발간등록번호
11-1311000-000330-10

소프트웨어 개발보안 가이드

행정안전부
Ministry of the Interior and Safety

KISA 한국인터넷진흥원
KOREA INTERNET & SECURITY AGENCY

소프트웨어 개발보안 가이드

제·개정 이력(Revision History)

	제·개정일	변경 내용
1	2011. 6.	• 소프트웨어 개발보안 가이드 제정
2	2013. 1.	• 구현단계 보안약점 기준 변경에 따른 내용 추가
3	2017. 1.	• 설계단계 보안약점 기준 신설에 따른 내용 추가
4	2019.11.	• 관련법령 및 용어 수정, 코드예제 내용 추가
5	2021.11.	• 구현단계 보안약점 기준 확대에 따른 내용 추가 및 수정

CONTENTS

소프트웨어 개발보안 가이드

제1장 개요

제1장

개요

제1절 배경

공격자의 초점이 지속적으로 애플리케이션 계층을 향해 이동함에 따라 소프트웨어(SW, Software) 자원 보호가 중요해졌다. 최근 발생되는 사이버공격 시도의 약 75%는 소프트웨어의 보안 취약점을 악용하는 것으로 특히 외부에 공개되어 불특정 다수를 대상으로 사용자의 정보를 처리하는 웹 애플리케이션의 취약점으로 인해 중요한 개인정보가 유출되는 침해사고가 빈번하게 발생되고 있다. 보안강화를 위해 구축해놓은 침입차단시스템과 같은 보안장비로는 응용프로그램의 취약점에 대한 공격을 완벽히 방어하는 것은 불가능하다.

'SW개발보안'은 소프트웨어 개발과정에서 개발자의 실수, 논리적 오류 등으로 인해 발생될 수 있는 보안 취약점, 보안약점들을 최소화하여 사이버 보안 위협에 대응할 수 있는 안전한 소프트웨어를 개발하기 위한 일련의 보안 활동을 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 나타나 있다.

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.

제2절 가이드의 목적과 구성

소프트웨어 보안의 목표는 성공적인 사업을 운영하기 위한 정보 자원의 기밀성, 무결성, 가용성을 유지하는 것이다. 이러한 목표를 달성하기 위해서 보안통제 기능의 구현이 요구되며, 이 가이드에서는 소프트웨어의 취약점을 완화시킬 수 있는 소프트웨어의 각 개발 단계별 기술적 통제 항목에 중점을 두고 있다.

개발자와 공격자의 접근방식은 기본적으로 차이가 있다. 개발자는 애플리케이션의 정상적인 의도에 초점을 맞춰 접근하고, 공격자는 정상적인 의도 외에 허용되는 모든 동작하는 것에 관심을 가진다. 즉 위협을 최소화하는 방법은 소프트웨어를 개발하는 초기단계부터 보안요구사항을 정의하고, 설계단계에서 보안을 적용하여 공격자에게 허용되는 위협들을 최소화해야 하는 것이다.

소프트웨어 보안 취약점은

- 보안 요구사항이 정의되지 않았거나,
- 설계에 논리적인 오류가 있거나,
- 보안에 취약한 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 생명 주기에 고려되어야 하는 보안위협을 최소화하기 위해 각 단계별 수행해야 하는 보안 활동들을 정의하여 안전한 소프트웨어 개발에 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	• 소프트웨어(SW) 개발자·운영자
범위	• 신규로 개발되거나 유지보수로 변경되는 소프트웨어 개발 시 적용 ※ 행정기관 등이 추진하는 정보화사업 중 소프트웨어 개발 프로세스의 전체 단계(분석·설계·구현·테스트)에 초점
구성	• (1장) 가이드의 목적과 구성 • (2장) 소프트웨어 개발보안 의무화 대상, 범위, 기준과 정보화사업 단계별, 주체별 개발보안 활동 소개 • (3장) 분석·설계단계 소프트웨어 보안강화 활동 소개 • (4장) 구현단계 보안약점 제거 기준에 대한 설명 및 보안대책과 JAVA, C 및 C#언어로 작성된 안전하지 않은 코딩 예제 기반의 시큐어코딩 예시 소개 • (부록) 설계단계 보안설계 기준, 구현단계 보안약점 제거 기준, 설계단계 보안설계 적용 계획서 및 산출물 예시, 용어정리, 참고자료
활용	• (발주자) 안전한 소프트웨어를 구축하기 위해 분석·설계·구현·테스트의 각 단계별로 보안 강화 정책이 적용될 수 있도록 요구사항 도출 시 활용 • (사업자) SW개발보안을 적용하여 소프트웨어 개발 시 활용 • (감리법인) 설계단계의 보안설계 적용 확인, 구현단계의 소스코드에 대한 보안약점 진단 및 제거, 테스트단계의 SW취약점 진단 및 조치 확인 시 활용

제2장 소프트웨어 개발보안

제2장

소프트웨어 개발보안

제1절 개요

1. 소프트웨어 개발보안 필요성

SW개발보안은 해킹 등 사이버공격의 원인인 보안약점을 소프트웨어(SW, Software) 개발 단계에서 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, 시큐어코딩을 위한 표준이 확립되어야 한다. 재사용 가능한 보안 라이브러리를 작성하여 비슷한 기능을 수행하는 프로젝트에 도입함으로써 일반적인 안전성을 확보할 수 있도록 해야 하며, 보안통제의 효과성 점검으로 향후 새로운 프로젝트에서 효과적인 보안정책이 적용될 수 있도록 해야 한다.

2021년 1월에 개정·고시된 '행정기관 및 공공기관 정보시스템 구축·운영 지침(행정안전부고시 제2021-3호)'은 개발보안과 관련하여 행정기관 및 공공기관이 정보화사업을 추진할 경우 준수해야 할 소프트웨어 개발보안 활동 및 보안약점 기준 등이 정의되어 있다. 지침에 나타난 소프트웨어 개발보안 적용 대상 및 범위·기준 등을 요약하면 다음과 같다.

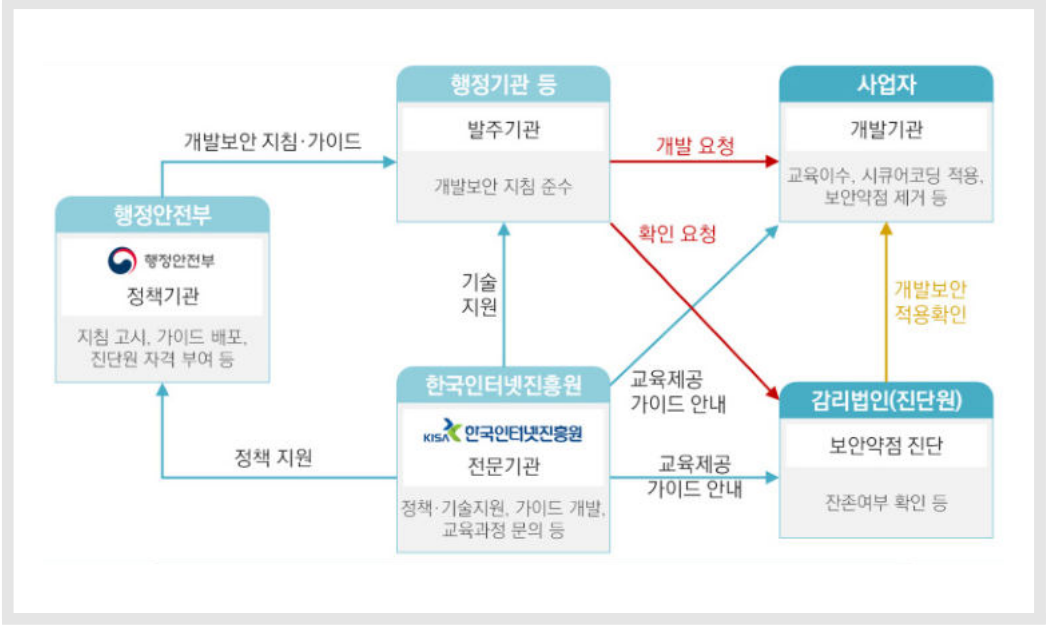
표 2-1 소프트웨어 개발보안 대상 및 진단기준

구분	내용	비고
대상	• 정보시스템 감리대상 사업 ※ 감리대상사 사업도 자체적으로 소프트웨어 개발보안 적용	'전자정부법 시행령 제71조제1항' 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
보안약점 진단기준	• 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·운영지침 제52조, 제53조' 참조
기타	• 정보화사업 발주 시, 제안요청서에 명시 ※ 소프트웨어 보안약점 진단도구 사용여부, 개발절차와 방법의 적절성, 소프트웨어 개발보안 관련 교육계획의 적정성 등을 확인하고 평가에 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

제2절 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]와 같이 요약할 수 있다.

표 2-2 활동 주체별 개발보안 역할

구분	내용	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	-
발주기관	• 소프트웨어 개발보안 계획 수립 • 제안요청서에 '소프트웨어 개발보안 적용' 명시 • 소프트웨어 개발보안 역량을 갖춘 사업자 선정 • 사업자의 소프트웨어 개발보안 준수여부 점검 • 감리대상 사업의 경우 감리법인의 소프트웨어 보안약점 제거 여부 진단 ※ CC인증 받은 진단도구 사용, 진단절차·방법의 적절성 평가 등 • 사업종료 결과물 및 증빙서류 등 최종 확인 ※ 감리법인에 의뢰(감리대상) 또는 자체적으로 수행(감리대상 外)	행정기관 및 공공기관
사업자(개발자)	• 소프트웨어 개발보안 관련 기술수준 및 적용계획 명시 • 소프트웨어 개발보안 가이드를 참조하여 관련 교육 실시 후 개발에 투입 • 자체적으로 SW보안약점 진단 및 제거 • SW보안약점 관련 시정요구 이행 • SW보안약점이 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 • SW보안약점 제거여부 진단 및 조치결과 확인 ※ 진단도구 사용 시, CC인증(국정원) 받은 도구 사용 • 감리법인은 SW보안약점 진단 시, 진단원을 우선적으로 배치 권고	진단원 활용

제3절 소프트웨어 개발보안 방법론

1. 소프트웨어 개발 생명주기 이해

소프트웨어 개발 생명주기(SDLC)는 소프트웨어의 생성에서 소멸까지의 과정을 단계별로 나눈 것으로, 프로젝트의 진행관리를 위해 각 단계별 주요활동과 산출물로 프로젝트 진행 방향을 명확하게 파악하고, 관리를 용이하게 한다.

소프트웨어 개발 생명주기는 개발방법론에 따라 단계가 달라지지만 일반적으로 정의단계, 개발단계, 유지보수단계로 나눌 수 있다.

정의단계는 무엇(what)을 처리하는 소프트웨어를 개발할 것인지 정의하는 단계로 타당성 검토단계, 개발계획단계, 요구사항 분석단계로 나눌 수 있다.

개발단계는 어떻게(How)에 초점을 두고 실제로 소프트웨어를 개발하는 단계이다. 설계단계, 구현단계, 테스트단계로 나눌 수 있다.

유지보수단계는 소프트웨어를 직접 운영하며, 변경(Change)에 초점을 두고 여러 환경변화에 따라 소프트웨어를 적응 및 유지시키는 단계이다.

다양한 소프트웨어 개발 생명주기 모델을 소프트웨어 개발에 적용할 수 있으며, 다음은 대표적인 개발생명주기 모델들을 나열하였다.

① 개발수정 모델(Build-Fix Model)

요구사항, 분석/설계 단계 없이 일단 개발에 들어간 후 만족할 때까지 수정작업을 수행하는 방식

② 폭포수 모델(Waterfall Model)

순차적으로 소프트웨어를 개발하는 전형적인 개발모델로 소프트웨어 개발의 전 과정을 나누어 체계적이고 순차적으로 접근하는 방식.

③ 프로토타입 모델(Prototyping Model)

점진적으로 시스템을 개발해 나가는 접근 방식으로 원형(prototype)을 만들어 고객과 사용자가 함께 평가한 후 개발될 소프트웨어의 요구사항을 정제하여 보다 완전한 요구 명세서를 완성하는 방식

④ 나선형 모델(Spiral Model)

포수 모형과 원형모형의 장점을 수용하고 위험분석을 추가한 점증적 개발모델로 프로젝트 수행 시 발생하는 위험을 관리하고 최소화하기 위한 방식

2. 소프트웨어 개발 방법론 이해

소프트웨어 개발 방법론은 소프트웨어 공학 원리를 소프트웨어 개발 생명주기에 적용한 개념으로 정보시스템 개발을 위한 작업활동, 절차, 산출물, 기법 등을 체계적으로 정리한 것을 말한다.

1) 구조적 방법론

정형화된 분석 절차에 따라 사용자 요구사항을 파악하여 문서화하는 체계적인 방법으로 비즈니스 프로세스 자동화를 목표로 하고 있으며, 프로세스 중심의 개발 방법이다.

특징		• 데이터 흐름 지향, 즉 프로세스 위주의 분석과 설계 방식 • 모듈의 분할과 정복에 의한 하향식 설계 방식 • SDLC의 구조를 가진 폭포수 모델이 기본 • 소프트웨어의 개발이 목표인 프로세스와 산출물의 구성 • 데이터의 구성에 대한 설계 방안이 부족 • 프로젝트 관리 및 조직, 역할 등 방법론적 다른 요소들의 정의가 없음
라이프 사이클		• 폭포수 모델
개발방식		• TOP-DOWN
단계별 산출물 활용	계획	• 도메인 분석, 프로젝트 계획서 등
	분석	• DFD(Data Flow Diagram) 등
	설계	• Structure Chart, 프로그램 사양서 등

2) 객체지향 방법론

분석, 설계 및 개발에 있어서 객체지향 기법을 활용하여 시스템을 구축하는 방법이다.

특징		• 반복적, 그리고 점증적인(Iterative and Incremental) 개발 방식 • 재사용성의 강조 • 쉽고 표준화된 표기법 존재 • 분산객체 기술의 완벽한 지원 • DBMS의 원활한 연계 등 아직 개선의 여지가 많음
라이프 사이클		• 반복적 개발
개발방식		• BOTTOM-UP
단계별 산출물 활용	계획	• 비즈니스 프로세스, 개념 모델, 프로젝트 계획서 등
	분석	• 유즈케이스 다이어그램, 시퀀스 다이어그램, 클래스 다이어그램 등
	설계	• 시퀀스 다이어그램, 클래스 다이어그램, 컴포넌트 다이어그램, 배포 다이어그램 등

3) CBD 방법론

재사용이 가능한 컴포넌트의 개발 또는 사용 컴포넌트를 조합하여 애플리케이션 개발 생산성과 품질을 높이고, 시스템 유지보수 비용을 최소화할 수 있는 방법이다.

특징		• 컴포넌트 기반 개발 • 반복 점진적 개발프로세스 제공 • 표준화된 산출물 작성, 컴포넌트 제작 기법으로 재사용성 향상
라이프 사이클		• 반복적 개발
개발방식		• BOTTOM-UP
단계별 산출물 활용	계획	• 비즈니스 프로세스, 개념 모델, 프로젝트 계획서 등
	분석	• 유즈케이스 다이어그램, 유즈케이스 명세서, 요구사항 추적표 등
	설계	• 사용자 인터페이스 설계서, 컴포넌트 설계서, ERD 기술서, 데이터베이스 설계서 등

[표 2-3]은 한국지능정보사회진흥원에서 배포한 CBD SW개발 표준 산출물 관리가이드에서 제시하고 있는 각 단계별 산출물 목록이다.

표 2-3 산출물 목록

단계	코드	산출물	구분
분석	R1	사용자 요구사항 정의서	필수
	R2	유스케이스 명세서	
	R3	요구사항 추적표	필수
설계	D1	클래스 설계서	
	D2	사용자 인터페이스 설계서(화면설계서)	필수
	D3	컴포넌트 설계서	
	D4	인터페이스 설계서	
	D5	아키텍처 설계서	필수
	D6	총괄시험 계획서	
	D7	시스템시험 시나리오	
	D8	엔티티 관계 모형 기술서 (ERD)	필수
	D9	데이터베이스 설계서(테이블설계서)	필수
	D10	통합시험 시나리오	
	D11	단위시험 케이스	
	D12	데이터 전환 및 초기데이터 설계서	
구현	I1	프로그램 코드	
	I2	단위시험 결과서	
	I3	데이터베이스 테이블	
시험	T1	통합시험 결과서	필수
	T2	시스템시험 결과서	
	T3	사용자 지침서	
	T4	운영자 지침서	
	T5	시스템 설치 결과서	
	T6	인수시험 시나리오	
	T7	인수시험 결과서	

[표2-4]는 분석단계의 산출물인 요구사항 추적표 샘플이다. 기본적으로 프로젝트를 수행하면서 최소한으로 요구되는 산출물들의 연관관계를 추적할 수 있도록 하기 위한 표이며, 개발 프로젝트 수행 시 보안 요구항목 적용을 검토할 수 있는 문서로 활용할 수 있다.

표 2-4 요구사항 추적표

R3	요구사항 추적표				
시스템명		서브시스템명			
단계명		작성일자		버전	

분석 단계				설계 단계			구현 단계		시험 단계	
사용자 요구사항 명세서		유스케이스 명세서		사용자 인터페이스 설계서	컴포넌트 설계서	데이터베이스 설계서	프로그램 코드	단위시험 결과서	통합시험 결과서	시스템 시험 결과서
요구사항 ID	요구 사항명	유스 케이스 ID	유스 케이스명	화면ID	컴포넌트 ID	테이블 ID	파일 ID	단위시험 ID	시나리오 ID	시나리오 ID

3. 소프트웨어 개발보안 방법론 이해

이러한 기존의 개발방법론이 적용된 프로젝트에서 안전한 소프트웨어를 만들기 위해 요구되는 보안 활동들을 적용하는 개발방법을 SW개발보안 방법론이라 한다.

안전한 소프트웨어는 보안관련 기능을 수행하는 소프트웨어가 아니라, 신뢰성이 위협받는 상황에서도 시스템을 신뢰할 수 있는 상태로 유지할 수 있도록 만들어진 소프트웨어이다.

SW개발보안 방법론에서는 안전한 SW개발을 위해 SW개발 생명주기(SDLC: Software Development Life Cycle)에 걸쳐 다음과 같은 보안활동을 추가하고 있다.

1) 요구사항분석 단계

기본활동: 사용자의 문제를 구체적으로 이해하고 소프트웨어가 담당해야 하는 정보영역을 정의한다. 사용자의 기능, 성능, 신뢰도 등에 대한 요구는 요구사항 정의서(Requirements Specifications)로 문서화한다. 요구사항 정의서는 다시 세분화하여 업무, 기술, 성능, 운영 요구사항정의서로 분석할 수 도 있다. 이 단계에서는 요구사항정의서 이외에도 기능차트, 프로세스정의서, 인터페이스 정의서와 같은 산출물이 생성될 수 있다.

추가적인 보안활동: 요구사항 중 보안항목 요구사항을 식별하는 보안 활동을 추가한다. 어떤 정보들이 시스템화되어 관리되어야 하고, 이때 이 정보들은 얼마만큼의 보안등급(기밀성, 무결성, 가용성)을 가져야 하며, 법률적으로 관리의 중요성이 어느 정도 강조되는지에 대한 점검 작업이 요구된다.

2) 설계단계

기본활동: 소프트웨어의 구조와 그 성분을 명확하게 밝혀 구현을 준비하는 단계이다. 외부 시스템 및 사용자와의 인터페이스를 중시하는 외부설계와 시스템 내부를 설계하는 내부 설계로 분류되기도 하고 전체적 구조와 데이터 알고리즘을 설계하는 단계를 분리해 기본설계와 상세설계로 분류되기도 한다. 설계단계는 설계사양서(Design Specification)를 산출물로 만들어 내며, 이 산출물과 요구사양서를 토대로 사용자지침서와 시험계획서를 작성한다. 이 단계에서는 화면설계서, 데이터베이스 ERD(Entity Relationship Diagram), 테이블 목록, 테이블 정의서, 프로그램 목록, 개발표준정의서, 단위테스트 시나리오, 통합테스트 시나리오와 같은 산출물이 생성될 수 있다.

추가적인 보안활동: 시스템을 분석해 위협들을 도출해내는 위협 모델링, 보안통제 기준 설정과 같이 개발보안 가이드가 제시하는 작업을 기존 개발 프로세스에 추가해 진행한다. 특히 설계단계에서

수행되는 위협 모델링 작업으로 최대한 많은 위협들을 도출해 해당 위협들이 충분히 제거될 수 있도록 시스템이 설계되어야 한다.

3) 구현단계

기본활동: 프로그래밍을 하는 단계이다. 각 모듈의 코딩과 디버깅이 이루어지고 그 결과를 검증하는 단위테스트 또는 모듈테스트를 수행한다. 이 단계의 산출물로는 소스코드, 단위테스트 결과서, 결함/오류보고서, 오류코드 정의서와 같은 산출물이 생성될 수 있다.

추가적인 보안활동: 표준코딩정의서 또는 소프트웨어 개발보안가이드를 모든 개발자들이 준수하여 개발하는 것이 중요하다. 구현단계에서 단위테스트로 소프트웨어가 가질 수 있는 보안 취약점을 충분히 제거할 수 있도록 해야 하며, 코드 리뷰 또는 소스코드 진단 작업으로 소스 코드 수준의 안정성이 보장되도록 하여야 한다.

4) 테스트단계

기본활동: 개발된 모듈들을 통합시키며 시험하는 통합테스트, 완성된 시스템으로서 요구사항을 완벽하게 구현했는지를 확인하기 위한 시스템테스트, 그리고 사용자가 직접 자신의 사용 현장에서 검증해 보는 인수테스트 등을 수행한다. 이 단계의 산출물로는 통합테스트 결과서, 시스템 이행 계획서와 같은 산출물이 생성될 수 있다.

추가적인 보안활동: 설계단계에서 수행된 위협모델링으로 도출된 위협들이 구현단계에서 해당 취약점들이 없는 애플리케이션으로 개발되었는지를 동적 분석 도구를 이용하거나 모의 침투 테스트로 검증하는 작업을 수행하여 소프트웨어의 안전성이 보장되도록 하여야 한다.

5) 유지보수 단계

기본활동: 소프트웨어 이용 중에 나타나는 문제점을 수정하거나 새로운 기능을 추가하여 보다 안정적인 소프트웨어로 발전시키기 위한 작업을 수행한다.

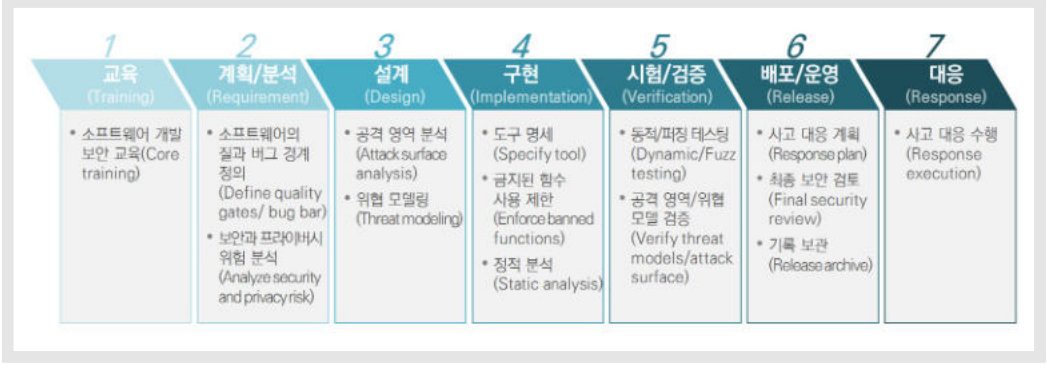
추가적인 보안활동: 각 개발 단계에서 안전한 소프트웨어를 만들기 위해 노력하였음에도 불구하고 발생될 수 있는 보안 사고에 대한 관리 및 사고 대응, 패치 관리 및 교육이 병행되어야 한다.

4. 소프트웨어 개발보안 적용 사례

SW 개발보안의 대표적인 모범사례로는 MS-SDL, Seven Touchpoints, CLASP 등이 있다.

1) MS-SDL(Microsoft - Secure Development Lifecycle)

마이크로소프트사는 보안수준이 높은 안전한 소프트웨어를 개발하기 위해 수행한 프로세스 개선 작업으로 자체수립한 SDL(Secure Development Lifecycle) 방법론을 적용하였으며, SDL이 적용된 소프트웨어는 이전 버전에 비해 50% 이상 취약점이 감소하였다고 발표했다.



교육 단계: 소프트웨어 개발팀의 구성원들이 매년 한 번씩 보안의 기초와 최신 보안 동향에 대한 정보를 교육받을 수 있도록 한다. 보안 교육은 안전설계, 위협모델링, 시큐어코딩, 보안테스팅, 프라이버시에 관한 내용을 포함한다.

계획·분석 단계: 안전한 소프트웨어를 구축하기 위한 기본 보안 요구사항과 프라이버시 요구사항을 정의한다. 이 단계에서는 'SDL 방법론 적용 여부 결정, 보안책임자(Security Advisor) 선정, 보안팀(Security Champion) 선정, 버그 리포팅 도구 정의, 보안 버그 경계(security bug bar) 정의, 보안 위험 평가'와 같은 필수 보안활동과 '보안 계획서 작성, 버그 추적 시스템 정의'와 같은 권장보안 활동을 수행한다.

설계 단계: 구현에서 배포에 이르기까지 수행해야 하는 작업 계획을 수립하는 단계이다. '보안 설계검토, 방화벽 정책 준수, 위협모델링, 위협모델 품질 보증, 위협모델 검토 및 승인'과 같은 필수 보안활동과 "보안 설계서" 문서 작성, 보안 디폴트 인스톨 실행, 모든 샘플 소스코드의 보안검토 수행, 안전하지 않은 함수와 코딩 패턴 알림, 설계변화 요구에 관한 보안관련 사항 문서화, 위협모델로 발견한 취약성 해결을 위한 작업 목록 작성'과 같은 권장보안 활동을 수행한다.

구현 단계: 보안 및 프라이버시 문제점을 발견하고 제거하기 위해 개발 시 최선의 방책을 수립하고 따르도록 한다. '최신 버전의 빌드 도구 사용, 금지된 API 사용 회피, Execute 허가하여 안전한 SQL 사용, 저장 프로시저에서 SQL 사용'과 같은 필수 보안 활동과, '안전하게 소프트웨어를 사용하기 위해 필요한 사용자 정보 식별, 사용자 중심의 보안 문서 계획, 보안 형상관리에 관한 정보 생성, 자동화된 금지 API 변환 실행, 프로젝트 팀 전체와 모든 모범 사례와 정책에 대한 정의, 문서화, 토론 등'과 같은 권장 보안 활동을 수행한다.

시험·검증 단계: 보안 및 프라이버시 테스팅과 보안 푸쉬(security push), 문서 리뷰로, 코드가 이전 단계에서 설정한 보안과 프라이버시 방책을 지키는지 확인한다. 보안 푸쉬는 팀 전체에 걸쳐 위협모델 갱신, 코드 리뷰, 테스팅에 초점을 맞춘 작업이다. '커널-모드 드라이버를 위한 테스팅 완료, COM 객체 테스팅 수행, 인증된 사이트 크로스 도메인 스크립팅을 위한 테스팅, 애플리케이션 검증테스트 수행, 파일 fuzzing 수행, 위협모델 검토 및 수정 등' 필수 보안 활동과 '보안 테스팅 계획 완료, 침투테스트 수행, 시큐어 코드 검토, 보안 푸쉬를 시작하기전 모든 코드에 대한 우선순위 결정, 보안문서 계획서 검토 등'과 같은 권장 보안 활동을 수행한다.

배포·운영 단계: 사고 대응 계획을 준비하는 것은 시간이 지남에 따라 나타날 수 있는 새로운 위협요소를 해결하는 데 중요하다. 관련 담당자의 긴급 연락처 식별 및 조직 내 다른 그룹이나 타사에서 개발된 소프트웨어에 대한 보안서비스 계획 수립 등이 포함되어야 한다. 수행 된 모든 보안활동에 대한 검토로 소프트웨어 배포 준비 상태를 보장할 수 있으며, 배포 전에 소프트웨어 인증으로 보안 및 개인 정보 보호 요구 사항을 충족시킬 수 있다.

모든 관련 데이터를 보관하여 배포 이후 작업을 수행하는데 도움이 되도록 하며 장기적으로 비용을 낮추는 데도 도움이 되도록 한다.

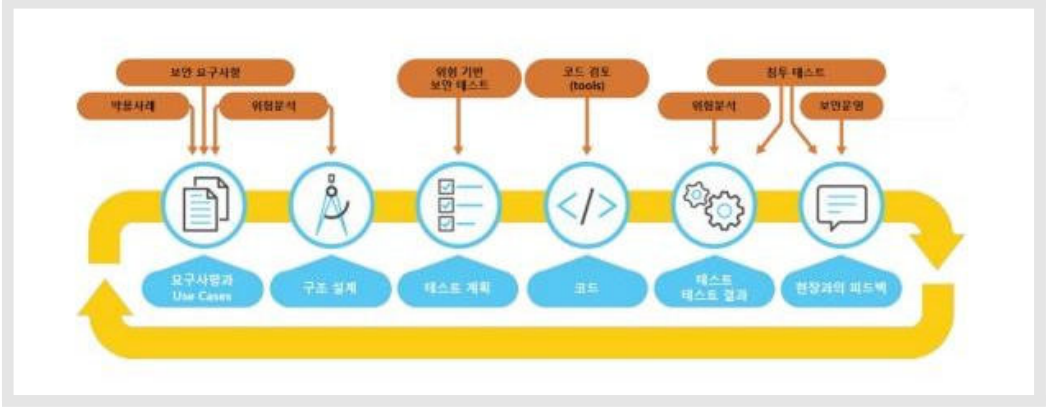
대응 단계: 배포 단계에서 만들어진 사고 대응 계획을 구현할 수 있다는 것은 소프트웨어 보안 또는 개인 정보 보호 취약성으로부터 고객을 보호하는 데 필수적이다.

