는 응답헤더에 쓰이는 외부 입력값 필터링으로 HTTP 응답분할 취약점 방어 기법과 리다이렉트에 사용되는 외부 입력값 제한으로 오픈 리다이렉트 방어 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.7.1. 응답헤더에 쓰이는 외부 입력값 필터링

외부 입력값을 응답 헤더의 값으로 사용하는 경우, 개행문자열 포함 여부를 확인하고 제거한 후 사용되도록 한다. 해당 기능은 공통모듈로 등록하여 필터링 기능이 일괄되게 적용되도록 한다.

```
public String filterCrLf(String
 s) { if (s != null) {
 return s.replaceAll("[₩₩r₩₩n]", "");
 }
```

1.7.1. 응답헤더에 쓰이는 외부 입력값 필터링

```
 return s;
}
 :
private void doSomething(...) {
 :
 String p = request.getParameter("data");
 response.setHeader(filterCrLf(p));
 :
}
```

1.7.2. 리다이렉트에 사용되는 외부 입력값 제한

외부 입력값을 페이지 이동에 사용하는 경우, 연결할 URL과 도메인을 화이트 리스트로 정의하고 범위 내에서만 이동 할 수 있도록 제한한다.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
 throws ServletException, IOException {
 // 외부로 연결할 URL과 도메인 목록
 // 내용이 많거나 변경이 잦은 경우, 관리자 기능으로 DB화하여
 관리 String allowedURL[] = {
 "http://url1.com","http://url2.com","http://urlcom"
 };
 String nurl = request.getParameter("nurl"); if (nurl
 == null) nurl = "0";
 try {
 Integer n =
 Integer.parseInt(nurl); if (n >=
 0 && n < 3) {
 response.sendRedirect(allowedURL[n])
 }
 } catch (NumberFormatException nfe) {
 :
 }
}
```

1-8. 허용된 범위내 메모리 접근

다음은 “허용된 범위내 메모리 접근” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “허용된 범위내 메모리 접근” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구 사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요도	해결방안	검수기준	비고
SR-010801	메모리 버퍼 경계 설정 및 검사	비 기능	C나 C++ 같이 메모리를 프로 그래머가 관리하는 플랫폼을 사용 하는 경우 메모리 버퍼의 경계값 을 넘어서 메모리를 읽거나 저장 하지 않도록 경계설정 또는 검사 를 반드시 수행해야 한다.	RFP 보안요구 사항 1.8	플랫폼이 C/C++ 같이 메모리를 프로그래머 가관리해야 한다.	상	메모리 버퍼를 사용 하는 경우, 적절한크기의 버퍼를 설정하고, 설정된 범위 내에서 읽고 쓸 수 있도록 통제한다.	메모리 버퍼 오버플로우가 발생 하지 않는다.	
SR-010802	취약한 API 사용통제	비 기능	개발 시, 메모리 버퍼오 버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 통제해야 한다.	RFP 보안요구 사항 1.8	리다이렉트 기능을 제공 해야한다.	상	취약한 API를 문서화 하여 사용을 금지하고, 소스 코드 리뷰로 사용 여부를 주기적으로 확인한다.	취약한 API를 사용하지 않는다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “허용된 범위내 메모리 접근” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-010801	메모리 버퍼 경계 설정 및 검사		아키텍처 요구사항 및 구현방안 SR- 010801	1.8.1			
SR-010802	취약한 API 사용 통제		아키텍처 요구사항 및 구현방안 SR- 010802	1.8.2			

아키텍처 설계서의 소프트웨어 아키텍처에서는 버퍼 오버플로우에 위험한 함수와 안전한 함수를 제시하고 있으며, 아키텍처 요구사항 및 구현방안에서는 버퍼 오버플로우 방어 기법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

1. 소프트웨어 아키텍처

다음의 위험한 함수는 버퍼의 경계값을 체크하지 않아 버퍼 오버플로우를 발생시킬 수 있는 반면, 안전한 함수는 버퍼의 크기를 미리 지정하고, 버퍼의 크기 내에서만 처리하도록 제한하고 있다. 그러므로, 버퍼 오버플로우를 막기 위해서는 아래의 안전한 함수를 사용하여야 한다.

위험한 함수	안전한 함수
strcat	strlcat
strcpy	strlcpy
strncat	strlcat
strncpy	strlcpy
sprintf	snprintf, asprintf
vsprintf	vsnprintf, vasprintf

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010801

요구사항 내용

C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계값을 넘어서 메모리를 읽거나 저장하지 않도록 경계설정 또는 검사를 반드시 수행해야 한다.

구현방안

① 리눅스 환경에서 ASLR을 설정하여, 스택이나 힙, 라이브러리 등의 주소를 랜덤으로 프로세스 주소 공간에 배치함으로써 실행할 때 마다 데이터의 주소가 바뀌게 하여 메모리상의 공격을 어렵게 한다.
② StackGuard 컴파일러를 이용하여 경계 검사가 이루어지도록 한다.
③ 버퍼 오버플로우에 안전한 함수를 사용한다.
④ 메모리 사용시 경계값을 검사하고 사용한다.

요구사항ID	SR-010802

요구사항 내용

개발 시, 메모리 버퍼오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 통제해야 한다.

구현방안

① 메모리 버퍼오버플로우에 취약한 함수와 대체 함수를 식별한다.
② 코딩 룰에 따라 안전한 대체 함수를 사용한다.
③ 코드리뷰를 주기적으로 실시하여 취약한 함수 사용 여부를 점검한다.

개발가이드에는 버퍼 오버플로우 공격을 방어하기 위한 메모리 버퍼 경계 설정 및 검사 방법, 버퍼 오버플로우를 유발할 수 있는 위험한 함수와 안전한 함수 등을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.8.1. 메모리 버퍼 경계 설정 및 검사

메모리 버퍼를 사용할 경우 적절한 버퍼의 크기를 설정하고, 설정된 범위에서 올바르게 읽거나 쓸 수 있게 통제하여야 한다. 특히, 복사된 문자열은 널(NULL) 문자로 종료되도록 한다.

```
typedef struct _charvoid
 { char x[16];
 void
 * y;
 void
 * z;
} charvoid
static void
 goodCode()
 { charvoid
 cv_struct
 cv_struct.y = (void *) SRC_STR;
 printLine((char *) cv_struct.y);
 /* sizeof(cv_struct.x)로 변경하여 포인터 y의 덮어쓰기를
 방지함 */ memcpy(cv_struct.x, SRC_STR,
 sizeof(cv_struct.x));
 /* 문자열 종료를 위해 널 문자를 삽입함 */
 cv_struct.x[(sizeof(cv_struct.x)/sizeof(char))-1] =
 '₩0'; printLine((char *) cv_struct.x);
 printLine((char *) cv_struct.y);
}
```

1.8.1. 메모리 버퍼 경계 설정 및 검사

위험한 함수는 버퍼의 경계값을 체크하지 않아 버퍼 오버플로우를 발생시킬 수 있으므로, 버퍼 오버플로우 공격을 막기 위해서는 아래의 안전한 함수를 사용하여야 한다.

위험한 함수	안전한 함수
strcat	strlcat
strcpy	strlcpy
strncat	strlcat
strncpy	strlcpy
sprintf	snprintf, asprintf
vsprintf	vsnprintf, vasprintf
gets	fgets

코드 리뷰로 위험한 함수 사용 여부를 주기적으로 확인해야 한다.

1-9. 보안기능 입력값 검증

다음은 “보안기능 입력값 검증” 보안요구사항에 대한 안전한 설계 산출물의 예시 이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “보안기능 동작에 사용되는 입력값 검증” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010901	중요정보는 서버측의 세션이나 DB에 저장해서 사용	비기능	사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다.	RFP 보안요구사항 1.9	없음	상	처리에 필요한 중요정보를 식별하고, 서버 세션으로 관리한다.	중요 정보는 서버 세션의 값을 활용하고, 사용자 입력을 사용하지 않는다.	
SR-010902	외부 입력값은 검증 후 제한적으로 보안기능 구현에 사용	비기능	쿠키값, 환경변수, 파라미터 값 등 외부 입력값이 보안기능을 수행 하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증 작업을 수행 한 뒤 제한적으로 사용해야 한다.	RFP 보안요구사항 1.9	없음	상	사용자 입력에 의존해야 하는 값인지를 확인하고, 외부 입력에 의존해야 하는 값 인 경우에는 충분한 검증 과정과 함께 구현한다.	중요 기능에 처리에 사용되는 입력값에 대한 검증 기능이 구현되어 있다.	

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-010903	중요정보가 포함된 쿠키 암호화	비기능	중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송 되지 않아야 하며, 불가피하게 전송해야 하는 경우에는 해당 정보를 암호화해서 전송해야 한다.	RFP 보안요구사항 1.9	없음	상	중요 정보가 포함된 쿠키는 암호화하고, HttpOnly속성을 활성화하여 브라우저로 내려 보낸다.	중요 정보가 포함된 쿠키가 암호화 되고, HTTPS 통신을 할 때만 서버로 전송된다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “보안기능 동작에 사용되는 입력값 검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		...	아키텍처정의서	개발 가이드	...	...	...
요구사항ID	요구사항 명						
SR-010901	중요정보는 서버측의 세션이나 DB에 저장해서 사용		아키텍처 요구사항 및 구현방안 SR- 010901	1.9.1			
SR-010902	외부 입력값은 검증 후 제한적으로 보안기능 구현에 사용		소프트웨어 아키텍처 아키텍처 요구사항 및 구현방안 SR-010102	1.9.2			
SR-010903	중요정보가 포함된 쿠키 암호화		아키텍처 요구사항 및 구현방안 SR- 010903	1.9.3			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 서버에서 관리해야 할 중요정보를 명시하고, 중요정보가 포함된 쿠키의 운용방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-010901
요구사항 내용	
사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다.	
구현방안	
① 사용자의 역할, 권한을 결정하는 정보는 세션에 저장하고 관리한다. ② 로그인에 성공한 사용자에 대해서 해당 정보를 세션에 저장하고 세션 ID를 전달한다. ③ 세션에 저장되는 정보의 이름과 내용은 아래와 같다.	

2. 아키텍처 요구사항 및 구형방안

구현방안

세션 변수 이름		내용	비고
USER_ID	사용자 ID	로그인한 사용자의 ID	
USER_NM	사용자 이름	로그인한 사용자의 이름	
USER_GROUP	사용자 그룹	로그인한 사용자의 그룹	TBL_GRP_INFO 참조
USER_ROLE	사용자 역할	로그인한 사용자의 역할	TBL_RLE_INFO 참조

- 세션 ID는 "세션 ID 고정" 공격을 방어할 수 있도록 로그인 이후에 재할당하고, 로그아웃 이후에 할당된 세션을 완전히 제거한다.

요구사항ID	SR-010903

요구사항 내용

중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송해야 하는 경우에는 해당 정보를 암호화해서 전송해야 한다.

구현방안

① 세션에 저장되는 정보는 쿠키로 전달되지 않도록 한다.
② 쿠키는 보안(Secure) 쿠키로 HttpOnly 속성을 활성화하여, 지속시간을 최소로 설정하여 사용한다.

개발가이드에는 서버 세션의 값을 이용한 처리 방법, 사용자 입력값을 이용한 처리 방법, 쿠키 사용 규칙 등을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

1.9.1. 중요 정보 세션 저장

상태 정보나 인증, 인가, 권한에 관련한 중요 정보는 서버측 세션이나 DB에 저장하고 사용되도록 하고, 사용자 입력에 의존하지 않도록 한다.

① 사용자가 인증(로그인)에 성공하면 이후 처리에 사용할 중요 정보를 세션에 저장하고, 해당 정보에 접근할 수 있는 세션 ID를 생성하여 클라이언트 브라우저로 전달한다.
② 세션에 저장할 중요 정보는 아래와 같이 정의하여 관리할 수 있도록 한다.

세션 변수 이름		내용	비고

- 세션 변수 이름 : 세션 변수의 영문 이름과 한글 이름을 기술
- 내용 : 세션 변수에 대한 설명을 기술
- 비고 : 참고할 부가 설명을 기술

1.9.2. 입력값을 이용한 기능 구현

보안기능과 같은 중요 기능을 구현할 때, 사용자 화면에서 전달되는 쿠키, 환경변수, 히든필드의 값에 의존하지 않아야 하며, 해당 입력에 의존해야 하는 경우에는 충분한 검증 과정을 함께 구현해야 한다.

사용자의 권한, 역할 등은 서버의 세션에 저장하고 세션 정보를 활용하여 구현하도록 한다.

```
// 올바르지 않은 방법
// userRole = request.getParameter("USER_ROLE");

// 올바른 방법
userRole = (String)session.getAttribute("USER_ROLE");
```

사용자 화면에서 전달되는 모든 입력에 의존해서 요청을 처리하지 말고, 사용자 입력 필요 여부를 확인하고 가능한 서버가 보유하고 있는 정보를 활용할 수 있도록 한다.
아래의 코드에서 품목과 단가는 서버가 보유하고 있는 정보이므로, 수량만 사용자 화면에서 전달된 값을 사용하고 품목과 단가는 서버가 보유하고 있는 정보를 활용하는 것이 올바른 방법이다.

```
// 올바르지 않은 방법
// String item = request.getParameter("item"); // 품목
// String price = request.getParameter("unit_price"); // 단가
// String quantity = request.getParameter("quantity"); // 수량
// int total = Integer.parseInt(price) * Integer.parseInt(quantity);

// 올바른 방법
ItemModel item = service.getOneItem(id); // 품목 정보 추출
String item = item.getItem(); // 품목
int price = item.getUnitPrice(); // 단가
String quantity = request.getParameter("quantity"); //
수량 int total = price * quantity;
```

1.9.3. 쿠키 사용 규칙

중요 상태 정보 또는 인증, 권한 결정에 사용되는 정보는 쿠키로 전송되지 않도록 하며, 불가피하게 쿠키를 사용해야 하는 경우에는 해당 정보를 암호화해서 전송되도록 한다.
① 1.9.1에서 정의한 세션에 저장되는 정보들은 쿠키에 포함되지 않도록 한다.
② 개인정보, 금융정보, 인증정보와 같은 중요 정보는 쿠키에 포함되지 않도록 한다.

중요 정보를 쿠키로 전송할 경우에는 아래의 규칙들을 적용할 수 있도록 한다.
① 쿠키의 내용을 안전한 암호화 방식으로 암호화한다.
② Secure 속성을 활성화하여 HTTPS 통신으로써만 쿠키가 전달될 수 있도록 한다.
③ HttpOnly 속성을 활성화하여 브라우저에서 쿠키 조작이 어렵도록 한다.
④ 쿠키의 지속시간(Expired)을 최소화하여 XSS와 같은 공격으로 쿠키가 유출되는 것을 완화시키도록 한다.

1.9.3. 쿠키 사용 규칙

Secure 속성 및 HttpOnly 속성 설정

```
// 올바르지 않은 방법
// Cookie c = new Cookie("name", "value");
// c.setMaxAge(60*60*24*365);
// c.setSecure(false);
// response.addCookie(c);

// 올바른 방법
Cookie c = new Cookie("name", "value");
c.setMaxAge(-1);
c.setSecure(true);
response.addCookie(c);
```

HttpOnly 속성 설정

① Tomcat

- Tomcat 5.5.28, Tomcat 6.0.19부터 지원
- /conf/context.xml에 설정

```
<?xml version="1.0" encoding="UTF-8"?>
  <Context path="/myWebApplicationPath" useHttpOnly="true"> :
```

② JEUS

- JEUS 6.0 fix9부터 지원 (default가 true)
- WEBMain.xml 또는 jeus-web-dd.xml에 설정

```
<session-config>
 <session-cookie>
   <jsessionid-name>JSESSIONID</jsessionid-name>
   <version>0</version>
   <domain>.tmax.co.kr</domain>
   <path>/</path>
   <max-age>-1</max-age>
   <secure>false</secure>
   <http-only>true</http-only>
 </session-cookie>
</session-config>
```

1.9.3. 쿠키 사용 규칙

③ JBoss 5.0.1 및 JBOSS EAP 5.0.1
- /server/<myJBossServerInstance>/deploy/jbossweb.sar/context.xml에 설정

```
<Context cookies="true" crossContext="true">
  <SessionCookie secure="true" httpOnly="true" />
```

④ Servlet 3.0이 지원되는 경우 web.xml에 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema
instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee"

target="_blank">http://java.sun.com/xml/ns/javaee/web-
          app_3_0.xsd">
                :
  <session-config>
   <cookie-config>
    <http-only>true</http-only>
   </cookie-config>
  </session-config>
                :
```

1-10. 업로드·다운로드 파일 검증

다음은 “업로드·다운로드 파일 검증” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “업로드·다운로드 파일 검증” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-011001	업로드 파일 검증	비기능	업로드되어 저장되는 파일의 타입, 크기, 개수, 실행 권한을 제한해야 한다.	RFP 보안요구사항 1.10	없음	상	2kb 미만의 이미지 파일을 3개까지만 업로드 할 수 있도록 하고, 저장시 실행 속성을 제거한다.	2kb 미만의 이미지 파일 3개까지만 업로드되며, 저장 파일의 실행 속성이 제거되어 있다.	
SR-011002	업로드 파일 접근 제한	비기능	업로드되어 저장되는 파일은 외부에서 식별 되지 않아야 한다.	RFP 보안요구사항 1.10	없음	상	저장 파일명과 경로를 외부에서 알 수 없도록 한다.	원본 파일명과 저장 파일명이 상이 하며, 경로를 포함 한 맵핑 정보를 관리하고 있다.	
SR-011003	다운로드 파일명 검증	비기능	파일 다운로드 시, 요청 파일명에 대한 검증작업을 수행해야 한다.	RFP 보안요구사항 1.10	없음	상	요청 파일명에 경로 조작 문자열 포함 여부를 확인하고 사용 하며, 파라미터로 전달된 값 을 저장 파일을 참조하는데 직접적으로 사용하지 않는다.	경로조작 문자열을 필 터링하며, 파라미터로 전달된 파일명과 맵핑된 저장 파일명과 경로를 이용하여 파일을 다운로드 한다.	
SR-011004	다운로드 파일 무결성 검사	비기능	다운로드 받은 소스코드나 실행 파일은 무결성 검사를 수행해야 한다.	RFP 보안요구사항 1.10	없음	상	파일의 무결성을 검사를 수행한 후 파일을 다운로드 한다.	업로드한 파일과 동일한 경우에만 다운로드 한다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “업로드·다운로드 파일검증” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-011001	업로드 파일 검증		아키텍처 요구사항 및 구현방안 SR- 011001	1.10.1			
SR-011002	업로드 파일 접근 제한		아키텍처 요구사항 및 구현방안 SR- 011002	1.10.1			
SR-011003	다운로드 파일명 검증		아키텍처 요구사항 및 구현방안 SR- 011003	1.10.2			
SR-011004	다운로드 파일 무결성 검사		아키텍처 요구사항 및 구현방안 SR- 011004	1.10.2			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 파일 업로드 취약점과 다운로드 취약점의 방어 기법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-011001
요구사항 내용	
업로드 되어 저장되는 파일의 타입, 크기, 개수, 실행 권한을 제한해야 한다.	
구현방안	
① 업로드 가능한 파일의 타입으로 확장자가 gif, jpg, jpeg, png인 이미지 파일로 제한한다. ② 업로드 가능한 파일의 크기는 2kb로 제한하고, 한번에 3개의 파일만 업로드할 수 있도록 제한한다. ③ 업로드한 파일은 스토리지에 날짜별로, 실행 권한을 제거하고 저장한다.	