2) Seven Touchpoints

Seven Touchpoints는 실무적으로 검증된 개발보안 방법론 중 하나로써 소프트웨어 보안의 모범 사례를 소프트웨어 개발 라이프사이클에 통합하였다. 공통 위험 요소를 파악하고 이해하며, 보안을 설계하고 모든 소프트웨어 산출물에 대해 철저하고 객관적인 위험 분석 및 테스트를 거쳐 안전한 소프트웨어를 만들어는 내는 방법을 정의하고 있다. 이러한 보안 활동은 수행하는 동안 소프트웨어 위험 요소에 대한 위험 관리 프레임워크에 따라 명시적으로 추적하고 모니터링을 수행한다.

Seven Touchpoints는 SDLC(Software Development Life Cycle)의 각 단계에 관련된 아래 그림과 같이 7개의 보안강화활동을 개발자에게 집중적으로 관리하도록 요구한다.

Software Security Touchpoints



Seven Touchpoints는 SDLC내의 개발단계와 이와 관련된 7개의 보안강화활동을 아래와 같이 정의한다.

요구사항과 Use Cases 단계: 오용사례와 위험분석으로 설계보안항목에 대한 정의와 명세를 작성하고, 오용사례에 대한 정의 및 케이스 예시를 작성한다.

구조설계 단계: 공격저항 분석(attack resistance analysis), 모호성 분석, 허점 분석 등으로 위험요소를 분석한다.

테스트 계획 단계: 공격 패턴, 위험 분석 결과, 악용 사례를 기반으로 위험기반 보안테스트를 수행한다.

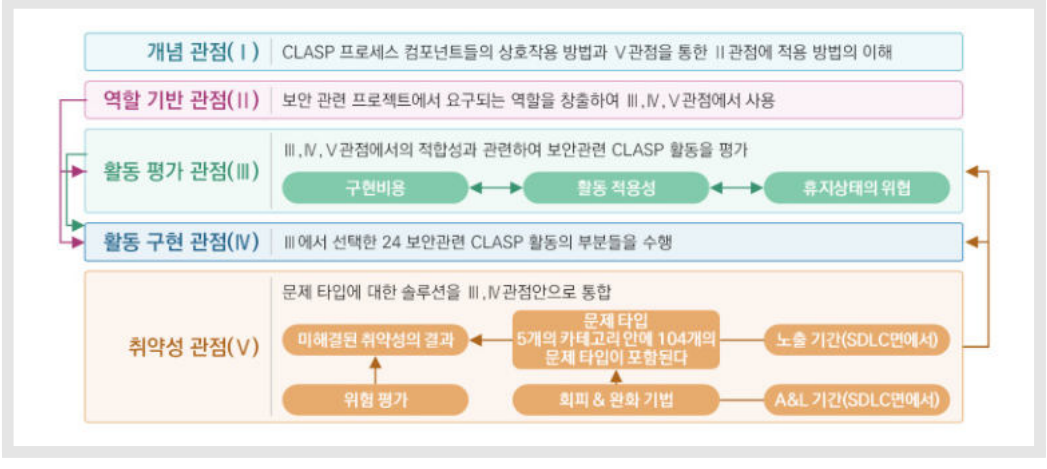
코드 단계: 구현 오류(implementation bug)에 중점을 두며 특히 소스코드에 존재하는 취약성을 발견할 목적으로 수행되는 코드 정적분석에 중심을 둔다.

테스트 및 테스트결과 단계: 위험 분석 및 침투 테스트를 수행한다. 침투테스트로 실제 작동 환경에서의 필드 소프트웨어에 대한 좋은 이해를 제공한다.

현장과의 피드백 단계: 보안 운영으로 얻은 공격자와 공격 도구에 대한 경험과 지식은 개발자에게 다시 피드백한다.

3) CLASP(Comprehensive, Lightweight Application Security Process)

SW개발보안 방법론 중 하나인 CLASP은 SDLC 초기단계에 보안 강화를 목적으로 하는 정형화된 프로세스로써, 활동중심·역할기반의 프로세스로 구성된 집합체이다. 이미 운영 중인 시스템에 적용하기 좋다.



CLASP은 안전한 소프트웨어를 개발하기 위해 5가지 관점에 따라 개발보안 프로세스를 수행할 것을 제안한다.

개념 관점(Concepts View): CLASP의 구조와 CLASP 프로세스 구성요소 간의 종속성의 개요를 제공하고, CLASP 프로세스 컴포넌트들의 상호작용 방법과 취약성 관점으로써, 어떻게 역할기반 관점에 적용하는지를 기술한다.

역할기반 관점(Role-Based View): 프로젝트에서 24개의 보안관련 CLASP 활동들에 대한 각 역할을 창출하고, 팀의 구성원이 각자 맡게 될 역할들을 정의하여 활동평가 관점, 활동구현 관점, 취약성 관점에서 사용한다.

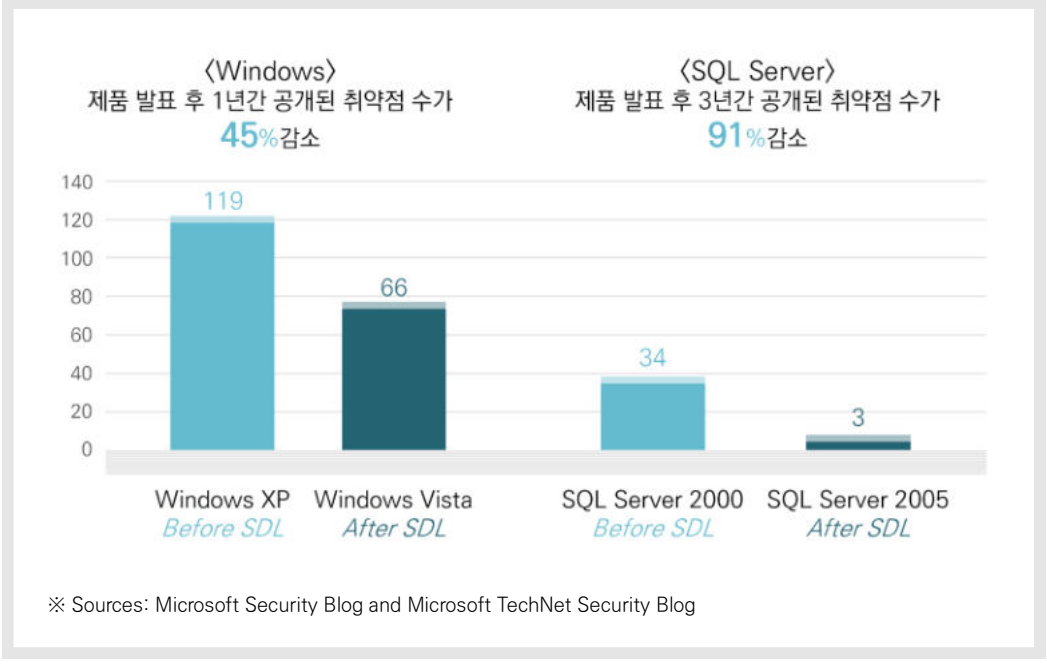
활동평가 관점(Activity-Assessment View): 활동구현 관점에서의 적합성과 관련하여 보안관련 CLASP 활동들에 대해 타당성을 평가할 수 있도록 도와줌으로써, 프로젝트 매니저와 프로세스 엔지니어링팀의 부담을 덜어준다.

활동구현 관점(Acitivity-Implementation View): 활동평가 관점에서 선택한 24가지 보안관련 CLASP 활동들을 수행한다.

취약성 관점(Vulnerability View): 문제 타입에 대한 솔루션을 활동평가 관점과 활동구현 관점으로 통합한다.

5. 소프트웨어 개발보안 적용 효과

MS(MicroSoft사)에서는 소프트웨어 개발보안 방법론을 적용하여 프로젝트를 수행한 경우, 취약점이 대폭 감소하여 보안패치와 같은 유지보수 비용이 확연하게 감소하였다고 발표하였다.



※ Sources: Microsoft Security Blog and Microsoft TechNet Security Blog

NIST에서 제품출시단계에 발견되는 결함을 제거하기 위해서는 30배의 비용이 요구된다는 자료를 발표하였다. 이러한 결함을 초기단계에서부터 제거하게 된다면 유지보수비용의 30배를 절감할 수 있다는 것이다.

구분	설계단계	코딩단계	통합단계	베타제품	제품출시
설계과정 결함	1배	5배	10배	15배	30배
코딩과정 결함		1배	10배	20배	30배
통합과정 결함			1배	10배	20배

제4절 프로젝트 참여 직무별 보안활동

프로젝트에 참여하는 구성원들이 각각의 직무별 보안활동을 정의하여 프로젝트가 수행되는 동안 책임감을 가지고 보안활동을 수행하도록 하는 것도 안전한 소프트웨어를 개발하기 위한 방법 중 하나가 될 수 있다. 다음은 CLASP의 SW개발보안 방법론에서 설명하고 있는 프로젝트 수행기간 동안 프로젝트 참여 직무별로 수행할 보안 활동에 대한 정의이다.

1. 역할(Role)별 보안활동

1) 프로젝트 관리자(Project Manager)

- 프로젝트 관리자는 팀 구성원에게 응용프로그램의 보안 전략을 알려야 한다.
- 보안위험과 비즈니스에 응용 프로그램 보안의 영향을 이해시킨다. 예를 들어 프로젝트 일정과 보안위험의 상관관계 등을 이해시킨다.
- 조직의 상태를 모니터링 한다. 기본적인 비즈니스 매트릭스 조합을 정의하고 단계별 적용한다.

2) 요구사항 분석가(Requirement Specifier)

- 요구사항 분석가는 아키텍트가 고려해야 할 여러 가지 보안관련 비즈니스 요구사항들을 자세히 설명할 수 있어야 한다.
- 프로젝트 팀이 고려해야 할 구조를 정의한 뒤, 해당 구조에 존재하는 자원에 대한 보안 요구사항이 무엇인지 결정한다.
- 보안수준을 추상화할 때 다른 프로젝트에 적용되었던 보안 요구사항을 재사용하여 시간을 절약할 수 있어야 한다.
- 유즈케이스에 대한 보안 고려사항을 기반으로 오용사례를 정의할 수 있어야 한다.

3) 아키텍트(Architect)

- 아키텍트는 명백한 보안 오류를 도입하지 않도록 충분히 보안 기술의 문제를 이해 할 수 있어야 한다.
- 시스템에 사용되는 모든 리소스를 가능한 자세하게 정의한다.
- 시스템에서 각각 리소스의 역할에 적절한 보안 요구사항이 적용되도록 한다.
- 각 리소스가 시스템 라이프 사이클로 서로 간의 상호작용을 이해할 수 있게 해야 한다.

4) 설계자(Designer)

- 설계자는 특정기술이 설계보안항목을 만족하는지 확인하고 제대로 그 기술이 사용될 수 있는 방법을 파악해야 한다.
- 일반적으로 결과를 평가하고 최선의 문제해결 방법을 결정해야 한다.
- 애플리케이션 보안 노력에 대한 품질 측정을 지원해야 한다. 즉, 설계자는 모든 기존 개발 역할의 보안 관련 작업을 수행할 수 있어야 한다.
- 식별되지 않은 보안 위험을 가지고 있는 경우 요구사항 분석단계를 다시 추진해야 한다.
- 고가의 수정을 요구하는 위험을 최소화 하기 위해 로드맵을 제공해야 한다.
- 타사의 소프트웨어 통합 시 보안 위험을 이해하고 있어야 한다.
- 일반적으로 소프트웨어에서 식별된 보안 위협에 대응할 수 있어야 한다.

5) 개발자(Implementer)

- 코드를 구현하는 개발자는 고도로 구조화된 개발 환경에서 프로그램을 구현하기 위해 안전한 코딩표준을 준수하여 개발하여야 한다.
- 제 3자가 소프트웨어 안전 여부를 쉽게 판단할 수 있도록 문서화해야 한다.

6) 테스트 분석가(Test Analyst)

- 테스트 분석가는 요구사항과 구현결과를 반복적으로 테스트 해야 한다. 테스트 그룹은 반드시 보안 전문가일 필요는 없으며, 테스트가 가능할 정도의 위험에 대한 학습이나 툴 사용방법을 숙지하고 있으면 된다.

7) 보안감사자(Security Auditor)

- 보안 감사자는 프로젝트의 현재 상태를 검사하고 현재 상태의 보안을 보장한다.
- 요구사항을 검토할 때는 요구사항이 적합하고 완전한지 확인한다.
- 설계단계에서는 일반적으로 취약성으로 이어질 수 있는 사항이 있는지 점검한다.
- 구현단계에서는 보안 문제를 발견할 수 있도록 시도해야 한다.
- 보안감사자는 프로젝트의 전체 단계에서 활동하여야 한다.

제3장 분석·설계단계 보안강화 활동

제3장

분석·설계단계 보안강화 활동

제1절 분석·설계단계 보안강화 활동

1. 분석·설계단계 개발보안 필요성

분석·설계단계에서는 기능 및 비기능 요구사항을 충족시키기 위한 소프트웨어의 구조와 그 성분을 명확하게 밝혀 구현을 준비하는 단계이다. 분석·설계단계에서 사전에 보안항목을 반영하지 않으면 이후 구현단계에서 소프트웨어의 일관성이 떨어지거나, 단순한 수정만으로 보안항목을 만족시킬 수 없는 경우가 발생할 수 있다. 설계에서 반영하지 못한 보안항목을 다시 반영하기 위해서는 구현단계에서는 5배, 제품 출시 이후에는 30배까지 추가 비용이 들 수 있기 때문에 사전 설계 단계에서 반영하는 것이 중요하다.

2. 정보에 대한 보안항목 식별

분석단계에서는 처리대상 정보와 이를 처리하는 기능에 대해 적용되어야 하는 보안항목들을 식별하는 작업이 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 시스템에서 처리될 때 철저하게 보호되어야 하는 중요정보들을 정의하고 있으며 각 기관은 내부정책자료, 외부정책자료 등을 기반으로 보안항목을 식별하고 있다. 내부 정책자료로는 기관 내 개인정보 보호 규정, 정보보안 관련 규정 등이 사용되고 있으며, 외부정책 자료로는 개인정보 보호법, 정보통신망법, 금융거래법 등 다양한 법, 법령, 관련 지침 등이 사용되고 있다.

가. 외부환경 분석(법, 제도, 규정 등)으로 보안항목 식별

현재 시스템에서 처리하는 정보 중 중요정보로 분류되어 있는 정보의 대부분은 개인정보이며, [표3-1]은 개인정보보호에 관련 법규들이다.

표 3-1 개인정보보호 관련 법규

관련법규	주요내용
개인정보 보호법	• 개인정보의 처리 및 보호에 관한 사항을 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	• 「개인정보보호법」 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 IT 기술관련 규정

관련 지침	주요내용
개인정보의 기술적·관리적 보호조치 기준 해설서	• 「개인정보 보호법」 제29조와 같은 법 시행령 제48조의2제3항에 따라 정보통신서비스 제공자 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 정보들 중 법적 의무사항으로 필수적인 안전조치를 해야 하는 정보들 이외에도 물리적, 기술적 접근 허용범위와 정보유출 시 예상되는 피해를 기준으로 보안등급을 결정하거나, 정보의 자산가치를 기준으로 중요정보를 식별하고 법적 의무사항에 준하는 보안 강도를 적용하여 정보가 처리될 수 있도록 설계되어야 한다.

3. 설계단계 보안설계 기준

분석 단계에서는 정보처리시스템의 각 기능들을 안전하게 서비스하기 위해 필요한 보안 항목들을 식별하여, 산출물인 요구사항 정의서에 다음과 같은 설계항목을 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 처리방법 설계

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

번호	설계항목	설명	비고
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	입출력 검증
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	
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	
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	
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	
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	
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	
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	
9	보안기능 입력값 검증	• 보안기능(인증, 권한부여 등) 입력 값과 함수(또는 메소드)의 외부입력값 및 수 행결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법 설계	
10	업로드·다운로드 파일 검증	• 업로드·다운로드 파일의 무결성, 실행권한 등에 관한 유효성 검증방법 설계 및 부적합한 파일에 대한 처리방법 설계	파일 검증

나. 보안기능

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

번호	설계항목	설명	비고
1	인증 대상 및 방식	• 중요정보·기능의 특성에 따라 인증방식을 정의하고 정의된 인증방식을 우회하지 못하게 설계	인증 관리
2	인증 수행 제한	• 반복된 인증 시도를 제한하고 인증 실패한 이력을 추적하도록 설계	
3	비밀번호 관리	• 생성규칙, 저장방법, 변경주기 등 비밀번호 관리정책별 안전한 적용 방법 설계	
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 설계	접근 권한 관리
5	암호키 관리	• 암호키 생성, 분배, 접근, 파기 등 암호키 생명주기별 암호키 관리 방법을 안전하게 설계	암호 관리
6	암호연산	• 국제표준 또는 검증필 암호모듈로 등재된 안전한 암호 알고리즘을 선정하고 충분한 암호키 길이, 솔트, 충분한 난수값을 적용한 안전한 암호연산 수행방법 설계	
7	중요정보 저장	• 중요정보(비밀번호, 개인정보 등)를 저장·보관하는 방법이 안전하도록 설계	중요 정보 처리
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	설계항목	설명	비고
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

HTTP를 이용하여 연결을 유지하는 경우 세션을 안전하게 할당하고 관리하여 세션정보 노출이나 세션 하이재킹과 같은 침해사고가 발생하지 않도록 설계

번호	설계항목	설명	비고
1	세션통제	• 다른 세션 간 데이터 공유금지, 세션 ID 노출금지, (재)로그인시 세션 ID 변경, 세션종료(비활성화, 유효기간 등) 처리 등 세션을 안전하게 관리할 수 있는 방안 설계	세션 통제

4. 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 • 경로조작 및 자원삽입 • 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 않은 URL 주소로 자동접속 연결 • HTTP 응답분할
	허용된 범위내 메모리 접근	• 포맷스트링 삽입 • 메모리 버퍼 오버플로우
	보안기능 입력값 검증	• 보안기능 결정에 사용되는 부적절한 입력값 • 정수형 오버플로우 • Null Pointer 역참조
	업로드 · 다운로드 파일검증	• 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 제한기능 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 주요 정보
	암호연산	• 취약한 암호화 알고리즘 사용 • 충분하지 않은 키 길이 사용 • 적절하지 않은 난수 값 사용 • 부적절한 인증서 유효성 검증 • 솔트 없이 일방향 해쉬함수 사용
	중요정보 저장	• 암호화 되지 않은 중요정보 • 사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출
	중요정보 전송	• 암호화 되지 않은 중요정보
에러 처리 (1개)	예외처리	• 오류 메시지 정보노출
세션 통제 (1개)	세션통제	• 잘못된 세션에 의한 데이터 정보 노출

구현단계 보안약점 제거 기준(49개)을 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.

입력데이터 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 인증없는 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 하드 디스크에 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 메소드 부터 반환된 Private 배열	private 배열에 public 데이터 할당		
API 오용 (2개)	DNS Lookup에 의존한 보안 결정	취약한 API 사용				

5. 설계단계 개발보안 적용 효과

설계단계에서 개발보안을 적용할 때 얻을 수 있는 효과는 아래와 같다.

- 개발보안을 일관성 있게 적용

  크로스사이트 스크립트(XSS)와 같은 보안 취약점들은 설계 시에 소프트웨어에서 사용할 공통 라이브러리나 프레임워크에서 고려하면 소프트웨어에서 일관성 있게 취약점 방어가 가능해진다. 설계 시 XSS를 방지하기 위한 공통 입력값 검증 모듈을 정의 하여 사용하게 하거나, 시스템 설정으로 방어할 수 있다. 설계 시에 이러한 고려가 없다면, 이후 구현 단계에서 개발자/팀별로 각기 다른 방법으로 취약점을 방어하기 때문에 이후 유지보수가 어렵고, 완전하게 취약점을 방어하지 못하는 경우가 발생한다.

- 구현 단계에서 개발보안 항목을 명확히 알 수 있음

  설계 시에 개발보안 항목에 대한 명확한 기술이 없으면, 구현 시 개발 보안을 적용할 때 모호하거나 개발자 임의로 처리하는 경우가 발생할 수 있다. 크로스사이트 스크립트와 같은 전형적인 경우는 문제가 없을 수도 있지만 인증이나 인가와 같이 사전에 명확한 보안사항이 정의되어 있지 않으면 구현 시 요구사항과 다른 구현이 진행되어 이 후 추가 수정이 필요하거나 혹은 보안 취약점을 가지고 배포될 위험이 있다.

제2절 기능설계 시 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 커스터마이징하여 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.

행정기관 및 공공기관 정보시스템 구축·운영 지침('별표3')에서는 정보처리시스템 구축 시 우선적으로 고려되어야 하는 보안설계 기준을 정의하고 있다. 이 설계항목들을 설계단계에서 어떻게 반영할 것인가에 대한 계획수립이 요구되며, [표3-3]은 설계항목에 대한 적용계획을 수립하기 위해 활용될 수 있는 『보안요구항목 적용계획서』양식이다. 설계항목 적용계획서 작성하여 미들웨어 프레임워크 또는 공통라이브러리 구축과 같은 시스템의 구조적인 보안설계로 보안을 강화하거나 개발자들이 구현 시 시큐어코딩을 적용할 수 있도록 표준코딩정의서 또는 시큐어코딩 가이드를 문서화하는 작업을 설계단계에 수행할 수 있도록 계획한다.

표 3-3 설계항목 적용계획서(예)

유형	입력데이터 검증 및 표현	요구사항번호	SR1-1
설계항목	DBMS 조회 및 결과 검증		
보안대책	DBMS 조회시 질의문(SQL) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.		
설계항목 적용 계획	애플리케이션에서 DB연결을 위해 사용되는 계정은 "애플리케이션명_시리얼번호"로 설정하며, 해당 계정에 대해서는 "애플리케이션명_테이블명"에 대해 조회, 수정, 삭제, 업데이트 권한만 설정한다.		
비고	표준코딩정의서에서 정의		

설계자는 구현하고자 하는 기능을 설명하기 위해 유즈케이스 다이어그램, 플로우차트, DFD(데이터 흐름 다이어그램)와 같은 그림을 그린다. 이러한 기능설명을 하는 산출물을 기준으로 설계항목을

적용하여 시스템이 구현하고자 하는 각각의 기능들이 안전하게 동작될 수 있도록 설계단계에서 보안이 고려되도록 한다.

다음 예시들은 설계단계의 산출물에 설계항목을 표기하여 개발자들이 각 기능에 대해 고려되어야 하는 설계항목들을 숙지할 수 있도록 한 것이다.

| 설계항목 적용예 1 _ 유즈케이스 다이어그램 |

<table>
<tr><th colspan="3">유즈케이스 ID: 사용자 로그인</th></tr>
<tr><td>사전조건</td><td colspan="2">포스트-잇 프로그램을 가동한다.</td></tr>
<tr><td colspan="3">시나리오</td></tr>
<tr><td colspan="3">[주요 이벤트 경로]

사용자는 포스트-잇 프로그램을 가동한다.
자동 로그인이 실행되어 사용자는 바로 자신의 개인화된 화면으로 이동한다.

[대안 또는 예외 경로]

a. 로그인 옵션이 수동 로그인으로 적용되어 있을 때:
b. 프로그램은 아이디/비밀번호를 묻는 창을 띄운다.
c. 프로그램은 서버에서 아이디/비밀번호를 검증한다.
d. 아이디/비밀번호가 잘못되었을 경우 1.a로 돌아간다.

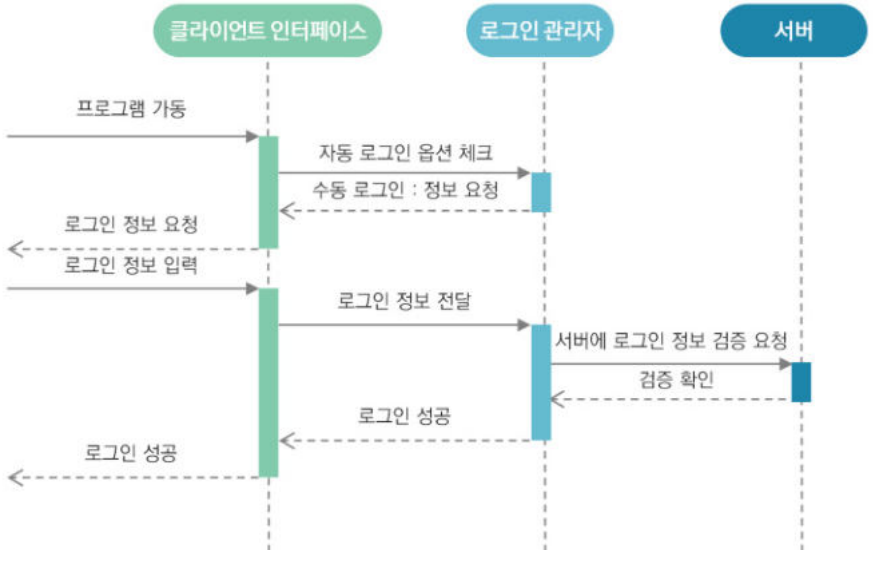
첫번째 로그인일 경우

a. 프로그램은 아이디/비밀번호를 묻는 창을 띄운다.
b. 프로그램은 서버에서 아이디/비밀번호를 검증한다.
c. 아이디/비밀번호가 잘못되었을 경우 2.a로 돌아간다.
d. 사용자에게 자동 로그인 옵션과 수동 로그인 옵션 중 하나를 선택하도록 묻는다.
e. 사용자가 입력 옵션 선택을 하지 않을 경우, 기본적으로 자동 로그인 옵션을 선택하도록 한다.</td></tr>
<tr><td>사후조건</td><td colspan="2">사용자는 로그인에 성공한다.</td></tr>
<tr><td colspan="3">* 설계항목(예시)</td></tr>
<tr><td>설계항목</td><td>주요 점검사항</td><td>비고</td></tr>
<tr><td>SR1-1 DBMS 조회 및 결과 검증</td><td>• 사용자가 입력한 아이디/비밀번호가 SQL 삽입을 일으키지 않도록 해야 한다.</td><td>FR-001 공통적용</td></tr>
<tr><td>SR2-2 인증수행제한</td><td>• 인증시도 횟수가 제한 되도록 해야 한다.</td><td>FR-002 공통적용</td></tr>
<tr><td>SR2-8 중요정보 전송</td><td>• 아이디/비밀번호는 암호화된 통신채널을 이용하여 전송되어야 한다.</td><td></td></tr>
<tr><td>SR3-1 에러처리</td><td>• 에러메시지에 시스템의 정보가 노출되지 않아야 한다.</td><td></td></tr>
</table>

※ FR-001 기능은 SQL삽입 필터링을 위한 필터링 기능이라고 가정
※ FR-002 기능은 인증시도 횟수제한 기능 이라고 가정

| 설계항목 적용예 2 _ 순차 다이어그램 |

[순차 다이어그램: 사용자 로그인 – 수동 로그인]



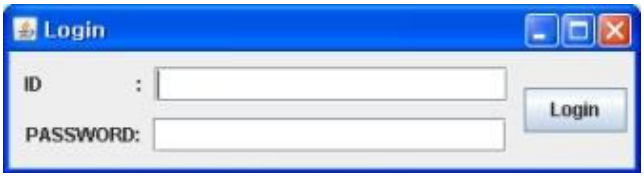
* 설계항목(예시)		
설계항목	주요 점검사항	비고
SR1-1 DBMS 조회 및 결과 검증	• 사용자가 입력한 아이디/비밀번호가 SQL삽입을 일으키지 않도록 해야 한다.	FR-001 공통적용
SR2-2 인증수행제한	• 인증시도 횟수가 제한 되도록 해야 한다.	FR-002 공통적용
SR2-8 중요정보 전송	• 아이디/비밀번호는 암호화된 통신채널을 이용하여 전송되어야 한다.	
SR3-1 에러처리	• 에러메시지에 시스템의 정보가 노출되지 않아야 한다.	

※ FR-001 기능은 SQL삽입 필터링을 위한 필터링 기능이라고 가정
※ FR-002 기능은 인증시도 횟수제한 기능 이라고 가정

| 설계항목 적용예 3 _ 화면설계서 |

ID	SP-LOGIN-01		
업무시스템	포스트잇 클라이언트	화면명	사용자 로그인 화면
업무기능	사용자 로그인 정보 입력	관련 TABLE	
단위시스템	사용자 로그인		

A. 화면



B. 처리 개요

사용자의 로그인 정보를 입력 받는 화면을 구성한다. 사용자에게 입력 받는 로그인 정보는 ID와 PASSWORD 이다. 사용자 입력 후 Login 버튼을 누르면 유즈케이스 '사용자 로그인' 의 대안 또는 예외 경로로 로그인 정보를 검증한다.

C. 화면 입/출력 정보일람

번호	I/O	이벤트	포인트	기능(링크 포함)
1	I	입력	ID	
2	I	입력	PASSWORD	
3	I	클릭	Login	입력된 ID/PASSWORD로 로그인 시도

D. 업무 흐름

1. 사용자는 ID/PASSWORD 입력창에 각각 자신의 로그인 정보를 입력한다.
2. PASSWORD 입력창에 입력된 글귀는 바로 보이지 않고 점(dot) 또는 별(*)로써 표시된다.
3. 개인 정보 입력 후 Login 버튼을 누르면 시스템은 사용자의 로그인 정보를 이용하여 로그인을 시도한다.

* 설계항목(예시)

설계항목	주요 점검사항	비고
• SR1-1 DBMS 조회 및 결과 검증	• 사용자가 입력한 아이디/비밀번호가 SQL삽입을 일으키지 않도록 해야 한다.	FR-001 공통적용
• SR2-2 인증수행제한	• 인증시도 횟수가 제한 되도록 해야 한다.	FR-002 공통적용
• SR2-8 중요정보전송	• 아이디/비밀번호는 암호화된 통신채널을 이용하여 전송되어야 한다.	
• SR3-1 에러처리	• 에러메시지에 시스템의 정보가 노출되지 않아야 한다.	

※ FR-001 기능은 SQL삽입 필터링을 위한 필터링 기능이라고 가정
※ FR-002 기능은 인증시도 횟수제한 기능 이라고 가정

프로젝트 수행 시, 각 개발단계의 산출물들은 일관성을 가지고 관리되어야 하며, [표3-4]는 분석단계 산출물인 요구사항 추적표 샘플이다. 기본적으로 프로젝트를 수행하면서 최소한으로 요구되는 산출물들의 연관관계를 추적할 수 있도록 하기 위한 산출물이며, 개발 프로젝트 수행 시 보안 요구항목 적용을 검토할 수 있는 문서로 활용할 수도 있다.

표 3-4 요구사항 추적표

분석 단계				설계 단계			구현 단계		시험 단계	
사용자 요구사항 명세서		유스케이스 명세서		사용자 인터페이스 설계서	컴포넌트 설계서	데이터베이스 설계서	프로그램 코드	단위시험 결과서	통합시험 결과서	시스템 시험 결과서
요구사항 ID	요구사항명	유스케이스 ID	유스케이스명	화면 ID	컴포넌트 ID	테이블 ID	파일 ID	단위시험 ID	시나리오 ID	시나리오 ID
FR-001	SQL 필터링				CO-001		PG-001	UT-001	TT-001	TR-001
FR-002	인증 점검				CO-002		PG-002	UT-002	TT-002	TR-002
…	..	…	..	…	…	…	…	…	…	…
FR-100	로그인	UC-100	로그인 처리	UI-100	CO-100	DB-100	PG-100	UT-100	TT-100	TR-100

제3절 보안설계 기준 정의 및 설계시 고려사항

1. 입력데이터 검증 및 표현

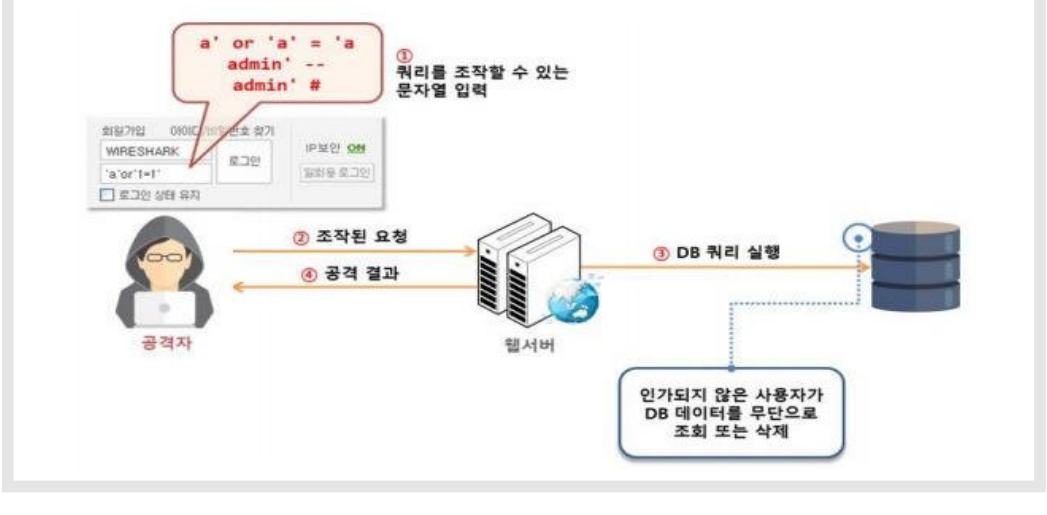
1.1 DBMS 조회 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	DBMS 조회 및 결과 검증
설명	DBMS 조회 시 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 외부 입력값이 삽입되는 SQL질의문을 동적으로 생성해서 실행하지 않도록 해야 한다. ③ 외부 입력값을 이용해 동적으로 SQL질의문을 생성해야 하는 경우, 입력값에 대한 검증을 수행한 뒤 사용해야 한다.

가. 취약점 개요

데이터베이스(DB)와 연동된 웹 응용프로그램에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 폼 및 URL 입력란에 SQL 질의문을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안취약점을 말한다.

그림 3-1 SQL 삽입 취약점



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

나. 설계 시 고려사항

① 애플리케이션에서 DB연결으로 데이터를 처리하는 경우 최소 권한이 설정된 계정을 사용해야 한다.

취약한 애플리케이션으로 인해 침해사고가 발생하더라도 나머지 부분에 대해 공격자가 접근 권한을 가지지 않도록 애플리케이션에서 DB연결을 위해 사용되는 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 지정한다.

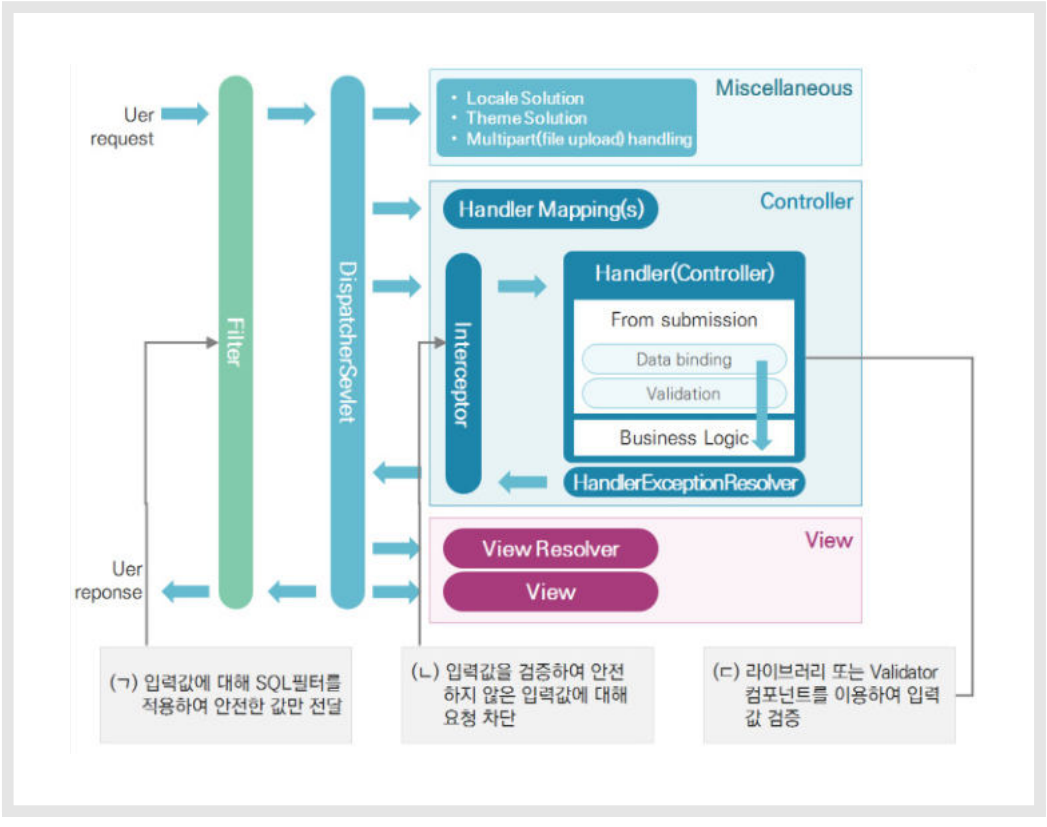
ORM프레임워크를 사용하여 안전한 정적 쿼리 구조로 SQL문을 수행할 수 있도록 개발환경을 설정하고, ORM프레임워크에서 제공하는 함수를 활용하여 외부 입력값에 의해 SQL 질의문의 구조가 변경되지 않도록 한다.

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 입력 값 검증

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



(ㄴ) 인터셉트를 이용한 입력 값 검증

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

(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 최소화하고 SQL문을 안전하게 처리할 수 있도록 다음과 같은 프레임워크나 라이브러리의 사용을 고려할 수 있다.

표 3-5 SQL삽입 보안약점 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 자유 소프트웨어이다.
	• MyBatis: http://blog.mybatis.org/ • 객체지향언어인 자바의 관계형 데이터베이스 프로그래밍을 좀 더 쉽게 도와주는 개발 프레임 워크이다. • Apache Software License v2.0[2] 정책에 따라 자유롭게 사용가능하다.
	• JPA(Java Persistence API): https://jcp.org/aboutJava/communityprocess/final/jsr220/index.html • 관계형 데이터베이스에 접근하기 위한 표준 ORM 기술을 제공하며, 자바플랫폼, 표준 에디션 및 자바플랫폼, 엔터프라이즈에디션을 사용하는 어플리케이션에서의 관계형데이터 관리를 목적으로 하는 프로그래밍 인터페이스다.
ASP.NET (확장가능)	• AntiSQLi 라이브러리: http://ironbox.github.io/AntiSQLi/ • SQL인젝션의 위협이 있는 데이터를 자동으로 파라미터화하는 오픈 소스 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 2.0 은 누구든 자유롭게 아파치 소프트웨어를 다운 받아 부분 혹은 전체를 개인적 혹은 상업적 목적으로 이용할 수 있으며 재배포시에는 원본소스코드 또는 수정한 소스코드를 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.

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	SQL 삽입

라. 사고사례

"OO 해킹사고, 허술한 웹 취약점 관리가 발단" (2015-10-20, IT조선)

지난 9월 11일 온라인 커뮤니티 'OO' 해킹사고는 웹 취약점을 악용한 데이터베이스(DB) 공격이 발단이었던 것으로 나타났다.

미래창조과학부는 20일 약 196만 명의 회원정보를 유출한 OO 홈페이지 침해사고와 관련해 해킹방법, 사고원인 등에 대한 민·관 합동조사단의 조사결과를 발표했다.

> 조사단은 해킹방법과 정보탈취에 악용된 보안취약점 확인을 위해 OO 에 남아있는 약 10만 건의 웹서버 로그와 약 2890만 건의 개인정보 DB 등을 분석했다. 그 결과, 해커는 OO 홈페이지 구조 및 취약점을 파악하고, SQL인젝션에 취약한 웹페이지를 확인한 후 SQL인젝션으로 개인정보를 탈취한 것으로 드러났다.

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 Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

⑥ Dynamic SQL & SQL Injection, Raul Garcia, http://blogs.msdn.microsoft.com/raulga/2007/01/04/dynamic-sql-sql-injection/

⑦ SQL Injection Signatures Evasion, IMPERVA, http://www.imperva.com/docs/IMPERVA_HII_SQL-Injection-Signatures-Evasion.pdf

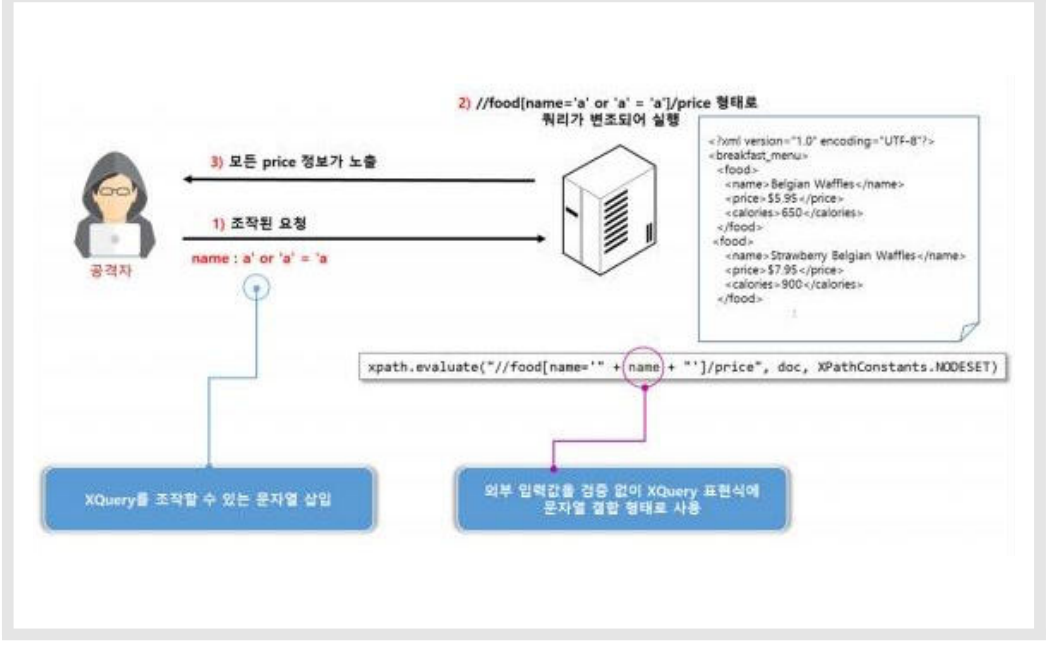
1.2 XML 조회 및 결과 검증

유형	입력데이터 검증 및 표현
설계항목	XML조회 및 결과 검증
설명	XML 조회 시 질의문(XPath, XQuery 등) 내 입력값과 그 조회결과에 대한 유효성 검증방법(필터링 등) 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① XML문서를 조회하는 기능을 구현해야 하는 경우 XML질의문에 사용되는 파라미터는 반드시 XML 질의문을 조작할 수 없도록 필터링해서 사용하거나, 미리 작성된 질의문에 입력 값을 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 XQuery 삽입 취약점



나. 설계 시 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

외부 입력값에서 XML삽입 공격이 가능한 문자열들을 필터링하는 Validator 컴포넌트를 개발하여 XML조회를 수행하는 애플리케이션 작성 시 입력값에 대한 검증 작업이 일괄 적용되도록 설계 한다.

(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 외부 입력값이 질의문의 구조를 바꿀 수 없는 API(예. Java API - XQuery)를 사용하도록 시큐어코딩 규칙을 정의한다.

다. 연관된 구현단계 보안약점 항목

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

라. 사고사례

XML해킹공격, PC정보 유출 이렇게 이루어져! (2013-08-28, 데일리시큐)

여름휴가를 마치고 돌아온 OO 대리(31세. 가명)는 가족과 여행지에서 함께 찍은 사진과 동영상파일이 외부 사이트에 노출된 것을 발견했다.

평소 PC는 백신 등 보안은 꼼꼼히 챙겼다고 자부했지만 자신의 PC 정보가 유출돼 크게 당황한 것이다. 이 대리는 평소 친분이 있는 보안전문가인 지인에게 자초지종을 설명한 결과 뜻밖의 이야기를 들었다.

유출된 경로를 추적한 결과 기존에 잘 알려지지 않은 XML 공격으로 정보 유출이 이루어진 것을 알게 된 것이다.

XML 공격기법의 경우 오래전부터 OWASP를 비롯해 알려진지 오래됐다. 하지만 한정된 진단 기간과 정보 유출과 관련된 해킹기법이 SQL 인젝션, 파일업로드 등 체크리스트 항목에 한정된 허점을 파고들며 다시 해커들의 주요 공격기법이 된 것이다.

이 취약점이 동작하는 이유는 XML 파서(번역기)가 취약하게 설정돼 서버의 디렉토리와 파일정보를 공격자에 노출하게 되어 발생하는 것이다. 따라서 이 공격을 해결하기 위해 기업 보안 관리자는 XML 파서 설정을 안전하게 변경하여 운영할 필요가 있다.

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 HandlingVerificationRequirements, http://www.owasp.org/index.php/Category: OWASP_Application_Security_Verification_Standard_Project

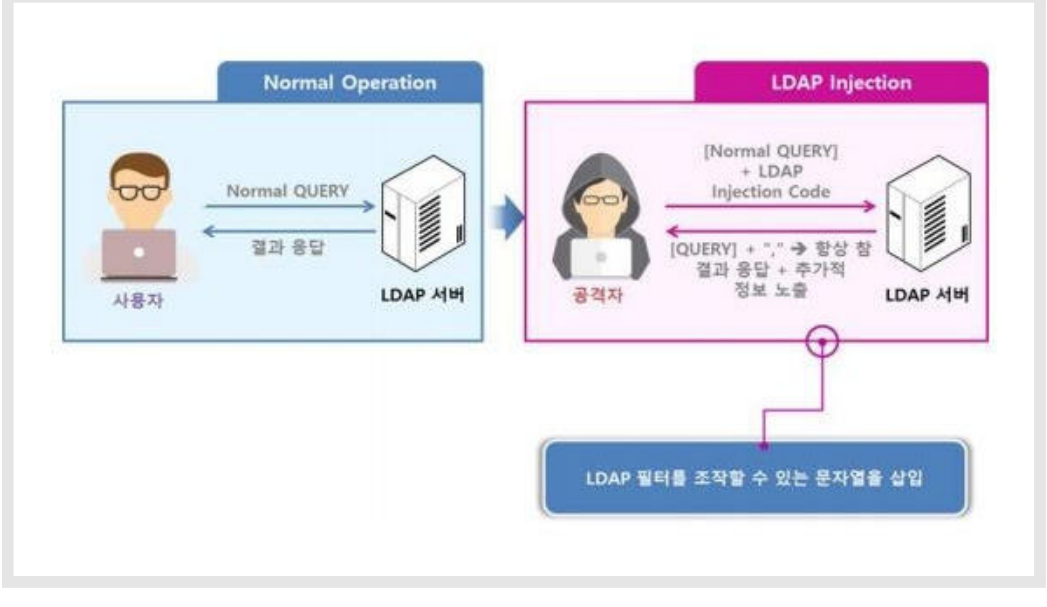
1.3 디렉토리 서비스 조회 및 결과 검증

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

가. 취약점 개요

외부입력값이 LDAP 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



나. 설계 시 고려사항

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

LDAP 인증이 포함되는 기능 설계 시, 외부 입력값이 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 3-6 LDAP 필터 작성법

개발환경	활용가능한 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 검색필터로 액티브 디렉토리 조회 시의 검색기준을 정의하고 효율적인 검색을 수행할 수 있다. 위 MS 웹문서의 내용과 참조목록으로 필터 작성에 필요한 문법을 참고하여 특수문자를 필터링한다.

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	LDAP 삽입

라. 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

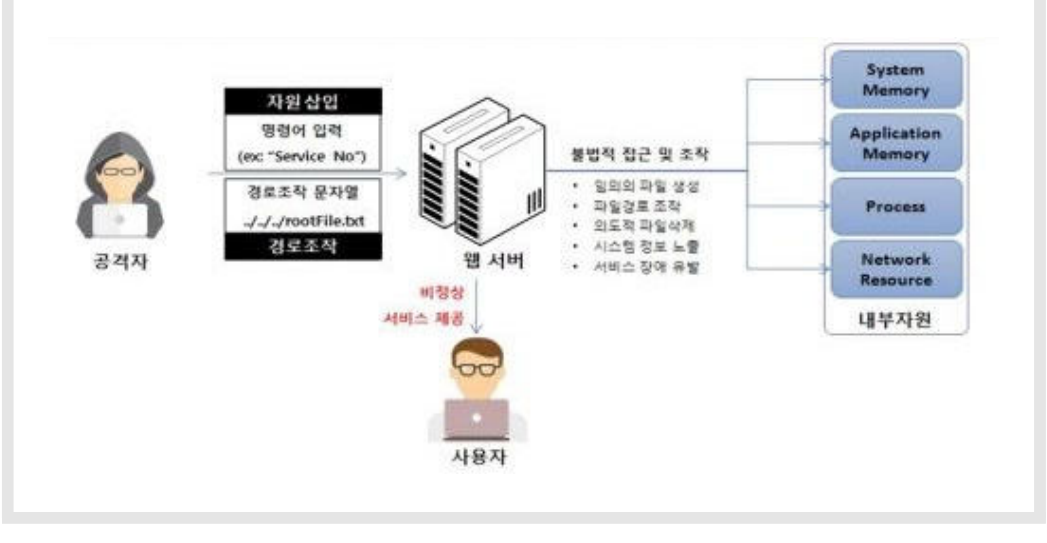
유형	입력데이터 검증 및 표현
설계항목	시스템 자원 접근 및 명령어 수행 입력값 검증
설명	시스템 자원접근 및 명령어를 수행할 때 입력값에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 외부 입력값을 이용하여 시스템자원(IP, PORT번호, 프로세스, 메모리, 파일 등)을 식별하는 경우 허가되지 않은 자원이 사용되지 않도록 해야 한다. ② 서버 프로그램 안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부 입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

가. 취약점 개요

사례1: 경로조작 및 자원삽입

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, 경로조작 및 자원 삽입으로 공격자가 허용되지 않은 권한을 획득하여, 설정에 관계된 파일을 변경하거나 실행시킬 수 있다.

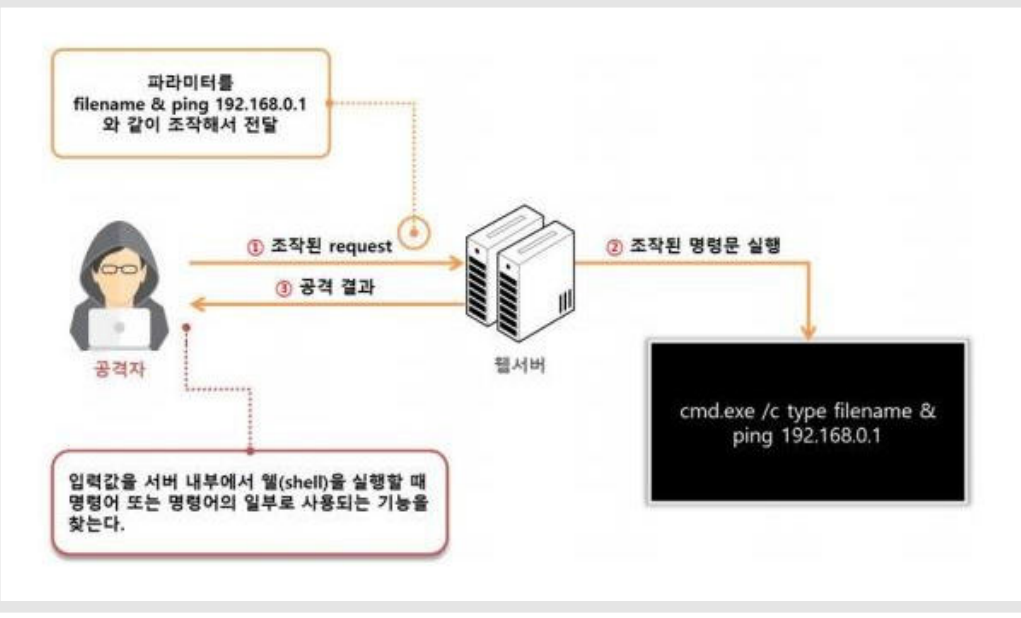
그림 3-4 경로조작 및 자원삽입



사례2: 입력값을 조작하여 허가되지 않은 명령 실행

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

그림 3-5 운영체제 명령어 삽입



나. 설계 시 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 설계 시 사용해야 하는 리소스 목록을 정의하여 지정된 범위 안에서 리소스를 선택하여 사용하도록 해야 하며, 리소스 목록은 프로퍼티파일이나 XML파일로 정의하여 리소스 정보를 변경하는 경우 프로그램 수정을 최소화할 수 있도록 관리한다.

사용자의 요청 리소스가 특정 디렉토리 내의 모든 파일인 경우, 모든 파일명을 목록화하는 것은 어렵다. 이런 경우는 입력값 중 경로조작을 일으킬 수 있는 문자(.. / ₩) 문자를 제거하고 사용하여 지정된 경로 내의 파일만 접근 가능하도록 시큐어코딩 규칙을 정의한다.

② 서버 프로그램 안에서 쉘을 생성하여 명령어를 실행해야 하는 경우 외부 입력값에 의해 악의적인 명령어가 실행되지 않도록 해야 한다.

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

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

이 때 외부 입력값은 목록화된 정보를 검색하는 인덱스값으로 사용하여 시스템 명령어 사용을 최소화 한다.

다. 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

라. 사고사례

[단독] 충격! OO 사이트 보안 취약성 확인....국민 정보가 "줄줄" (2011-05-01, 보안뉴스)

주소창에 주민번호 입력으로 주소 및 주요 정보 노출 가능

철통같이 지켜져야 할 OO 사이트에서 기본적인 보안문제가 방치돼 특히 별다른 해킹 프로그램 없이 인터넷 주소창에 주민번호와 특정 구문을 입력하는 것만으로도 개인의 민감한 정보가 노출되는 것으로 나타나 충격을 주고 있다.
이 사이트는 개발단계부터 체계적으로 보안을 적용하지 않아 로그인을 거치지 않아도 주요 정보를 열람할 수 있는 심각한 취약점을 가지고 있는 것으로 보안뉴스에서 처음 확인됐다.

▲

인터넷 주소창에 특정구문이 담긴 URL에 주민등록번호를 입력한다. ⓒ보안뉴스

▲

페이지가 이동하면서 빈창이 나타난다. 이때 소스보기를 하면 주소 및 주요정보를 볼 수 있다. ⓒ보안뉴스

해당 취약점은 로그인 후 입력되는 명령어에 대한 별다른 보안절차가 없어 나타난 것으로 보인다. 이에 따라 인터넷 주소창에 OO 사이트 주소를 비롯한 특정 구문과 주민등록번호를 입력하면 정부에 등록된 가입자의 실제 주소가 소스코드 보기로 상세하게 나타난다. 그 뿐 아니라 이름과 주민등록번호를 입력하면 조회 대상이 언제 직장에 입사했는지에 대한 정보도 알 수 있다

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 Directorytraversalattack,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 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 사이트스크립트(XSS) 필터링을 수행한 뒤 사용해야 한다.

가. 취약점 개요

사례1: 외부 입력값을 검증 없이 응답페이지 생성에 사용하는 경우

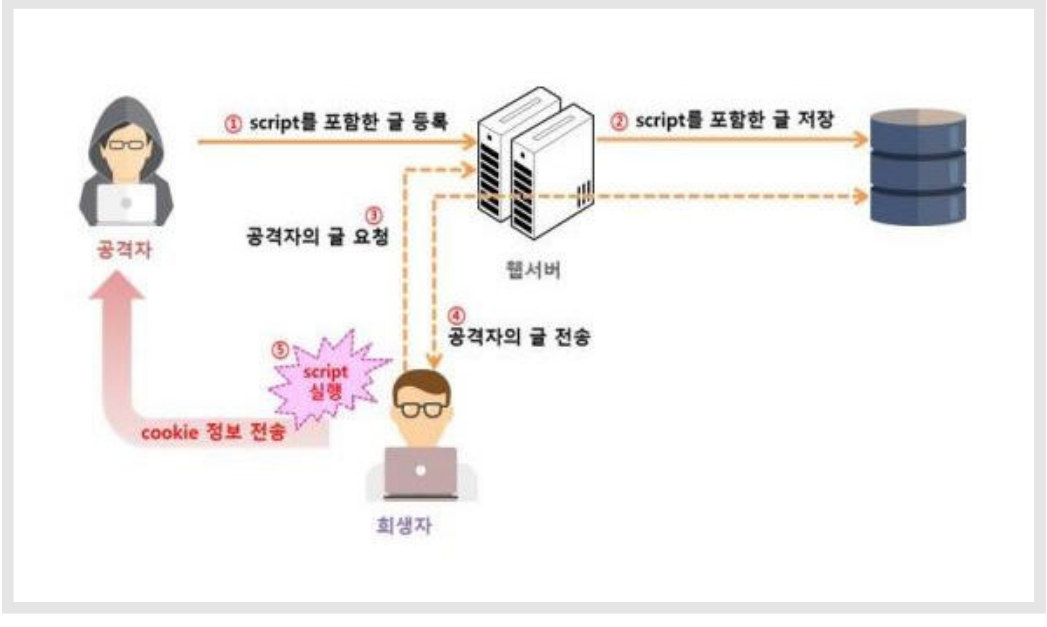
웹 페이지에 악의적인 스크립트가 포함될 수 있으며, 해당 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 실행되어 정보유출 등의 공격을 유발할 수 있다.

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. 설계 시 고려사항

① 사용자로부터 입력 받은 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

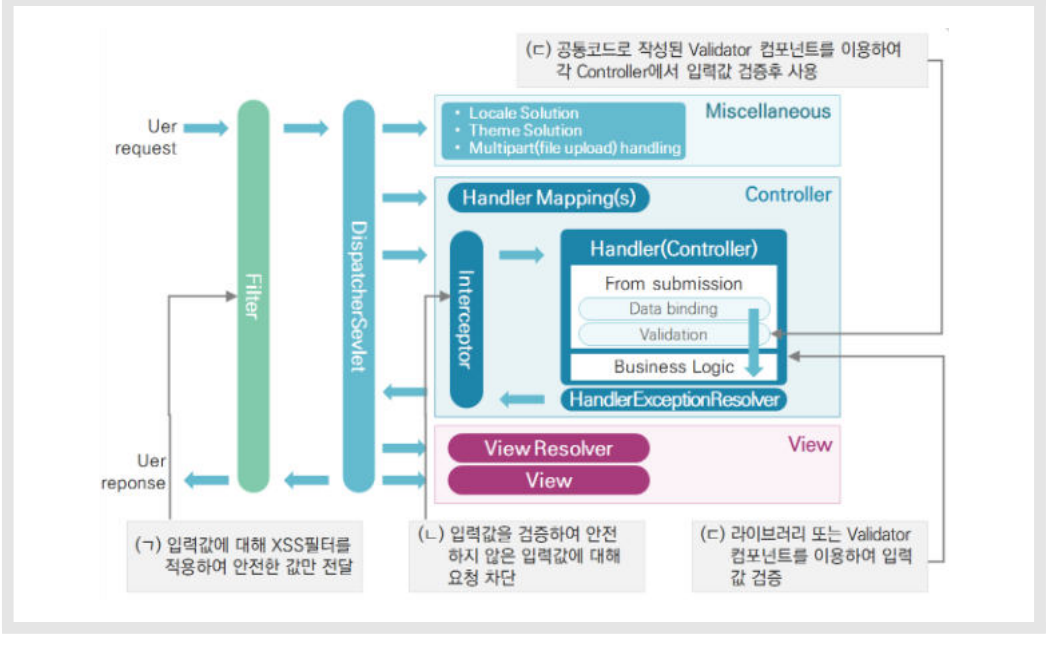
웹 컴포넌트인 Filter를 사용하여 사용자의 입력값에 대해 XSS 필터나 HTML인코딩을 적용하여 안전한 값으로 치환한 뒤 사용할 수 있도록 모든 애플리케이션에 일괄 적용한다.