요구사항ID	SR-011002
요구사항 내용	
업로드 되어 저장되는 파일은 외부에서 식별되지 않아야 한다.	
구현방안	
① 업로드한 파일은 외부에서 유추할 수 없으며, 중복되지 않는 이름으로 저장한다. - 공통 모듈에 저장 파일명을 생성하는 기능을 추가하고, 파일 저장 시 호출하여 사용한다. - 해당 모듈은 UUID와 같은 클래스를 이용하여 유일한 이름을 생성하여 반환하도록 한다. ② 업로드한 파일은 스토리지에 날짜별로 디렉토리를 만들어서 저장한다. ③ 저장 경로와 파일명은 시스템 외부에서 알 수 없도록 하며, 업로드한 파일을 참조할 수 있도록 원본 파일명과 함께 데이터베이스에 저장한다. - 데이터베이스에 파일 ID, 원본 파일명, 저장 경로, 저장 파일명, 파일 해시 등의 정보를 보관한다. - 시스템 외부에서 업로드한 파일을 참조하기 위해서는 파일 ID 또는 원본 파일명을 이용하며, 저장 경로, 저장 파일명은 외부에 노출되지 않도록 한다. ④ 시스템 외부에서 업로드한 파일을 참조하기 위해서는 파일 ID, 원본 파일명을 파라미터로 받는 파일 다운로드 프로세스를 이용한다.	

요구사항ID	SR-011003
요구사항 내용	
파일 다운로드 시, 요청 파일명에 대한 검증 작업을 수행해야 한다.	
구현방안	
① 시스템에 저장된 모든 파일(관리자 등록 파일, 사용자 등록 파일, ... 등)은 파일 다운로드 프로세스를 호출하여 다운로드한다. ② 파일을 다운로드하기 위해서는 파일 ID 또는 원본 파일명을 파라미터로 받는 파일 다운로드 프로세스를 호출해야 한다. ③ 파일 다운로드 프로세스는 파라미터로 넘어 온 원본 파일명에 경로를 조작할 수 있는 문자열 포함 여부를 확인해야 한다. 만약, 경로조작 문자열이 포함된 경우 지정된 오류 메시지("일치하는 파일이 존재하지 않습니다.")를 출력한다.	

2. 아키텍처 요구사항 및 구현방안

요구사항ID	SR-011004
요구사항 내용	
다운로드 받은 소스코드나 실행파일은 무결성 검사를 수행해야 한다.	
구현방안	
① 파일 다운로드 프로세스는 다운로드 할 파일에 대해 무결성 검사를 수행해야 한다. - 관리자가 업로드한 파일의 경우, 업로드 시점에 생성한 파일의 해시값과 다운로드할 파일의 해시값을 비교하여 일치하는 경우에만 다운로드되도록 한다. - 사용자가 업로드한 파일의 경우, 다운로드할 파일의 종류가 이미지이고, 업로드 시점에 생성한 파일의 해시값과 다운로드할 파일의 해시값을 비교하여 일치하는 경우에만 다운로드 되도록 한다. ② 무결성 검사에 실패한 경우 지정된 오류 메시지("[파일 다운로드 오류] 무결성 검사에 실패하였습니다.")를 출력하고, 관리자가 원인을 파악할 수 있도록 로그와 알림 메시지를 관리자에게 전달한다.	

개발가이드에는 파일 업로드 및 다운로드 기능을 구현할 때 유의해야 할 사항을 제시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

1.10.1. 파일 업로드

파일 업로드 기능을 구현할 경우에는 서버의 디스크 및 커넥션 자원 고갈을 막기 위해 업로드 파일의 크기와 개수를 제한해야 한다.

파일의 크기는 업로드 프로세스에서 제한할 수도 있으며, Spring 프레임워크를 사용하는 경우 MultipartResolver 컴포넌트의 속성 설정으로도 제한할 수 있다.

```
MultipartFile file = request.getFile("file");
if (file != null && file.getSize() >
    1024000) { return "파일 크기 제한을
    초과했습니다.";
}
```

```
<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMul
tipartResolver">
 <!- max upload size in bytes ->
 <property name="maxUploadSize" value="20971520" /> <!- 20MB
->
```

1.10.1. 파일 업로드

```
〈!– max size of file in memory (in bytes) –〉
〈property name="maxInMemorySize" value="1048576" /〉 〈!– 1MB
–〉
〈/bean〉
```

파일 업로드 기능을 구현할 경우에는 웹쉘과 같은 악성 프로그램의 실행을 막기 위해 업로드 파일의 종류를 제한하고, 저장 경로와 저장 파일명을 외부에서 알 수 없도록 해야 하며, 실행 속성을 제거하고 저장해야 한다.

파일의 종류는 파일의 확장자, 파일의 Content-Type, 파일의 시그니쳐(signature) 등을 이용하여 검증할 수 있다. 파일의 종류를 검증할 때는 허용 목록(white list)을 정의하고, 허용 범위 내에서 사용될 수 있도록 해야 하며, 파일의 확장자와 Content-Type은 위변조가 쉬우므로 파일의 시그니쳐를 이용해 파일의 종류를 검증하는 것이 안전하다.

```
public boolean isImageFile(File filename) throws Exception
 { Magic magic = new Magic();
 FileInputStream is = new
 FileInputStream(filename); byte[] data = new
 byte[512];
 is.read(data);
 MagicMatch match = magic.getMagicMatch(data); if
 (match.getMimeType().contains("images")) {
    return true;
 } else {
    return false;
 }
}
```

업로드 파일은 외부에서 알 수 없는 경로에 유일한 파일명으로 저장해야 한다. 일반적으로 분리된 스토리지에 저장하며, 저장 파일명은 랜덤한 숫자 또는 시간과 같이 유일한 값을 사용한다.

```
String savedFileName =
UUID.randomUUID().toString(); File uploadFile = new
File(uploadPath + savedFileName); try {
 file.transferTo(uploadFile);
} catch (Exception e)
 { System.out.println("upload
 error");
}
```

1.10.1. 파일 업로드

저장에 사용한 경로와 파일명은 다운로드 기능에서 참조할 수 있도록 원본 파일명과 함께 DB화하여 관리한다.

업로드 파일이 저장되는 디렉토리에 디렉토리 또는 파일을 생성할 때 부여되는 퍼미션을 확인하고, 실행 권한이 부여되지 않도록 한다.

1.10.2. 파일 다운로드

업로드한 파일은 반드시 다운로드 기능을 이용해서 내려 받도록 한다.

다운로드 기능은 파일 ID 또는 원본 파일명을 파라미터로 가지며, 파라미터로 전달된 값과 일치하는 저장 경로와 파일 명의 파일을 읽어서 내려 보낸다.

다운로드 기능은 전달된 파라미터에 경로조작 문자열 포함 여부를 확인해야 하며, 경로조작 문자열이 포함된 경우 다운로드 기능을 중지하고, 오류 메시지를 출력한다.

다운로드 기능은 다운로드 되는 파일의 위변조 여부를 확인하기 위해, 시스템에서 허용한 유형의 파일인지 여부를 확인하고, 원본 파일을 저장할 때 생성한 해시값과 다운로드할 파일의 해시값을 비교하여 일치하는 경우에만 다운로드 되도록 한다.

2. 보안기능

2-1. 인증 대상 및 방식

다음은 “인증 대상 및 방식” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “인증 대상 및 방식” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020101	인증 후 중요 기능/리소스 허용	비기능	중요기능이나 리소스 에 대해서는 인증 후 사용 정책이 적용되어야 한다.	RFP 보안요구사항 2.1	없음	상	인증 여부를 체크하는 서블릿 필터 컴포넌트를 추가하여 중요기능을 처리하기 전에 인증 여부를 확인한다.	로그인하지 않고 중요 기능을 요청 할 경우, 인증 오류 메시지가 출력된다.	
SR-020102	안전한 인증방식 사용	비기능	안전한 인증방식을 사용하여 인증우회나 권한상승이 발생하지 않도록 해야 한다.	RFP 보안요구사항 2.1	없음	상	외부 입력값을 Query Map에 바인딩할 경우 에는 #을 이용하도록 개발가이드에 명시하고, 개발보안 교육으로 개발자에게 전파하여 지켜질 수 있도록 한다.	유효한 아이디와 비밀번호를 입력한 경우, 인증정보 확인하여 세션을 생성한다.	
SR-020103	중요기능에 대해 2단계 인증 적용	비기능	중요기능에 대해 2단계(2 factor)인증을 고려해야 한다.	RFP 보안요구사항 2.1	없음	상	아키텍처 정의에 따라 모든 쿼리는 MyBatis 프레임워크의 쿼리 맵으로 정의, 실행 한다. 이때, 변수 바인 딩은 SR-010102 에 따라 # 기호를 이용한다.	중요기능요청시 인증서 선택창이 뜨고, 인증서 비밀번호가 일치하는 경우에만 요청이 서버로 전달, 처리된다.	

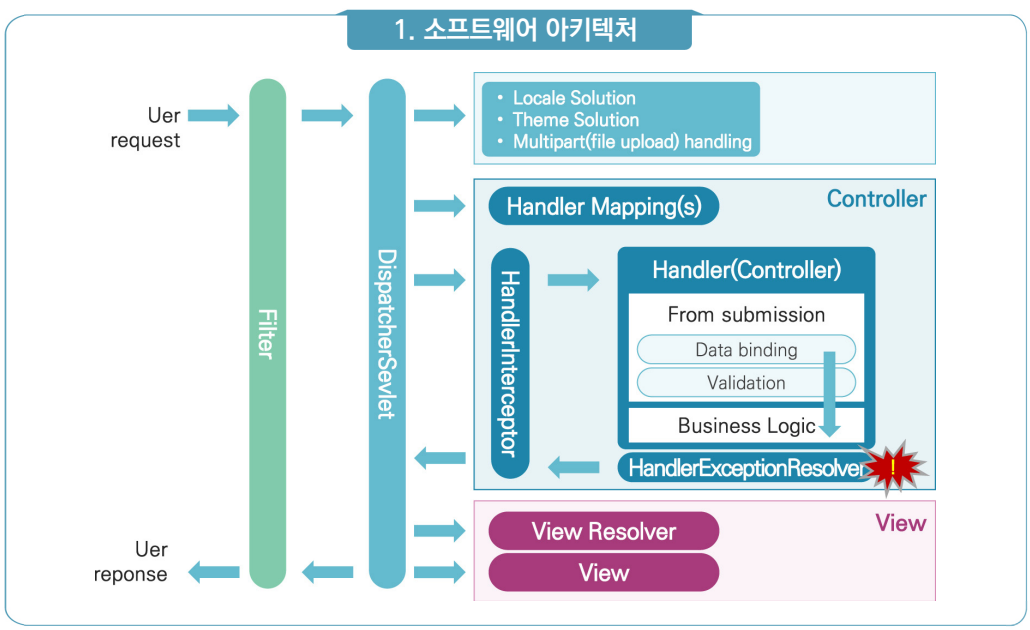
요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “인증 대상 및 방식” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이즈
요구사항ID	요구사항 명						
SR-020101	인증 후 중요 기능/리소스 허용		아키텍처 요구사항 및 구현방안 SR-020101	2.1.1			UTC_020101 UTC_020102
SR-020102	안전한 인증방식 사용		아키텍처 요구사항 및 구현방안 SR-020102	2.1.1			UTC_020103
SR-020103	중요기능에 대해 2단계 인증 적용		아키텍처 요구사항 및 구현방안 SR-020103	2.1.2			UTC_020104

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 중요 기능/리소스의 노출을 방어하기 위해 인증 후 중요 기능/리소스를 허용하는 것과 안전한 인증 방법, 중요기능에 대해 2단계 인증 적용 등을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서



2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020101

요구사항 내용

중요기능이나 리소스에 대해서는 인증 후 사용 정책이 적용되어야 한다.

구현방안

① 기능목록 또는 기능명세서를 이용하여 인증이 필요한 중요기능과 리소스를 식별한다.
② Filter 또는 Interceptor 컴포넌트를 이용하여 인증 여부를 확인 후 중요 기능 및 리소스에 접근할 수 있도록 한다.
③ 오류처리 공통모듈에 인증오류(unauthentication)를 추가하고, 인증되지 않은 사용자가 중요 기능 및 리소스에 접근할 경우 인증오류가 발생하도록 한다.
④ 인증오류가 발생하면 증적유지 및 감사추적을 할 수 있도록 접근정보를 로그파일로 남긴다.

요구사항ID	SR-020102

요구사항 내용

안전한 인증방식을 사용하여 인증우회나 권한상승이 발생하지 않도록 해야 한다.

구현방안

① 세션을 생성, 관리하는 폼 기반 인증(form-based authentication) 방식을 이용하여 인증 요청을 처리한다.
② 아이디와 비밀번호는 다음의 규칙을 만족해야 하며, 서버측에서 입력값 검증을 수행한다.
  - 아이디는 영문자, 숫자 사용이 가능하며, 길이는 8~10자리로 제한한다.
  - 비밀번호는 영문자, 숫자, 특수문자 사용이 가능하며, 길이는 8~12자리로 제한한다.
③ 사용자 정보 테이블에 일치하는 아이디와 비밀번호가 존재하는지 확인하는 방식으로 인증을 처리한다. 이때, SQL 삽입 취약점이 발생하지 않도록 시큐어코딩 가이드에 따라 SQL문을 만들고 실행한다.
④ 인증에 성공하면 세션ID를 재생성하고, 세션에 처리에 필요한 정보를 담은 후 일반 사용자와 관리자를 구분하여 제공될 페이지로 리다이렉트 한다.

요구사항ID	SR-020103

요구사항 내용

중요기능에 대해 2단계(2 factor)인증을 고려해야 한다.

구현방안

① 2단계 인증을 적용할 중요기능을 도출한다.

기능	내용	적용할 추가 인증기법
비밀번호 변경	로그인한 사용자의 비밀번호를 변경	사용자의 현재 사용 중인 비밀번호
사용자 권한 변경	관리자 기능으로 일반 사용자의 권한을 변경	관리자 비밀번호
서버 재기동	운영 중인 서버를 재기동	서버 인증서

② ①번 과정에서 도출한 기능에 대해 2단계 인증이 적용될 수 있도록 상세 설계에 반영한다.
④ 2단계 인증을 적용한 각 기능에 대해서는 추가 인증 없이 처리가 완료되는지 여부를 확인한다.

개발가이드에는 중요 기능/리소스의 노출을 막기 위한 인증 여부 확인 방법, 폼 기반의 인증 방법, 2단계 인증 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.1.1 인증 여부 확인

인증 여부를 확인하는 공통 모듈로 개발한다.

```
public boolean isAuthenticated(HttpSession s)
{
 boolean authenticated = false;

 try {
     authenticated
     = getBooleanSessionAttribute(s, getUserId() +
".isAuthenticated");
 } catch (ParameterNotFoundException e) {
  // error
 }
 return authenticated;
}

protected boolean getBooleanSessionAttribute(HttpSession s,
String
 name) throws ParameterNotFoundException
{
 boolean value = false;

 Object attribute = s.getRequest().getSession().getAttribute(name);
if (attribute ==
 null) {
    throw new ParameterNotFoundException();
 } else {
    value = ((Boolean) attribute).booleanValue();
 }
 return value;
}
```

인증 여부 확인 모듈을 Filter 또는 Interceptor와 같은 컴포넌트에 등록하여 유입되는 모든 요청에 대해 자동으로 인증 여부를 확인할 수 있도록 한다.

2.1.1 인증 여부 확인

```
〈mvc:interceptors〉
        〈bean class="kr.co.openeg.lab.common.interceptor.SessionInterceptor" /〉
〈/mvc:interceptors〉
public class SessionInterceptor extends HandlerInterceptorAdapter
  { @Override
  public boolean preHandle(HttpServletRequest
                  request, HttpServletResponse
                  response,
                  Object handler) throws
    Exception { if
    (isAuthenticated(request.getSession())) {
      return true;
    } else {
       return false;
    }
  }

  @Override
  public void postHandle(HttpServletRequest request,
                HttpServletResponse response, Object handler,
                ModelAndView modelAndView) throws Exception {
  }
}
```

인증 여부 확인 모듈을 Filter 또는 Interceptor와 같은 컴포넌트에 등록하여 유입되는 모든 요청에 대해 자동으로 인증 여부를 확인할 수 있도록 한다.

2.1.1 인증 여부 확인

사용자가 입력한 아이디와 비밀번호를 POST 방식으로 서버로 전달한다. 요청 파라미터로 전달된 아이디와 비밀번호와 일치하는 사용자 정보가 존재하는 경우 "사용자ID.isAuthenticated" 형식의 세션ID를 생성하고, 처리에 필요한 정보를 세션에 저장한다.

인증 처리과정에서 유의할 사항은 다음과 같다.
- 아이디와 비밀번호는 POST 방식으로 전달한다.
- 아이디와 비밀번호는 서버에서 입력값 검증 후 사용한다.
- 비밀번호는 서버에서 일방향 해시함수로 변환한다.
- 사용자 정보를 조회할 때 SQL 삽입과 같은 취약점이 발생하지 않도록 시큐어코딩 기법에 따른다.
- 세션에 저장할 내용과 식별자는 분석/설계 단계에 정의하여 관리한다.

```
String id = request.getParameter("id");
```

2.1.1 인증 여부 확인

```
String pw = request.getParameter("pw");

if (!CommonUtils.isValid("ID", id) || !CommonUtils.isValid("PW", pw)) {
    // 입력값 검증에
    실패 return false;
}
pw = CommonUtils.getHash(pw);

if (!CommonUtils.isValidUser(id, pw)) {
// 인증
실패
return
false;
}

// 세션 생성
String sessionId = id + ".isAuthenticated";
session.setAttribute(sessionId, "true");
```

2.1.3 2단계 인증

단계 인증이 필요한 중요 기능을 도출하여 각 기능에 적용할 추가 인증 기법을 정의한다.

2단계 인증 기법은 기능의 중요성과 적용 가능성 등을 고려하여 아래의 방법들 중 하나를 선택한다.