(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 필터링할 수 있도록 설계한다.



② DB조회결과를 동적으로 생성되는 응답페이지에 사용하는 경우 HTML인코딩 또는 크로스 사이트스크립트(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 수행해야 하는데, 이러한 설정은 개발가이드 구현단계에 항목을 작성하여 적용할 수 있도록 해야 한다.

HTML, URL 등의 문자를 인코딩 및 필터링하여 XSS 취약점을 최소화할 수 있도록 다음과 같은 라이브러리의 사용을 고려할 수 있다.

표 3-7 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 어플리케이션 개발 시, XSS공격으로부터 안전할 수 있도록 도와주는 인코딩 라이브러리이다.
	• MS AntiXSSEncoder 클래스: http://social.msdn.microsoft.com/Search/en-US?query=antixssencoder&pgArea=header&emptyWatermark=true&ac=4 • HTML, XML, CSS, URL 문자열을 인코딩하여 XSS공격의 위협에 대응할 수 있는 클래스이다. HttpEncoder클래스의 HttpUtility, HttpServerUtility, HttpResponse Header 메소드를 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 방지가능하다. MIT License에 따라 사용가능하다.
Java	• LUCY XSS Filter: http://github.com/naver/lucy-xss-filter • 웹 애플리케이션을 XSS공격으로부터 보호하기 위한 화이트리스트 설정방식의 자바 라이브러리이다. Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
	• OWASP ESAPI XSS Filter: https://www.owasp.org/index.php/Category: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. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	크로스사이트 스크립트

라. 사고사례

"국가·공공기관 대상 해킹 메일 알고보니…" (2010-01-26, 디지털타임스)

국가정보원 국가사이버안전센터가 지난 25일 저녁 '관심' 단계로 사이버위협 수준을 높인 후 경고한 국가·공공기관 대상 해킹메일이 OO 취약점을 악용한 것으로 분석됐다.

> 국정원은 26일 이번 해킹 메일이 주요 상용메일 포털사이트의 웹 서비스 취약점 중 하나인 크로스 사이트 스크립팅(XSS) 취약점과 지난해 12월 16일 발표된 OO 관련 보안 취약점을 악용한 것이라고 밝혔다.

국정원은 이 해킹 메일을 분석한 결과 클릭 시 로그인 화면이 표시되는데 동시에 아이디와 비밀번호 입력여부와 상관없이 iFrame 태그 등을 이용해 은밀히 악성 PDF파일을 다운로드하고 실행시켜 해당 PC에 자료를 유출하는 악성코드를 설치한다고 설명했다.

마. 참고자료

① OWASP Top 10 A7:2017-Cross-Site Scripting(XSS), OWASP, http://www.owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html

② "Feed Injection In Web2.0: Hacking RSS and Atom Feed Implementations" By Robert Auger, http://www.cgisecurity.com/papers/HackingFeeds.pdf

③ "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests", http://www.cert.org/advisories/CA-2000-02.html

④ "Cross-site Scripting Explained" By Amit Klein, http://crypto.stanford.edu/cs155/papers/CSS.pdf

⑤ "DOM Based Cross Site Scripting or XSS of the Third Kind" By Amit Klein (WASC article) http://www.webappsec.org/projects/articles/071105.shtml

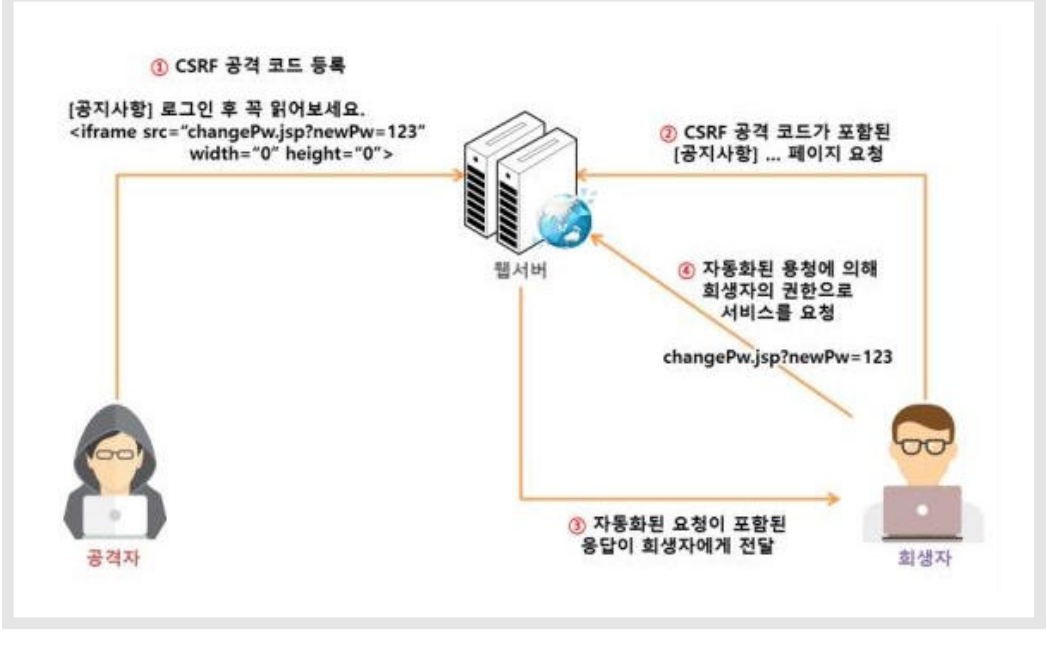
1.6 웹 기반 중요 기능 수행 요청 유효성 검증

유형	입력데이터 검증 및 표현
설계항목	웹 기반 중요기능 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



나. 설계 시 고려사항

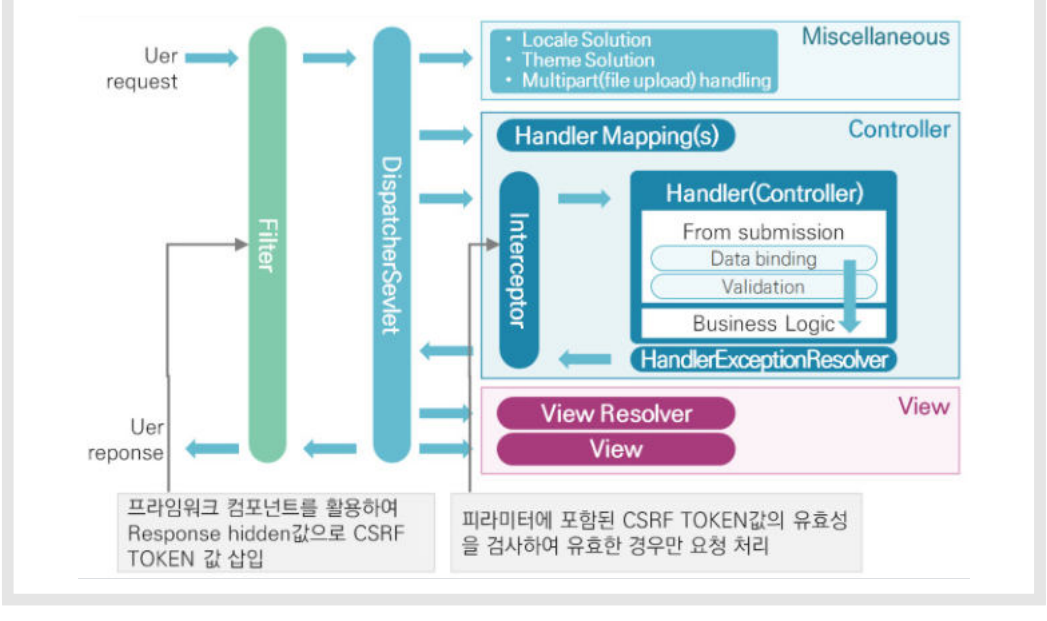
① 시스템으로 전송되는 모든 요청에 대해 정상적인 사용자의 유효한 요청인지, 아닌지 여부를 판별할 수 있도록 해야 한다.

(ㄱ) CSRF 토큰 사용

웹은 URL기반으로 요청을 처리하는 구조이다. 해당 요청이 특정사용자의 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 설계하는 것이다. Spring이나 Struts와 같은 MVC 프레임워크의 경우 사용자의 요청을 중간에 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 다음과 같은 프레임워크나 라이브러리의 사용을 고려할 수 있다.

표 3-8 CSRF 방어를 위한 프레임워크 및 라이브러리

개발환경	활용가능한 프레임워크 또는 라이브러리
Java	• Spring Security: http://spring.io/ • 인증, 인가 등 기업 애플리케이션의 보안기능을 제공하는 Java/Java EE • 프레임워크이며, 3.2이상의 버전에서는 CSRF공격에 대한 방어기능을 제공한다. • Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
	• Apache Struts: http://struts.apache.org/ • Java EE 웹 애플리케이션 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 CSRF토큰을 사용할 수 있도록 하는 자바EE 필터를 제공하는 라이브러리이다. BSD License에 따라 사용할 수 있다.
Python	• Django: https://www.djangoproject.com/ • 파이썬으로 작성된 오픈소스 웹 어플리케이션 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

CSRF토큰 방식도 XSS 취약점이 있는 사이트를 공격하게 되면 무력화될 수 있으므로 CAPTCHA와 같은 사용자와 상호 처리 가능한 기법을 적용하여 위조된 요청이 차단될 수 있도록 설계한다.

(ㄷ) 재인증 요구

중요기능의 경우 재인증으로 안전하게 실제 요청여부를 확인하도록 설계한다.

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	크로스사이트 요청위조

라. 사고사례

OO 개인정보 유출 사건 (위키백과)

2008년 OO 전체회원 1863만여 명의 개인 정보가 유출되었다. OO은 자체적으로 2008년 2월 4일에 자사의 정보가 유출된 사실을 감지하였으나, 실제 피해규모보다 적은 1081만 명의 정보만이 유출된 것으로 파악했다.

이 사건은 중국의 해커에 의해 이루어진 것으로 추정되며, 사이트 간 요청 위조(CSRF) 공격 방식을 이용한 것으로 알려졌다. 해커는 OO의 관리자들에게 공격 코드가 포함된 전자 우편을 대량으로 송신하였다. 관리자가 전자 우편을 읽는 순간 거기에 포함된 코드를 실행하게 되었고, 그 결과 해커는 관리자의 인증 정보를 얻었다.

> 해커는 이를 이용하여 OO에 가입된 사용자들의 정보를 빼내었으며, 이후 유출된 개인정보를 인질로 OO 측에 금전을 요구하였다.

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-request-forgery-demystified,47/

⑥ Cross-SiteRequestForgeries: ExploitationandPrevention,WilliamZellerandEdwardW. 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

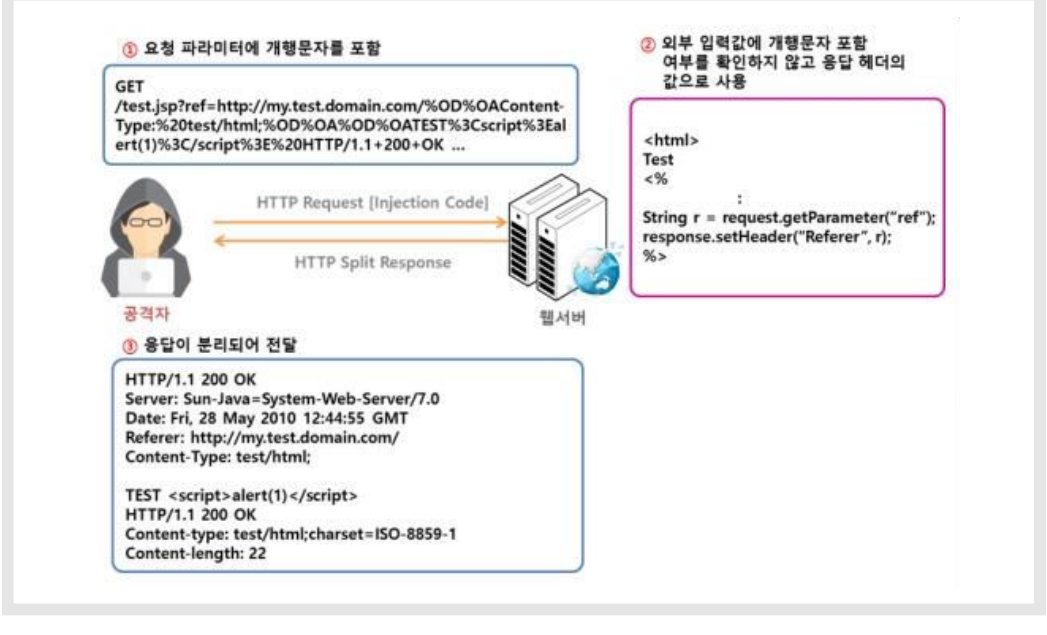
유형	입력데이터 검증 및 표현
설계항목	HTTP 프로토콜 유효성 검증
설명	비정상적인 HTTP 헤더, 자동연결 URL 링크 등 사용자가 원하지 않은 결과를 생성하는 HTTP 헤더·응답결과에 대한 유효성 검증방법 설계 및 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 외부 입력값을 쿠키 및 HTTP 헤더정보로 사용하는 경우, HTTP 응답분할 취약점을 가지지 않도록 필터링해서 사용해야 한다. ② 외부 입력값이 페이지이동(리다이렉트 또는 포워드)을 위한 URL로 사용되어야 하는 경우, 해당 값은 시스템에서 허용된 URL목록의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

가. 취약점 개요

사례1: HTTP응답분할

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.

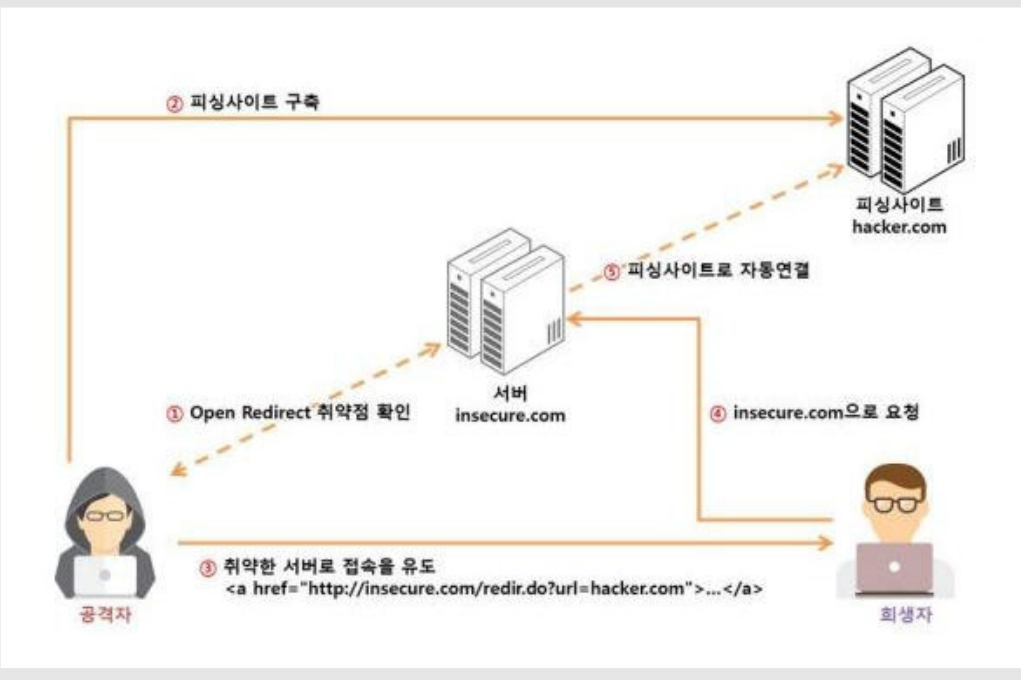
그림 3-9 HTTP 응답 분할



사례2: 신뢰되지 않은 URL로 자동 접속연결

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

그림 3-10 신뢰되지 않는 URL 주소로 자동접속 연결



나. 설계 시 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 HTTP 응답 헤더에 삽입되는 외부 입력값은 반드시 ₩r₩n 문자를 제거하여 사용할 수 있도록 시큐어코딩 규칙을 정의한다.

프로그램에서 Cookie 설정, 응답헤더 설정, 페이지 리다이렉트를 위한 Location정보 삽입 등 응답헤더에 외부 입력값이 삽입되는 경우, HTTP 응답분할을 일으킬 수 있는 문자(₩r₩n)를 필터링하도록 검증절차를 적용한다.

② 외부 입력값이 페이지이동(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. 연관된 구현단계 보안약점 항목

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

라. 사고사례

OOO 등 주요 인터넷사이드 동시다발 DDoS 공격에 불통 충격 (2009-07-08, 디지털데일리)

OO 등 국내 주요 대형 인터넷 사이트가 동시에 분산서비스거부(DDoS) 공격을 받아 서비스가 마비되는 사태가 발생했다.

7일 오후 6~7시부터 8일 자정이 지난 현재, OO 등 국내 사이트가 접속이 안되거나 불안한 상태다.

방송통신위원회와 한국정보보호진흥원(KISA)은 이날 오후 7시경부터 이들 주요 인터넷 사이트에 서비스가 중단되는 현상이 발생하자, 긴급히 상황 분석 작업을 벌였다.

그 결과, 대량 유해 트래픽을 수반하는 DDoS 공격으로 인해 국내 일부 사이트에 대한 인터넷 접속이 지연되거나 접속이 되지 않는 것으로 파악했다.

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.blogspot.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/preventing-open-redirection-attacks

1.8 허용된 범위내 메모리 접근

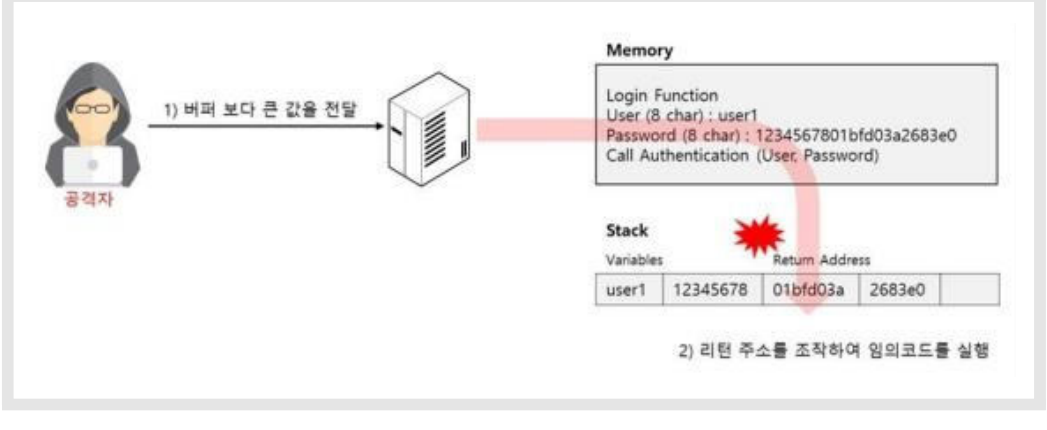
유형	입력데이터 검증 및 표현
설계항목	허용된 범위내 메모리 접근
설명	해당 프로세스에 허용된 범위의 메모리 버퍼에만 접근하여 읽기 또는 쓰기 기능을 하도록 검증방법과 메모리 접근요청이 허용범위를 벗어났을 때 처리방법을 설계해야 한다.
보안대책	① C나 C++ 같이 메모리를 프로그래머가 관리하는 플랫폼을 사용하는 경우 메모리 버퍼의 경계 값을 넘어서 메모리를 읽거나 저장하지 않도록 경계설정 또는 검사를 수행해야 한다. ② 개발 시, 메모리 버퍼오버플로우를 발생시킬 수 있는 취약한 API를 사용하지 않도록 해야 한다.

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: 포맷스트링 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 경계 값을 넘어서 메모리를 읽거나 저장하지 않도록 경계 설정 또는 검사를 수행해야 한다.

실행되는 시스템에 Non-executable Stack, 랜덤스택(ASLR), 스택가드(StackGuard)와 같은 메모리 보호 정책이 적용되도록 해야 하며, 프로그램 코딩 시 안전한 함수를 사용하고 메모리 사용 시 경계 값을 검사하고 사용할 수 있도록 시큐어코딩 규칙을 정의하여 개발자들이 준수할 수 있도록 한다.

② 개발 시, 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. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
입력데이터 검증 및 표현	메모리 버퍼 오버플로우
입력데이터 검증 및 표현	포맷스트링 삽입

라. 사고사례

'국방융합기술.hwp' 파일… 열지마세요 (2012-06-27, 아이티투데이)

OO는 지난 15일에 이어 27일 또다시 OO에 존재하는 코드 실행 취약점을 악용한 악성코드 유포 사례가 발견됐다며 주의를 당부했다.

국내 특정 조직들을 대상으로 발송된 이메일의 첨부 파일 형태로 유포된 이 한글파일은 HncTextArt_hplg에 존재하는 스택(Stack)의 경계를 체크하지 않아 발생하는 버퍼오버플로우(Buffer Overflow) 취약점이며, 해당 취약점은 2010년부터 지속적으로 악용돼 왔던 OO 소프트웨어 취약점들 중 하나다.

마. 참고자료

① CWE-119Improper Restriction of Operations with in 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 Buffer Overflow(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/Format%20String

1.9 보안기능 입력값 검증

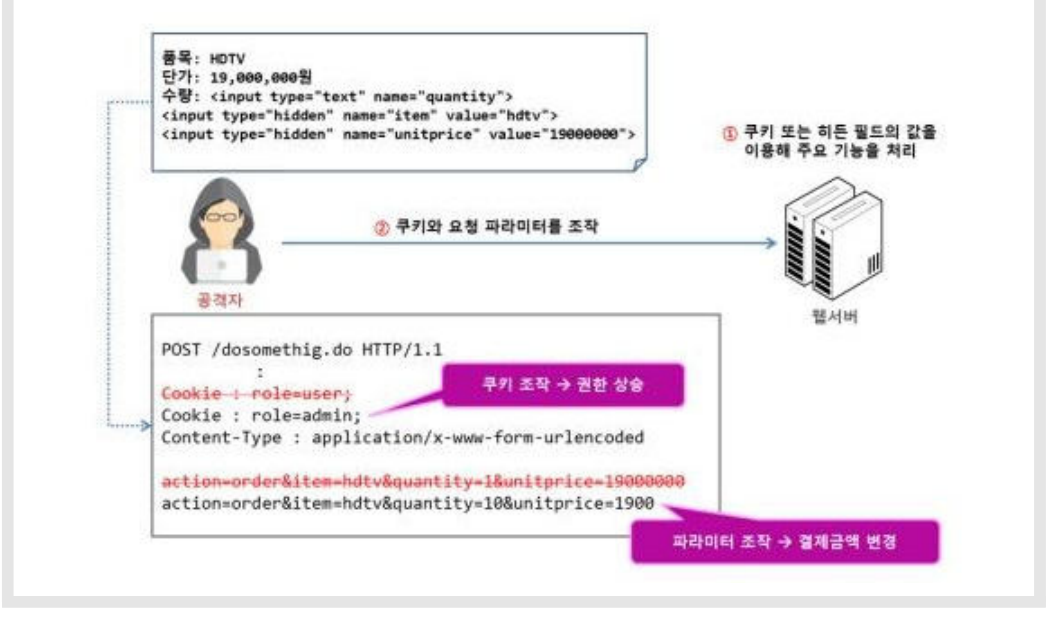
유형	입력데이터 검증 및 표현
설계항목	보안기능 입력값 검증
설명	보안기능(인증, 권한부여 등) 입력값과 함수(또는 메소드)의 외부입력 값 및 수행결과에 대한 유효성 검증방법과 유효하지 않은 값에 대한 처리방법을 설계해야 한다.
보안대책	① 사용자의 역할, 권한을 결정하는 정보는 서버에서 관리해야 한다. ② 쿠키, 환경변수, 파라미터 등 외부 입력값이 보안기능을 수행하는 함수의 인자로 사용되는 경우, 입력값에 대한 검증작업을 수행한 뒤 제한적으로 사용해야 한다. ③ 중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송해야 하는 경우에는 해당 정보를 암호화해서 전송해야 한다.

가. 취약점 개요

사례1: 보안기능결정에 사용되는 부적절한 입력값

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

그림 3-12 보안기능 결정에 사용되는 부적절한 입력값



사례2: 정수형 오버플로우

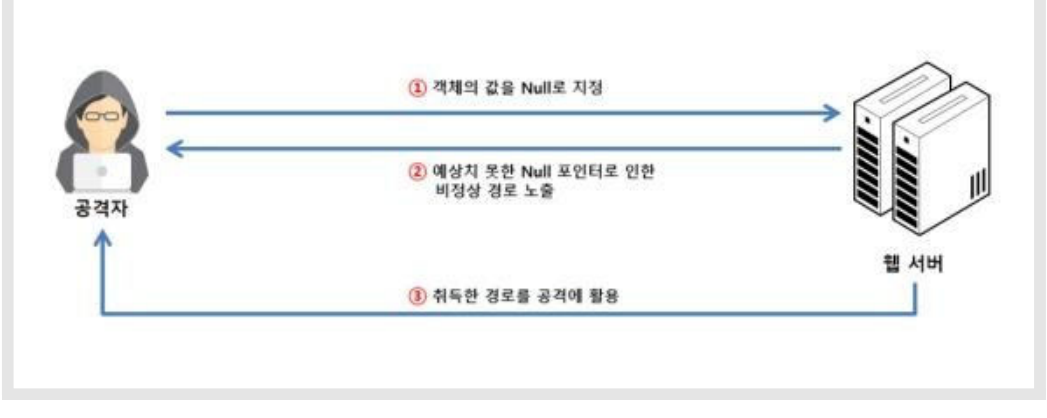
정수형 변수의 오버플로우는 정수 값이 증가하면서, 허용된 가장 큰 값보다 더 커져서 실제 저장되는 값이 의도하지 않게 아주 작은 수이거나 음수가 되어 발생한다. 특히 반복문 제어, 메모리 할당, 메모리 복사 등을 위한 조건으로 사용하는 외부입력값이 오버플로우 되는 경우 보안상 문제를 유발할 수 있다.

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 쿠키, 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 클라이언트 측에서 수행하는 검증방식과 서버에서 수행하는 검증방식이 동일해야 한다.

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

③ 중요상태정보나 인증, 권한결정에 사용되는 정보는 쿠키로 전송되지 않아야 하며, 불가피하게 전송해야 하는 경우에는 암호화해서 전송해야 한다.

중요정보가 포함되는 쿠키는 암호화하여 전송하거나 HMAC과 같은 메시지인증코드를 이용하여 서버 측에서 무결성을 검증한다.

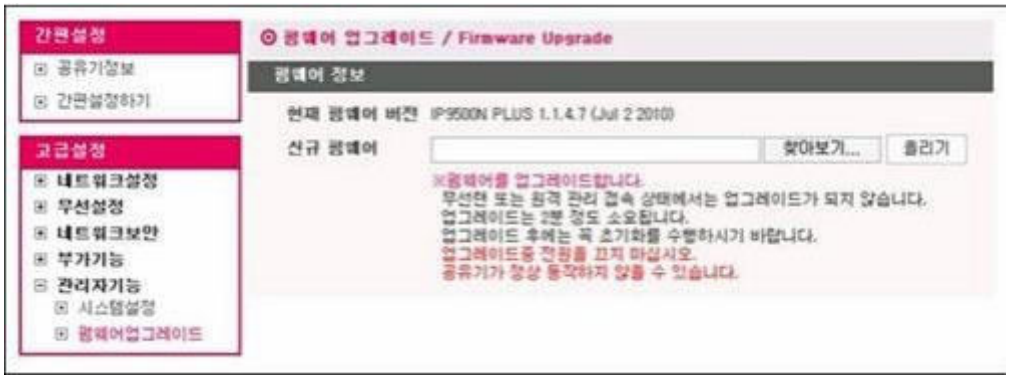
다. 연관된 구현단계 보안약점 항목

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

라. 사고사례

OO 인터넷 유무선 관리자 인증 취약점 보안 업데이트 권고 (2013-05-14, 보안뉴스)

OO사의 OO 제품군 유무선 공유기에서 관리자 인증을 우회할 수 있는 취약점이 발견되어 사용자들의 주의가 필요하다.



이 취약점은 관리자 암호가 설정되더라도 쿠키 값을 변조해 공유기의 관리자 설정 페이지로 접속 가능하다.