구분	인증기법	예
TYPE1	지식기반인증	비밀번호, PIN
TYPE2	소유기반인증	토큰, 스마트카드
TYPE3	생체기반인증	지문, 홍채

단위 테스트 케이스에는 관리자 기능, 회원정보 수정과 같은 중요 기능을 인증 후 접근할 수 있도록 제한하고 있는지 여부와 인증 우회가 가능한지 여부를 테스트하는 방법과 절차가 제시되어 있다.

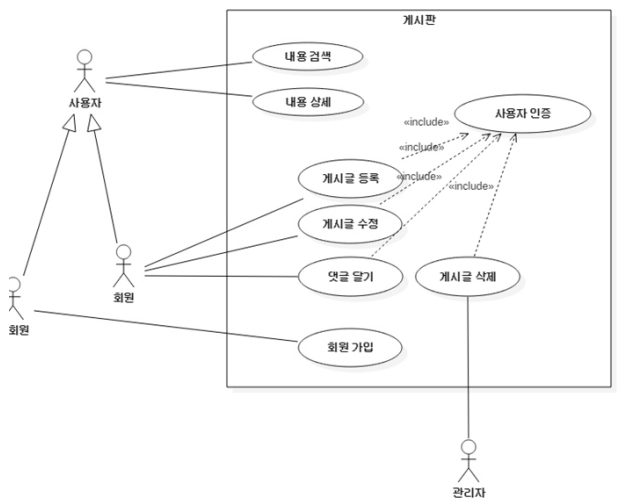
안전한 보안설계의 예 : 단위 테스트 케이스

요구사항 ID	UTC_0201					
설명	중요기능 및 리소스에 대해서는 인증한 사용자만 접근할 수 있도록 제한하고, 인증하지 않은 사용자의 접근에 대해서는 미리 정의한 오류 메시지가 표출되는지 확인한다.					
관련 컴포넌트 ID	CMT_0201			관련 프로그램 ID	PRG_0201	
케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC_020101	관리자 기능 접근	관리자	관리자 기능 접근 URL	① 브라우저 주소창에 관리자 페이지 주소를 입력하고 엔터 ② 관리자 기능 표출 여부 및 오류 메시지 출력여부를 확인 ③ 접근 로그 생성 여부 확인	① 관리자로 로그인한 경우, 해당 페이지 표출 ② 관리자로 로그인하지 않았거나, 로그인 하지 않은 경우, “잘못된 접근 입니다.” 메시지 출력 후 index 페이지로 이동 ③ 요청시간, 요청IP, 접속URL, 처리 결과, 메시지 등의 정보를 포함한 접근 로그를 생성	
UTC_020102	회원정보 수정 페이지 접근	전체	관리자 기능 접근 URL	① 브라우저 주소창에 회원정보 수정 페이지 주소를 입력하고 엔터 ② 수정 페이지 표출 여부 및 오류 메시지 출력 여부를 확인 ③ 접근 로그 생성 여부 확인	① 로그인한 경우, 회원정보 수정 페이지 표출 ② 로그인하지 않은 경우, “로그인 후 사용 할 수 있습니다.” 메시지 출력 후 로그인 페이지로 이동 ③ 요청시간, 요청IP, 접속URL, 처리 결과, 메시지 등의 정보를 포함한 접근 로그를 생성	
UTC_020103	인증우회	전체	생성 규칙을 위배하는 아이디/비밀번호 SQL 삽입을 유발하는 아이디/비밀번호	① 로그인 페이지에서 시험 데이터(비정상적인 아이디/패스 워드)를 입력 후 엔터 ② 로그인 성공 여부 및 오류 메시지 출력 여부 확인 ③ 접근 로그 생성 여부 확인	① “일치하는 계정이 존재하지 않습니다.” 메시지 출력 후 로그인 페이지로 이동 ② 생성 규칙에 위배되는 아이디/비밀번호를 입력하거나 SQL 삽입을 유발하는 아이디/비밀번호를 입력 하는 경우, 요청시간, 요청IP, 접속 URL, 처리결과, 메시지 등의 정보를 포함한 접근 로그를 생성	
UTC_020104	비밀번호 변경	전체	관리자 기능 접근 URL	① 로그인후 비밀번호 변경 페이지 호출 ② 현재 비밀번호를 입력 하지 않은 상태에서 비밀번호 변경 버튼 클릭 ③ 또는 현재 비밀번호를 파라미터에서 제외 하고 비밀번호변경처리URL 을 직접 호출	① 현재 비밀번호를 입력하지 않은 상태에서 비밀번호 변경 버튼을 클릭 한 경우, “현재 비밀번호를 입력하세요”라는 JavaScript 검증 결과 메시지 출력 ② 현재 비밀번호를 파라미터에서 제외하고 비밀번호 변경 처리 URL을 직접 호출한 경우, “현재 비밀번호를 입력 하세요” 메시지 출력 후 비밀번호 변경 페이지로 이동 ③ ②번과 같은 접근이 있을 경우, 요청 시간, 요청IP, 접속URL, 처리결과, 메시지 등의 정보를 포함한 접근 로그를 생성	

유즈케이스 명세서에는 로그인한 사용자(회원 또는 관리자)와 로그인하지 않은 사용자(비회원)가 접근할 수 있는 기능을 구분하고 작업흐름을 명시하고 있다.

안전한 보안설계의 예 : 유즈케이스 명세서

2. 유즈케이스 다이어그램(UCD)			
UCD ID	UCD-020101	UCD 명	게시판
관련 서브 시스템 ID	UCD-0201	관련 서브 시스템 명	고객지원

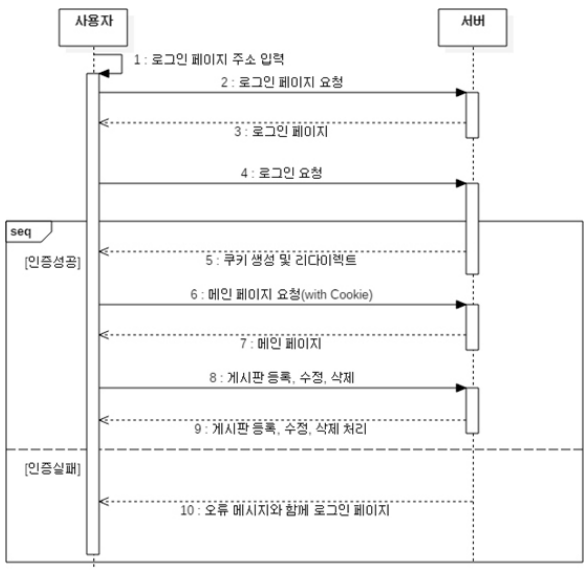


5. 유즈케이스 기술서			
요구사항 ID	게시판 회원 기능	액터명	사용자(회원)
유즈케이스 개요 및 설명	회원이 게시판 내용을 등록, 수정, 댓글달기, 내용검색, 내용상세조회를 요청		
사전조건	사용자의 회원, 비회원 여부를 확인할 수 있도록 사용자 인증을 수행		
사후조건			
작업흐름			
정상흐름	1. 회원이 로그인 2. 회원이 게시글을 등록 3. 회원이 게시글을 조회 및 상세 조회 4. 회원이 게시글을 수정 또는 댓글 달기		
대안흐름	1. 관리자는 게시글 삭제		
예외흐름	비회원은 회원가입		

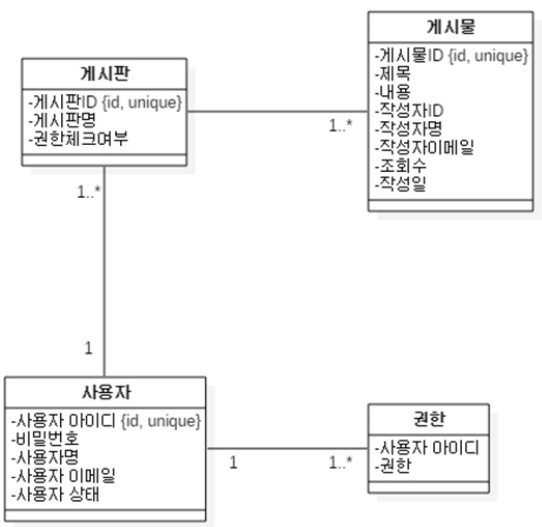
클래스 설계서의 시퀀스도에는 인증에 성공한 사용자만 게시판 기능을 사용할 수 있도록 명시하고 있으며, 설계 클래스도에는 게시판 사용자별 권한을 관리할 수 있도록 명시하고 있다.

안전한 보안설계의 예 : 클래스 설계서

2. 시퀀스도

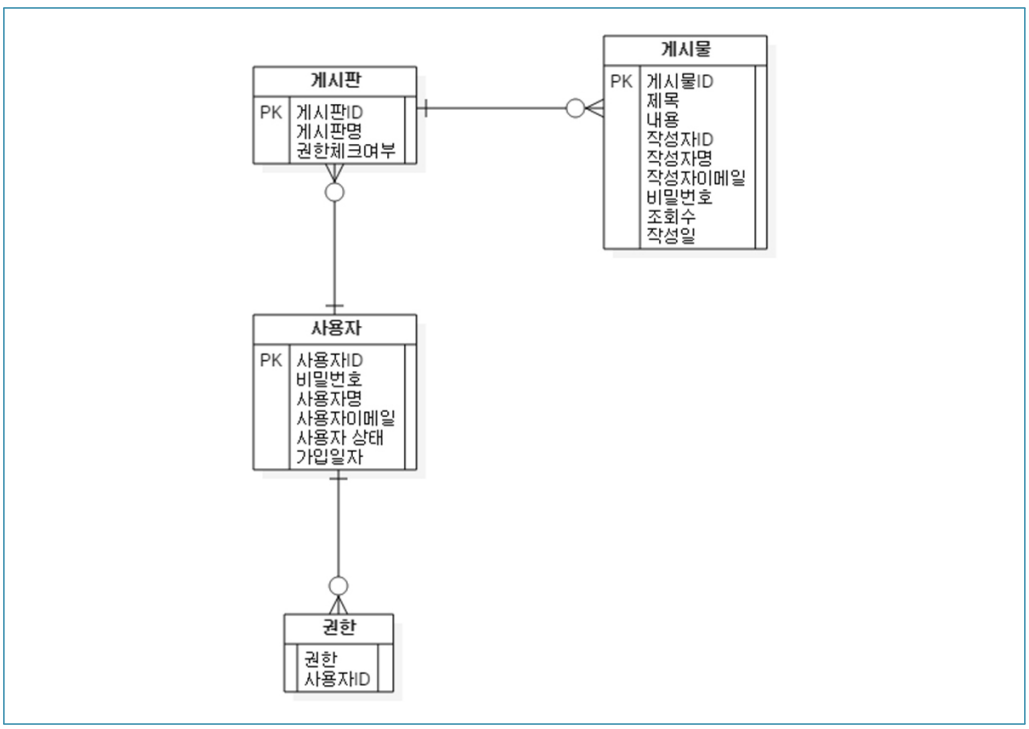


3. 설계 클래스도



ERD와 테이블 정의서에는 게시판별 사용자 권한을 관리할 수 있도록 권한 테이블을 정의하고 있다.

안전한 보안설계의 예 : ERD



안전한 보안설계의 예 : 테이블 정의서

USER_INFO									
컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
id	사용자 아이디	사용자 정보	Y	VARCHAR2	10	Y	N		사용자 아이디
password	사용자 비밀번호	사용자 정보	Y	VARCHAR2	100	N	N		사용자 비밀번호
Name	이름	사용자 정보	Y	VARCHAR2	100	N	N		이름
Email	이메일	사용자 정보	Y	VARCHAR2	20	N	N		이메일
Status	가입상태	사용자 정보	Y	CHAR		N	N		사용자 아이디 사용자 비밀번호 이름 이메일
reg_de	가입일자	사용자 정보	Y	DATE		N	N		

USER_INFO									
컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
permission	권한	권한정보	Y	VARCHAR2	10	Y	N		권한
user_id	사용자 아이디	권한정보	Y	VARCHAR2	10	N	N		사용자 아이디

2-2. 인증 수행 제한

다음은 “인증 수행 제한” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “인증 수행 제한” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구 사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요도	해결방안	검수기준	비고
SR-020201	인증 시도 횟수 제한	비 기능	로그인 기능 구현 시, 인증 시도 횟수를 제한하고 초과된 인증 시도에 대해 인증 제한 정책을 적용해야 한다.	RFP 보안요구사항 2.2	없음	상	인증시도 횟수를 5회로 제한하고, 5회를 초과한 경우 24시간 동안 계정 잠금을 수행한다.	인증 실패가 5번 이상 발생하면 24시간 동안 계정 잠금이 수행 된다.	
SR-020202	실패한 인증 시도 로깅	비 기능	실패한 인증 시도에 대한 정보를 로깅하여 인증 시도 실패가 추적 될 수 있게 해야 한다.	RFP 보안요구사항 2.2	없음	상	로그인 성공/실패 시 로그 파일에 사용자 ID, 로그인 실패 횟수, 로그인 시간, IP 주소, 계정상태 등을 남긴다.	인증 시도 정보가 로그 파일에 남는다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “인증 수행 제한” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화 되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이스	통합시험 케이스	…
요구사항ID	요구사항 명								
SR-020201	인증시도 횟수 제한		아키텍처 요구사항 및 구현방안 SR-020201	2.1.1			TCU 020201	TTC 020201	
SR-020202	실패한 인증시도 로깅		아키텍처 요구사항 및 구현방안 SR-020202	2.1.1			TCU 020201	TTC 020202	

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 인증시도 횟수 제한과 인증 실패 시 로깅 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020201
요구사항 내용	
로그인 기능 구현 시, 인증시도 횟수를 제한하고 초과된 인증시도에 대해 인증제한 정책을 적용해야 한다.	
구현방안	
① 인증 시도 정보를 DB에 기록한다. - 사용자 ID, 로그인 실패 횟수, 로그인 시간, IP 주소, 계정 상태 등의 정보를 저장한다. ② 로그인 실패 횟수가 5회 이상인 경우, "인증시도 횟수초과로 24시간 동안 계정을 사용할 수 없다"는 내용의 메시지를 출력한다. ③ 계정이 잠긴 상태에서도 로그인 시도 정보는 계속해서 DB에 저장하며, 로그인 시도가 지속될 경우, 관리자에게 알림을 전달한다.	

요구사항ID	SR-020202
요구사항 내용	
실패한 인증시도에 대한 정보를 로깅하여 인증시도 실패가 추적될 수 있게 해야 한다.	
구현방안	
① 인증 시도 정보를 DB에 기록한다. - 사용자 ID, 로그인 실패 횟수, 로그인 시간, IP 주소, 계정 상태 등의 정보를 저장한다. ② 관리자가 인증 시도 정보를 열람할 수 있도록 조회 기능을 제공한다. - 관리자 인증 후 사용 가능하며, 로그인 실패 횟수 현황, 계정 잠금 현황 등의 정보를 제공한다.	

개발가이드에는 인증시도 횟수 제한 방법으로 무차별대입 공격 방어 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.2.1. 인증 시도 횟수 제한

사용자 인증 기능을 구현할 때, 인증 시도 정보(사용자 ID, 접속 IP, 접속 시간, 실패 횟수, 계정 상태 등)를 저장해야 하며, 관리자가 인증 시도 횟수, 인증 실패 횟수, 계정 잠금 등의 정보를 조회하고 분석할 수 있도록 관리자 기능을 제공해야 한다.

무차별 대입 공격과 같은 방식으로 인증 정보가 노출되는 것을 막기 위해서는 인증 실패 시 실패 횟수를 카운트하고 일정 횟수를 초과하면 지정된 시간 동안 계정을 사용할 수 없도록 계정 잠금을 수행해야 한다.

단위 테스트 케이스와 통합 테스트 케이스에는 인증시도 횟수를 제한하고 있는지 여부를 테스트 하는 방법과 절차가 제시되어 있다.

안전한 보안설계의 예 : 단위 테스트 케이스

요구사항 ID	UTC_0202					
설명	사용자가 로그인한다. - 아이디와 비밀번호를 입력하고 로그인한다. - 로그인 성공 시 메인 페이지로 이동하고, 로그인 실패 시 아이디와 비밀번호를 다시 입력하도록 한다. - 로그인 실패 제한횟수(5회)를 초과하면 계정을 잠근다.					
관련 컴포넌트 ID	CMT_0202			관련 프로그램 ID	PRG_0202	
케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC_020201	로그인	전체	로그인 성공 아이디 : testuer 비밀번호 : Tes-t1User%@	① 아이디, 비밀번호 입력 ② 로그인 버튼 클릭	① 정상적으로 로그인되고 메인페이지로 이동한다.	
UTC_020202	로그인	전체	로그인 실패 아이디 : testuer 비밀번호 : Test1User	① 아이디, 비밀번호 입력 ② 로그인 버튼 클릭	① 로그인에 실패하고, 아이디와 비밀번호를 다시 입력하도록 팝업이 생성된다. ② 로그인 실패 횟수가 6회가 되면 계정이 잠기고 추가 인증을 받아 로그인을 하도록 팝업이 생성된다.	

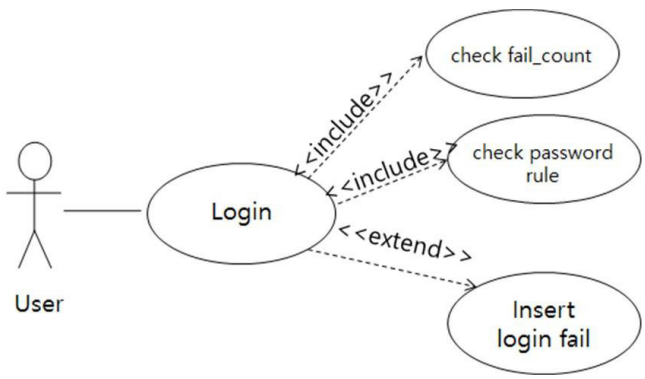
안전한 보안설계의 예 : 통합 테스트 케이스

시나리오 ID	시나리오 명	상세설명(흐름도)	검증포인트
SR-020201	로그인	로그인 후 마이페이지 접근 및 이메일 주소 변경 ① 사용자 로그인 ② 마이페이지 메뉴 선택 ③ 개인정보 수정하기 선택 ④ 이메일 주소 수정	- 사용자 로그인 - 마이페이지 메뉴 확인 - 개인정보 수정하기 확인 - 아이디 수정하기 에러 발생 검증
TTC-020202	로그인 실패	로그인 시 등록된 아이디와 잘못된 비밀번호 입력 ① 잘못된 비밀번호 입력 ② 5회 반복 ③ 6회 오류 발생 ④ 계정 잠금	- 로그인 실패 - 아이디와 비밀번호 재입력 팝업 생성 - 6회 오류시 계정잠금

유즈케이스 명세서에는 로그인 시 인증시도 횟수를 카운트하고 시도 횟수를 초과하는 경우 계정 잠금 처리 기능을 작업흐름에 명시하고 있다.

안전한 보안설계의 예 : 유즈케이스 명세서

2. 유즈케이스 다이어그램(UCD)			
UCD ID	UCD-020201	UCD 명	로그인
관련 서브 시스템 ID	UCD-0202	관련 서브 시스템 명	회원관리

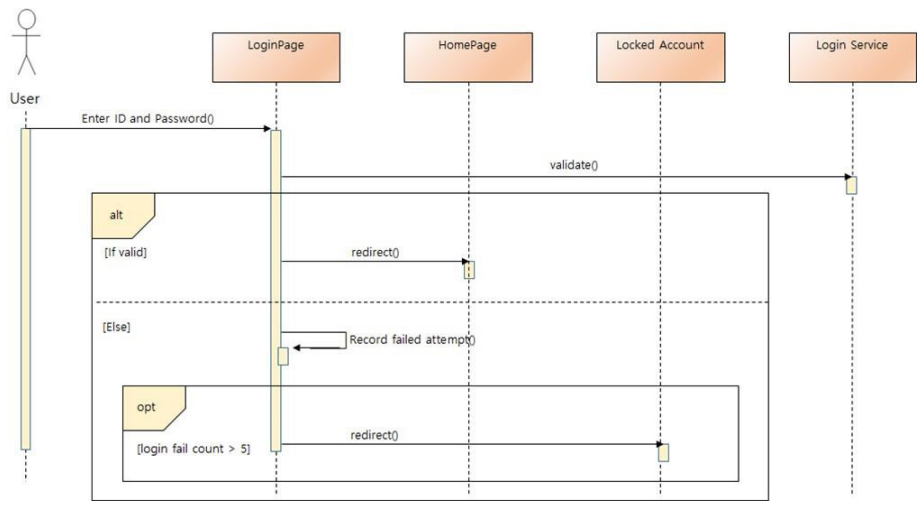


5. 유즈케이스 기술서			
요구사항 ID	로그인	액터명	사용자
유즈케이스 개요 및 설명	사용자가 아이디와 비밀번호를 이용하여 로그인 함		
사전조건			
사후조건			
작업흐름			
정상흐름	1. 사용자는 로그인 화면에 사용자 아이디와 비밀번호를 입력한다. 2. 아이디와 비밀번호가 입력되었는지 확인한다. 3. 입력된 아이디와 비밀번호가 유효한지 검사한다. 4. 아이디와 비밀번호가 유효하면 메인화면을 보여준다.		
대안흐름			
예외흐름	1. 정상흐름 1에서 유효하지 않은 아이디와 비밀번호를 입력한 경우, 로그인 실패 횟수를 1 증가시키고 사용자에게 아이디와 비밀번호를 다시 입력하도록 한다. 2. 실패 횟수가 5회 초과한 경우, 해당 계정을 24시간 동안 사용할 수 없도록 한다.		

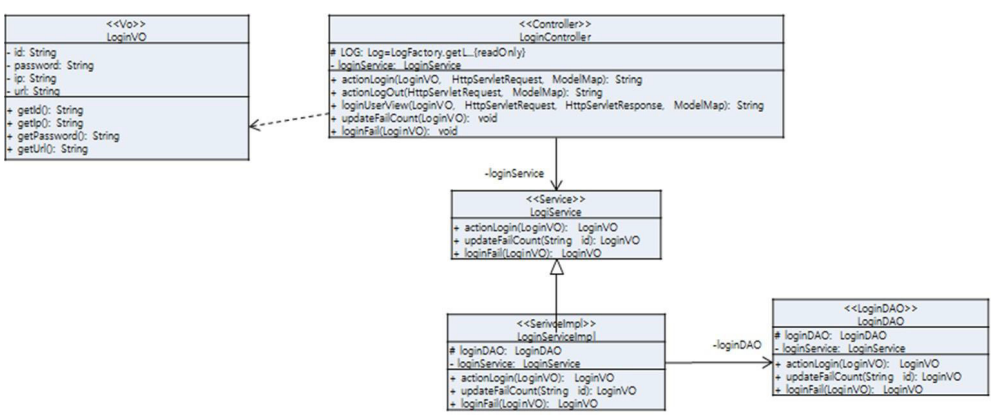
클래스 설계서의 시퀀스도에는 인증에 성공한 사용자만 게시판 기능을 사용할 수 있도록 명시하고 있으며, 설계 클래스도에는 게시판 사용자별 권한을 관리할 수 있도록 명시하고 있다.

안전한 보안설계의 예 : 클래스 설계서

2. 시퀀스도



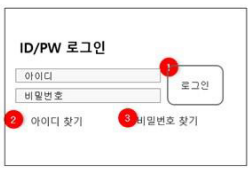
3. 설계 클래스도



화면 설계서는 로그인을 처리할 때, 로그인에 실패한 경우 인증시도 횟수를 증가시키고 로그인 실패 정보를 기록하도록 명시하고 있다.

안전한 보안설계의 예 : 화면 설계서

홈페이지 화면 정의서					
시스템명	대표홈페이지	서브시스템	홈페이지 접속 로그인	작성일	
페이지ID	login.jsp	페이지명	로그인	작성자	
개요	홈페이지 초기 로그인				



화면 정의

1. 사용자 : 회원
2. 사용주기 : 수시
3. 프로세스 :
   아이디 패스워드 입력하고 로그인 버튼 클릭하여 사용자 인증되면 로그인 실행하고 사용자 홈화면 이동

항목 및 기능설명

1.로그인 :
사용자 아이디와 패스워드를 입력하고 사용자가 로그인 한다.
2.아이디찾기 :
아이디를 분실한 경우 가입한 아이디 확인을 요청한다.
3.비밀번호 찾기 :
패스워드를 분실한 경우 아이디 정보를 기반으로 패스워드를 확인한다.

1. 입력한 아이디, 패스워드로 인증이 실패한 경우 MEMBER 테이블에 로그인 실패횟수를 1 증가시킨다.
2. 로그인 실패시 LOGINFAIL 테이블에 ID, IP, 로그인실패회수, 시간, 잠금여부를 기록한다.

ERD와 테이블 정의서에는 인증실패 횟수, 계정잠금 여부 등 인증시도 횟수 제한에 필요한 정보를 저장할 테이블과 컬럼을 정의하고 있다.

안전한 보안설계의 예 : ERD

MEMBER		
ID	VARCHAR2(100)	PK
NAME	VARCHAR2(100)	
PASSWD	VARCHAR2(100)	
EMAIL	VARCHAR2(100)	
ADDRESS	VARCHAR2(100)	
HP	VARCHAR2(100)	
FAIL_COUNT	NUMBER(1)	
IS_LOCK	CHAR(1)	
LAST_LOGIN	DATE	

LOGINFAIL	
SEQ	CHAR(18)
ID	VARCHAR2(100)
IP	VARCHAR2(100)
FAIL_COUNT	NUMBER(1)
IS_LOCK	CHAR(1)
TIME	DATE

안전한 보안설계의 예 : 테이블 정의서

테이블 명	테이블 유형	관련 엔티티명	테이블 설명	보존 기간	발생 주기	비고
MEMBE R	일반테이블	사용자 정보	사용자 정보	5년	실시간	
LOGINF AIL	일반테이블	로그인 실패 정보	로그인 실패 정보	5년	실시간	
RESOUR CE	일반테이블	디스크 사용 현황	디스크 사용 현황	5년	실시간	
STORA GE	일반테이블	저장소 디스크 정보	저장소 디스크 정보	5년	실시간	
NODE	일반테이블	노드 정보	노드 정보	5년	실시간	

MEMBER

컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
ID	사용자ID	사용자 정보	Y	VARCHAR2	100	Y	N		사용자ID
NAME	이름	사용자 정보	Y	VARCHAR2	100	N	N		사용자이름
PASSWD	비밀번호	사용자 정보	Y	VARCHAR2	100	N	N		비밀번호
EMAIL	이메일주소	사용자 정보	Y	VARCHAR2	100	N	N		이메일주소
ADDRESS	주소	사용자 정보	Y	VARCHAR2	100	N	N		주소
HP	핸드폰번호	사용자 정보	Y	VARCHAR2	100	N	N		핸드폰번호

LOGINFAIL

컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
SEQ	시퀀스	사용자 정보	Y	CHAR	18	Y	N		사용자ID
ID	사용자ID	사용자 정보	Y	VARCHAR2	100	N	N		사용자이름
IP	사용자IP	사용자 정보	Y	VARCHAR2	100	N	N		비밀번호
FAIL_COUNT	로그인실패횟수	사용자 정보	Y	NUMBER	1	N	N		이메일주소
IS_LOCK	잠금여부	사용자 정보	Y	CHAR	1	N	N		주소
TIME	로그인 시간	사용자 정보	Y	DATE		N	N		핸드폰번호

2-3. 비밀번호 관리

다음은 “비밀번호 관리” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “비밀번호 관리” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020301	안전한 비밀번호 생성규칙 적용	비기능	비밀번호를 설정할 때 한국인터넷진흥원의 『암호이용안내서』의 비밀번호 생성규칙을 적용해야 한다.	RFP 보안요구사항 2.3	없음	상	암호이용안내서의 비밀번호 생성규칙을 적용한다.	암호이용 안내서의 비밀번호 생성규칙을 따른다.	
SR-020302	네트워크로 비밀번호 전송시 암호화	비기능	네트워크를 통 해 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화 하거나 암호화된 통신 채널을 이용해야 한다.	RFP 보안요구사항 2.3	없음	상	로그인, 회원정보수정 등 비밀번호 정보를 포함하는 요청은 HTTPS로 처리한다.	비밀번호 정보가 포함된 요청은 HTTP로 처리되지 않고 HTTPS로 처리된다.	
SR-020303	비밀번호 저장시 솔트가 적용된 안전한 해시함수 사용	비기능	비밀번호 저장 시, 솔트가 적용된 안전한 해시함수를 사용해야 하며, 해시 함수 실행은 서버에서 해야 한다.	RFP 보안요구사항 2.3	없음	상	해시를 추출하는 공통 모듈 구현 시 임의의 난수를 생성하여 솔트로 사용하고, 해시값과 함께 반환하도록 한다.	해시를 추출할 때 난수 형태의 솔트가 적용되고 사용한 솔트를 확인할 수 있다.	
SR-020304	비밀번호 재설정/변경 시 안전한 변경 규칙 적용	비기능	비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다.	RFP 보안요구사항 2.3	없음	상	항시 비밀번호 변경이 가능하도록 하고, 동일 비밀번호를 장기간 사용하는 경우 3개월 주기로 비밀번호를 변경하도록 강제한다. 비밀번호를 분실한 경우, 비밀번호 찾기 기능으로 가입 시 입력한 이메일로 임시 비밀번호를 발급한다.	비밀번호 변경, 3개월 주기로 변경, 비밀번호 찾기 기능을 제공한다.	
SR-020305	비밀번호 관리 규칙 적용	비기능	비밀번호 관리 규칙을 정의해서 적용해야 한다.	RFP 보안요구사항 2.3	없음	상	비밀번호 변경주기를 3개월로 하고, 만료기간을 6개월로 한다.	비밀번호 변경주기와 만료기간 정책이 반영 되었다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “비밀번호 관리” 보안요구사항의 세부 보안 요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이즈	통합시험 케이즈	…
요구사항ID	요구사항 명								
SR-020301	안전한 비밀번호 생성 규칙 적용		아키텍처 요구사항 및 구현방안 SR-020301	2.3.1					
SR-020302	네트워크로 패스 워드 전송시 암호화		아키텍처 요구사항 및 구현방안 SR-020302	2.3.2					
SR-020303	비밀번호 저장시 솔트가 적용된안쪈환해시함수 사용		아키텍처 요구사항 및 구현방안 SR-020303	2.3.3					
SR-020304	비밀번호 재설정/변경 시 안전한 변경규칙 적용		아키텍처 요구사항 및 구현방안 SR-020304	2.3.4					
SR-020305	비밀번호 관리규칙 적용		아키텍처 요구사항 및 구현방안 SR-020305	2.3.5					

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 안전한 비밀번호 생성, 전송, 저장, 변경, 관리규칙을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020301
요구사항 내용	
비밀번호를 설정할 때 한국인터넷진흥원의 『암호이용안내서』의 비밀번호 생성규칙을 적용해야 한다.	
구현방안	
① 비밀번호는 영문자, 숫자, 특수문자를 조합하여 8글자 이상을 사용하도록 한다. ② 많이 사용하는 유형의 비밀번호를 DB에 등록하고, 입력한 비밀번호가 해당 유형에 포함되면 사용할 수 없도록 한다. ③ 전화번호와 같이 사용자의 신상정보가 포함된 비밀번호는 사용할 수 없도록 한다. ④ 동일 문자의 반복, 키보드의 배열과 같이 일정한 패턴을 포함한 비밀번호는 사용할 수 없도록 한다.	

요구사항ID	SR-020302
요구사항 내용	
네트워크로 비밀번호를 전송하는 경우 반드시 비밀번호를 암호화하거나 암호화된 통신 채널을 이용해야 한다.	

2. 아키텍처 요구사항 및 구형방안

구현방안
① 비밀번호 정보를 전달하는 기능을 도출한다. - 회원가입, 회원정보 변경, 로그인, ... 등 ② 해당 기능은 HTTPS를 사용하여 스니핑과 같은 공격으로부터 데이터를 보호할 수 있도록 한다. - HTTP로 접속하는 경우, HTTPS로 자동 전환되도록 서버와 애플리케이션을 설정한다.

요구사항ID	SR-020303
요구사항 내용	
비밀번호 저장 시, 솔트가 적용된 안전한 해시함수를 사용해야 하며, 해시함수 실행은 서버에서 해야 한다.	
구현방안	
① 공통모듈로 해시를 추출하는 기능을 추가한다. ② 해시 추출 시 임의의 난수를 생성하여 솔트로 사용하도록 한다. ③ 추출된 해시값과 솔트를 반환하도록 한다.	

요구사항ID	SR-020304
요구사항 내용	
비밀번호 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다.	
구현방안	
① 사용자별 비밀번호 변경일시와 이전 비밀번호 정보를 관리한다. ② 마지막 비밀번호 변경일시가 3개월을 초과한 경우, 로그인과 동시에 비밀번호를 변경하도록 한다. ③ 비밀번호를 변경 시 비밀번호 이력을 참고하여 이전 비밀번호를 반복해서 사용할 수 없도록 한다. ④ 비밀번호 찾기를 요청하면, 회원 가입 시 입력한 이메일 주소로 임시 비밀번호를 전달한다. ⑤ 임시 비밀번호로 로그인한 경우에는 반드시 비밀번호를 변경 후 재 로그인하도록 한다.	

요구사항ID	SR-020305
요구사항 내용	
비밀번호 관리 규칙을 정의해서 적용해야 한다.	
구현방안	
① 사용자별 마지막 로그인 일시 정보를 관리한다. ② 비밀번호는 3개월 단위로 변경하도록 한다. ③ 6개월 동안 로그인하지 않은 계정은 만료 처리한다.	

개발가이드에는 안전한 비밀번호 생성, 전송, 저장 방법, 재설정 및 변경, 관리 규칙과 방법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.3.1. 비밀번호 생성 규칙

무작위대입 및 사전대입 공격을 방어하기 위해서는 다음의 생성 규칙에 따라 비밀번호를 생성하여야 한다.
- 세가지 유형의 문자(영문자, 숫자, 특수문자)를 조합하는 경우, 8자리 이상의 길이를 사용
- 두가지 유형의 문자를 조합하는 경우, 10자리 이상의 길이를 사용
- 비밀번호로 많이 사용되는 단어 사용 금지
- 전화번호와 같이 개인의 신상정보가 포함된 단어 사용 금지
- 반복 문자열, 키보드의 배열과 같이 일정한 규칙을 가진 문자열 사용 금지

공통모듈에 비밀번호 검증 기능을 추가하고, 비밀번호 생성 및 변경 시 검증 후 사용될 수 있도록 한다.

```
public static boolean checkPasswd(String
  password) { if (null == password ||
  password.equals("")) {
   return false;
  }

  if (password.trim().length() < 8)
    { LOG.info("[ValidatorUtil_checkPasswd()] checkPasswd MIN
    8!!"); return false;
  }
  for (int i = 0; i < password.length(); i++)
     { char ch = password.charAt(i);
     if (! ((ch >= '0' && '9' >= ch) || (ch >= 'a' && 'z' >= ch)
        || (ch >= 'A' && 'Z' >= ch)
        || ch == '!' || ch == '@' || ch == '$' || ch == '%'
        || ch == '^' || ch == '&' || ch == '*'))
        { LOG.info("password not [" + ch +
        "] !"); return false;
     }
  }
  return true;
}
```

2.3.2. 비밀번호 전송 규칙

로그인, 회원가입, 회원정보 변경과 같이 비밀번호 정보를 전송해야 하는 경우, HTTPS와 같은 안전한 통신 채널을 이용하여 전송하여야 한다.

이때, HTTP 프로토콜로 접속하더라도 HTTPS로 리다이렉트되도록 서버와 애플리케이션을 설정하여야 한다.

```
# Apache httpd.conf
〈VirtualHost
 xxx.xxx.xxx.xxx:80〉
 RewriteEngine On
 RewriteCond %{HTTPS}
 off
 RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
〈/VirtualHost〉
```

만약, 안전한 통신 채널을 이용할 수 없는 경우에는 안전한 암호화 방식으로 암호화하여 전송하여야 한다.

2.3.3. 비밀번호 저장

비밀번호는 개인정보보호법, 정보통신망법에서 일방향 해시로 암호화하여 저장하도록 규정하고 있다. 해시를 사용할 경우에는 SHA2 이상의 알고리즘을 사용하여야 하며, 해시의 크래킹을 방어하기 위해 솔트를 반드시 적용하여야 한다.

```
public String getPasswordHash(String password, byte[] salt) throws
 Exception { MessageDigest md = MessageDigest.getInstance("SHA-
 256"); md.update(password.getBytes());
 md.update(salt);

 byte byteData[] = md.digest();
 StringBuffer hexString = new
 StringBuffer(); for (int i=0;
 i〈byteData.length i++) {
   String hex=Integer.toHexString(0xff &
   byteData[i]); if (hex.length() == 1) {
    hexString.append('0');
    }
   hexString.append(hex);
 }
 return hexString.toString();
}
```

2.3.4. 비밀번호 재설정 및 변경

로그인한 사용자는 언제든지 비밀번호를 변경할 수 있도록 기능을 제공해야 하며, 장기간 비밀번호를 변경하지 않고 사용자에 대해서는 3개월~6개월 단위로 비밀번호 변경을 유도해야 한다.

비밀번호 변경 시 이전 비밀번호 이력을 조회하여 이전 비밀번호를 연속해서 사용하는 것을 막아야 하며, 비밀번호 변경 시에도 비밀번호 생성 규칙과 동일 규칙을 적용해야 한다.