마. 참고자료

① CWE-807 Reliance on Untrusted Inputs in a Security Decision, MITRE, http://cwe.mitre.org/data/definitions/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 NullPointerDereference란?,Wisedog, http://story.wisedog.net/null-pointer-dereference-%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

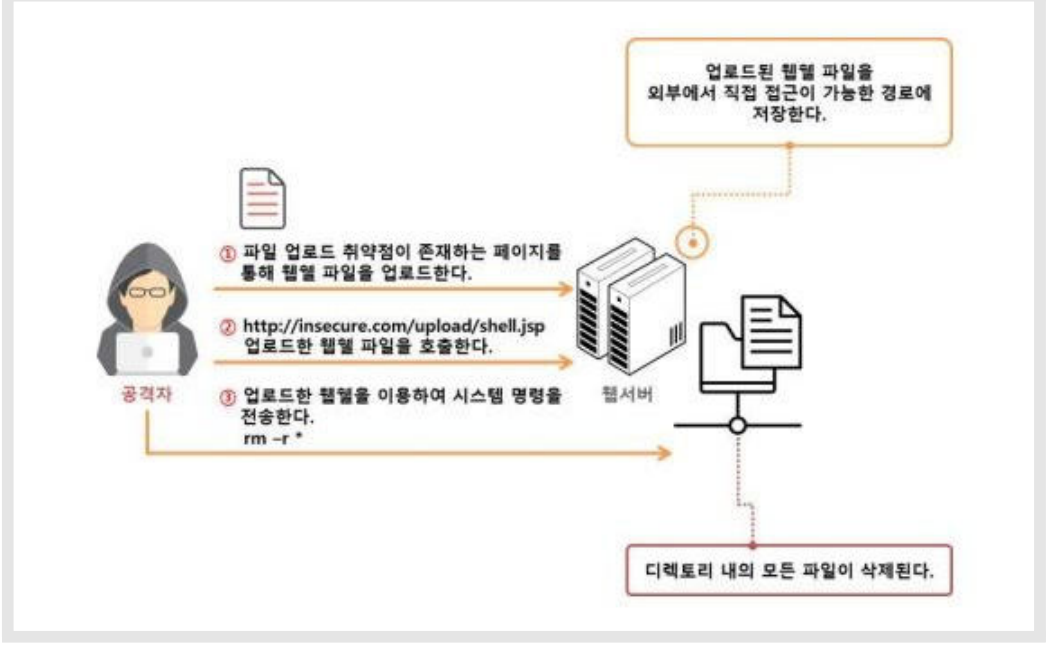
유형	입력데이터 검증 및 표현
설계항목	업로드·다운로드 파일검증
설명	업로드·다운로드 파일의 무결성, 실행권한 등에 관한 유효성 검증방법과 부적합한 파일에 대한 처리방법을 설계해야 한다.
보안대책	① 업로드 되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다. ② 업로드 되어 저장되는 파일은 외부에서 식별되지 않아야 한다. ③ 파일 다운로드 요청 시, 요청파일명에 대한 검증작업을 수행해야 한다. ④ 다운로드 받은 소스코드나 실행파일은 무결성 검사를 실행해야 한다.

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.

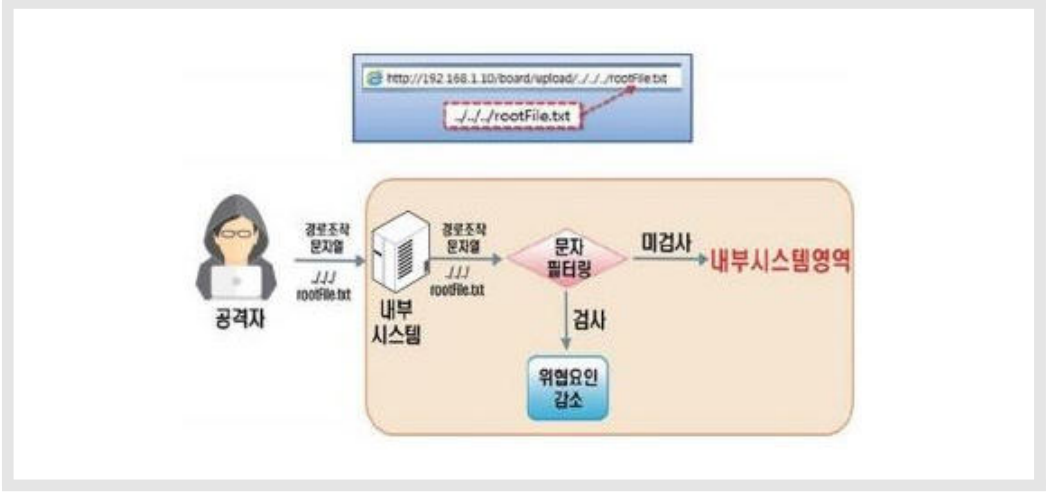
그림 3-15 위험한 형식 파일 업로드



사례2: 경로조작 문자를 이용한 파일 다운로드 취약점

외부입력값에 대해 경로조작에 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. 설계 시 고려사항

① 업로드 되어 저장되는 파일의 타입, 크기, 개수, 실행권한을 제한해야 한다.