사용자가 비밀번호를 분실한 경우, 비밀번호 찾기 기능으로 비밀번호를 재설정할 수 있도록 해야 한다.

비밀번호 찾기는 회원가입 시 등록한 이메일 주소로 임시 비밀번호를 발송하며, 사용자는 임시 비밀번호로 로그인할 수 있다.

임시 비밀번호로 로그인한 경우, 반드시 비밀번호를 변경 후 다시 로그인하도록 해야 한다.

2.3.5. 비밀번호 관리 규칙

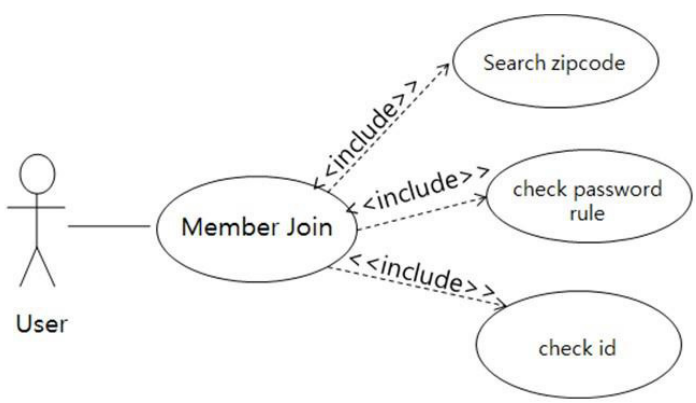
비밀번호는 다음의 관리 규칙에 따라 관리되어야 한다.
- 비밀번호는 3개월~6개월 간격으로 변경해야 한다.
- 비밀번호 변경 이력을 관리하고, 이전 비밀번호를 연속해서 사용할 수 없도록 해야 한다.
- 최소사용기간 정책에 따라 비밀번호 변경 후 일정 기간 동안 비밀번호를 변경할 수 없도록 해야 한다.
- 마지막 로그인 시간을 관리하고, 일정 기간 동안 로그인하지 않는 사용자의 계정은 만료 처리해야 한다.

유즈케이스 명세서에는 회원가입 시 비밀번호 조합규칙을 체크하는 기능을 구분하고 작업흐름을 명시하고 있다.

안전한 보안설계의 예 : 유즈케이스 명세서

2. 유즈케이스 다이어그램(UCD)

UCD ID	UCD-020101	UCD 명	게시판
관련 서브 시스템 ID	UCD-0201	관련 서브 시스템 명	고객지원

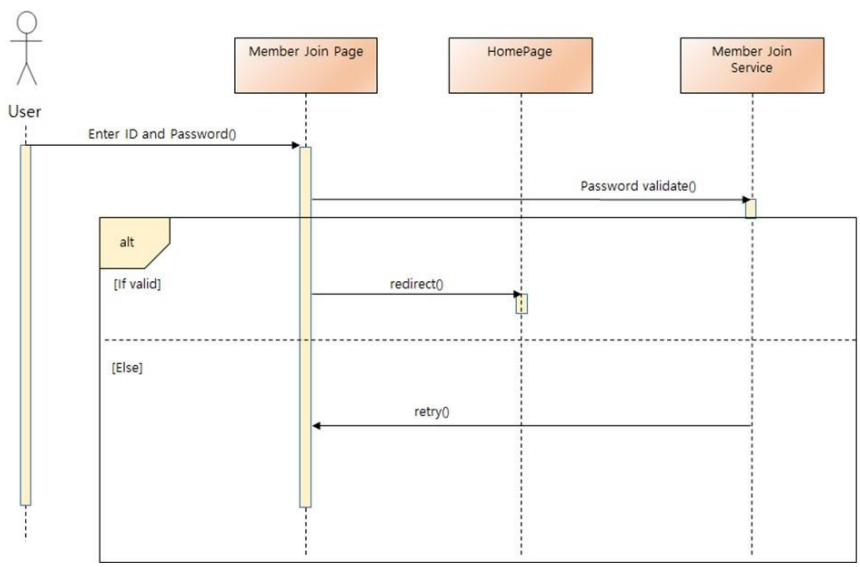


5. 유즈케이스 기술서			
요구사항 ID	회원가입하기	액터명	사용자
유즈케이스 개요 및 설명	사용자 정보를 입력하고 회원으로 가입함		
사전조건			
사후조건			
작업흐름			
정상흐름	1. 아이디, 비밀번호, 주소 등 개인정보를 입력한다. 2. 필수 항목이 입력되었는지 확인한다. 3. 입력된 아이디와 비밀번호가 유효한 지 검사한다. 4. 아이디와 비밀번호가 유효하면 메인화면을 보여준다.		
대안흐름			
예외흐름	①정상흐름 ③에서 입력된 비밀번호가 비밀번호 조합규칙을 만족하지 못하면 사용자에게 다시 비밀번호를 입력하게 한다.		

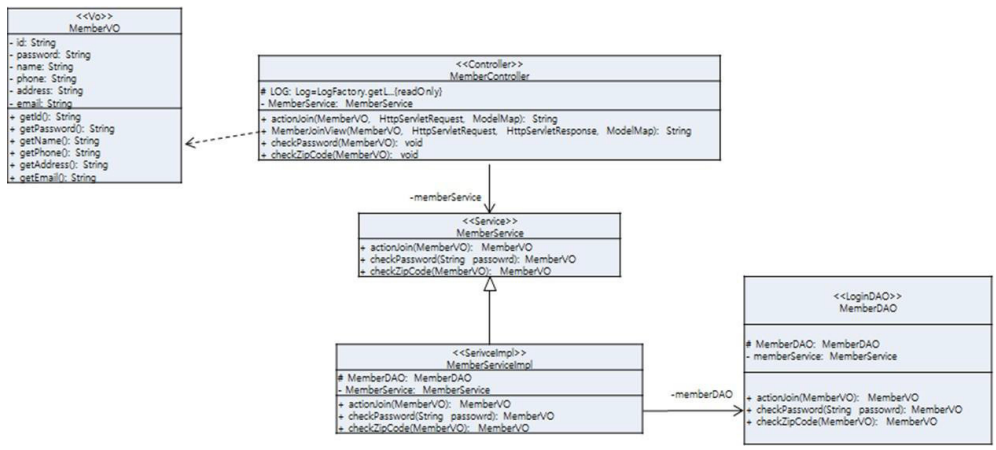
클래스 설계서의 시퀀스도에는 인증에 성공한 사용자만 게시판 기능을 사용할 수 있도록 명시하고 있으며, 설계 클래스도에는 게시판 사용자별 권한을 관리할 수 있도록 명시하고 있다.

안전한 보안설계의 예 : 클래스 설계서

2. 시퀀스도



3. 설계 클래스도



화면설계서에는 비밀번호 변경 시 비밀번호 설정 규칙을 만족하지 못할 경우에 대한 처리가 명시되어 있다.

안전한 보안설계의 예 : 화면 설계서

홈페이지 화면 정의서

시스템명	대표홈페이지	서브시스템	홈페이지 회원 가입	작성일	
페이지ID	memberJoin.jsp	페이지명	회원가입	작성자	
개요	홈페이지 회원가입				

일반회원 가입신청

일반회원아이디 [중복아이디 검색]
일반회원이름
비밀번호 (8~20자의 영문대소문자, 숫자, 특수문자만 가능합니다.)
비밀번호확인
비밀번호힌트 --선택하세요--
비밀번호정답
전화번호 - -
핸드폰번호
이메일주소
소속정보 일반기업 공공기관
소속명
우편번호 [검색]
주소
상세주소
인증서 DN [인증서등록] (공무원은 인증서 등록을 해 주시기 바랍니다.)
가입신청 취소

화면 정의

1. 사용자 : 비회원
2. 사용주기 : 수시
3. 프로세스 :
   사용자의 회원정보를 입력 받는 화면을 구성한다. 사용자에게 입력 받는 회원정보는 아이디/패스워드를 포함한 사용자의 개인정보이다. 사용자 입력 후 가입신청 버튼을 누르면 사용자의 회원정보가 DB에 추가된다.

항목 및 기능설명

1. 일반회원아이디 :
   아이디는 중복되서는 안되면 중복아이디 검색 후 사용해야 한다.
2. 비밀번호 :
   비밀번호 설정 규칙을 확인한 후 회원가입 절차를 진행한다.
3. 우편번호 :
   우편번호는 주소 검색을 통해 설정하도록 한다.

1. 패스워드가 패스워드 설정규칙을 만족하지 못하면 재설정 하도록 가입신청 화면을 제공한다.

2-4. 중요자원 접근통제

다음은 “중요자원 접근통제” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “중요자원 접근통제” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020401	중요자원에 대한 접근통제	비기능	중요자원에 대한 접근 통제 정책을 수립하여 적용해야 한다.	RFP 보안요구항 2.4	없음	상	관리자 페이지에 대한 접중요자원을 식별 하고, 사용자별 접근 통제를 구현한다.	중요자원이 식별 되고, 접근통제가 구현되어 있다.	
SR-020402	중요기능에 대한 접근통제	비기능	중요자원에 대한 접근 통제 정책을 수립하여 적용해야 한다.	RFP 보안요구항 2.4	없음	상	관리자 페이지에 대한 접중요자원을 식별하고, 사용자별 접근 통제를 구현한다.	중요자원이 식별 되고, 접근통제가 구현되어 있다.	
SR-020403	관리자 페이지에 대한 접근통제	비기능	중요자원에 대한 접근 통제 정책을 수립하여 적용해야 한다.	RFP 보안요구항 2.4	없음	상	사용자 페이지와 관리자 페이지를 분리 하고, 관리자 권한만 접근할 수 있도록 접근을 통제한다.	관리자 페이지가 분리 되고, 일반 사용자가 접근할 수 없다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “중요자원 접근통제“ 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계	
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이스	…
요구사항ID	요구사항 명							
SR-020401	중요자원에 대한 접근통제		아키텍처 요구사항 및 구현 방안 SR-020401	2.4.1			UTC-020401	
SR-020402	중요기능에 대한 접근통제		아키텍처 요구사항 및 구현 방안 SR-020402	2.4.2			UTC-020402	
SR-020403	관리자 페이지에 대한 접근통제		아키텍처 요구사항 및 구현 방안 SR-020403	2.4.3			UTC-020403	

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 중요 자원 및 기능에 대한 접근통제 방법과 관리자 페이지에 대한 접근통제 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020401

요구사항 내용

중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다.

구현방안

① 중요자원과 그룹별 권한을 식별하여 문서화한다.

자원	설명	권한 그룹			
		관리자	개발자	사용자	...
web.xml	서버 설정 정보를 포함한 파일	RW			
dispatcher- servlet.xml	스프링 컨텍스트 설정 정보를 포함한 파일	RW	R		
user-role.xml	사용자 권한 정보를 정의한 파일	RW	R		
:	:	:	:	:	:

② 식별한 중요자원에 대해 접근권한을 설정하고, 주기적으로 변경 여부를 확인한다.

요구사항ID	SR-020402

요구사항 내용

중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다.

구현방안

① 중요기능과 그룹별 권한을 식별하여 문서화한다.

자원	기능	설명	권한 그룹			
			관리자	내부 개발자	외부 사용자	...
A0170	관리자 로그인	시스템 관리자 인증	RW			
I0010	직원 로그인	내부 사용자(직원) 인증	RW	R		
O0010	회원 로그인	외부 고객(회원) 인증	RW	R		
:	:	:	:	:	:	:

2. 아키텍처 요구사항 및 구형방안

구현방안
② 식별한 중요기능에 대한 접근권한을 DB화하고, 기능, 사용자, 권한을 체크하는 공통모듈을 개발한다. ③ 기능, 사용자, 권한을 체크하는 공통모듈을 Filter 또는 Interceptor 컴포넌트에 적용하여 일관되게 접근 통제가 적용되도록 한다.

요구사항ID	SR-020403
요구사항 내용	
관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.	
구현방안	
① 관리자 기능을 일반 서비스 기능과 분리한다. ② 관리자 기능은 내부 망에서만 접근할 수 있도록 제한한다. ③ 관리자 기능은 8989 포트를 이용하여 서비스한다. (또는 80, 8080, 444, 443 등을 제외한 가용한 포트) ④ 일반 서비스 페이지에 관리자 기능으로의 연결 또는 링크가 포함되지 않도록 한다.	

개발가이드에는 중요 자원 및 기능, 관리자 페이지에 대한 접근통제 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.4.1. 중요자원에 대한 접근통제

중요자원과 권한그룹을 식별하여 문서화한다.

중요자원은 시스템 운영에 필요한 설정 및 연결 정보를 포함한 파일, 트랜젝션을 처리하는 프로세스, 메모리, 데이터베이스 등이 될 수 있다.

권한그룹은 시스템에 접근하여 운영, 사용하는 사용자의 역할에 따라 부여하는 권한을 그룹핑한 것으로 시스템 관리자, 업무 관리자, 내부 사용자, 외부 사용자 등 다양하게 구분할 수 있다.

일반적으로 시스템 중요자원은 시스템 관리자를 제외한 나머지 사용자(또는 사용자 그룹)는 접근하지 못하도록 하고, 접근이 필요한 경우에도 최소 권한만 부여하도록 한다.

식별한 중요자원과 권한그룹은 주기적으로 변경 여부를 확인해야 한다.

2.4.2. 중요기능에 대한 접근통제

중요기능과 권한그룹을 식별하여 문서화한다.

중요기능은 전체 시스템은 운영하기 위한 기능, 사용자 및 권한 관리 기능, 기초 데이터 관리 기능, 외부에 노출되어서는 안되는 데이터를 가지고 있는 기능 등 대부분 시스템 관리자 및 내부 관리자 기능이 해당된다.

중요기능에 접근하기 위해서는 반드시 인증 및 인가 과정을 거쳐야 하며, 인가는 다음의 3단계를 모두 적용하여야 한다.

2.4.2. 중요기능에 대한 접근통제

화면 수준의 접근통제

권한 없는 사용자에게 기능에 대한 버튼, 링크 등이 노출되지 않도록 한다. 예를 들어, 삭제 권한이 없는 사용자에게는 삭제 버튼을 제공하지 않으며, 일반 사용자(또는 외부 사용자)에게는 관리자 페이지로 접근할 수 있는 링크를 제공하지 않는다.

기능 수준의 접근통제

사용자가 기능을 요청했을 때, 요청을 받은 서버에서 요청한 사용자가 해당 기능에 대한 권한이 있는지 확인한 후 요청을 처리하도록 한다. 예를 들어, 삭제 권한이 없는 사용자가 주소창에 직접 삭제 기능 URL을 입력하고 요청을 하더라도 서버에서는 삭제 처리가 되지 않도록 해야 한다.

데이터 수준의 접근통제

요청 권한이 있는 사용자가 기능을 요청하더라도 해당 데이터에 대한 권한이 있는지 확인하여야 한다. 삭제 권한이 있는 사용자가 삭제 요청을 하더라도 해당 사용자가 해당 데이터에 대한 권한이 있는지 확인하고 삭제해야 한다.

중요기능에 대한 접근통제는 공통모듈로 개발하여 Filter 또는 Interceptor에 적용하여 일괄되게 접근통제가 이루어질 수 있도록 한다.

2.4.3. 관리자 페이지에 대한 접근통제

관리자를 제외한 다른 사용자의 접근을 막기 위해, 관리자 기능은 일반 서비스 기능과 물리적 또는 논리적으로 분리해서 운영하도록 한다.

그리고, 관리자 기능이 쉽게 노출되지 않도록 다음과 같은 규칙을 적용한다.
- 내부 망에서만 접근할 수 있도록 제한
- 일반적인 포트(80, 8080, 444, 443 등)가 아닌 유추하기 어려운 포트 번호를 사용
- 관리자 기능에 접근할 수 있는 외부 연결 또는 링크가 포함되지 않도록 한다.

단위 테스트 케이스 및 통합 테스트 케이스에는 중요 자원 및 기능, 관리자 페이지에 대한 접근통제가 이루어지고 있는지 여부를 테스트하는 방법과 절차가 제시되어 있다.

안전한 보안설계의 예 : 단위 테스트 케이스

요구사항 ID	UTC_0204					
설명	중요자원(프로그램 설정, 민감한 사용자 데이터 등)을 정의하고, 정의된 중요자원에 대한 접근을 통제하는 신뢰할 수 있는 방법(권한관리 포함) 및 접근통제 실패 시 대응방안 설계					
관련 컴포넌트 ID	CMT_0204			관련 프로그램 ID	PRG_0204	
케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC-020401	중요자원 접근통제	전체	사용자 권한별 시스템(서버) 접근 계정	① 사용자 권한별로 시스템(서버) 에로그인 ② 설정파일을 비롯한 권한 밖의 파일을 조작	① 시스템 관리자 계정의 경우, 로그인 및 파일 조작(RWX)이 정상 처리 ② 시스템 접속 권한이 없는 사용자의 경우, 로그인 실패 ③ 시스템 접속 권한은 있으나 파일 조작 권한이 없는 사용자의 경우, 로그인은 되나 파일 조작은 실패	

케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC-020402	중요자원 접근통제	전체	사용자 권한별 사이트 접속 계정	① 사용자 권한별로 사이트에 로그인 ② 권한 밖의 메뉴, 링크, 버튼 노출 여부 확인 ③ 권한 밖의 페이지URL을 직접 입력 하여 접근 시도	① 권한이 없는 사용자 계정으로 로그인 한 경우, 권한 밖의 메뉴, 링크, 버튼이 보이 지 않음 ② 권한 밖의 페이지 URL을 직접 입력 하여 접근한 경우, “잘못된 접근 입니다.” 메시지 출력 후 이전 페이지로 이동 ③ 권한 밖의 페이지URL을 직접 입력하여 접근한 경우, 요청시간, 요청 IP, 접속 URL, 처리결과, 메시지 등의 정보를 포함 한 접근 로그를 생성	
UTC-020403	관리자 페이지 접근 통제	전체	관리자 기능 접근 URL	① 브라우저 주소창에 관리자 페이지 주소를 입력하고 엔터 ② 관리자 기능 표출 여부 및 오류 메시지 출력 여부를 확인 ③ 접근 로그 생성 여부 확인	① 관리자로 로그인한 경우, 해당 페이지 표출 ② 관리자로 로그인하지 않았거나, 로그인 하지 않은 경우, “잘못된 접근 입니다.” 메시지 출력 후 index 페이지로 이동 ③ 요청시간, 요청IP, 접속URL, 처리 결과, 메시지 등의 정보를 포함한 접근 로그를 생성	

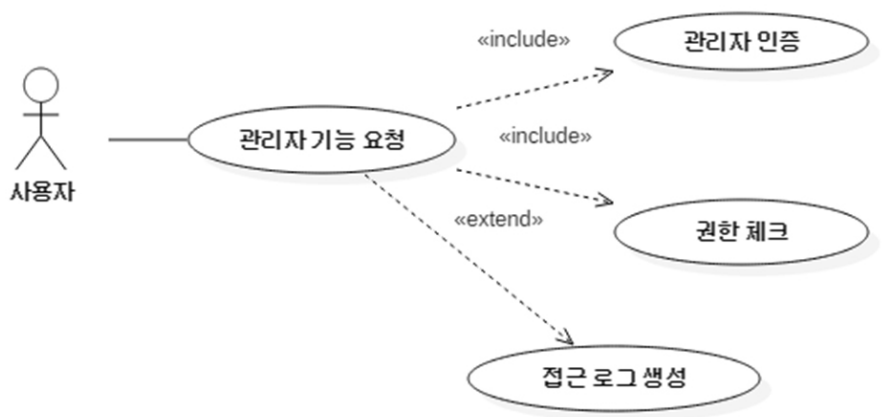
안전한 보안설계의 예 : 통합 테스트 케이스

시나리오 ID	시나리오 명	상세설명(흐름도)	검증포인트
IT-TS-COM-2401	인증 없이 관리자 기능 접근통제	관리자 기능 또는 페이지 주소를 직접 입력 → 관리자 기능 노출 및 실행 확인	- 인증 오류 처리 - 정의된 인증 오류 메시지 출력 - 로그인 페이지로 리다이렉트
IT-TS-COM-2402	비인가자 관리자 기능 접근통제	일반 권한의 사용자로 로그인 → 관리자 기 능 및 페이지 접근 URL 노출 확인 → 관리자 기능 또는 페이지 주소를 직접 입력 → 관리자 기능 노출 및 실행 확인	- 접근 권한 오류 처리 - 정의된 접근 권한 오류 메시지 출력 - 접근 로그 생성 - 이전 페이지로 리다이렉트

유즈케이스 명세서에는 관리자 기능을 구분하고 인증 및 권한을 체크하여 관리자 기능을 제공하도록 작업흐름을 명시하고 있다.

안전한 보안설계의 예 : 유즈케이스 명세서

2. 유즈케이스 다이어그램(UCD)			
UCD ID	UCD-020101	UCD 명	게시판
관련 서브 시스템 ID	UCD-0201	관련 서브 시스템 명	고객지원

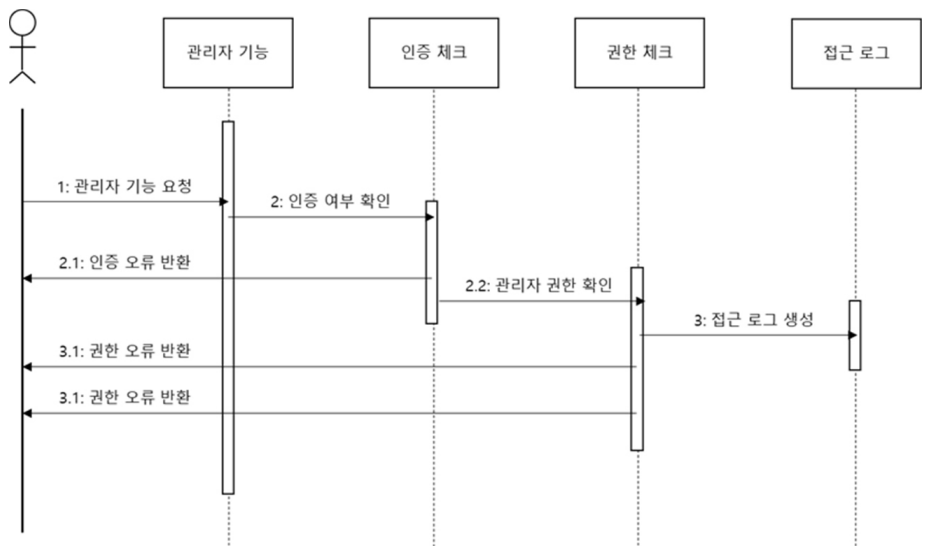


5. 유즈케이스 기술서			
요구사항 ID	관리자 기능 요청	액터명	사용자
유즈케이스 개요 및 설명	사용자가 관리자 기능 페이지에 접근 또는 요청		
사전조건			
사후조건			
작업흐름			
정상흐름	1. 사용자가 관리자 기능 페이지로의 링크를 클릭하거나 주소창에 입력 2. 관리자 인증 여부를 확인 3. 관리자 권한 체크 4. 관리자인 경우 요청한 기능을 처리		
대안흐름			
예외흐름	정상 흐름 2와 3에서 관리자가 아닌 경우, 접근 로그를 생성하고 이전 페이지로 이동		

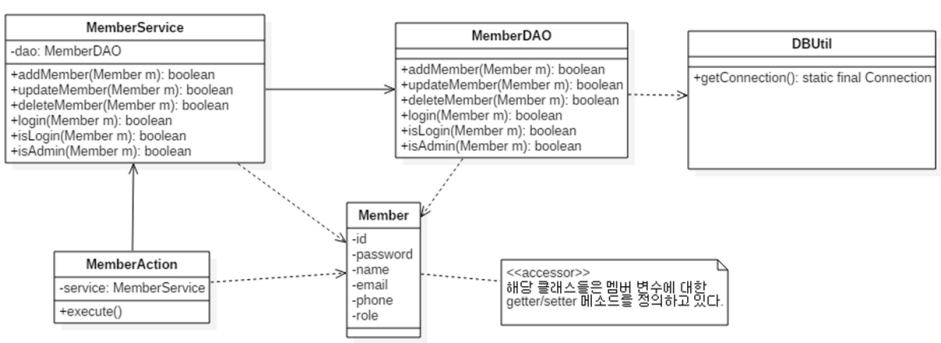
클래스 설계서의 시퀀스도에는 관라자 기능은 인증 및 권한 체크 후 제공되도록 절차를 명시하고 있으며, 설계 클래스도에는 관리자 여부를 확인할 수 있는 기능을 명시하고 있다.

안전한 보안설계의 예 : 클래스 설계서

2. 시퀀스도



3. 설계 클래스도



ERD와 테이블 정의서에는 접근통제에 사용할 사용자의 역할과 접근정보를 저장할 테이블과 컬럼을 정의하고 있다.

안전한 보안설계의 예 : ERD

MEMBER
id: VARCHAR2(10)
password: VARCHAR2(100)
name: VARCHAR2(100)
email: VARCHAR2(15)
phone: VARCHAR2(12)
role: CHAR(2)

ACCESS_LOG
id: NUMBER(8)
user_id: VARCHAR2(10)
access_dt: DATETIME
access_ip: VARCHAR(15)
access_url: VARCHAR(256)
result_cd: NUMBER(3)

안전한 보안설계의 예 : 테이블 정의서

MEMBER

컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
id	사용자 아이디	사용자 정보	Y	VARCHAR2	10	Y	N		사용자 아이디
password	사용자 비밀번호	사용자 정보	Y	VARCHAR2	100	N	N		사용자 비밀번호
name	이름	사용자 정보	Y	VARCHAR2	100	N	N		이름
email	이메일	사용자 정보	Y	VARCHAR2	15	N	N		이메일
phone	전화번호	사용자 정보	Y	VARCHAR2	12	N	N		전화번호
role	역할	사용자 정보	Y	CHAR	2	N	N		역할 - RT: 시스템 관리자 - AD: 업무 관리자 - IU: 내부 사용자 - OU: 외부 사용자

ACCESS_LOG

컬럼명 (영문)	컬럼명 (한글)	연관 엔티티명	NN 여부	데이터 타입	길이	PK 정보	FK 정보	제약 조건	컬럼설명
id	로그 아이디	접속로그 정보	Y	NUMBER	8	Y			로그 아이디
user_id	사용자 아이디	접속로그 정보	N	VARCHAR2	10	N			인증하지 않은 사용자인 경우 NULL
access_dt	접속 시간	접속로그 정보	Y	DATE		N			접속 시간
access_url	접속 URL	접속로그 정보	Y	VARHCAR	256	N			접속 URL
result_cd	접속 코드	접속로그 정보	Y	NUMBER	3	N			접속 코드 (Response Result Code)

2-5. 암호키 관리

다음은 “암호키 관리” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “암호키 관리” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020501	DB 데이터 암호화에 사용되는 암호키 관리	비기능	DB데이터 암호화에 사용 되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다.	RFP 보안요구사항 2.5	없음	상	암호이용안내서에 따라 암호키를 생성, 관리한다.	암호이용안내서에 따라 암호키를 생성, 관리 하고 있다.	
SR-020502	설정파일 내의 중요정보 암호화에 사용되는 암호키 관리	비기능	설정파일(xml,Properties)내의 중요정보 암호화에 사용되는 암호 키는 암호화해서 별도의 디렉터리에 보관 해야 한다.	RFP 보안요구사항 2.5	없음	상	설정파일내의 중요정보암호화에 사용되는 암호키는 관리자만 접근 할 수 있는 디렉토리에 보관한다.	설정파일내의 중요정보 암호화에 사용되는 암호키를 관리자만 접근할 수 있는 디렉토리에 보관하고 있다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “암호키 관리” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이즈	통합시험 케이즈	…
요구사항ID	요구사항 명								
SR-020501	DB 데이터 암호화에 사용되는 암호키 관리		아키텍처 요구사항 및 구현방안 SR-020501	2.5.1~ 2.5.4					
SR-020502	설정파일 내의 중요정보암호화에 사용되는 암호키 관리		아키텍처 요구사항 및 구현방안 SR-020502	2.5.5					

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 DB 데이터 암호화에 사용되는 암호키 관리 방안과 설정파일 내의 중요정보 암호화에 사용되는 암호키 관리 방안을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020501

요구사항 내용

DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다.

구현방안

① 아래의 내용을 문서화하여 관리한다.
- 암호화 키를 생성, 사용, 관리하는 사람
- 암호화 키 생성에 사용하는 프로그램과 절차
- 생성한 키의 종류와 변경 주기
- 키 폐기 절차
- 키 복구 방법

② 암호키를 데이터가 저장되는 데이터베이스와 물리적으로 분리된 장소에 별도로 보관한다.
③ 대칭키 암호알고리즘에 사용되는 비밀키의 송신자 사용기간은 최대 2년, 수신자 사용기간은 최대 5년으로 설정한다.
④ 공개키 암호알고리즘에서 사용되는 암호화 공개키는 최대 2년, 복호화 개인키는 최대 2년, 검증용 공개키는 최소 3년, 서명용 개인키는 최대 3년으로 설정한다.
⑤ 암호화키 사용이 일정 시간을 넘은 경우, 사용자 인터페이스로 키 사용기간이 경과했음을 알리고 새로운 키 생성을 권장하도록 한다.

요구사항ID	SR-020502

요구사항 내용

설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉토리에 보관해야 한다.

구현방안

① 설정파일에 포함된 중요정보 암호화에 사용되는 암호키는 암호화하여 별도의 디렉토리에 보관한다.
② 해당 디렉토리는 시스템 관리자만 접근할 수 있도록 접근 권한을 설정한다.

개발가이드에는 암호화 키 관리 규칙과 설정파일 내의 중요정보 암호화 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.5.1. 암호화 키 관리 규칙 작성

다음과 같은 사항을 고려하여 암호화 키 관리 규칙을 만들고 이에 따라 관리한다.

1. DB 데이터 암호화에 사용되는 암호키는 데이터가 저장되는 데이터베이스와 물리적으로 분리된 장소에 별도로 보관한다.
2. 암호화 키를 생성, 분배, 사용, 폐기하는 키의 생명주기관리를 위한 명시적인 암호화 정책을 적용한다.
3. 비밀번호나 암호화 키는 메모리에 저장하지 않는다.
4. 비밀번호나 암호키가 메모리에 저장되어야 하는 경우 사용종료 후 메모리를 0으로 초기화한다.
5. 암호키 생성 및 변경 시 암호키에 대한 백업기능을 구현한다.
6. 대칭키 암호알고리즘에 사용되는 비밀키의 송신자 사용기간은 최대 2년, 수신자 사용기간은 최대 5년으로 설정한다.
7. 공개키 암호알고리즘에서 사용되는 암호화 공개키는 최대 2년, 복호화 개인키는 최대 2년, 검증용 공개키는 최소 3년, 서명용 개인키는 최대 3년으로 설정한다.

2.5.2. 키 생명주기 기준 암호화 키 관리 프로세스 구축

키 생성, 사용, 폐기에 맞춰 관리 내용과 절차를 문서화한다.

키 생성

암호화 키와 비밀번호를 생성, 사용, 관리하는 사람 등을 명시하고 키를 생성하는데 사용하는 프로그램 등 어떠한 방법으로 생성하는지에 대한 절차를 명시한다.

키 사용

암호화 키와 비밀번호를 어떠한 방법으로 사용하는지에 대한 절차, 생성한 키의 종류에 따른 변경주기, 인가된 사용자만 키에 접근할 수 있는 접근통제 방법 및 요구사항 등을 명시한다.

키 폐기

키의 사용주기가 다 된 경우 및 사용 용도가 끝난 경우 등 생성한 키를 폐기하여야 하는 경우를 명시하고, 암호화 키와 비밀번호를 안전하게 폐기하는 절차 및 요구사항 등을 명시한다.

2.5.3. 키 복구 방안 문서화

사용자 퇴사 등으로 인해 사용자 이외의 사람에게 키 복구가 필요한 경우, 암호화 키는 정보보호담당자의 관리 하에 암호화 키 관리대장 등에서 복구하고, 비밀번호는 정보보호담당자가 임시 비밀번호를 발급하는 등 키 복구에 대한 방안을 마련하도록 한다.

2.5.4. 암호키 사용 유효기간

암·복호화 키의 사용이 일정시간을 넘은 경우 사용자 인터페이스로 키 사용기간이 경과했음을 알리고 새로운 키 생성을 권장하도록 설계한다.

2.5.5. 설정파일 내의 중요정보 암호화

설정파일에 포함된 중요정보는 안전한 암호화 알고리즘을 이용하여 암호화해야 한다. 암호화 과정에 사용된 암호화 키는 별도의 디렉토리에 안전하게 저장, 관리되도록 해야 한다.

2-6. 암호연산

다음은 “암호연산” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “암호연산” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020601	안전한 양방향 암호화 알고리즘 사용	비기능	대칭키 또는 비대칭키를 이용해서 암호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호이용안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다.	RFP 보안요구사항 2.6	없음	상	공통모듈에 암호화 기능을 추가하고, 구현시 대칭키 암호 알고리즘으로는 AES를, 비대칭 암호 알고리즘으로는 RSA를 사용하고, 각각의 키는 256, 2048의 길이를 사용한다.	대칭키 암호 알고리즘으로는 AES를, 비대칭 암호 알고리즘으로는 RSA를 사용하고 있으며, 각각의 키는 256, 2048의 길이를 이용 하고 있다.	
SR-020602	안전한 일방향 암호화 알고리즘 사용	비기능	복호화 되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트 값을 적용하여 암호화해야 한다.	RFP 보안요구사항 2.6	없음	상	공통모듈에 해시 기능을 추가하고, 안전한 SHA2 알고리즘을 이 용하여구현한다. 이때 , 난수를 이용한 솔트가 적용될 수 있도록 한다.	해시 추출 시 SHA2 계열 알고리즘을 사용하고, 솔트를 적용하고 있다.	
SR-020603	안전한 난수생성 알고리즘 사용	비기능	난수생성 시 안전한 난수생성 알고리즘을 사용해야 한다.	RFP 보안요구사항 2.6	없음	상	공통모듈에 난수 생성 기능을 추가하고, java. util.Random 클래스를 이용하여 난수를 생성 한다. seed로는 현재 시간을 사용한다.	java.util.Ran-dom 클래스를 이용하여 난수를 생성하고 있으며, 시간을 seed로 사용하고 있다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “암호연산” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계	
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이즈	…
요구사항ID	요구사항 명							
SR-020601	안전한 양방향 암호화 알고리즘 사용		아키텍처 요구사항 및 구현 방안 SR-020601	2.6.1				
SR-020502	안전한 일방향 암호화 알고리즘 사용		아키텍처 요구사항 및 구현 방안 SR-020602	2.6.2				

분석 단계			설계 단계			구현 단계	시험 단계	
사용자 요구사항 명세서		...	아키텍처정의서	개발 가이드	...	...	단위시험 케이즈	...
요구사항ID	요구사항 명							
SR-020503	안전한 난수생성 알고리즘 사용		아키텍처 요구사항 및 구현 방안 SR-020603	2.6.3				

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 명시적인 예외와 런타임 예외 대응 방법과 오류 메시지로 정보노출이 되지 않도록 하는 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020601
요구사항 내용	
대칭키 또는 비대칭키를 이용해서 암호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호이용안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다.	
구현방안	
① 공통모듈로 대칭 암/복호화 기능과 비대칭 암/복호화 기능을 추가한다. ② 대칭 암/복호화 기능은 AES 알고리즘을 이용하며, 키 길이는 256으로 설정한다. ③ 비대칭 암/복호화 기능은 RSA 알고리즘을 이용하며, 키 길이는 2048로 설정한다.	

요구사항ID	SR-020602
요구사항 내용	
복호화 되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트값을 적용하여 암호화해야 한다.	
구현방안	
① 공통모듈로 해시 추출 기능을 추가한다. ② 해시 추출 시 SHA2 계열의 안전한 알고리즘을 사용하고, 임의의 난수를 생성하여 솔트로 사용하도록 한다. ③ 추출된 해시값과 솔트를 반환하도록 한다.	

요구사항ID	SR-020603
요구사항 내용	
난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.	
구현방안	
① 공통모듈로 난수 생성 기능을 추가한다. ② 난수 생성 시 java.util.Random 클래스를 사용하며, 시간을 seed 값으로 사용한다.	

개발가이드에는 안전한 암호화 알고리즘 사용, 안전한 해시 함수 사용, 안전한 난수 생성 함수 사용 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.6.1. 안전한 암호화 알고리즘 사용

암호화 기능 구현에 사용되는 암호화 알고리즘은 학계에서 안전성이 검증된 알고리즘을 사용하여야 한다.

대칭키 암호화 알고리즘	비대칭키 암호화 알고리즘
SEED	RSA
ARIA-128/192/256	KCDSA(전자서명용)
AES-128/192/256	RSAES-OAEP
Camelia-128/192/256	ElGamal
Blowfish	ECC
MISTY1	ECKCDSA 등
KASUMI 등	

암호화에 사용되는 키의 길이는 알고리즘의 안전성을 보장할 수 있도록 충분히 큰 값을 사용해야 하며, 대칭키의 경우 128비트 이상을, 비대칭키의 경우 2048비트 이상을 사용해야 한다.