(ㄱ) 업로드 파일의 크기 제한

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

```
〈bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver"〉
〈!-- 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

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

(ㄷ) 업로드 되어 저장되는 파일은 실행권한 제거

업로드 된 파일은 반드시 실행권한을 가지지 않도록 시스템이 설정되어 있어야 한다. 업로드 된 파일이 저장되는 디렉터리의 파일 또는 디렉터리 생성 시 기본 권한을 확인하고, 업로드 된 파일에 실행 권한이 주어지지 않도록 시스템 설정을 확인한다. 파일이 읽기 권한을 가지고 있기만 해도 서버에서 해당 파일을 읽어서 실행할 수 있으므로 서버 설정으로 업로드 경로의 파일이 실행되지 않도록 설정해야 한다.

② 업로드 되어 저장되는 파일은 외부에서 식별되지 않아야 한다.

(ㄱ) 업로드 되는 파일의 저장경로는 외부에서 직접 접근이 불가능한 경로사용.

파일이 저장되는 경로는 URL으로 직접적으로 접근 가능하지 않은 디렉터리를 사용한다. 이 경우 외부에서 URL 조작으로 업로드 된 파일에 접근이 불가능하므로, 업로드 되어 저장된 파일에 대한 통제가 가능하다. 이렇게 업로드 된 파일을 다운로드 하기 위해서는 다운로드 처리를 수행하는 컨트롤러를 구현해야 하며, 컨트롤러 구현 시 안전하게 파일을 다운로드 처리하기 위한 통제 기능을 적용할 수 있다.

(ㄴ) 업로드 되어 저장되는 파일의 파일명은 랜덤하게 생성하여 사용.

저장된 파일을 공격자가 찾을 수 없도록 파일명을 랜덤하게 생성하여 디렉터리에 저장한다. 이 경우 업로드 한 파일명과 저장된 파일명이 매핑 되어 관리될 수 있도록 DB 테이블의 칼럼을 설계할 때 업로드 파일명, 저장된 파일명이 함께 저장되도록 설계한다.

③ 파일 다운로드 요청 시, 요청 파일명에 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 접근통제

시스템에 저장되어 있는 파일을 다운로드 서비스 하는 경우, 허용 파일에 대한 목록을 작성하고 목록의 범위안의 파일에 대해서만 제한적으로 접근 또는 다운로드 할 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 사용할 수 있도록 설계한다.

파일 업로드 기능 구현 시 아래와 같은 프레임워크의 사용을 고려하여 기능의 목적에 부합하는 형식의 파일만 업로드 하도록 허용한다.

표 3-9 위험한 형식의 파일 업로드 대응 프레임워크

개발환경	활용가능한 프레임워크 또는 라이브러리
Java	• Spring: http://spring.io/ • 자바 플랫폼을 위한 오픈소스 어플리케이션 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 구현 시 허용할 파일유형과 파일크기제한을 정의할 수 있다. 2.x버전은 OSLv3.0에 따라, 3.x버전은 MIT License에 따라 사용가능하다.
ASP.NET	• .NET Framework: https://www.microsoft.com/net • 동적 웹페이지 구축을 돕는 웹 애플리케이션 프레임워크로써, Content-Type 속성파일의 MIME-Type 형식을 가져와 허용할 파일 확장명을 지정할 수 있다.

다. 연관된 구현단계 보안약점 항목

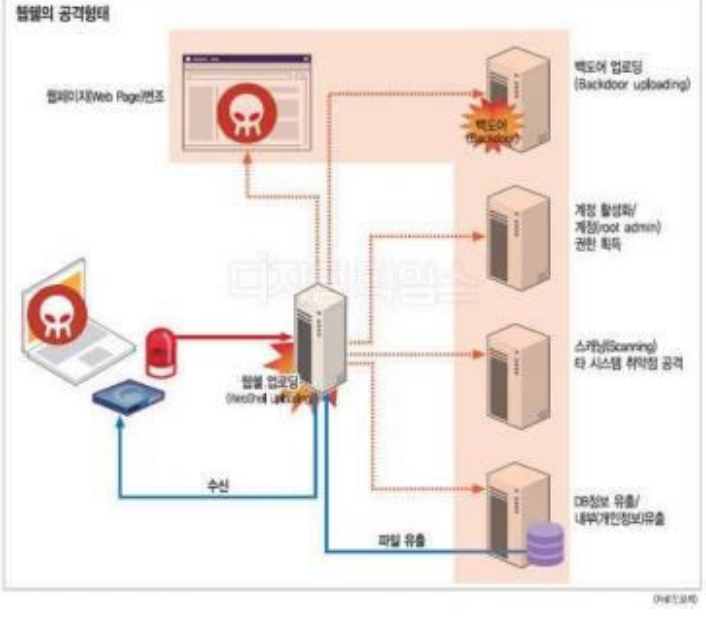
유형	보안약점 항목
입력데이터 검증 및 표현	위험한 형식 파일 업로드
입력데이터 검증 및 표현	무결성 검사 없는 코드 다운로드
보안기능	부적절한 전자서명 확인

라. 사고사례

무심코 연 첨부파일, 해킹에 '속수무책' (2014-06-10, 디지털타임스)

일반적으로 웹 게시판이나 자료실에는 사진이나 문서를 올리는 '파일첨부' 기능이 포함되어 있습니다. 이 때, txt, jpg, doc와 같은 데이터 파일종류 이외에 악의적으로 제작된 스크립트 파일인 '웹쉘(Web-Shell)'을 업로드 해 사이버 공격을 하는 사고가 빈번히 발생합니다.

웹쉘이란 공격자가 원격에서 대상 웹서버에 명령을 수행할 수 있도록 작성한 웹 스크립트(asp, jsp, php, cgi) 파일을 말합니다. zip, jpg, doc와 같은 데이터 파일종류 이외에 악의적으로 제작된 스크립트 파일인 웹쉘을 정상 파일처럼 숨겨 업로드 해 웹서버를 해킹하는 것이지요. 최근에는 파일 업로드뿐만 아니라 SQL삽입(Injection)와 같은 웹 취약점을 공격한 후 지속적으로 피해시스템을 관리할 목적으로 웹쉘을 생성해 업로드 하는 고도의 해킹도 발생하고 있습니다.



… 한 해 동안 분석했던 사이버공격 피해 웹서버 중 웹쉘이 발견된 웹서버는 총 91%에 달하기도 했습니다. …

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, https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices

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-implement-secure-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

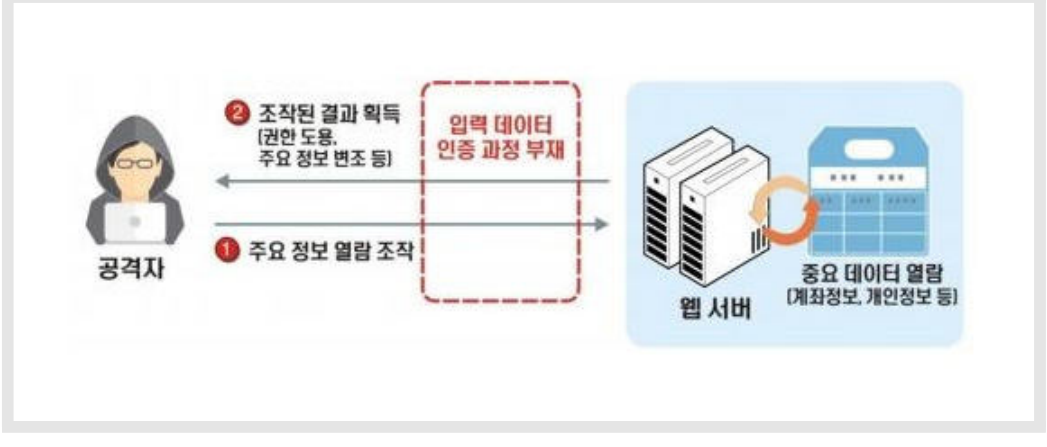
2.1 인증 대상 및 방식

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(2factor)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. 설계 시 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 설계 시 중요기능을 분류하고 식별된 중요기능에 대해 일괄적으로 인증을 요구하도록 시스템이 구성되도록 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 패스워드 사용 시

일회용 패스워드를 적용하는 경우 타서비스나 시스템과의 연동이 보장되도록 설계해야 한다. 일회용 패스워드를 도입하는 경우 다음과 같은 규칙을 적용하여 설계한다.

A. 일회용 패스워드는 시각정보, 이벤트정보, 질의-응답 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 패스워드에는 시간적 제한을 설정해야 한다(금융영역에서는 30~60초).

D. 일회용 패스워드는 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 전자서명 인증기술을 이용한 2단계 인증을 고려해야 한다.

2단계인증은 Type1(패스워드/PIN 등 지식기반인증), Type2(토큰/스마트카드 등 소유기반

인증), 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 3-10 인증기능을 제공하는 프레임워크

개발환경	활용가능한 프레임워크 또는 라이브러리
Java	• Spring Security: http://spring.io/ • 인증, 인가 등 기업어플리케이션의 보안기능을 제공하는 Java/Java EE 프레임워크이다. • Apache Software License v2.0 정책에 따라 자유롭게 사용가능하다.
ASP.NET	• .NET Framework: https://www.microsoft.com/net • 동적 웹페이지 구축을 돕는 웹어플리케이션 프레임워크로써, 사용자 인증 및 권한 부여를 위한 확장 가능한 프레임워크를 제공한다.
PHP	• Zend Framework: http://framework.zend.com/ • 객체지향 웹어플리케이션을 위한 프레임워크로써 사용자 인증모듈을 제공한다. New BSD License에 따라 사용할 수 있다.
	• Symfony: http://symfony.com/legacy • 재사용가능한 PHP 컴포넌트/라이브러리를 포함한 웹 어플리케이션 프레임워크이다. MIT License에 따라 사용할 수 있다.

다. 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

라. 사고사례

검색엔진 인증우회 못 막아, 관리자 인식부족 노출 방지 (2007-01-30, 한겨레)

무려 452곳에 이르는 공공기관의 홈페이지에 5400여건의 개인정보가 노출된 사건은 공공기관 홈페이지의 시스템 결함과 더불어 개인정보의 중요성에 대한 해당 공공기관들의 인식부족 탓인 것으로 밝혀졌다.

> 보안전문가들은 공공기관 사이트들이 로그인 절차를 갖추고 있음에도 개인정보가 노출된 것은 인증·권한확인 절차의 누락이 원인이라고 설명했다. OO처럼 비공개 문서의 내려받기가 가능했던 것도 홈페이지 내에 파일처리에 관한 인증·권한확인 절차를 두지 않았기 때문인 것으로 추정했다.

OO사의 전임컨설턴트는 "기술적인 측면에서 게시판의 '사용자 모드'로 내부 관리자만 접근이 가능하도록 하거나 쓰기, 수정, 보기 등의 각 단계마다 일일이 보안을 걸어둬야 한다"며 "특히 관리자와 게시자만 볼 수 있게 만든 게시판의 경우에는 더욱 그렇다"고 말했다.

OO 홈피 보안 허술… 비용 줄이려 보안코드 무시? (2016-10-12, JTBC)

OO '고객의 소리' 게시판에 글을 남기기 위해선 인증 과정을 거쳐야합니다.

> 이렇게 접속하는 방법을 정상 접근이라고 하는데, OO은 정상 접근에 대한 인증 절차만 만들고 인터넷 주소에 나타나는 고유 ID 숫자를 바꿔 접속하는 우회 접근에 대한 보안은 허술했습니다.
> [보안업체 관계자: 홈페이지를 요청하는 사람이 인증된 사람이냐를 확인해 줄 수 있는 3줄 정도의 시큐어코딩만 들어갔어도 이런 문제가 생기지 않는데…]

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.mmmphp/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/digital-authentication-the-basics

⑥ 일회용 패스워드(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&totalSu=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

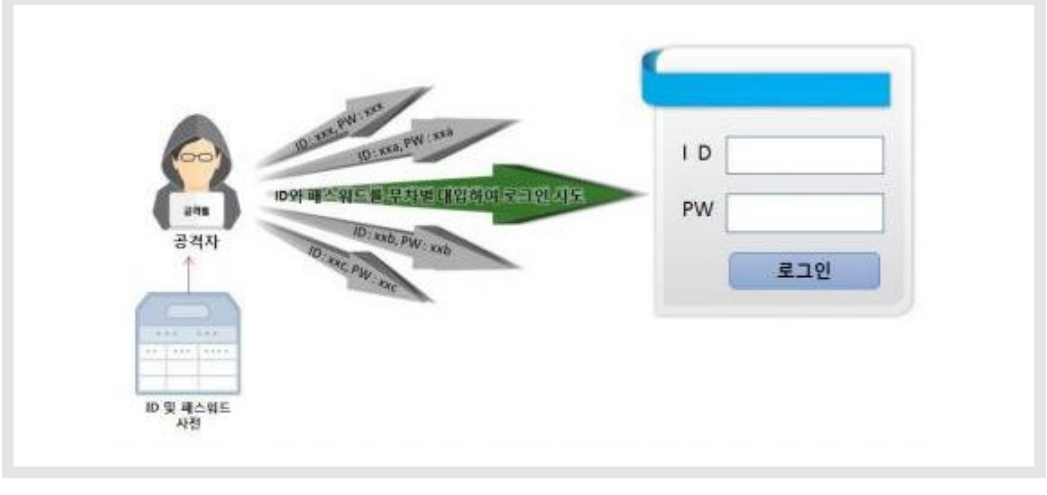
2.2 인증 수행 제한

유형	보안기능
설계항목	인증 수행 제한
설명	반복된 인증 시도를 제한하고 인증 실패한 이력을 추적하도록 설계해야 한다.
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 패스워드 무차별 대입 공격이 시도되어 계정정보가 노출될 수 있다.

그림 3-18 반복된 인증시도 제한 기능 부재



나. 설계 시 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 시간정보, 로그아웃한 시간정보 등을 저장할 수 있도록 설계하고 일정횟수 이상 연속적으로 로그인 실패 시 사용자ID와 패스워드 외의 추가적인 정보를 확인하도록 한다.

계정정보 입력 시 자동입력 방지문자와 같은 장치를 마련하도록 설계한다.

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

표 3-11 CAPTCHA 기능을 제공하는 서비스 및 솔루션

개발환경	활용가능한 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. 보안문자를 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. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	반복된 인증시도 제한기능 부재

라. 사고사례

유명 만화사이트, 로그인 취약점으로 개인정보 유출 위험 (2016-04-12, 데일리시큐)

모 유명 만화 사이트의 로그인 취약점으로 인해 평문 비밀번호가 그대로 노출되고 있으며 이로 인해 회원들의 개인정보 유출로 이어질 수 있어 신속한 보안조치가 필요한 상황이다.

> 해당 취약점을 발견하고 데일리시큐에 제보한 OO제보자는 "해당 만화 사이트는 회원가입 시 아이디를 요구하지만 비밀번호 입력 횟수에 제한이 없기 때문에 부르트포스(Brute force) attack(무차별대입공격)으로 불법접근이 가능할 수 있다"며 "해당사이트에서 아이디로 사용되는 이메일 주소는 블로그나 SNS 등을 이용해 알아낼 수 있기 때문에 손쉽게 계정탈취로 이어질 수 있어 위험한 상황"이라고 설명했다.

제보자는 비밀번호 입력 횟수에 제한을 둬 무차별대입공격으로 인한 개인정보가 유출을 차단해야 한다"고 덧붙였다.

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 Requirements, 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/Authentication_Cheat_Sheet#Prevent_Brute-Force_Attacks

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

2.3 비밀번호 관리

유형	보안기능
설계항목	비밀번호 관리
설명	비밀번호 생성규칙, 저장방법, 변경주기 등 비밀번호 관리 정책별 안전한 적용방법을 설계해야 한다.
보안대책	① 비밀번호 설정 시 한국인터넷진흥원『패스워드 선택 및 이용 안내서』의 패스워드 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

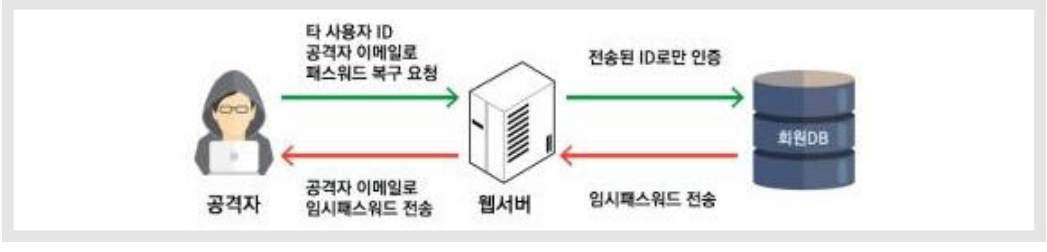
회원가입 시 안전한 비밀번호 생성규칙이 적용되지 않아서 취약한 비밀번호로 회원가입이 가능할 경우 무차별 대입 공격으로 패스워드가 누출될 수 있다.

그림 3-19 취약한 비밀번호 허용

사례2: 취약한 비밀번호 복구

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

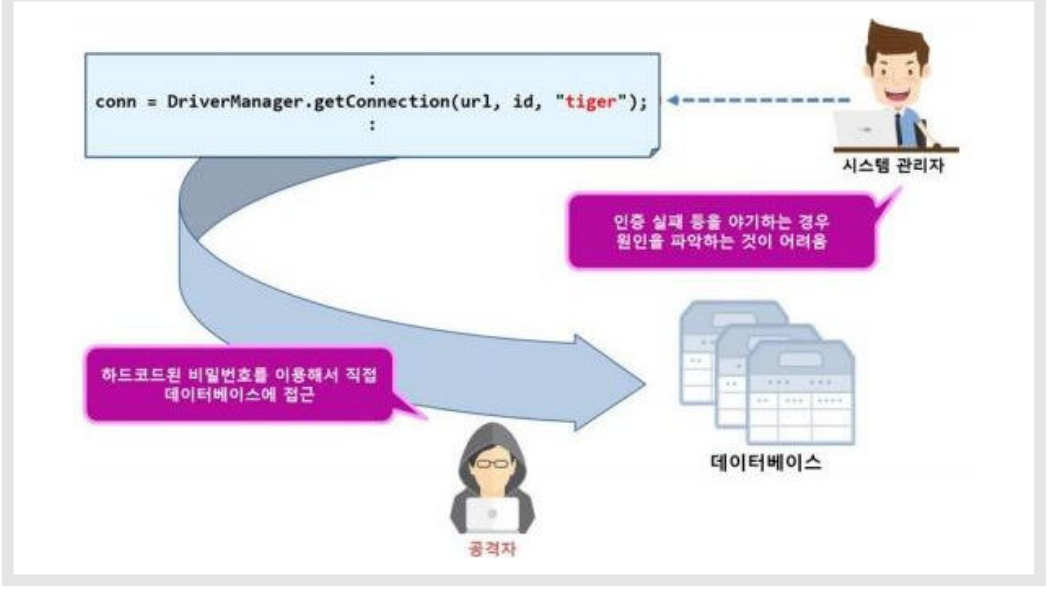
그림 3-20 취약한 비밀번호 복구



사례3: 하드코드된 비밀번호

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 야기하는 경우, 시스템 관리자가 그 실패의 원인을 파악하기 쉽지 않다.

그림 3-21 하드코드된 비밀번호



나. 설계 시 고려사항

① 비밀번호 설정 시 한국인터넷진흥원『패스워드 선택 및 이용 안내서』의 안전한 패스워드 생성규칙을 적용 한다.

표 3-12 안전한 패스워드 생성규칙

구분	안전한 패스워드	안전하지 않은 패스워드
설명	• 두 종류 이상의 문자구성과 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 패스워드 ※ 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 패스워드 • 이용자 ID를 이용 ※ 이용자 ID가 'kisa'일 경우 패스워드를 'kisa1' 등으로 설정 • 한글, 영어 등 사전적 단어로 구성(예, '바다나라', 'love12' 등) • 특정 인물이나 널리 알려진 단어를 포함하는 패스워드 ※ 사이트, 기업명, 연예인 이름 등의 특정 명칭을 의미 • 숫자와 영문자를 비슷한 문자로 치환한 형태로 구성 ※ 영문자 'O'을 숫자'0'으로, 영문자 'I'를 숫자 '1'로 치환 등 • 기타 ※ 시스템에서 초기 설정되거나 예제로 제시된 패스워드 ※ 한글의 발음을 영문으로, 영문단어의 발음을 한글로 변경한 형태의 패스워드

[표 3-13]과 같은 솔루션 사용을 고려하여 사용자가 안전한 비밀번호 사용을 유도한다.

표 3-13 패스워드 안전성 검증 솔루션

개발환경	활용가능한 프레임워크 또는 라이브러리
PHP, JSP, ASP	• 패스워드 안전성 검증 S/W: http://seed.kisa.or.kr/iwt/ko/sup/EgovSafePwdLb.do • 웹 사이트의 회원가입, 패스워드 변경 및 로그인 메뉴에서 사용자가 입력한 패스워드에 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 서버간의 통신 등 인터넷과 같은 공중망 환경에서는 패스워드와 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, 안전한 암호 알고리즘을 적용하는지 등을 확인해야 한다.

- TLS 최신 버전(예: TLS 1.2 및 TLS 1.3)만 지원하고, 한국인터넷진흥원『암호 알고리즘 및 키 길이 이용 안내서』를 참고하여 안전한 암호 알고리즘 조합 적용

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

비밀번호는 복호화 되지 않는 일방향 해시 함수를 사용해서 암호화하여 저장해야 한다.

[참고 3] 일방향 해시 함수

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 해시 함수의 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 패스워드가 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, 비밀번호 변경 시 이전의 비밀번호와 연관성이 없어야 한다.

(ㄴ) 비밀번호 재설정

비밀번호를 잊어버렸거나 분실하는 경우 비밀번호 재설정이 필요하다. 이 경우에 “비밀번호 찾기” 기능을 사용해야 한다면 이메일과 질의-답변으로 반드시 해당 사용자에게만 비밀번호 정보를 전달해야 하며 올바른 비밀번호가 입력되기 전에는 이메일 등 개인정보를 수정할 수 없도록 해야 한다. 질의-답변 검증 시 일정횟수 이상 정답을 맞히지 못하면 비밀번호 찾기 기능을 사용하지 못하도록 설정해야 한다. 검증 후 기존의 비밀번호가 아닌 임시비밀번호를 발급하도록 설계해야 하며 사용자는 임시 비밀번호를 발급받은 즉시 새로운 비밀번호로 재설정해야 한다.

시스템은 언제든지 사용자가 자신의 비밀번호를 변경할 수 있게 비밀번호 변경기능을 제공하도록 설계한다.

⑤ 비밀번호 관리 규칙을 정의해서 적용해야 한다.

안전한 비밀번호 관리를 위해 다음과 같은 항목을 고려할 수 있다.

(ㄱ) 변경주기

비밀번호는 3개월(또는 6개월)마다 주기적으로 변경하도록 해야 한다.

(ㄴ) 만료기간 설정

일정기간 시스템 사용자에 대해서는 비밀번호 만료기간을 설정해야 한다.

사용자 테이블에 개인정보 변경주기를 추가한 뒤 일단위로 해당 필드가 업데이트되도록 한다. 비밀번호 기간이 만료되면 로그인 시 사용자에게 비밀번호 변경을 요청하고, 비밀번호 변경 시 비밀번호 변경주기를 초기화하도록 시큐어코딩 규칙을 정의한다.

(ㄷ) 성공한 로그인 시간 관리

마지막으로 성공한 로그인 시간 정보를 관리해야 한다. 사용자 테이블에 마지막으로 로그인한 시간정보를 저장하고 사용자에게 알림으로써 계정도용 여부를 점검할 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 설치 시 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. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	하드코드된 중요정보
	취약한 비밀번호 허용

라. 사고사례

가장 털리기 쉬운 패스워드 100선 공개, '내 비밀번호' 있나 확인하세요 (2015-01-21, 아시아경제)

가장 털리기 쉬운 패스워드 100선이 공개됐다.
미국 패스워드관리업체 OO는 '2014년 최악의 패스워드 25개' 명단을 발표했다. 이는 작년 한 해 동안 패스워드를 포함한 계정 정보가 인터넷에 유출된 사례 300만건을 조사한 결과다.
조사결과 '123456'과 'password'가 2년 연속으로 각각 1,2위를 차지했다. 3~7위는 각각 '12345', '12345678', 'qwerty', '234567890', '1234' 등 자판에서 연속으로 나오는 키를 누르는 조합이었고, '12345678'(11위)도 마찬가지였다.

똑같은 숫자를 여러 차례 누르는 경우도 '111111'(15위), '696969'(22위), '123123'(23위) 등 흔했다. 'abc123'(14위)은 자판에서는 연속이 아니지만 알파벳 순서상으로 연속인 글자와 숫자를 누르는 조합이었다. 이외에 운동경기나 스포츠팀 이름, 사람 이름, 네 자리로 된 1990년 전후의 출생년도 등이 100위권 내에 많이 포함됐다.

OO는 "흔한 패스워드를 사용할수록 해커가 당신의 이메일이나 은행계정에서 정보를 빼내기가 쉬워진다"며 이런 패스워드를 사용하지 않는 것이 바람직하다고 조언했다.

OO 오픈마켓에서 물건 팔려다 '탈탈' (2013-10-11, 보안뉴스)

5인터넷 종합쇼핑몰인 OO에서 이름, 아이디, 메일주소만 알면 쉽게 해킹이 가능해 문제가 됐다.

> 해당 문제점을 제보한 OO씨는 "비밀번호를 찾다가 비밀번호를 찾는 방법 중 '등록한 e-mail 주소로 비밀번호 찾기' 부분에서 해킹의 위험이 있는 것을 발견해 제보하게 됐다"며, "'등록한 e-mail 주소로 비밀번호 찾기' 기능은 이름, 아이디, 메일주소만 알면 누구나 쉽게 임시비밀번호를 발급받을 수 있어 문제가 된다"고 전했다.

또한, 그는 "OO의 판매자의 경우에는 이름과 메일주소가 그대로 노출돼 있고, 메일주소로 아이디를 유추할 수 있어 더욱 심각하다"며, "새로운 이메일주소로 임시 비밀번호를 발급하는 기능을 제거하거나 다른 방식으로 사용자 본인 확인 후 다른 이메일로 비밀번호를 전송할 수 있도록 해야 한다"고 전했다.

▲

아이디, 이름, 등록한 이메일 주소만 알면 본인확인이 안된 다른 메일주소로 임시비밀번호를 받을 수 있음

마. 참고자료

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

② OWASP Top 10 - A07:2021 Identification and Authentication Failures, OWASP, https://www.owasp.org/www-project-top-ten

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

⑦ Password Storage Cheat Sheet, OWASP, http://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

2.4 중요자원 접근통제

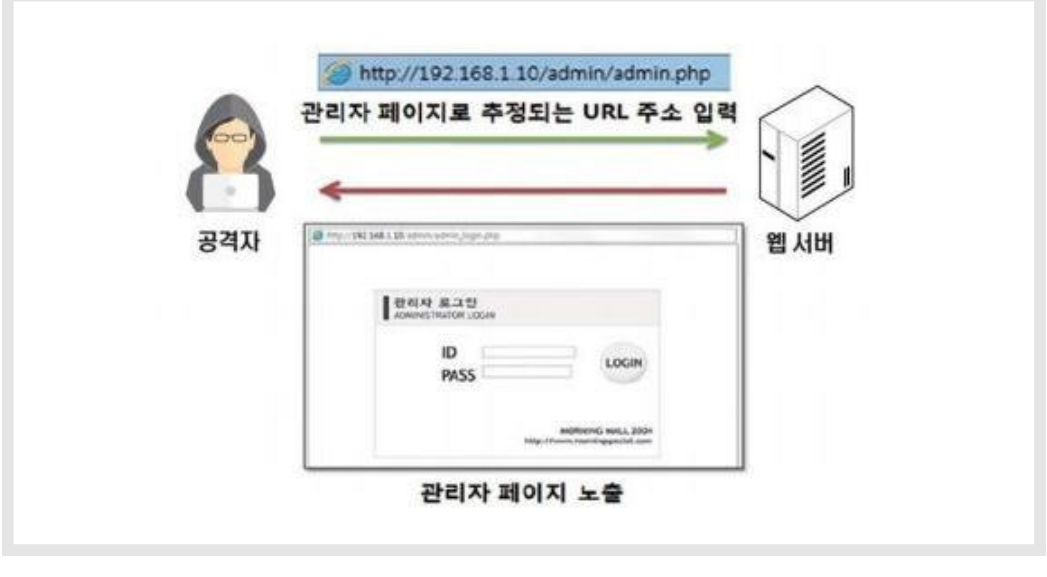
유형	보안기능
설계항목	중요자원 접근 통제
설명	중요자원(프로그램 설정, 민감한 사용자 데이터 등)을 정의하고, 정의된 중요자원에 대한 접근을 통제하는 신뢰할 수 있는 방법(권한관리 포함) 및 접근통제 실패 시 대응방안을 설계해야 한다.
보안대책	① 중요자원에 대한 접근통제 정책을 수립하여 적용해야 한다. ② 중요기능에 대한 접근통제 정책을 수립하여 적용해야 한다. ③ 관리자 페이지에 대한 접근통제 정책을 수립하여 적용해야 한다.

가. 취약점 개요

사례1: 관리자 페이지 노출

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

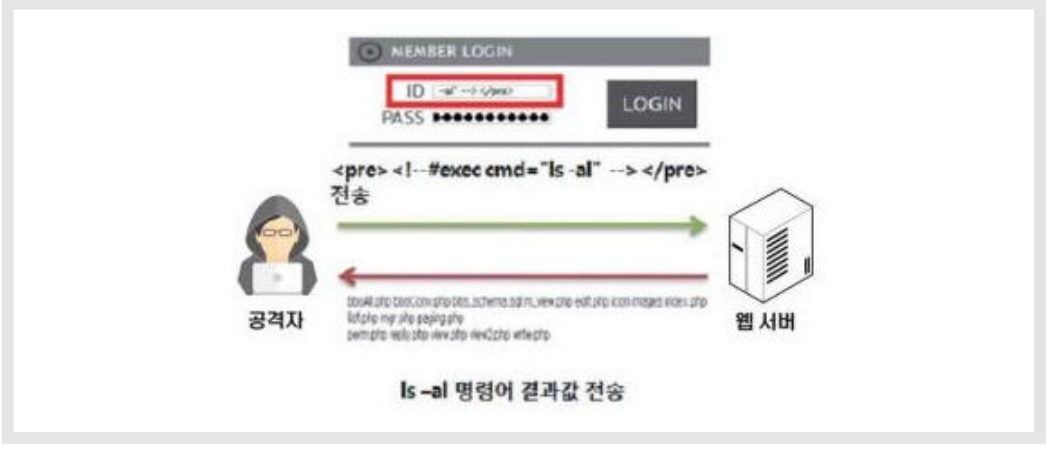
그림 3-22 관리자페이지 노출



사례2: SSI 삽입

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

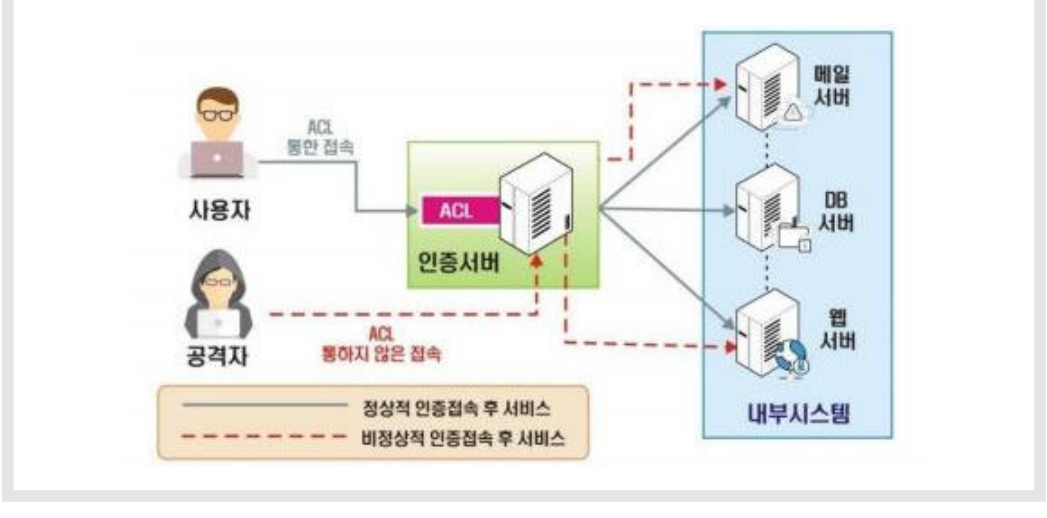
그림 3-23 SSI 인젝션



사례3: 부적절한 인가

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

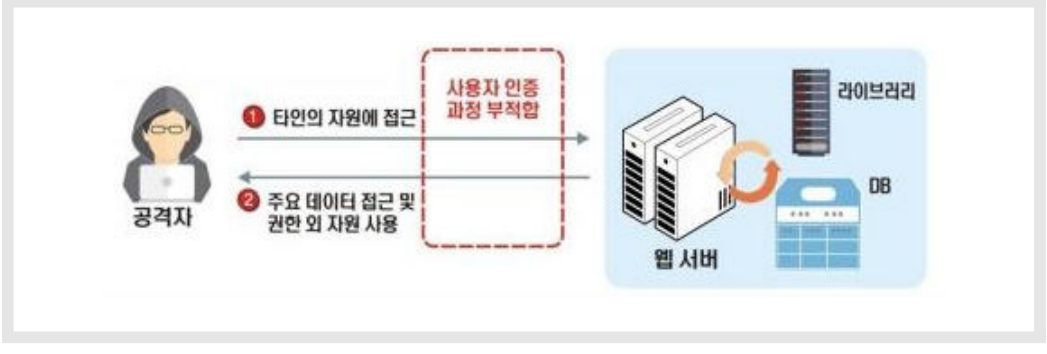
그림 3-24 부적절한 인가



사례4: 중요자원에 대한 잘못된 권한 설정

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

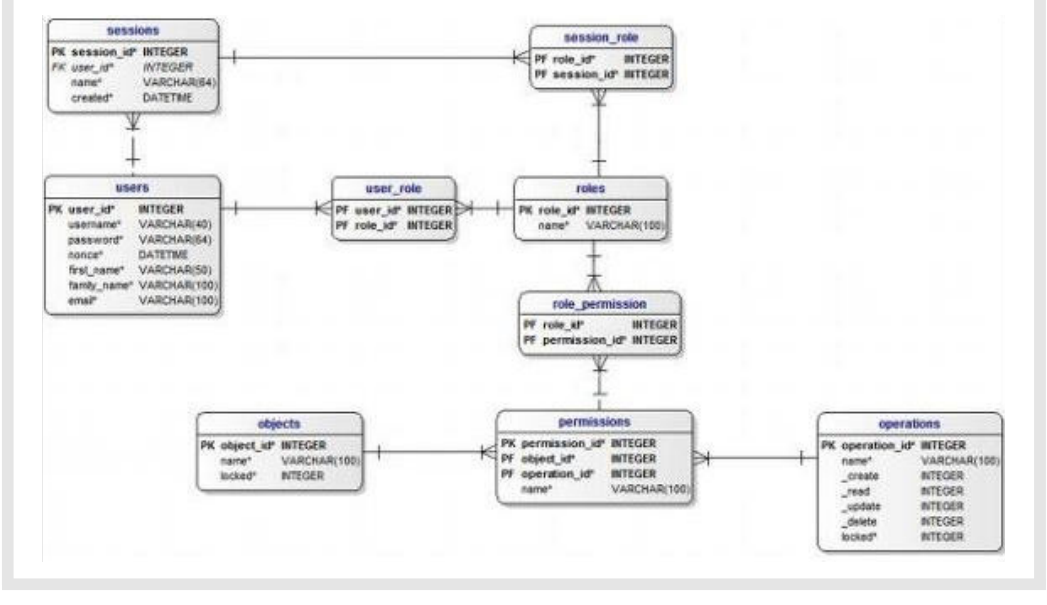
그림 3-25 중요한 자원에 대한 잘못된 권한 설정



나. 설계 시 고려사항

RBAC(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–)

예를 들면, Spring Security 프레임워크 사용 시 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. 메소드 호출에 앞서, 각 사용자가 객체에 대해 특정 작업을 수행할 권한이 있는지 검증한다.
3. 메소드 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 접근 통제 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(1)의 원칙과 권한분리(2) 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 접근 통제는 소프트웨어를 익명·일반·특권사용자와 관리자 영역으로 구분하여 역할기반 접근통제(RBAC) 정책 및 비즈니스 로직에 따라 접근통제가 실시되도록 다음과 같은 조건에 따라 설계한다.

- 중요기능에 대한 접근권한은 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 수행 하고 수행종료 즉시 원상 복귀한다.

③ **관리자 페이지에 대한 접근 통제 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.
- 관리자페이지에 접속 시 추가인증을 요구하도록 해야 한다.

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

표 3-15 접근제어를 제공하는 프레임워크 및 라이브러리

개발환경	활용가능한 프레임워크 또는 라이브러리
Java	• Spring Security: http://spring.io/ • 인증, 인가 등 기업어플리케이션의 보안기능을 제공하는 Java/Java EE 프레임워크이며, 접근제어리스트(ACL) 및 역할기반접근제어(RBAC)으로 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 Software License v2.0 정책에 따라 자유롭게 사용가능하다.

다. 연관된 구현단계 보안약점 항목

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

라. 사고사례

[OO 개인정보 유출] OO, 같은 수법에 왜 또 당했나? (2014-03-06, 아주경제)

2년전 OO는 정보 유출 프로그램을 제작, 고객정보시스템을 조회하는 것처럼 꾸며 소량으로 고객정보를 가져가는 식으로 5개월간 870만 건의 개인정보를 빼냈다.

> 6일광역수사대에 따르면 전문 해커 김모씨와 정모씨 등은 OO 홈페이지를 해킹해 OO 홈페이지에 로그인 후 개인정보를 빼내왔다. 이들은 홈페이지 이용대금 조회란에 고유숫자 9개를 무작위로 자동 입력시키는 이 프로그램으로 OO 가입고객의 9자리 고유번호를 맞춰 개인정보를 탈취했다.

성공률이 높을 땐 하루 20만~30만 건의 개인정보를 탈취하는 등 최근 1년간 1천200만 명의 고객정보를 털었다. 이들이 확보한 개인정보는 이름, 주민등록번호, 휴대전화번호, 집주소, 직업, 은행계좌 등이다.

'OO 경영포털' 해외 해커 침입… 보안 취약 (2013-10-10, 보안뉴스)

'OO 사이트(www.OO.net)'가 해외 해커에 의해 침입 당한 흔적과 함께 디렉토리 리스팅 취약점과 파일 업로드 취약점이 발견됐다.

▲ 디렉토리 내부의 모든 파일이 보여지는 디렉토리 리스팅 취약점.

> 이 취약점은 디렉토리는 물론, 내부의 모든 파일들이 보이게 되어 공격자는 웹 어플리케이션의 구조를 파악해 민감한 정보가 포함된 설정 파일을 조회하거나 웹에 게시하지 않은 각종 파일을 유출할 수 있다.

OO 씨는 "OO 경영포털 사이트의 디렉토리 리스팅 취약점은 웹서버 관리 미흡으로 발생하는 취약점으로 공격자는 이를 이용하여 웹 애플리케이션의 구조를 파악하고 민감한 데이터를 조회하거나 추가적인 공격방법을 구상할 수 있다"면서 "이에 대응하기 위해서는 웹상에서 디렉토리 검색을 차단해야 한다."라고 밝혔다.

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/definitions/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

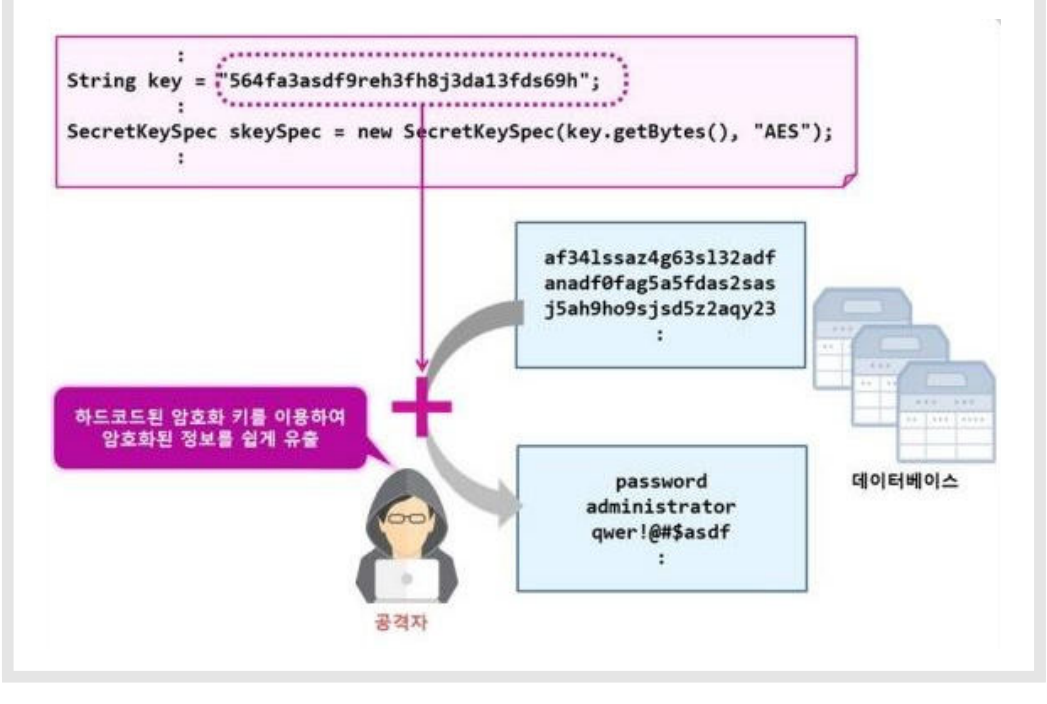
유형	보안기능
설계항목	암호키 관리
설명	암호키 생성, 분배, 접근, 파기 등 안전하게 암호키 생명 주기별 암호키 관리방법을 안전하게 설계해야 한다.
보안대책	① DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호이용안내서」에서 정의하고 있는 방법을 적용해야 한다. ② 설정파일(xml, Properties)내의 중요정보 암호화에 사용되는 암호키는 암호화해서 별도의 디렉터리에 보관해야 한다.

가. 취약점 개요

사례1: 하드코드된 암호키

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

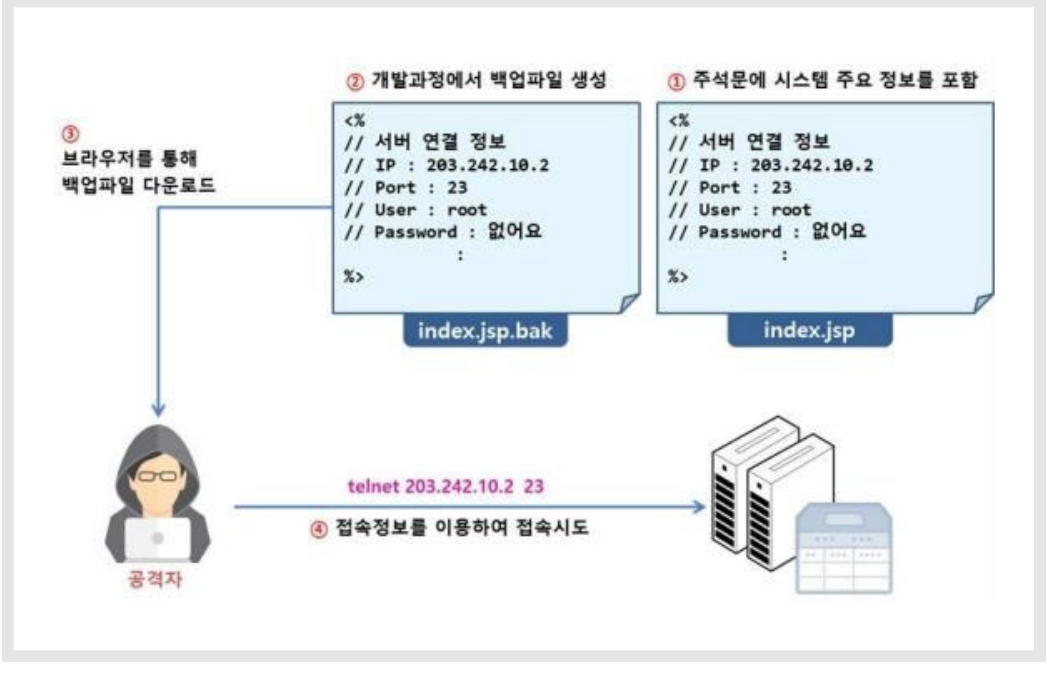
그림 3-26 하드코드된 암호화 키



사례2: 주석문 안에 포함된 암호키

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

그림 3-27 주석문 안에 포함된 시스템 주요정보



나. 설계 시 고려사항

① DB데이터 암호화에 사용되는 암호키는 한국인터넷진흥원의 「암호 키 관리 안내서」에서 정의하고 있는 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. 패스워드나 암호화키는 메모리에 저장하지 않는다.
4. 패스워드나 암호키가 메모리에 저장되어야 하는 경우 사용종료 후 메모리를 0으로 초기화 한다.

5. 암호키 생성 및 변경 시 암호키에 대한 백업기능을 구현한다.
6. 암호알고리즘에 사용되는 키 종류에 따라 사용 유효기한을 설정한다.

(ㄴ) 조직의 보호 목적에 따라 암호키 관리 수준을 지정

NIST에서 제정한 FIPS 140-2의 레벨로써, 조직의 보호목적에 따라 적절히 채택한다.

표 3-16 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 이상변화 시, 암호키를 삭제한다.

(ㄷ) 키 생명주기 기준 암호화 키 관리 프로세스를 구축

키 생성 – 암호화 키와 패스워드를 생성, 사용, 관리하는 사람 등을 명시하고 키를 생성하는데 사용하는 프로그램 등 어떠한 방법으로 생성하는지에 대한 절차를 명시한다.

키 사용 – 암호화 키와 패스워드를 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 패스워드를 안전하게 폐기하는 절차 및 요구사항 등을 명시한다.

(ㄹ) 키 복구 방안

사용자 퇴사 등으로 인해 사용자 이외의 사람에게 키 복구가 필요한 경우, 암호화 키는 정보보호담당자의 관리 하에 암호화 키 관리대장 등에서 복구하고, 패스워드는 정보보호담당자가 임시 패스워드를 발급하는 등 키 복구에 대한 방안을 마련하도록 한다.

(ㅁ) 암호키 사용 유효기간

암·복호화 키의 사용이 일정시간을 넘은 경우 사용자 인터페이스로 키 사용기간이 경과했음을 알리고 새로운 키 생성을 권장하도록 설계한다. [표3-17]은 NIST에서 권고하는 암호키 사용 유효기간이다.

표 3-17 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	
	서명용 개인키	1~3년	
	검증용 공개키	키 크기에 따라 다름	

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

설정파일내의 중요 정보 암호화에 사용된 암호키는 마스터키를 이용하여 암호화하여 별도의 디렉터리에 보관한다.

다. 연관된 구현단계 보안약점 항목

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

라. 사고사례

보안업체 코드서명 정보 유출, 파장은? (2016-02-22, 지디넷코리아)

보안업체가 인터넷에서 자사 프로그램(보안 모듈)을 배포하는데 쓰는 일종의 인감도장과 같은 성격의 코드 서명 정보가 유출돼 악성 프로그램 유포에 악용되는 사건이 벌어졌다.

> 코드서명은 인터넷에 프로그램을 유포하기 위해 자사가 만든 것이 맞다는 것을 증명하기 위해 일종의 인감도장을 찍는 것과 같다. 이 과정에서 사용되는 것이 인증서와 개인키다. 이 사건의 경우 금융권 등 고객사에 제공하는 일부 보안모듈을 우리가 개발한 것이 맞다고 증명하는데 필요한 인감도장 정보(개인키)가 유출됐다.

공격자는 이 정보로 코드서명을 거쳐 악성 프로그램을 배포했다. 마치 실제 보안회사가 배포하는 것처럼 위장했다.

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 암호 키 관리 안내서, 2014, 한국인터넷진흥원, http://seed.kisa.or.kr/

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 Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_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-coded_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

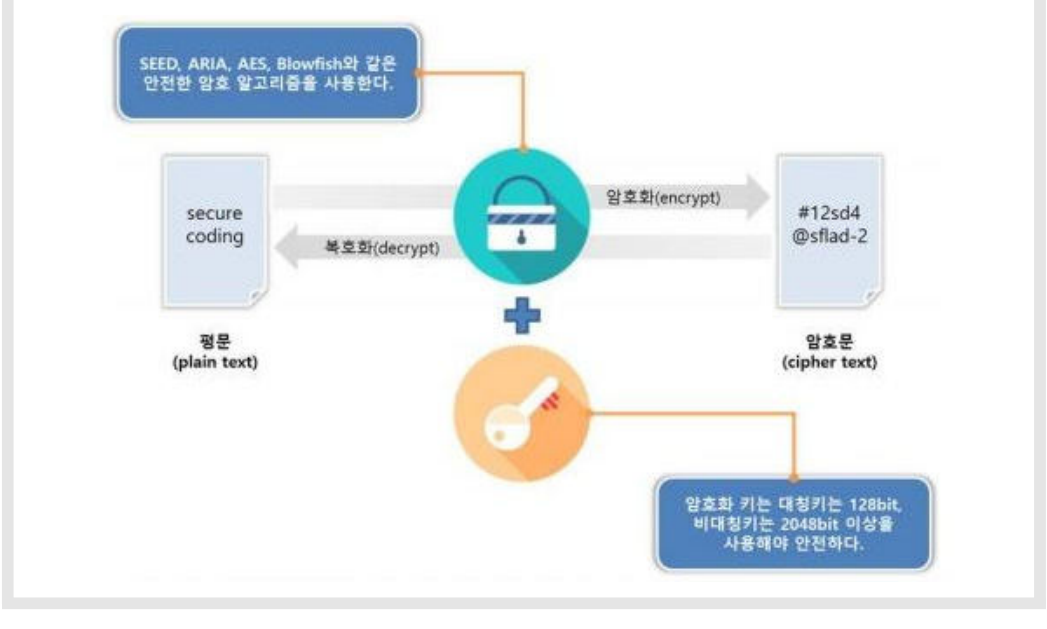
유형	보안기능
설계항목	암호연산
설명	국제표준 또는 검증필 프로토콜로 등재된 안전한 암호 알고리즘을 선정하고 충분한 암호키 길이, 솔트, 충분한 난수 값을 적용한 안전한 암호연산 수행방법을 설계해야 한다.
보안대책	① 대칭키 또는 비대칭키를 이용해서 암호화를 수행해야 하는 경우 한국인터넷진흥원의 『암호이용 안내서』에서 정의하고 있는 암호화 알고리즘과 안전성이 보장되는 암호키 길이를 사용해야 한다. ② 복호화 되지 않는 암호화를 수행하기 위해 해시함수를 사용하는 경우 안전한 해시 알고리즘과 솔트 값을 적용하여 암호화해야 한다. ③ 난수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

가. 취약점 개요

사례1: 취약한 암호알고리즘 사용

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

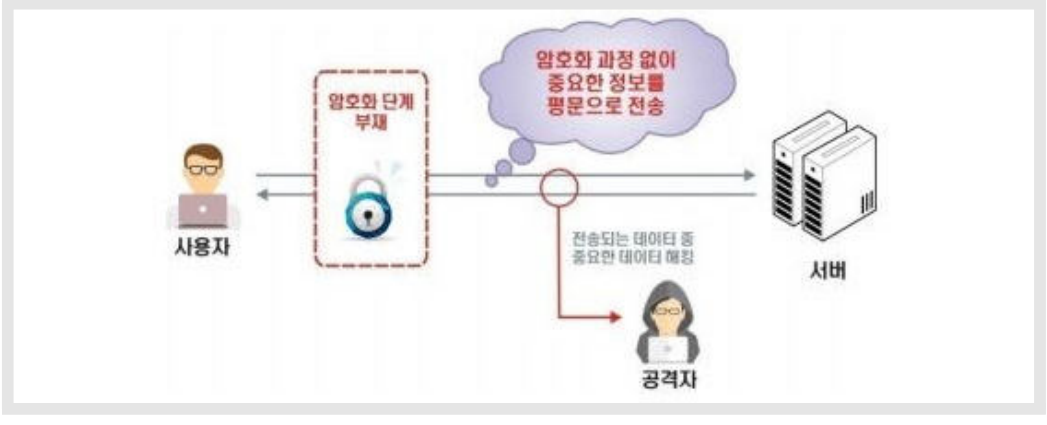
그림 3-28 취약한 암호화 알고리즘 사용



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 패스워드를 복호화 할 수 있게 된다.

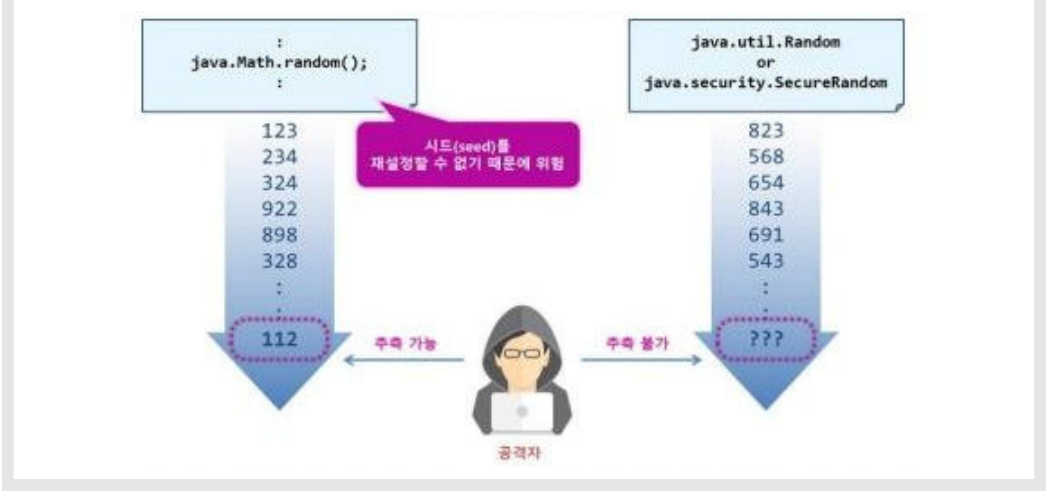
그림 3-29 충분하지 않은 키 길이 사용



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 해쉬함수

패스워드 저장 시 일방향 해쉬함수의 성질을 이용하여 패스워드의 해쉬값을 저장한다. 만약 패스워드를 솔트(Salt)없이 해쉬하여 저장한다면, 공격자는 레인보우 테이블과 같이 가능한 모든 패스워드에 대해 해쉬값을 미리 계산하고, 이를 이용한 전수조사로 패스워드를 찾을 수 있게 된다.

그림 3-31 솔트 없이 일방향 해쉬함수 사용

나. 설계 시 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 3-18 NIST 알고리즘 안전성 유지기간 및 최소 키 길이 권고

알고리즘 안전성 유지기간	보안 강도 (비트)	대칭키 암호 알고리즘 (보안강도)	비대칭키 암호 알고리즘 (비트)			
			인수분해 (ex.RSA)	이산대수 (ex. KCDSA)		타원곡선 암호 (ex. ECC)
				공개키	개인키	
2011년에서 2030년까지	112	112	2048	2048	224	224

알고리즘 안전성 유지기간	보안 강도 (비트)	대칭키 암호 알고리즘 (보안강도)	비대칭키 암호 알고리즘 (비트)			
			인수분해 (ex.RSA)	이산대수 (ex. KCDSA)		타원곡선 암호 (ex. ECC)
				공개키	개인키	
2030년 이후	128	128	3072	3072	256	256
	192	192	7680	7680	384	384
	256	256	15360	15360	512	512

표 3-19 국·내외 사용 권고 알고리즘

분류		국내(2018)	NIST(미국, 2015)
대칭키 암호 알고리즘 (블록암호)		SEED HIGHT ARIA LEA	AES 3TEDA
해시함수		SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512 LSH-224 LSH-256 LSH-348 LSH-512 LSH-512/224 LSH-512/256	SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256 SHA3-224 SHA3-256 SHA3-384 SHA3-512
공개키 암호 알고리즘	키 공유용	DH ECDH	DH ECDH MQV ECMQV
	암·복호화용	RSAES	RSA
	전자서명용	RSA-PSS KCDSA ECDSA EC-KCDSA	RSA DSA ECDSA

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

해시함수는 사용목적에 따라 메시지인증/키 유도/난수생성용과 단순 해시(메시지압축) /전자서명용으로 나눠지며, 사용목적과 보안강도에 따라 선택하여 이용한다.

③ 난수 생성 시 안전한 난수 생성 알고리즘을 사용해야 한다.

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

다. 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

라. 사고사례

인터넷뱅킹에도 쓰는 암호화 기술 보안 '우려' (2015-10-26, 지디넷코리아)

국내 주요 인터넷뱅킹 사이트를 포함해 대부분 암호화 통신을 제공하는 웹사이트가 지원하는 암호화 알고리즘(SHA1)이 이르면 올해 말부터 심각한 보안문제에 노출될 수 있는 것으로 나타났다.

> SHA1이 두 가지 서로 다른 정보를 입력했을 때, 같은 해쉬값을 만들어 낼 수 있는 시점이 온다는 것이다. 이를 악용한 공격을 '충돌공격(collision attack)'이라고 부른다.
> 이런 시점이 되면 더 이상 SHA1을 활용한 해쉬값은 안전하다고 보기 어렵다.

앞서 MD5라는 암호화 알고리즘 역시 SHA1과 같은 용도로 활용됐었지만 이란을 대상으로 한 미국, 이스라엘 첩보기관이 수행한 사이버첩보활동에 악용된 '플레임(Flame)' 악성코드가 이러한 MD5에 대한 충돌공격을 활용해 각종 정보를 수집하는데 악용됐다.

마. 참고자료

① CWE-327 Use of a Broken or Riscky Cryptographic Algorithm, MITRE, http://cwe.mitre.org/data/definitions/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 Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_VerificationStandard_Project

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

⑥ 암호 알고리즘 및 키 길이 이용 안내서, 2018, 한국인터넷진흥원, http://seed.kisa.or.kr/

2.7 중요정보 저장

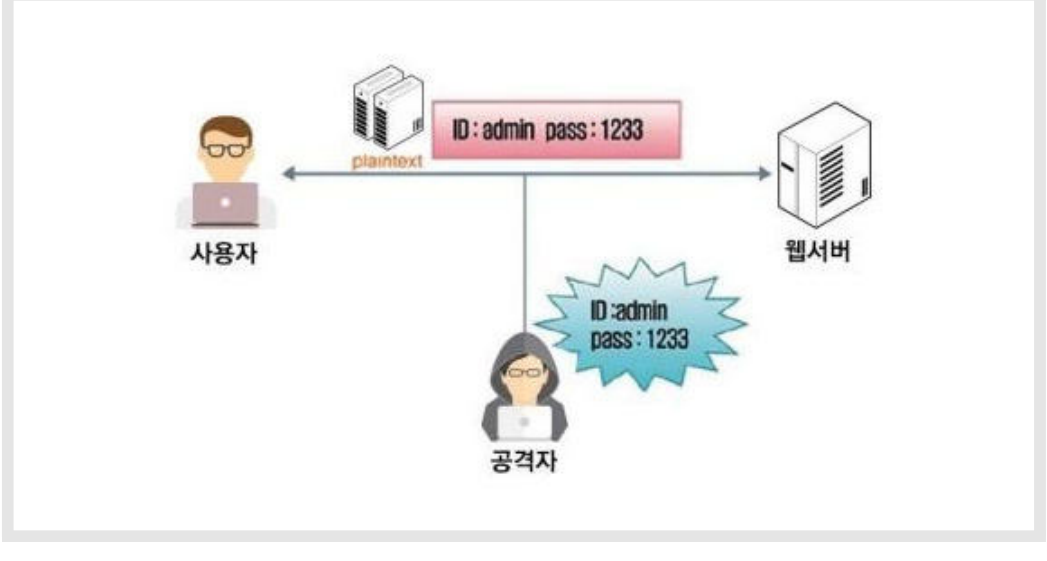
유형	보안기능
설계항목	중요정보 저장
설명	중요정보(비밀번호, 개인정보 등) 저장 시 안전한 저장 및 보관 방법을 설계해야 한다.
보안대책	① 중요정보 또는 개인정보는 암호화해서 저장해야 한다. ② 불필요하거나 사용하지 않는 중요정보가 메모리에 남지 않도록 해야 한다.

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.

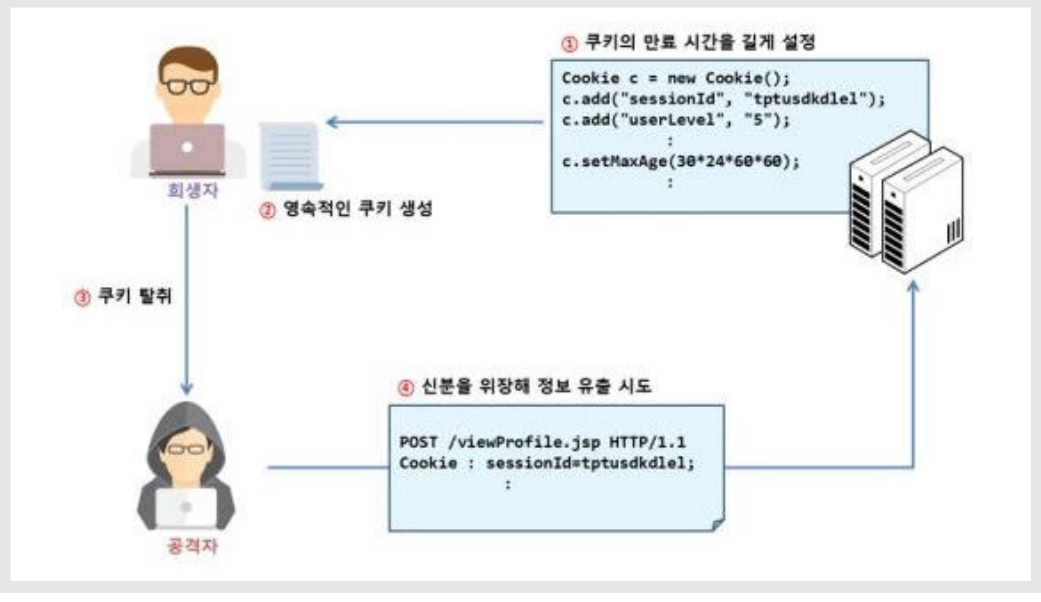
그림 3-32 **중요정보 평문저장**



사례2: 사용자 하드디스크에 저장된 쿠키를 통한 정보 노출

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. 설계 시 고려사항

① 중요정보 또는 개인정보는 암호화해서 저장해야 한다.

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

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

특히 쿠키, HTML5 로컬저장소와 같은 클라이언트 측 하드 드라이브에는 중요정보가 저장되지 않도록 설계해야 하며, 부득이 중요정보를 저장해야 하는 경우에는 반드시 클라이언트 측에 저장되는 민감 정보를 암호화한다.

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

② 중요정보가 메모리에 남지 않도록 해야 한다.

개인정보 또는 특정 금융정보를 처리하는 기능 구현 시 더 이상 필요하지 않은 데이터에 대해 메모리를 0으로 초기화하여 중요데이터가 메모리에 남지 않도록 시큐어코딩 규칙을 정의한다.

특히 민감한 정보를 포함하는 페이지는 사용자 측 캐싱을 비활성화 하도록 제한적인 캐시정책을 수립하여야 하며, 부득이 캐싱을 해야 하는 경우 캐싱되는 정보는 암호화하여 저장하도록 설계한다.

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

입력 폼 자동완성 비활성화