2.6.2. 안전한 해시 함수 사용

해시 함수를 사용할 경우, 안전성이 검증된 SHA2 계열 이상의 해시를 사용해야 한다. 또한 해시의 크래킹을 방어하기 위해서는 솔트를 적용해야 하며, 솔트는 추측할 수 없도록 난수를 생성하여 적용하며, 사용한 솔트는 안전한 방식으로 보관해야 한다.

2.6.3. 안전한 난수 생성 함수 사용

java.Math 클래스의 random() 함수는 시드를 재설정할 수 없으므로 난수 생성 시 사용하지 않고, java.util.Random 클래스를 이용하거나, java.security.SecureRandom 클래스를 이용하여 난수를 생성해야 한다.

시드는 항상 변하는 시간과 같은 값을 사용한다.

2-7. 중요정보 저장

다음은 “중요정보 저장” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “중요정보 저장” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020701	중요정보 또는 개인정보 암호화 저장	비기능	중요정보 또는 개인정 보는 암호화해서 저장해야 한다.	RFP 보안요구사항 2.7	없음	상	중요정보를 식별하고, 암호화해서 저장한다.	중요정보가 식별 되고, 암호화해서 저장된다.	
SR-020702	중요정보 삭제 및 메모리 초기화	비기능	중요정보가 메모리에 남지 않도록 해야 한 다.	RFP 보안요구사항 2.7	없음	상	중요정보가 쿠키를 통 해서 전달되지 않도록 하고, 쿠키로 전달되는 경우 쿠키의 유효 기간을 최소로 한다.	중요정보가 쿠키로 전달되는 경우, 세션이 만료되면 쿠키가 사라진다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “중요정보 저장” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이스	통합시험 케이스	…
요구사항ID	요구사항 명								
SR-020701	중요정보 또는 개인정보 암호화 저장		아키텍처 요구사항 및 구현방안 SR-020701	2.7.1					
SR-020702	중요정보 삭제 및 메모리 초기화		아키텍처 요구사항 및 구현방안 SR-020702	2.7.2					

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 중요정보가 유출되는 것을 막기 위해 중요정보를 안전하게 저장하는 방법과 완벽하게 삭제하는 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020501

요구사항 내용

중요정보 또는 개인정보는 암호화해서 저장해야 한다.

구현방안

① 암호화가 필요한 중요정보와 개인정보 항목을 식별한다.

항목	설명	비고
비밀번호	사용자 인증에 사용되는 정보	일방향 해시함수 사용 (SHA256)
계좌번호	사용자의 이용은행 계좌번호	AES 알고리즘 사용
:	:	:

② 공통 모듈(Utility)에 암호화 함수를 추가하고, 도출된 암호화 대상 정보에 대해서는 해당 함수의 반환값을 저장한다.

③ 암호화 함수는 다음의 조건을 준수해야 한다.

- 일방향 해시는 SHA256을, 대칭 암호화는 AES를 비대칭 암호화는 RSA를 사용한다.
- 키길이로 AES는 128비트를, RSA는 2048비트를 사용한다.
- 반드시 해시값을 추출할 때는 솔트를 적용한다.
- 키와 솔트는 분리된 공간에 안전하게 저장, 관리한다.

요구사항ID	SR-020702

요구사항 내용

중요정보가 메모리에 남지 않도록 해야 한다.

구현방안

① SDR-020701에서 식별한 중요정보가 쿠키로 전달되지 않도록 한다.

② 중요정보를 쿠키에 담는 경우, 해당 정보를 안전한 방식으로 암호화하고, 쿠키의 유효기간을 최소화한다.

③ 중요정보를 입력하는 입력폼에 대해서는 자동완성 기능을 비활성화 한다.

④ 중요정보를 담은 변수가 더 이상 필요하지 않은 경우, 0으로 초기화하여 메모리에 남지 않도록 한다.

개발가이드에는 중요정보를 암호화하여 안전하게 저장하는 방법과 세션으로 전달할 때 고려해야 할 사항들을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

2.7.1. 주요 데이터 암호화

중요 데이터(개인정보, 인증정보, 금융정보)는 안전한 방식으로 암호화하여 저장한다.

암호화 과정에 사용하는 알고리즘은 자신만의 암호화 알고리즘을 개발하는 것은 위험하며, 안전성이 검증된 알고리즘을 사용해야 한다.

```
// 올바르지 않은 방법
// Cipher c = Cipher.getInstance("DES");
// c.init(Cipher.ENCRYPT_MODE, key);
:
// 올바른 방법
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding"); c.init(Cipher.ENCRYPT_MODE,
key);
:
```

암호화 알고리즘의 안전성을 보장할 수 있도록 충분한 키 길이를 적용해 줘야 한다. 일반적으로 대칭 암호화 방식은 128비트 이상을, 비대칭 암호화 방식은 2048비트 이상을 사용해야 한다.

```
// 올바르지 않은 방법
// KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
// keyGen.initialize(512);
:
// 올바른 방법
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA"); keyGen.initialize(2048);
:
```

2.7.2. 주요 데이터 세션 쿠키 사용

쿠키에는 주요 데이터가 포함되지 않도록 설계해야 하며, 부득이 주요 데이터가 포함되어야 할 경우에는 해당 데이터를 암호화하고 세션 쿠키로 설정해서 전송해야 한다.

```
Cookie c = new Cookie("name",
"value"); c.setMaxAge(-1);
c.setSecure(true); response.addCookie
(c);
```

세션 쿠키 설정 방법은 다음과 같다.

2-8. 중요정보 전송

다음은 “중요정보 전송” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “중요정보 전송” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-020801	민감 정보 암호화 전송	비기능	인증정보와 같은 민감 한 정보 전송시 안전하게 암호화해서 전송해야 한다.	RFP 보안요구사항 2.8	없음	상	민감한 정보를 식별하고, 전송시 암호화해서 전송한다.	민감한 정보를 암호화해서 전송하고 있다.	
SR-020802	쿠키 중요정보 암호화 전송	비기능	쿠키에 포함되는 중요 정보는 암호화해서 전 송해야 한다.	RFP 보안요구사항 2.6	없음	상	쿠키로 중요 정보를 전달하는 경우, 해당 정보를 암호화해서 전송 한다.	쿠키에 포함된 중요 정보를 암호화 하고 있다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “중요정보 전송” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이스	통합시험 케이스	…
요구사항ID	요구사항 명								
SR-020801	민감 정보 암호화 전송		아키텍처 요구사항 및 구현방안 SR-020801	2.8.1					
SR-020802	쿠키 중요정보 암호화 전송		아키텍처 요구사항 및 구현방안 SR-020802	2.8.2					

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 중요정보가 유출되는 것을 막기 위해 중요정보를 안전하게 저장하는 방법과 완벽하게 삭제하는 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-020801
요구사항 내용	
인증정보와 같은 민감한 정보 전송시 안전하게 암호화해서 전송해야 한다.	
구현방안	
① 중요한 정보를 식별한다. ② 중요한 정보가 전송되어야 하는 경우, HTTPS을 이용하여 클라이언트와 서버 간에 안전한 통신이 되도록 한다. ③ HTTP 프로토콜로 접속하면 강제로 HTTPS로 리다이렉트 되도록 한다.	

요구사항ID	SR-020802
요구사항 내용	
쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.	
구현방안	
① SDR-028-001에서 식별한 중요 정보는 쿠키에 포함되지 않도록 한다. ② 만약, 쿠키에 포함되어야 한다면, 공통 모듈(utilities)에 정의, 구현한 암호화 모듈을 이용하여 암호화하여 전달 되도록 한다. ③ 중요 정보가 포함된 쿠키는 세션 쿠키로 설정한다.	

개발가이드에는 네트워크로 중요 정보를 전송하는 경우, 중요 정보를 노출하지 않고 전송하는 기법과 쿠키를 이용하여 중요 정보를 전송하는 경우, 중요 정보를 안전하게 처리하는 기법을 제시 하고 있다.

안전한 보안설계의 예 : 개발가이드

2.8.1. 중요정보 전송

중요정보를 네트워크로 전송해야 하는 경우, 안전한 암호화 모듈로 암호화하여 전송하거나 안전한 통신 채널을 사용하도록 설계한다.

HTTPS를 적용한 경우, HTTP 프로토콜로 접속하면 HTTPS로 리다이렉트 되도록 해야 한다. 아파치의 경우 httpd. conf 파일에 아래와 같이 설정한 후 서비스를 재시작하여 적용할 수 있다.

```
〈VirtualHost
xxx.xxx.xxx.xxx:80〉
RewriteEngine  On
RewriteCond  %{HTTPS}
off
RewriteRule  (.*)  https://%{HTTP_HOST}%{REQUEST_URI}
〈/VirtualHost〉
```

2.8.2. 중요정보를 포함한 쿠키

쿠키에 중 정보가 포함되어야 하는 경우, 해당 정보를 암호화하고, 세션 쿠키로 설정한다. 세션 쿠키 설정 방법은 다음과 같다.

```
Cookie  c  =  new  Cookie(“name”,
“value”);  c.setMaxAge(-1);
response.addCookie(c);
```

3. 에러처리

3-1. 예외처리

다음은 “예외처리” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “예외처리” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약사항	중요도	해결방안	검수기준	비고
SR-030101	명시적 예외처리	비기능	명시적인 예외의 경우 예외처리 블럭을 이용 하여 예외발생시 수행해야 하는 기능이 구현 되도록 해야 한다.	RFP 보안요구항 3.1	없음	상	명시적인 예외에 대해 처리한다.	명시적인 예외에 대한 처리가 구현 되어 있다.	
SR-030102	런타임 예외처리	비기능	런타임 예외의 경우 입력값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다.	RFP 보안요구사항 3.1	없음	상	정적분석으로 런타임 예외가 발생할 수 있는 경우에 대해 검증 기능을 추가한다.	런타임 예외가 발생하지 않는다.	
SR-030103	오류 메시지로 정보노 출 방지	비기능	에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.	RFP 보안요구사항 3.1	없음	상	서버 설정으로 지정된 오류 페이지가 사용자에게 보여지도록 한다.	오류 발생시 지정된 오류 페이지가 보여진다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “예외처리” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	…
요구사항ID	요구사항 명						
SR-030101	명시적 예외처리		아키텍처 요구사항 및 구현방안 SR-030101	3.1.1			
SR-030102	런타임 예외처리		아키텍처 요구사항 및 구현방안 SR-030102	3.1.2			
SR-030103	오류 메시지로 정보노출 방지		아키텍처 요구사항 및 구현방안 SR-030103	3.1.3			

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 명시적인 예외와 런타임 예외 대응 방법과 오류 메시지로 정보노출이 되지 않도록 하는 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-030101
요구사항 내용	
명시적인 예외의 경우 예외처리 블럭을 이용하여 예외 발생시 수행해야 하는 기능이 구현되도록 해야 한다.	
구현방안	
① 명시적으로 예외가 발생할 수 있는 부분은 try 블록으로 감싸고, catch 블록에는 적절한 에러 대응을 추가한다.	

요구사항ID	SR-030102
요구사항 내용	
런타임 예외의 경우 입력값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다.	
구현방안	
① NULL을 반환할 수 있는 함수의 결과를 값으로 받는 변수는 반드시 NULL 여부를 확인하고 사용될 수 있도록 한다. ② 배열의 인덱스, 반복문의 반복회수 등에 사용되는 숫자형 변수는 반드시 허용 범위를 검증하고 사용될 수 있도록 한다. ③ 기타 런타임 예외가 발생할 수 있는 부분은 입력값의 범위를 체크해서 애플리케이션이 정상적으로 동작할 수 있는 값만 사용될 수 있도록 한다.	

요구사항ID	SR-030103
요구사항 내용	
에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.	
구현방안	
① 웹 애플리케이션 서버를 설정해 특정 에러나 예외가 발생했을 때, 지정된 페이지가 사용자에게 보여질 수 있도록 한다. ② 발생할 수 있는 오류 종류와 코드, 메시지 등을 정의하여 사용될 수 있도록 한다. ③ 사용자에게 보여지는 오류 메시지에는 개인정보, 시스템정보, 민감정보와 같은 중요정보가 포함되지 않도록 한다.	

개발가이드에는 명시적 예외에 대한 처리, 런타임 예외에 대한 처리, 오류 메시지로 정보노출 방지 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

3.1.1. 명시적 예외처리

예외가 발생할 수 있는 부분은 try 구문으로 감싸고, catch 구문에 적절한 예외 처리를 추가한다. 해제해 줘야 할 자원을 생성한 경우, finally 구문을 이용하여 자원 해제를 처리한다.

```
try {
// 예외 발생 코드
} catch (예외) {
// 예외 처리
} finally {
// 자원 해제
}
```

3.1.2. 런타임 예외처리

입력값에 따라 런타임에 발생할 수 있는 예외는 입력값의 용도에 맞춰 범위를 체크하고 사용하도록 한다. NULL을 반환 할 수 있는 함수의 결과값을 받는 변수는 반드시 NULL 여부를 체크하고 사용하도록 한다.

```
String s = request.getParameter("abc");
if (s != null && s.equals("hello")) {
:
}
```

배열의 인덱스, 반복문의 반복회수 등에 사용되는 숫자형 변수는 반드시 허용 범위를 검증하고 사용하도록 한다.

```
int ii =
Integer.parseInt(request.getParameter("no")); if (ii
< 0) return;
String[] values = new String[ii];
```

3.1.3. 오류메시지로 정보노출 방지

시스템에서 발생할 수 있는 오류의 종류와 내용, 코드, 메시지 등을 정리해서 문서화하고, 오류 메시지에 중요정보가 포함되지 않도록 한다.

3.1.3. 오류메시지로 정보노출 방지

웹 애플리케이션 서버를 설정해 특정 에러나 예외사항에 대해 지정된 페이지가 사용자에게 보여지도록 한다.

```
〈error-page〉
〈error-code〉404〈/error-code〉
〈location〉/WEB-INF/jsp/common/error/404error.jsp〈/location〉
〈/error-page〉
〈error-page〉
〈exception-type〉java.lang.Throwable〈/exception-type〉
〈location〉/WEB-INF/jsp/common/error/error.jsp〈/location〉
〈/error-page〉
```

4. 세션통제

4-1. 세션통제

다음은 “세션통제” 보안요구사항에 대한 안전한 설계 산출물의 예시이며, 제시된 설계 산출물의 종류와 내용은 개발되는 소프트웨어의 특성, 환경, 방법론 등에 따라 다를 수 있다.

사용자 요구사항 정의서에는 “세션통제” 보안요구사항의 세부 보안요구사항을 비기능 보안요구항목으로 등록하고 해결방안과 검수기준을 제시하고 있다.

안전한 보안설계의 예 : 사용자 요구사항 정의서

요구 사항 ID	요구사항명	구분	요구사항설명	요구사항 출처	제약 사항	중요 도	해결방안	검수기준	비고
SR-040101	세션 간 데이터 공유 예방	비 기능	세션간 데이터가 공유 되지 않도록 설계해야 한다.	RFP 보안요구 사항 4.1	없음	상	컨트롤러 컴포넌트를 상속받는 클래스에 멤버 변수가 포함 되지 않도록 하고, JSP 페이지의 선언부에 변수를 선언하지 않도록 한다.	컨트롤러 컴포넌트를 상속받은 클래스에 멤버 변수가 포함되지 않고, JSP 페이지의 선언부에 변수가 선언되지 않았다.	
SR-040102	안전한 세션 관리	비 기능	세션이 안전하게 관리 되도록 해야 한다.	RFP 보안요구 사항 4.1	없음	상	모든 화면 오른쪽 상단에 “로그아웃” 버튼을 제공하여 쉽게 로그아웃을 요청할 수 있도록 하고, 로그아웃 후 세션에 저장된 모든 정보가 삭제되도록 한 다.	오른쪽 상단에 로그아웃 버튼이 제공되고, 로그아웃 후 세션 정보가 삭제된다.	
SR-040103	안전한 세션ID 관리	비 기능	세션ID가 안전하게 관리 되도록 해야 한다.	RFP 보안요구 사항 4.1	없음	상	인증에 성공하면 인증 전에 할당한 세션 ID를 파기하고, 세션ID를 재할당하여 전달한다.	인증전후 다른 세션 ID를 사용 한다.	

요구사항 추적표에는 사용자 요구사항 정의서에 등록된 “세션통제” 보안요구사항의 세부 보안요구사항이 분석, 설계, 구현, 시험 단계에서 어떻게 구체화되는지 추적할 수 있도록 관련 산출물의 참조 정보를 제시하고 있다.

안전한 보안설계의 예 : 요구사항 추적표

분석 단계			설계 단계			구현 단계	시험 단계		
사용자 요구사항 명세서		…	아키텍처정의서	개발 가이드	…	…	단위시험 케이즈	통합시험 케이즈	…
요구사항ID	요구사항 명								
SR-040101	세션 간 데이터 공유 예방		아키텍처 요구사항 및 구현방안 SR-040101	4.1.1			UTC 040101		
SR-040102	안전한 세션 관리		아키텍처 요구사항 및 구현방안 SR-040102	4.1.2			UTC 040102	TTC 040101	
SR-040103	안전한 세션ID 관리		아키텍처 요구사항 및 구현방안 SR-040103	4.1.3			UTC 040103	TTC 040101	

아키텍처 설계서의 아키텍처 요구사항 및 구현방안에서 세션 간 데이터가 공유되지 않도록 하는 방법과 세션 및 세션 ID를 안전하게 관리하는 방법을 명시하고 있다.

안전한 보안설계의 예 : 아키텍처 설계서

2. 아키텍처 요구사항 및 구형방안

요구사항ID	SR-040101
요구사항 내용	
세션 간 데이터가 공유되지 않도록 설계해야 한다.	
구현방안	
① 컨트롤러 컴포넌트를 상속받은 클래스에 멤버 변수가 포함되지 않도록 한다. ② JSP 페이지의 선언부에 변수를 선언하지 않도록 한다.	

요구사항ID	SR-040102
요구사항 내용	
세션이 안전하게 관리되도록 해야 한다.	
구현방안	
① 모든 페이지에서 로그아웃을 요청할 수 있도록 로그아웃 버튼을 지정된 위치에 노출되도록 한다. ② 사용자가 로그아웃을 요청하면 session.invalidate() 메소드를 사용하여 세션에 저장된 정보를 완전히 제거한다. ③ 세션 타임아웃은 15분으로 설정하고, 이전 세션이 종료되지 않은 상태에서 새로운 세션이 생성되지 않도록 한다. ④ 일정시간 동안 사용되지 않는 세션은 강제적으로 삭제한다. ⑤ 세션 ID가 포함된 쿠키는 HttpOnly 속성을 설정하여 전달한다. ⑥ 사용자가 비밀번호를 변경한 경우, 현재 활성화된 세션을 모두 삭제하고 다시 할당한다.	

요구사항ID	SR-040103
요구사항 내용	
세션ID가 안전하게 관리되도록 해야 한다.	
구현방안	
① 세션ID는 서버에서 생성한 것을 사용한다. ② URL Rewrite 기능이 사용되지 않도록 한다. ③ 인증에 성공하면 인증전에 할당한 세션ID를 파기하고 세션ID를 재할당하여 전달한다. ④ 주기적으로 세션ID를 재할당하도록 한다.	