```
〈input type='text' name='id' autocomplete='off'/〉
〈input type='password' name='pw' autocomplete='off'/〉
```

입력 폼 자동완성 비활성화

```
〈form autocomplete='off'〉
```

다. 연관된 구현단계 보안약점 항목

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

라. 사고사례

OO카드, 주민번호 암호화 안해 (2014-01-23, 위클리오늘)

사상 최악의 개인정보 유출 사태를 촉발시킨 OO가 주민번호를 암호화 하지 않아 2차 피해에 심각하게 노출된 것으로 드러났다.

> 주민번호가 암호화돼 있으면 유출되더라도 도용할 수 없지만 이들 카드사들이 주민번호를 암호화하지 않으면서 스스로 사태를 확대시킨 셈이고 "2차 피해 가능성은 없다"는 카드사들의 주장은 신빙성을 잃어가고 있다.

이 실장은 "이들 카드사가 무분별하게 개인정보를 수집해놓고 관리에는 허술하게 해 개인정보 유출에 따른 피해는 고스란히 금융소비자가 떠안게 됐다"비난했다.

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

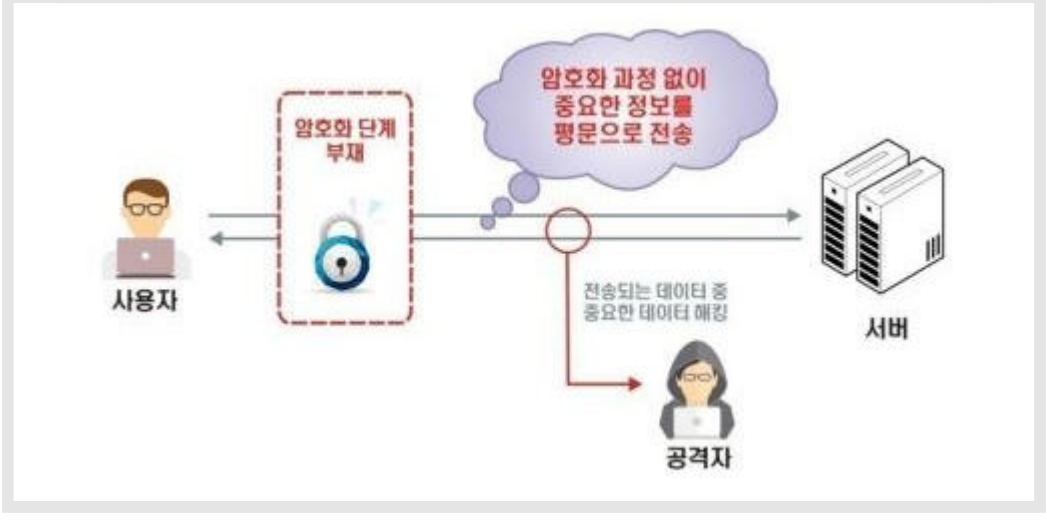
2.8 중요정보 전송

유형	보안기능
설계항목	중요정보 전송
설명	중요정보(비밀번호, 개인정보, 쿠키 등) 전송 시 안전한 전송방법을 설계해야 한다.
보안대책	① 인증정보와 같은 민감한 정보 전송 시 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. 설계 시 고려사항

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

분석단계에서 정의된 중요정보를 네트워크로 전송해야 하는 경우 안전한 암호모듈로 암호화 한 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 포함되어 전송되지 않도록 설계한다.

② 쿠키에 포함되는 중요정보는 암호화해서 전송해야 한다.

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

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
보안기능	암호화되지 않은 중요정보

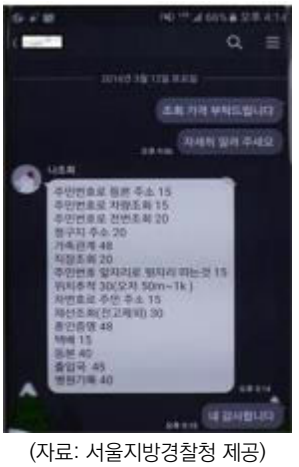
라. 사고사례

경찰, 이통사 위치정보 서버 해킹해 판매한 일당 검거 (2016-07-05, 보안뉴스)

국내 한 이동통신사의 휴대전화 위치정보 서버를 해킹해 이용자들의 개인정보를 판매하고 부당 이득을 챙긴 일당이 경찰에 붙잡혔다.

또한 이들에게 위치정보 추적과 미행 등을 의뢰한 의뢰인 34명도 불구속 입건했다. 의뢰인 34명 중 약 80%는 외도가 의심되는 배우자의 사생활 뒷조사를 위해 의뢰했고 기타 채권·채무자나 헤어진 여자친구의 소재를 파악해 달라는 의뢰도 있었다.

OO의 위치정보 서버는 위치정보를 암호화하지 않은 평문으로 전송한 것으로 논란이 될 것으로 보인다.



(자료: 서울지방경찰청 제공)

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_Protection_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

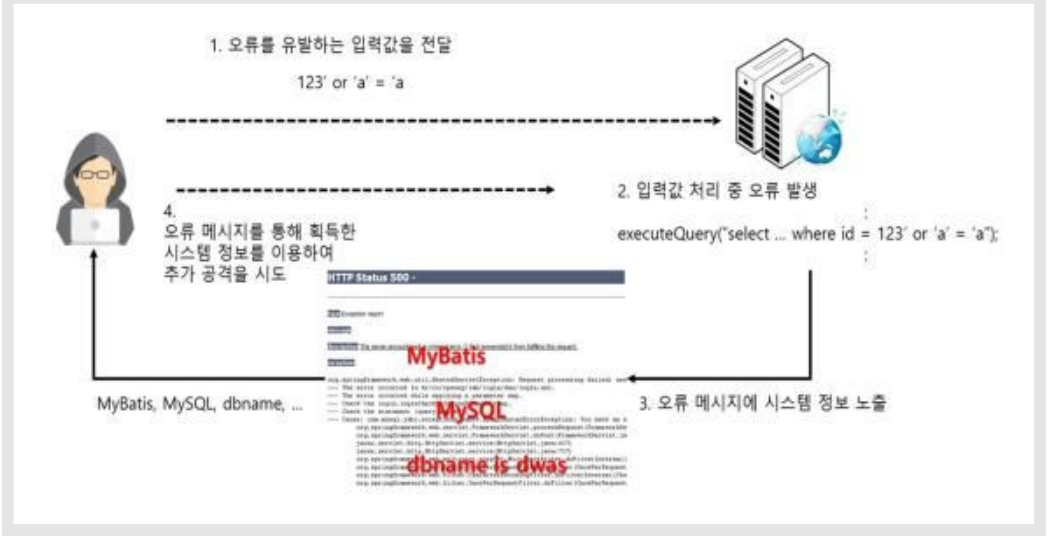
유형	보안기능
설계항목	예외처리
설명	오류메시지에 중요정보(개인정보, 시스템 정보, 민감 정보 등)가 노출되거나, 부적절한 에러 및 오류처리로 인하여 의도치 않은 상황이 발생하지 않도록 설계한다.
보안대책	① 명시적인 예외의 경우 예외처리 블럭을 이용하여 예외발생 시 수행해야 하는 기능이 구현되도록 해야 한다. ② 런타임 예외의 경우 입력 값의 범위를 체크하여 애플리케이션이 정상적으로 동작할 수 있는 값만 사용되도록 보장해야 한다. ③ 에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.

가. 취약점 개요

사례1: 오류메시지로 정보노출

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

그림 3-35 오류메시지로 정보노출



사례2: 시스템 정보 노출

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

그림 3-36 시스템 데이터 정보노출

나. 설계 시 고려사항

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

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

(예시) 자바 플랫폼 사용 시

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.

입력 값에 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 페이지를 사용자에게 에러 공지.

에러가 발생한 경우 프로그램 내에서 지정된 에러페이지로 리다이렉트 되도록 개발가이드 구현단계를 작성하거나, 웹 애플리케이션 서버 설정으로 특정 에러나 예외사항에 대해 지정된 페이지가 사용자에게 보일 수 있도록 설계한다.

(예시) web.xml 파일을 이용하여 에러 시 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 오류메시지에 중요정보가 포함되지 않도록 함.

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

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
에러처리	오류 메시지 정보노출

라. 사고사례

'오류 보고 메시지', 해커의 정보 획득 수단?
[보안뉴스] 2014-01-10 11:05 김경애 기자

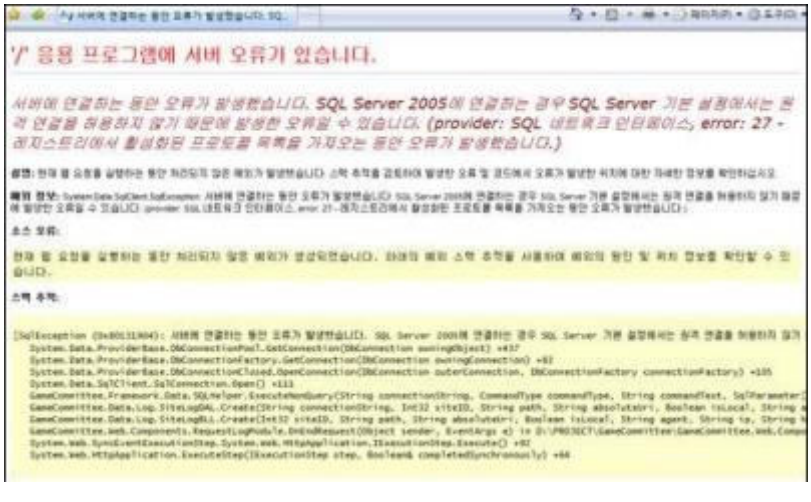
운영체제가 시스템 오류 발생, 하드웨어 변경 등의 상황을 OO 서버로 전송하기 위해 생성하는 윈도우 보고 메시지가 악의적 목적을 가진 공격자의 정보 획득 수단이 될 수 있다는 가능성이 제기돼 주목되고 있다.

> 인터넷침해대응센터는 10일 OO 오류 발생 시 생성되는 오류 보고 메시지에는 사용자의 하드웨어 및 운영체제 정보 등의 사용자 정보가 포함되어 있다고 밝혔다.

OO 오류 보고는 시스템 충돌 발생 시 자동으로 생성되며, 사용자 동의하에 인터넷으로 해당 보고서를 OO 서버로 전송할 수 있도록 구성된다. 보고서 내용에는 PC 모델명, PC ID, OS 버전, 문제를 유발시킨 프로그램명 등이 있다.

OO 홈페이지 해킹시도 당해 하루 종일 '먹통' (2008-12-31, 지디넷코리아)

정부산하 기관인 OO의 공식홈페이지가 해킹시도를 당해 서비스가 불가능한 상태인 것으로 확인 되었다.



OO 홈페이지의 해킹시도로 인해 20일 개최하기로 되어 있던 OO회의가 열리지 못하고 22일로 연기 되었다.

[참고] 위 사례의 오류페이지에서는 스택추적(StackTrace) 정보가 포함되어 있다. 서버에 오류 또는 예외가 발생한 경우 위와 같이 예외이름이나 스택추적정보를 출력하도록 되어있다면 공격자가 프로그램 내부구조를 쉽게 파악할 수 있기 때문에 공격자의 악성행위를 도울 수 있다는 위협이 발생한다.

마. 참고자료

① CWE-209 Information Exposure Through an Error Message, MITRE, http://cwe.mitre.org/data/definitions/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/definitions/754.html

④ Error Handling, Auditing and Logging, OWASP, http://www.owasp.org/index.php/Error_Handling,_Auditing_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%20Leakage

4. 세션통제

4.1 세션통제

유형	세션통제
설계항목	인증 대상 및 방식
설명	다른 세션 간 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

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

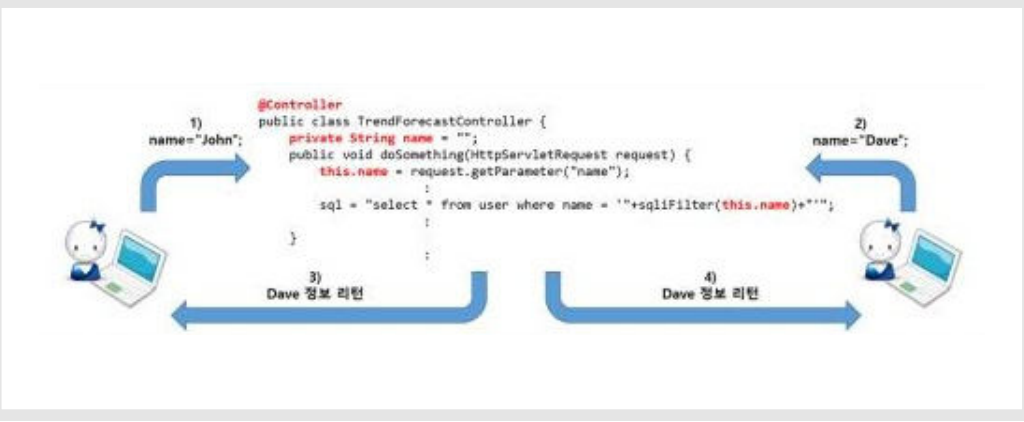
그림 3-37 불충분한 세션관리

사례2: 잘못된 세션에 의한 정보노출

다중 스레드 환경에서는 싱글톤(Singleton)객체 필드에 경쟁조건(race Condition)이 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 설계 시 읽고 쓰기가 가능한 변수를 사용하지 않도록 설계해야 한다.

② 세션이 안전하게 관리 되도록 설계해야 한다.

- 시스템 내의 모든 페이지에 대하여 로그아웃이 가능하도록 UI를 설계하고, 로그아웃을 요청하면 사용자에게 할당된 세션을 완전히 제거하는 API를 사용하도록 개발가이드 구현단계를 작성한다. Java의 경우 session.invalidate() 메소드를 사용하여 세션에 저장된 정보를 완전히 제거할 수 있다.
- 세션 타임아웃 시간은 중요기능의 경우 2~5분, 위험도가 낮은 경우에는 15~ 30분으로 설정하고, 이전 세션이 종료되지 않은 상태에서 새로운 세션이 생성되지 않도록 해야 한다.

- 웹 브라우저 종료로 인한 세션종료는 서버 측에서 인지할 수 없으므로, 일정시간 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 패스워드를 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 성공 시 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 재할당 하도록 설계한다.

다. 연관된 구현단계 보안약점 항목

유형	보안약점 항목
캡슐화	잘못된 세션에 의한 데이터 정보노출

라. 사고사례

온라인 뱅킹시 하이재킹 시도 맬웨어 발견 주의! (2012-03-05, 보안뉴스)

개인이 온라인 뱅킹 거래를 위해 사용자 계정에 로그인 했을 때 망 환경에서 사용자 간 또는 컴퓨터 간의 대화를 위한 연결과정인 세션을 하이재킹 하는 사례가 발견돼 주의가 요구된다고 영국 IT 전문지가 보도했다.

'Shylock'라고 명명된 이 맬웨어는 온라인 뱅킹 고객을 대상으로 고객이 계정을 로그인하면 세션을 하이재킹 할 수 있다.

이 맬웨어로 공격자는 실시간 채팅 창을 열고 은행 고객상담센터 담당자인 것처럼 속여 고객에게 세션이 보류됐다고 알린 후, 실시간 채팅으로 고객들의 정보를 빼내는 수법을 사용한 것으로 알려졌다.

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 Requirements, http://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_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

제4장 구현단계 시큐어코딩 가이드

제4장

구현단계 시큐어코딩 가이드

제1절 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 동적 쿼리(Dynamic Query)[4]를 생성하기 때문에 개발자가 의도하지 않은 쿼리가 생성되어 정보유출에 악용될 수 있다.

4 동적쿼리(Dynamic Query) : DB에서 실시간으로 받는 쿼리. Parameterized Statement가 동적 쿼리가 됨

나. 보안대책

PreparedStatement[5]객체 등을 이용하여 DB에 컴파일 된 쿼리문(상수)을 전달하는 방법을 사용한다. PreparedStatement를 사용하는 경우에는 DB 쿼리에 사용되는 외부 입력값에 대하여 특수문자 및 쿼리 예약어를 필터링하고, 스트러츠(Struts), 스프링(Spring) 등과 같은 프레임워크를 사용하는 경우에는 외부 입력값 검증모듈 및 보안모듈을 상황에 맞추어 적절하게 사용한다.

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

```
//외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
String gubun = request.getParameter("gubun");
.....
String sql = "SELECT * FROM board WHERE b_gubun = '" + gubun + "'";
Connection con = db.getConnection();
Statement stmt = con.createStatement();
//외부로부터 입력받은 값이 검증 또는 처리 없이 쿼리로 수행되어 안전하지 않다.
ResultSet rs = stmt.executeQuery(sql);
```

5 PreparedStatement : 컴파일된 쿼리 객체로 MySQL, Oracle, DB2, SQL Server 등에서 지원하며, Java의 JDBC, Perl의 DBI, PHP의 PDO, ASP의 ADO를 이용하여 사용가능

이를 안전한 코드로 변환하면 다음과 같다. 파라미터(Parameter)를 받는 PreparedStatement 객체를 상수 스트링으로 생성하고, 파라미터 부분을 setString, setParameter 등의 메소드로 설정 하여, 외부의 입력이 쿼리문의 구조를 바꾸는 것을 방지해야 한다.

안전한 코드의 예 JDBC API

```
String gubun = request.getParameter("gubun");
.....
//1. 사용자에 의해 외부로부터 입력받은 값은 안전하지 않을 수 있으므로, PreparedStatement
     사용을 위해 ?문자로 바인딩 변수를 사용한다.
String sql = "SELECT * FROM board WHERE b_gubun = ?";
Connection con = db.getConnection();
//2. PreparedStatement 사용한다.
PreparedStatement pstmt = con.prepareStatement(sql);
//3. PreparedStatement 객체를 상수 스트링으로 생성하고, 파라미터 부분을 setString 등의
     메소드로 설정하여 안전하다.
pstmt.setString(1, gubun);
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

```
〈?xml version="1.0" encoding="UTF-8" ?〉
〈!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN“
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"〉
.....
〈select id="boardSearch" parameterType="map" resultType="BoardDto"〉
//$기호를 사용하는 경우 외부에서 입력된 keyword값을 문자열에 결합한 형태로 쿼리에 반영되므로
  안전하지 않다.
  select * from tbl_board where title like '%$ {keyword }%' order by pos asc
〈/select〉
```

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
.....
<select id="boardSearch" parameterType="map" resultType="BoardDto">
//$ 대신 #기호를 사용하여 변수가 쿼리맵에 바인딩 될 수 있도록 수정하는 것이 안전하다.
  select * from tbl_board where title like '%'||# {keyword }||'%' order by pos asc
</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

```
import org.hibernate.Query
import org.hibernate.Session
.....
//외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
String name = request.getParameter("name");
//Hiberate는 기본으로 PreparedStatement를 사용하지만, 파라미터 바인딩 없이 사용 할 경우
  안전하지 않다.
Query query = session.createQuery("from Student where studentName = '" + name + "' ");
```

아래 코드와 같이 외부 입력값이 위치하는 부분을 ? 또는 :명명된 파라미터 변수로 설정하고, 실행시에 해당 파라미터가 전달되는 바인딩(binding)을 함으로써 외부의 입력이 쿼리의 구조를 변경시키는 것을 방지할 수 있다.

안전한 코드의 예 Hibernate

```
import org.hibernate.Query
import org.hibernate.Session
.....
String name = request.getParameter("name");
//1. 파라미터 바인딩을 위해 ?를 사용한다.
Query query = session.createQuery("from Student where studentName = ? ");
//2. 파라미터 바인딩을 사용하여 외부 입력값에 의해 쿼리 구조 변경을 못하게 사용하였다.
query.setString(0, name);
import org.hibernate.Query
import org.hibernate.Session
.....
String name = request.getParameter("name");
//1. 파라미터 바인딩을 위해 명명된 파라미터 변수를 사용한다.
Query query = session.createQuery("from Student where studentName = :name ");
//2. 파라미터 바인딩을 사용하여 외부 입력값에 의해 쿼리 구조 변경을 못하게 사용하였다.
query.setParameter("name", name);
```

다음 C# 코드는 외부 입력값을 SQL 쿼리에 직접 사용하고 있어, 쿼리의 구조가 변경될 위험이 있습니다.

안전하지 않은 코드의 예 C#

```
public void ButtonClickBad(object sender, EventArgs e)
{
  string connect = "MyConnString";
  string usrinput = Request["ID"];
// 외부로부터 입력받은 값을 SQL 쿼리에 직접 사용하는 것은 안전하지 않다.
  string query = "Select * From Products Where ProductID = " + usrinput;
  using (var conn = new SqlConnection(connect))
{
```

안전하지 않은 코드의 예 C#

```
      using (var cmd = new SqlCommand(query, conn))
{
       conn.Open();
       cmd.ExecuteReader(); /* BUG */
     }
   }
}
```

파라미터 바인딩을 사용하여 쿼리의 구조가 변경될 위험을 제거해야 합니다.