개발가이드에는 세션 간 데이터 공유가 일어나지 않도록 하는 기법과 세션 및 세션 ID를 안전하게 관리하는 기법을 제시하고 있다.

안전한 보안설계의 예 : 개발가이드

4.1.1. 세션 간 데이터 공유 방지

컨트롤러 컴포넌트를 상속받은 클래스에 멤버 변수나 클래스 변수는 세션 간에 공유가 발생하므로, 클래스 설계시 읽고 쓰기가 가능한 변수를 사용하지 않도록 한다.

```
@Controller
@RequestMapping("/board") public class BoardController {
private int pageNo = 1; /* 세션 간 공유가 발생 */
```

JSP 페이지의 선언부에 변수를 선언하면 서블릿으로 변환하는 과정에서 해당 클래스의 멤버 변수로 설정되어 세션 간 공유가 발생하므로, JSP 페이지의 선언부에 변수를 선언하지 않도록 한다.

```
<%!
String value = ""; /* 서블릿 클래스의 멤버 변수로 변환 */
:
%>
```

4.1.2. 안전한 세션 관리

시스템 내의 모든 페이지에 대하여 로그아웃이 가능하도록 UI를 설계하고, 로그아웃을 요청하면 session.invalidate() 메소드를 사용하여 세션에 저장된 정보를 완전히 제거한다.

세션 타임아웃 시간은 중요기능의 경우 2~5분, 위험도가 낮은 애플리케이션의 경우에는 15~30분으로 설정하고, 이전 세션이 종료되지 않은 상태에서 새로운 세션이 생성되지 않도록 한다.

일정시간 동안 사용되지 않는 세션 정보는 강제적으로 삭제한다.

중복 로그인을 허용하지 않는 경우, 새로운 로그인 세션 생성 시 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료하도록 한다.

세션ID가 포함된 쿠키에 대해 HttpOnly 속성을 설정하여 XSS 공격에 대응하도록 한다. 사용자가 비밀번호를 변경하는 경우 현재 활성화된 모든 세션을 삭제하고 다시 할당한다.

4.1.3. 안전한 세션ID 관리

세션ID는 안전한 서버에서 생성해서 사용되도록 한다. 세션ID는 최소 128비트의 길이로 생성되어야 하며, 안전한 난수 알고리즘을 적용하여 예측이 불가능한 값이 사용되도록 한다.

URL Rewrite 기능을 사용하는 경우 세션ID가 URL에 노출될 수 있으므로, URL Rewrite 기능을 사용하지 않도록 설계한다.

로그인 성공 시 로그인 전에 할당받은 세션ID는 파기하고 새로운 값으로 재할당하여 세션ID 고정 공격에 대응하도록 한다.

장기간 접속되어 있는 경우 세션ID의 노출 위험이 커지므로, 일정시간 주기적으로 세션ID를 재할당하도록 한다.

단위 테스트 케이스와 통합 테스트 케이스에는 세션 간 데이터 공유가 일어나는지 여부와 세션 및 세션 ID가 안전하게 생성 관리되는지 여부를 테스트하는 방법과 절차가 제시되어 있다.

안전한 보안설계의 예 : 단위 테스트 케이스

요구사항 ID	UTC_041					
설명	다른 세션 간 데이터 공유금지, 세션 ID 노출금지, (재)로그인시 세션ID 변경, 세션종료(비활성화, 유효 기간 등) 처리 등 세션을 안전하게 관리할 수 있는 방안 설계					
관련 컴포넌트 ID	CMT_041			관련 프로그램 ID	PRG_041	
케이스 ID	케이스 명	작업 권한	시험 데이터	시험항목 및 처리절차	예상결과 및 검증방법	시험 결과
UTC-041001	세션 간 데이터 공유 예방	전체	페이지 구분이 가능한 게시물	① 서로 다른 브라우저 또는 컴퓨터를 이용하여 게시판 조회 페이지를 호출 ② 하나는 1페이지를 다른 하나는 2페이지를 클릭 (거의 동시에)	① 각각 클릭한 페이지 번호의 내용이 출력 ② 만약, 동일한 페이지 번호의 내용이 보인다면 세션 간 데이터가 공유되는지 확인 (소스 검증)	
UTC-041002	안전한 세션 관리	전체	인증에 필요한 아이디/비밀번호	① 유효한 아이디/비밀번호로 로그인 ② 로그아웃 버튼 위치 확인 ③ 15분 동안 사용하지 않고 대기	① 인증에 성공하면 로그아웃 버튼이 동일 한 위치에 노출(제공) ② 15분이 경과하면 세션을 자동으로 종료	
UTC-041003	안전한 세션ID 관리	전체	인증에 필요한 아이디/비밀번호	① 유효한 아이디/비밀번호로 로그인	① 인증에 성공하면 인증전에 할당한 세션 ID를 파기하고 세션ID를 재할 당하여 전달	

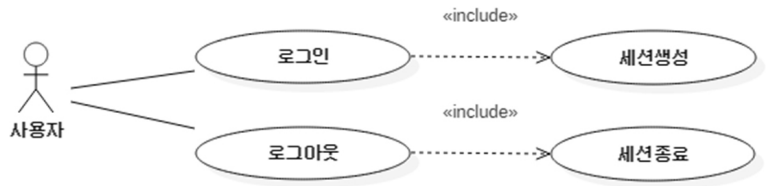
안전한 보안설계의 예 : 통합 테스트 케이스

시나리오 ID	시나리오 명	상세설명(흐름도)	검증포인트
TTC-040101	세션ID 관리	유효한 아이디/비밀번호로 로그인 -> 페이지 이동 -> 로그아웃 * 동일 과정을 반복	- 로그인 전후 세션ID가 바뀌는지 확인 - 생성된 세션ID가 규칙성을 가지는지 확인 - 동일한 세션ID가 중복되어 사용되는지 확인 - 인증 후 로그아웃 버튼이 동일한 위치에서 제공되는지 확인

유즈케이스 명세서에는 로그인과 로그아웃 기능을 구분하고 각각의 유즈케이스별로의 작업흐름을 명시하고 있다.

안전한 보안설계의 예 : 유즈케이스 명세서

2. 유즈케이스 다이어그램(UCD)			
UCD ID	UCD-040101	UCD 명	로그인/로그아웃
관련 서브 시스템 ID	UCD-0401	관련 서브 시스템 명	공통기능

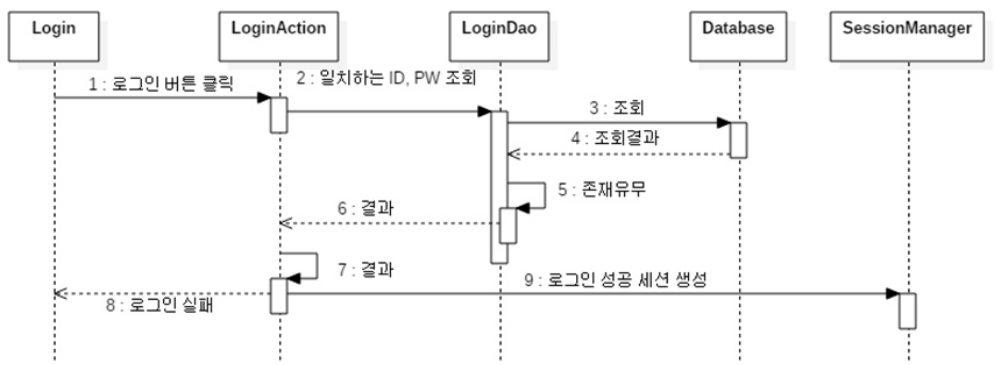


5. 유즈케이스 기술서			
요구사항 ID	로그인	액터명	회원
유즈케이스 개요 및 설명	사용자가 로그인을 요청		
사전조건			
사후조건			
작업흐름			
정상흐름	1. 사용자가 아이디, 비밀번호를 입력하고 로그인 버튼을 클릭 2. 사용자가 입력한 아이디, 비밀번호와 일치하는 정보를 조회 3. 일치하는 경우, 세션ID를 재할당하고 메인 페이지로 리다이렉트		
대안흐름			
예외흐름	정상흐름 2의 결과, 일치하는 정보가 존재하지 않는 경우, 오류 메시지와 함께 로그인 페이지를 리턴		
요구사항 ID	로그아웃	액터명	회원
유즈케이스 개요 및 설명	사용자가 로그아웃을 요청		
사전조건	사용자가 로그인한 상태		
사후조건			
작업흐름			
정상흐름	1. 사용자가 로그아웃 버튼을 클릭 2. 세션에 저장된 정보를 완전히 제거한 후 기본 페이지로 이동		
대안흐름	1.		
예외흐름	1.		

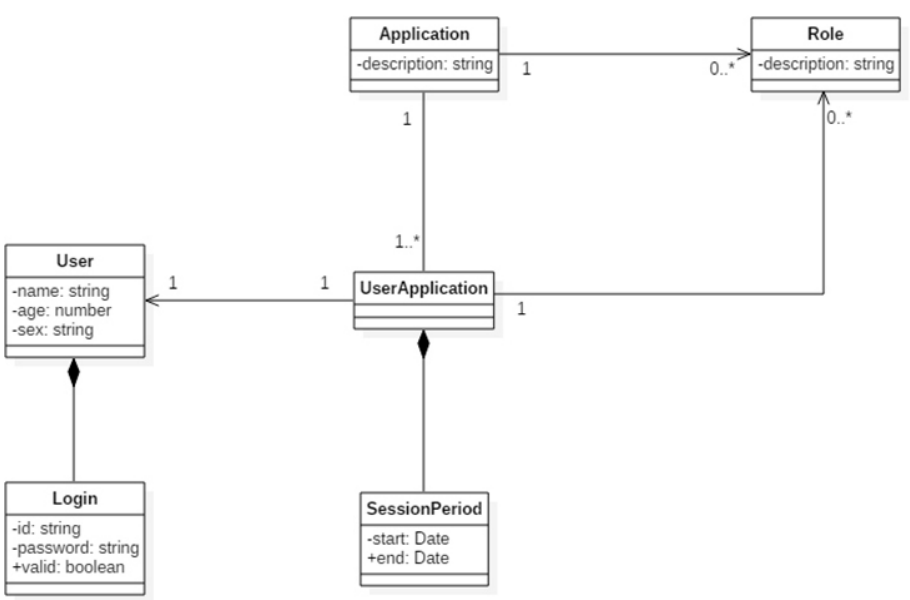
클래스 설계서에는 인증 처리 및 세션 생성 절차와 필요한 기능을 명시하고 있다.

안전한 보안설계의 예 : 클래스 설계서

2. 시퀀스도



3. 설계 클래스도



| 제 4 절 | 용어정리

AES(Advanced Encryption Standard)

미국 정부 표준으로 지정된 블록 암호 형식으로 이전의 DES를 대체하며, 미국 표준 기술 연구소(NIST)가 5년의 표준화 과정을 거쳐 2001년 11월 26일에 연방 정보처리표준(FIPS 197)으로 발표하였다.

CAPTCHA(Completely Automated Public Turning test to tell Computers and Humans Apart, 완전 자동화된 사람과 컴퓨터 판별)

HIP(Human Interaction Proof) 기술의 일종으로, 어떠한 사용자가 실제 사람인지 컴퓨터 프로그램인지를 구별하기 위해 사용되는 방법이다.

DES 알고리즘

DES(Data Encryption Standard)암호는 암호화 키와 복호화키가 같은 대칭키 암호로 64비트의 암호화키를 사용한다. 전수공격(Brute Force)공격에 취약하다.

FIPS

FIPS(Federal Information Processing Standards)는 미연방 정부 기관 및 정부 계약자를 위한 미국 연방 정부에서 개발하여 공개한 컴퓨터시스템 사용 표준이다.

FIPS 140-2

연방 정보 처리 표준(FIPS) 간행물 140-2 (FIPS PUB 140-2)는 암호화 모듈을 승인하는데 사용되는 미국 정부 컴퓨터 보안 표준으로 제목은 "암호화 모듈에 대한 보안 요구 사항"이다.

HMAC(Hash-based Message Authentication Code)

해시 기반 메시지 인증 코드, 메시지 다이제스트 알고리듬 5(MD-5), SHA-1 등 반복적인 암호화 해시 기능을 비밀 공용키와 함께 사용하며, 체크섬을 변경하는 것이 불가능하도록 한 키 기반의 메시지 인증 알고리즘이다.

HTTPS(Hypertext Transfer Protocol over Secure Socket Layer)

WWW(월드 와이드 웹) 통신 프로토콜인 HTTP의 보안이 강화된 버전이다.

LDAP(Lightweight Directory Access Protocol)

TCP/IP 위에서 디렉토리 서비스를 조회하고 수정하는 응용 프로토콜이다.

NIST

미국국립표준기술연구소(NIST, National Institute of Standards and Technology)는 국립표준국(NBS, National Bureau of Standards)이라고 알려진 측정 표준 실험실로 미국 상무부 산하의 비규제 기관이다.

PreparedStatement

컴파일된 쿼리 객체로 MySQL, Oracle, DB2, SQL Server 등에서 지원하며, Java의 JDBC, Perl의 DBI, PHP의 PDO, ASP의 ADO를 이용하여 사용 가능하다.

RC5

1994년 RSA Security사의 Ronald Rivest에 의해 고안된 블록 암호화 알고리즘이다.

SHA(Secure Hash Algorithm)

해시알고리즘의 일종으로 MD5의 취약성을 대신하여 사용한다. SHA, SHA-1, SHA-2(SHA-224, SHA-256, SHA-384, SHA-512) 등의 다양한 버전이 있으며, 암호 프로토콜인 TLS, SSL, PGP, SSH, IPSec 등에 사용된다.

Spring 프레임워크

개발 효율성과 생산성을 최대로 높일 수 있도록 지원하기 위해 개발된 J2EE기반의 오픈소스 개발 프레임워크이다.

Synchronized

JAVA에서 임계코드를 동기화하기 위해서 제공하는 구문이다.

Umask

파일 또는 디렉토리의 권한을 설정하기 위한 명령어이다.

개인키(Private Key)

공개키 기반구조에서 개인키란 암·복호화를 위해 비밀 메시지를 교환하는 당사자만이 알고 있는 키이다.

경로순회(directory traversal)

상대경로 참조 방식(“./”, “../”등)을 이용해 다른 디렉토리의 중요파일에 접근하는 공격방법으로 경로 추적이라고도 한다.

공개키(Public Key)

공개키는 지정된 인증기관에 의해 제공되는 키값으로서, 이 공개키로부터 생성된 개인키와 함께 결합되어, 메시지 및 전자서명의 암·복호화에 효과적으로 사용될 수 있다. 공개키를 사용하는 시스템을 공개키 기반구조(Public Key Infrastructure, PKI)라 한다.

동적 SQL(Dynamic SQL)

프로그램의 조건에 따라 SQL문이 다르게 생성되는 경우, 프로그램 실행시에 전체 쿼리문이 완성되어 DB에 요청하는 SQL문을 말한다.

동적쿼리(Dynamic Query)

컬럼이나 테이블명을 바꿔 SQL쿼리를 실시간 생성해 DB에 전달하여 처리하는 방식

샌드박스(Sandbox) 기법

어린이가 다치는 것을 방지하기 위해 만든 모래 통(Sandbox)에서 유래되었다. JAVA가 지원하는 기본 보안 SW로써, 외부에서 받은 프로그램을 JVM(Java Virtual Machine)이라는 보호된 영역 안에 가둔 뒤 작동시키는 방법으로 프로그램이 폭주하거나 악성 바이러스가 침투하는 경우를 대비한다.

서블릿(Servlet)

자바 서블릿은 자바를 사용하여 웹페이지를 동적으로 생성하는 서버 측 프로그램 혹은 그 사양을 말한다.

소프트웨어 개발보안

소프트웨어 개발과정에서 개발자 실수, 논리적 오류 등으로 인해 소프트웨어에 내재된 보안취약점을 최소화하는 한편, 해킹 등 보안위협에 대응할 수 있는 안전한 소프트웨어를 개발하기 위한 일련의 과정을 의미한다. 넓은 의미에서 소프트웨어 개발보안은 소프트웨어 생명주기의 각 단계별로 요구되는 보안활동을 모두 포함하며, 좁은 의미로는 SW개발과정에서 소스코드를 작성하는 구현 단계에서 보안취약점을 배제하기 위한 ‘시큐어코딩(Secure Coding)’을 의미한다.

소프트웨어 보안약점

소프트웨어 결함, 오류 등으로 해킹 등 사이버공격을 유발할 가능성이 있는 잠재적인 보안취약점을 말한다.

소프트웨어 보안약점 진단도구

개발과정에서 소스코드상의 소프트웨어 보안약점을 찾기 위하여 사용하는 도구를 말한다.

소프트웨어 보안약점 진단원

소프트웨어 보안약점이 남아있는지 진단하여 조치방안을 수립하고 조치결과 확인 등의 활동을 수행하는 자를 말한다.

스트러츠(Struts)

웹 개발을 단순화하는데 효과적인 오픈소스 프레임워크로 아파치 그룹에서 개발하여 제공하고 있다.

싱글톤 패턴(Singleton Pattern)

하나의 프로그램 내에서 하나의 인스턴스만을 생성해야만 하는 패턴이다. Connection Pool, Thread Pool과 같이 Pool 형태로 관리되는 클래스의 경우 프로그램 내에서 단하나의 인스턴트로 관리해야 하는 경우를 말함. java에서는 객체로 제공된다.

임계구역(Critical Section)

병렬컴퓨팅에서 둘 이상의 스레드가 동시에 접근해서는 안되는 공유자원(자료 구조 또는 장치)을 접근하는 코드부분을 말한다.

정적 SQL(Static SQL)

동적 SQL과 달리 프로그램 소스코드에 이미 쿼리문이 완성된 형태로 고정되어 있다.

해시함수

주어진 원문에서 고정된 길이의 의사난수를 생성하는 연산기법이며, 생성된 값은 '해시값'이라고 한다. MD5, SHA, SHA-1, SHA-256 등의 알고리즘이 있다.

화이트리스트(Whitelist)

블랙리스트(Black List)의 반대개념으로 신뢰할 수 있는 사이트나 IP주소 목록을 말한다.

소프트웨어 보안약점 진단가이드

인 쇄 2021년 11월

발 행 2021년 11월

발행처 **행정안전부**
세종특별자치시 정부2청사로 13
한국인터넷진흥원
전라남도 나주시 진흥길 9

〈비매품〉

※ 본 가이드 관련 최신본은 행정안전부 홈페이지(www.mois.go.kr), 한국인터넷진흥원 홈페이지(www. kisa.or.kr) 에서 얻으실 수 있습니다.