안전한 코드의 예 C#

```
void ButtonClickGood(object sender, EventArgs e)
 {
  string connect = "MyConnString";
      string usrinput = Request["ID"];
    //파라미터 바인딩을 위해 @ 을 사용합니다. 외부입력 값에 의해 쿼리 구조 변경을 할 수 없습니다.
  string query = "Select * From Products Where ProductID = @ProductID";
  using (var conn = new SqlConnection(connect))
  {
    using (var cmd = new SqlCommand(query, conn))
    {
      cmd.Parameters.AddWithValue("@ProductID",
      Convert.ToInt32(Request["ProductID"]);
      conn.Open();
      cmd.ExecuteReader();
    }
  }
```

라. 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

```
public class CodeInjectionController {
@RequestMapping(value = "/execute", method = RequestMethod.GET)
        public String execute(@RequestParam("src") String src)
        throws ScriptException {
                ScriptEngineManager scriptEngineManager = new
                ScriptEngineManager();
                ScriptEngine scriptEngine =
                scriptEngineManager.getEngineByName("javascript");
                // 외부 입력값인 src를 javascript eval 함수로 실행하고 있어 안전하지 않다.
                String retValue = (String)scriptEngine.eval(src);
                return retValue;
        }
}
```

다음 예제는 외부 입력 값을 javascript의 new Function()으로 동적으로 코드를 실행할 수 있다.

안전하지 않은 코드의 예 JAVA

```
〈body〉
 〈%
String name = request.getparameter("name");
  %〉
  ...
 〈script〉
   // 외부 입력값인 name을 javascript new Function()을 이용하여 문자열을 함수로 실행하고 있다.
        (new Function(〈%=name%〉))();
 〈/script〉
 〈/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

```
@RequestMapping(value = "/execute", method = RequestMethod.GET)
public String execute(@RequestParam("src") String src) throws ScriptException {
            // 정규식을 이용하여 특수문자 입력시 예외를 발생시킨다.
            if (src.matches("[ \ \w]*") == false) {
                        throw new IllegalArgumentException();
            }
            ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
      ScriptEngine scriptEngine = scriptEngineManager.getEngineByName("javascript");
            String retValue = (String)scriptEngine.eval(src);
            return retValue;
}
```

스크립트 실행이 필요한 경우는 화이트리스트 방식을 적용하여 유효한 문자인 경우에만 실행되도록 하고 그 외의 경우는 모두 예외 처리한다.

안전한 코드의 예 JAVA

```
@RequestMapping(value = "/execute", method = RequestMethod.GET)
public String execute(@RequestParam("src") String src) throws ScriptException {
        // 유효한 문자 "_" 일 경우 실행할 메소드 호출한다.
        if (src.matches("UNDER_BAR") == true) {
                ...
                // 유효한 문자 "$" 일 경우 실행할 메소드 호출한다.
                } else if (src.matches("DOLLAR") == true) {
                ...
                // 유효하지 않은 특수문자 입력시 예외를 발생시킨다.
                } else {
                        throw new IllegalArgumentException();
                }
        ...

}
```

라. 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 외부 입력값으로 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 파일을 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)3) 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.

안전하지 않은 코드의 예 JAVA

```
//외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
  String fileName = request.getParameter("P");
  BufferedInputStream bis = null;
  BufferedOutputStream bos = null;
  FileInputStream fis = null;
  try {
```

안전하지 않은 코드의 예 JAVA

```
        response.setHeader("Content-Disposition", "attachment;filename="+fileName+";");
...
      //외부로부터 입력받은 값이 검증 또는 처리 없이 파일처리에 수행되었다.
      fis = new FileInputStream("C:/datas/" + fileName);
      bis = new BufferedInputStream(fis);
      bos = new BufferedOutputStream(response.getOutputStream());
```

외부 입력값에 대하여 상대경로를 설정할 수 없도록 경로순회 문자열(/ ₩ & .. 등)을 제거하고 파일의 경로설정에 사용한다.

안전한 코드의 예 JAVA

```
String fileName = request.getParameter("P");
BufferedInputStream bis = null;
BufferedOutputStream bos = null;
FileInputStream fis = null;
try {
       response.setHeader("Content-Disposition", "attachment;filename="+fileName+";");
 ...
// 외부 입력받은 값을 경로순회 문자열(./₩)을 제거하고 사용해야한다.
      filename = filename.replaceAll("₩₩.", "").replaceAll("/", "").replaceAll("₩₩₩₩", "");
      fis = new FileInputStream("C:/datas/" + fileName);
      bis = new BufferedInputStream(fis);
      bos = new BufferedOutputStream(response.getOutputStream());
      int read;
while((read = bis.read(buffer, 0, 1024)) != -1) {
            bos.write(buffer,0,read);
      }
       }
```

인자값이 파일 이름인 경우에는 애플리케이션에서 정의(제한)한 디렉터리 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

```
public class ShowHelp {
    private final static String safeDir = "c:₩₩help_files₩₩";
    public static void main(String[] args) throws IOException {
        String helpFile = args[0];
        try (BufferedReader br = new BufferedReader(new FileReader(safeDir + helpFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
            ...
        }
```

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

```
public class ShowHelpSolution {
    private final static String safeDir = "c:₩₩help_files₩₩";
//경로조작 문자열 포함 여부를 확인하고 조치 후 사용하도록 한다.
    public static void main(String[] args) throws IOException {
      String helpFile = args[0];
      if (helpFile != null) {
          helpFile = helpFile.replaceAll("₩₩. {2, }[/₩₩₩₩]", "");
      }
try (BufferedReader br = new BufferedReader(new FileReader(safeDir + helpFile))) {
...
```

다음 C# 코드는 외부 입력값을 파일명에 바로 사용하고 있다. 이는 의도치 않은 파일의 손상을 가져올 수 있다.

안전하지 않은 코드의 예 C#

```
//외부 입력 값이 검증 없이 파일처리에 사용 되었다.
string file = Request.QueryString["path"];
    if (file != null)
    {
        File.Delete(file); 6:
    }
```

외부 입력값에 경로 조작 문자열을 제거하여 조작 위험을 없앨 수 있다.

안전한 코드의 예 C#

```
string file = Request.QueryString["path"];
    if (file != null)
{
//경로조작 문자열이 있는지 확인하고 파일 처리를 하도록 한다.
        if (file.IndexOf('₩₩') > -1 || file.IndexOf('/') > -1)
        {
            Response.Write("Path Traversal Attack");
        }
        else
        {
            File.Delete(file);
        }
    }
```

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

```
char* filename = getenv("reportfile");
FILE *fin = NULL;
// 외부 설정 값에서 받은 파일 이름을 그대로 사용한다.
fin = fopen(filename, "r");
while (fgets(buf, BUF_LEN, fin)) {
  // 파일 내용 출력
}
```

아래 C 코드는 외부에서 불러온 파일 이름을 그대로 사용하지 않고 경로 조작 가능성이 있는 문자열을 검증하고 사용한다.

안전한 코드의 예 C#

```
FILE *fin = NULL;
 regex_t regex;
Int ret;
 char* filename = getenv("reportfile");
 ret = regcomp(&regex, ".*₩₩.₩₩..*", 0);
 // 경로 조작 가능성 있는 문자열 탐지
 ret = regexec(&regex, filename, 0, NULL, 0);
 If (!ret) {
   // 경로 조작 문자열 발견, 오류 처리
}
 // 필터링된 파일 이름으로 사용
 fin = fopen(filename, "r");
 while (fgets(buf, BUF_LEN, fin)) {
   // 파일 내용 출력
}
```

라. 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

외부 입력값 또는 출력값에 스크립트가 삽입되지 못하도록 문자열 치환 함수를 사용하여 & < > " ' /( ) 등을 & < > " ' / ( )로 치환하거나, JSTL 또는 잘 알려진 크로스 사이트 스크립트 방지 라이브러리를 활용한다. HTML 태그를 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, 파일등으로 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 실행되면서 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

```
〈% String keyword = request.getParameter("keyword"); %〉
//외부 입력값에 대하여 검증 없이 화면에 출력될 경우 공격스크립트가 포함된 URL을 생성 할 수 있어
  안전하지 않다.(Reflected XSS)
검색어 : 〈%=keyword%〉

//게시판 등의 입력form으로 외부값이 DB에 저장되고, 이를 검증 없이 화면에 출력될 경우
  공격스크립트가 실행되어 안전하지 않다.(Stored XSS)
검색결과 : $ {m.content}

〈script type="text/javascript"〉
//외부 입력값에 대하여 검증 없이 브라우저에서 실행되는 경우 서버를 거치지 않는 공격스크립트가
  포함된 URL을 생성 할 수 있어 안전하지 않다. (DOM 기반 XSS)
document.write("keyword:" + 〈%=keyword%〉);
〈/script〉
```

외부 입력값 파라미터나 게시판등의 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

```
〈% String keyword = request.getParameter("keyword"); %〉
// 방법1. 입력값에 대하여 스크립트 공격가능성이 있는 문자열을 치환한다.
keyword = keyword.replaceAll("&", "&amp;");
keyword = keyword.replaceAll("〈", "&lt;");
keyword = keyword.replaceAll("〉", "&gt;");
keyword = keyword.replaceAll("₩"", "&quot;");
keyword = keyword.replaceAll("'", "&#x27;");
keyword = keyword.replaceAll("/"", "&#x2F;");
keyword = keyword.replaceAll("(", "&#x28;");
```

안전한 코드의 예 JAVA

```
keyword = keyword.replaceAll("'", "&#x29;");
검색어 : <%=keyword%>
//방법2. JSP에서 출력값에 JSTL c:out 을 사용하여 처리한다.
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
검색결과 : <c:out value="$ {m.content}"/>

<script type="text/javascript">
//방법3. 잘 만들어진 외부 라이브러리를 활용(NAVER Lucy-XSS-Filter, OWASP ESAPI,
  OWASP Java-Encoder-Project)
document.write("keyword:"+
  <%=Encoder.encodeForJS(Encoder.encodeForHTML(keyword))%>);
</script>
```

다음 C# 코드는 외부 입력값을 출력에 바로 사용하고 있다. 이는 XSS 공격을 유발 할 수 있다.

안전하지 않은 코드의 예 C#

```
string usrInput = Request.QueryString["ID"];
// 외부 입력 값이 검증 없이 화면에 출력 됩니다.
string str = "ID : " + usrinput;
Request.Write(str);
```

AntiXss 패키지 등을 사용하여 XSS 공격을 예방할 수 있다.

안전한 코드의 예 C#

```
string usrInput = Request.QueryString["ID"];
string str = "ID : " + usrinput;
//AntiXss 패키지 등을 이용하여 외부 입력값을 필터링 합니다.
var sanitizedStr = Sanitizer.GetSafeHtmlFragment(str);
quest.Write(sanitizedStr);
```

아래 C 코드 예제는 외부 입력값으로 사용자로부터 받은 입력값을 검증 없이 바로 cgi에 출력하는 화면이다.

안전하지 않은 코드의 예 C

```
int XSS(int argc, char* argv[]) {
   unsigned int i = 0;
   char data[1024];
   ...
   // cgiFromString으로 받아온 사용자 입력값이 검증 없이 화면에 출력됩니다.
giFromString("user input", data, sizeof(data));
printf(cgiOut, "Print user input = %s<br/>", data);
   fprintf(cgiOut, "</body></html>₩n");
   return 0;
}
```

cgi에 출력하기 전에 사용자 입력값을 검증하여야 한다.

안전한 코드의 예 C

```
cgiFromString("user input", data, sizeof(data));
// data에 위험한 문자열을 검사하는 코드를 추가한다.
if(strchr(p, '<')) return;
if(strchr(p, '>')) return;
...
fprintf(cgiOut, "Print user input = %s<br/>", data);
fprintf(cgiOut, "</body></html>₩n");
```

라. 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 명령어 를 생성하거나 선택이 필요한 경우에는 명령어 생성에 필요한 값 들을 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

```
public static void main(String args[]) throws IOException {
// 해당 프로그램에서 실행할 프로그램을 제한하고 있지 않아 파라미터로 전달되는 모든 프로그램이
   실행될 수 있다.
    String cmd = args[0];
    Process ps = null;
    try {
        ps = Runtime.getRuntime().exec(cmd);
        ...
```

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

```
public static void main(String args[]) throws IOException {
// 해당 어플리케이션에서 실행할 수 있는 프로그램을 노트패드와 계산기로 제한하고 있다.
   List<String> allowedCommands = new ArrayList<String>();        "
   allowedCommands.add("notepad");  allowedCommands.add("calc");
   String cmd = args[0];
   if (!allowedCommands.contains(cmd)) {
        System.err.println("허용되지 않은 명령어입니다.");
        return;
     }
     Process ps = null;  try {
          ps = Runtime.getRuntime().exec(cmd);
                        ......
```

아래 코드는 외부입력값을 검증하지 않고 그대로 명령어로 실행하기 때문에 공격자의 입력에 따라 의도하지 않은 명령어가 실행될 수 있다.

안전하지 않은 코드의 예 JAVA

```
//외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.

String date = request.getParameter("date");
String command = new String("cmd.exe /c backuplog.bat");
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

```
String date = request.getParameter("date");
String command = new String("cmd.exe /c backuplog.bat");
//외부로부터 입력 받은 값을 필터링으로 우회문자를 제거하여 사용한다.
date = date.replaceAll("|","");
date = date.replaceAll(";","");
date = date.replaceAll("&","");
date = date.replaceAll(":","");
date = date.replaceAll(")",""); 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#

```
//외부 입력값이 프로세스가 실행할 파일 이름을 지정하고 있다.
string fileName = PgmTextBox.Text;
ProcessStartInfo proStartInfo = new ProcessStartInfo();
proStartInfo.FileName = fileName;
Process.Start(proStartInfo);
```

적절한 정규식이나, white list 등을 이용하여 검증을 한 후 사용하도록 한다.

안전한 코드의 예 C#

```
string fileName = PgmTextBox.Text;
//외부 입력값에 대해 정규식 등을 이용하여 검증해야 한다.
 if (Regex.IsMatch(fileName, "properRegexHere"))
{
ProcessStartInfo proStartInfo = new ProcessStartInfo();
proStartInfo.FileName = fileName;
Process.Start(proStartInfo);
}
```

공격자의 입력에 따라 의도하지 않은 명령어가 실행될 수 있다.

안전하지 않은 코드의 예 C

```
int main(int argc, char* argv[]) {
   char cmd[CMD_LENGTH];

   if (argc < 1 ) {
      // error
}
// 외부 입력값으로 커맨드를 직접 수행
cmd_data = argv[1];
snprintf(cmd, CMD_LENGTH, “cat %s”, cmd_data);
system(cmd);
……
}
```

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

```
int main(int argc, char* argv[]) {
   char cmd[CMD_LENGTH];  int len = 0;
   if (argc < 1 ) {
     // error
   }
   // 외부 입력값으로 커맨드를 직접 수행  cmd_data = argv[1];
   len = strlen(cmd_data);
for (int i = 0; i < len; i++) {
      if (cmd_data[i] == '|' || cmd_data[i] == '&' ||
         cmd_data[i] == ';' || cmd_data[i] == ':' || cmd_data[i] == '>') {
     // 멀티라인을 지원하는 특수문자나 파일 리다이렉트 특수문자가 존재하여
// 안전하지 않음
     return -1;
    }
}
 snprintf(cmd, CMD_LENGTH, "cat %s", cmd_data);
 system(cmd);
.....
}
```

라. 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=64651588#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 업로드가능하고, 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 실행 여부를 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

```
  MultipartRequest multi
  = new MultipartRequest(request,savePath,sizeLimit,"euc-kr",new
DefaultFileRenamePolicy());
.....
//업로드 되는 파일명을 검증없이 사용하고 있어 안전하지 않다.
     String fileName = multi.getFilesystemName("filename");
  .....
     sql = " INSERT INTO
  board(email,r_num,w_date,pwd,content,re_step,re_num,filename) "
          + " values ( ?, 0, sysdate(), ?, ?, ?, ?, ? ) ";
    preparedStatement pstmt = con.prepareStatement(sql);
pstmt.setString(1, stemail);
pstmt.setString(2, stpwd);
```

안전하지 않은 코드의 예 JAVA

```
pstmt.setString(3, stcontent);
pstmt.setString(4, stre_step);
pstmt.setString(5, stre_num);
pstmt.setString(6, fileName);
pstmt.executeUpdate();
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

```
MultipartRequest multi
      = new MultipartRequest(request,savePath,sizeLimit,"euc-kr",new
  DefaultFileRenamePolicy());
  ......
    String fileName = multi.getFilesystemName("filename");
    if (fileName != null) {
//1.업로드 파일의 마지막 "." 문자열의 기준으로 실제 확장자 여부를 확인하고, 대소문자 구별을
  해야한다.
        String fileExt =
     FileName.substring(fileName.lastIndexOf(".")+1).toLowerCase();
//2.되도록 화이트 리스트 방식으로 허용되는 확장자로 업로드를 제한해야 안전하다.
      if (!"gif".equals(fileExt) && !"jpg".equals(fileExt) && !"png".equals(fileExt))
{
        alertMessage("업로드 불가능한 파일입니다.");
        return;
      }
}
......
     sql = " INSERT INTO
board(email,r_num,w_date,pwd,content,re_step,re_num,filename) "
       + " values ( ?, 0, sysdate(), ?, ?, ?, ?, ? ) ";
PreparedStatement pstmt = con.prepareStatement(sql);
......
  Thumbnail.create(savePath+"/"+fileName, savePath+"/"+"s_"+fileName, 150);
```

업로드할 파일에 대한 유효성을 검사하지 않으면, 위험한 유형의 파일을 공격자가 업로드 하거나 전송할 수 있다.

안전하지 않은 코드의 예 C#

```
string fn = Path.GetFileName(FileUploadCtr.FileName);
//업로드 하는 파일명을 검증없이 사용하고 있다.
FileUploadCtr.SaveAs(fn);
StatusLabel.Text = "Upload status: File uploaed!";
```

파일 타입과 크기 등을 검사하여 제한하도록 한다.

안전한 코드의 예 C#

```
//파일 타입과 크기를 제한합니다.
if (FileUploadCtr.PostedFile.ContentType == "image/jpeg")
{
    if (FileUploadCtr.PostedFile.ContentLength < 102400)
    {
        string fn = Path.GetFileName(FileUploadCtr.FileName);
        FileUploadCtr.SaveAs(Server.MapPath("~/") + fn);
        StatusLabel.Text = "Upload status: File uploaed!";
}
else
      StatusLabel.Text = "Upload Status: The File has to be less than
   100 kb!";
}
else
StatusLabel.Text = "Upload Status: Only JPEG files are accepted!";
```

라. 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 링크로 희생자가 피싱 사이트 등으로 접근하도록 할 수 있다.

(예시 링크)

〈a href="http://bank.example.com/redirect?url=http://attacker.example.net"〉Click〈/a〉

안전하지 않은 코드의 예 JAVA

```
String id = (String)session.getValue("id");
String bn = request.getParameter("gubun");
//외부로부터 입력받은 URL이 검증없이 다른 사이트로 이동이 가능하여 안전하지 않다.
String rd = request.getParameter("redirect");
if (id.length() > 0) {
        String sql = "select level from customer where customer_id = ? ";
        conn = db.getConnection();
        pstmt = conn.prepareStatement(sql);
```

안전하지 않은 코드의 예 JAVA

```
        pstmt.setString(1, id);
        rs = pstmt.executeQuery();
rs.next();
if ("0".equals(rs.getString(1)) && "01AD".equals(bn)) {
        response.sendRedirect(rd);
        return;
}
```

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

```
//이동 할 수 있는 URL범위를 제한하여 피싱 사이트 등으로 이동하지 못하도록 한다.
String allowedUrl[] = { "/main.do", "/login.jsp", "list.do" };
.....
String rd = request.getParameter("redirect");
try {
    rd = allowedUrl[Integer.parseInt(rd)];
} catch(NumberFormatException e) {
    return "잘못된 접근입니다.";
} catch(ArrayIndexOutOfBoundsException e) {
    return "잘못된 입력입니다.";
}
if (id.length() > 0) {
.....
    if ("0".equals(rs.getString(1)) && "01AD".equals(bn)) {
        response.sendRedirect(rd);
        return;
}
```

외부 입력 값으로 받은 URL로 검증없이 연결되는 경우, 공격자의 입력에 따라 피싱사이트 등으로 연결될 수 있다.

안전하지 않은 코드의 예 C#

```
// 외부 입력값으로 받은 URL을 검증없이 연결하고 있다.
string url = Request["dest"];
Response.Redirect(url);
```

로컬 URL 검증이나 white list 등을 이용하여 검증이 필요하다.

안전한 코드의 예 C#

```
public void AttackOpenRedirect()
    {
        String url = Request["dest"];
// 외부 입력값이 로컬 URL인지 확인한다. MVC 3 이상의 프레임워크를 사용할 경우,
   System.Web.Mvc 에 정의되어있는 Url.isLocalUrl 을 바로 사용할 수 있다.
        if(isLocalUri(url)) Response.Redirect(url);
     }
     private bool IsLocalUrl(string url)
     {
      if(string.IsNullOrEmpty(url))
      {
       return false;
      }
     Uri absoluteUri;
     if(Uri.TryCreate(url, UriKind.Absolute, out absoluteUri))
    {
       return String.Equals(this.Request.Url.Host, absoluteUri.Host,
   StringComparison.OrdinalIgnoreCase);
    }
  else
   {
      bool isLocal = !url.StartsWith("http:",
  StringComparison.OrdinalIgnoreCase)
```

안전한 코드의 예 C#

```
        && !url.StartsWith("https:",
  StringComparison.OrdinalIgnoreCase)
        && Uri.IsWellFormedUriString(url, UriKind.Relative);
      return isLocal;
    }
}
```

라. 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

```
public void unmarshal(File receivedXml)
throws JAXBException, ParserConfigurationException, SAXException, IOException {
        JAXBContext jaxbContext = JAXBContext.newInstance( Student.class );
        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
        // 입력받은 receivedXml 을 이용하여 Document를 생성한다.
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document document = db.parse(receivedXml);
        // 외부 엔티티로 만들어진 document를 이용하여 마샬링을 수행하여 안전하지
        않다.
        Student employee = (Student) jaxbUnmarshaller.unmarshal( document );
}
```

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

```
import javax.xml.parsers.SAXParsers;
import javax.xml.parsers.SAXParserFactory;

class XXE {
        public static void main(String[] args)
        throws FileNotFoundException, ParserConfigurationException, SAXException,
IOException {
```

안전하지 않은 코드의 예 JAVA

```
SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            // 외부개체 참조 제한 설정 없이 secure.xml 파일을 읽어서 파싱하여 안전하지 않다.
            saxParser.parse(new FileInputStream("secure.xml"), new DefaultHandler());
    }
}
```

다음의 JAXP DocumentBuilderFactory를 사용하는 경우 아래와 같이 제한 설정 추가할 수 있다.

안전한 코드의 예 JAVA

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
// XML 파서가 doctype을 정의하지 못하도록 설정한다.
dbf.setFeature("http://apache.org/xml/featuresdisallow-doctype-decl", true);
// 외부 일반 엔티티를 포함하지 않도록 설정한다.
dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
// 외부 파라미터도 포함하지 않도록 설정한다.
dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
// 외부 DTD 비활성화한다.
dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
// XIncude를 사용하지 않는다.
dbf.setXIncludeAware(false);
// 생성된 파서가 엔티티 참조 노드를 확장하지 않도록 한다.
dbf.setExpandEntityReferences(false);
DocumentBuilder db = dbf.newDocumentBuilder();
Document document = db.parse(receivedXml);
Model model = (Model) u.unmarshal(document);
```

PHP에서는 libxml_disable_entity_loader 함수를 이용하여 외부 엔티티 사용을 비활성화 할 수 있다.

안전한 코드의 예 JAVA

```
value = libxml_disable_entity_loader(true);
$dom = new DOMDocument();
$dom -> loadXML($xml);
libxml_disable_entity_loader($value);
```

라. 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 인증절차를 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 executeQuery으로 생성하는 쿼리의 파라미터의 일부로 외부입력값 name)을 사용하고 있다. 만일 something' or '1'='1 을 name의 값으로 전달하면 다음과 같은 쿼리문을 수행할 수 있으며, 이 경우 파일 내의 모든 값을 출력할 수 있게 된다.

doc('users.xml')/userlist/user[uname='something' or '1'='1'

안전하지 않은 코드의 예 JAVA

```
// 외부 입력 값을 검증하지 않고 XQuery 표현식에 사용한다.
String name = props.getProperty("name");
.....
// 외부 입력 값에 의해 쿼리 구조가 변경 되어 안전하지 않다.
String es = "doc('users.xml')/userlist/user[uname='"+name+"']";
XQPreparedExpression expr = conn.prepareExpression(es);
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 만든다.
String squery =
        "for $user in doc(users.xml)//user[username='"
        + UserTextBox.Text
        + "'and pass='"
        + PwdTextBox.Text
        + "'] return $user";


    Processor processor = new Processor();

    XQueryCompiler compiler = processor.NewXQueryCompiler();

    XdmNode indoc = processor.NewDocumentBuilder().Build(new
Uri(Server.MapPath("users.xml")));
using (StreamReader query = new StreamReader(squery))
{
    XQueryCompiler compiler = processor.NewXQueryCompiler();
    XQueryExecutable exp = compiler.Compile(query.ReadToEnd());
```

안전하지 않은 코드의 예 C#

```
    XQueryEvaluator eval = exp.Load();
    eval.ContextItem = indoc;

    Serializer qout = new Serializer();
    qout.SetOutputProperty(Serializer.METHOD, "xml");
    qout.SetOutputProperty(Serializer.DOCTYPE_PUBLIC, "-//W3C//DTD
XHTML 1.0 Strict//EN");
    qout.SetOutputProperty(Serializer.DOCTYPE_SYSTEM,
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd");
    qout.SetOutputProperty(Serializer.INDENT, "yes");
    qout.SetOutputProperty(Serializer.OMIT_XML_DECLARATION, "no");

    qout.SetOutputWriter(Response.Output);
//별다른 검증없이 XML 데이터에 접근한다.
    eval.Run(qout);
}
```

문자열 필터링으로 쿼리문 조작을 막을 수 있다.

안전한 코드의 예 C#

```
String squery =
        "for $user in doc(users.xml)//user[username='"
        + UserTextBox.Text
        + "'and pass='"
        + PwdTextBox.Text
        + "'] return $user";
// 문자열 필터링으로 위험한 문자열을 제거한다.
string validatedQuery = squery.Replace('/','*');
    Processor processor = new Processor();
XQueryCompiler compiler = processor.NewXQueryCompiler();

    XdmNode indoc = processor.NewDocumentBuilder().Build(new
    Uri(Server.MapPath("users.xml")));
    using (StreamReader query = new StreamReader(validatedQuery))
```

안전한 코드의 예 C#

```
    { // tainted value propagated
        XQueryCompiler compiler = processor.NewXQueryCompiler();
        XQueryExecutable exp = compiler.Compile(query.ReadToEnd()); //
xquery created
        XQueryEvaluator eval = exp.Load();
        eval.ContextItem = indoc;

        Serializer qout = new Serializer();
        qout.SetOutputProperty(Serializer.METHOD, "xml");
        qout.SetOutputProperty(Serializer.DOCTYPE_PUBLIC, "-//W3C//DTD
    XHTML 1.0 Strict//EN");
        qout.SetOutputProperty(Serializer.DOCTYPE_SYSTEM,
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd");
        qout.SetOutputProperty(Serializer.INDENT, "yes");
        qout.SetOutputProperty(Serializer.OMIT_XML_DECLARATION, "no");

        qout.SetOutputWriter(Response.Output);
        eval.Run(qout);
    }
```

• 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
String nm = props.getProperty("name");
String pw = props.getProperty("password");
.....
XPathFactory factory = XPathFactory.newInstance();
```

안전하지 않은 코드의 예 JAVA

```
XPath xpath = factory.newXPath();
.....
// 검증되지 않은 입력값 외부 입력값 nm, pw 를 사용하여 안전하지 않은 질의문이  작성되어 expr
    변수에 저장된다.
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.
Object result = expr.evaluate(doc, XPathConstants.NODESET);
// result의 결과를 NodeList 타입으로 변환하여 nodes 저장한다.
NodeList nodes = (NodeList) result;
for (int i=0; i<nodes.getLength(); i++) {
     String value = nodes.item(i).getNodeValue();
     if (value.indexOf(">") < 0) {
          // 공격자가 이름과 패스워드를 확인할 수 있다.  System.out.println(value);
     }
}
```

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
String nm = props.getProperty("name");
String pw = props.getProperty("password");
Document doc = new Builder().build("users.xml");

// 파라미터화된 쿼리가 담겨있는 login.xq를 읽어와서 파라미터화된 쿼리를 생성한다.
XQuery xquery = new XQueryFactory().createXQuery(new File("login.xq"));
```

안전한 코드의 예 JAVA

```
Map vars = new HashMap();
// 검증되지 않은 외부값인 nm, pw를 파라미터화된 쿼리의 파라미터로 설정한다.
vars.put("loginID", nm);
vars.put("password", pw);
// 파라미터화된 쿼리를 실행하므로 외부값을 검증없이 사용하여도 안전하다.
Nodes results = xquery.execute(doc, null, vars).toNodes();
for (int i=0; i<results.size(); i++) {
      System.out.println(results.get(i).toXML());
}
```

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
public String XPathFilter(String input) {
    if (input != null) return input.replaceAll("['₩₩[]", "");
    else return "";
}
.....
// 외부 입력값에 사용
String nm = XPathFilter(props.getProperty("name"));
String pw = XPathFilter(props.getProperty("password"));
.....
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
.....
// 외부 입력값인 nm, pw를 검증하여 쿼리문을 생성하므로 안전하다.
XPathExpression expr = xpath.compile("//users/user[login/text()='"+nm+"' and
  password/text()='"+pw+"']/home_dir/text()");
Object result = expr.evaluate(doc, XPathConstants.NODESET);
NodeList nodes = (NodeList) result;
for (int i=0; i<nodes.getLength(); i++) {
```

안전한 코드의 예 JAVA

```
    String value = nodes.item(i).getNodeValue();
    if (value.indexOf(">") < 0) {
        System.out.println(value);
    }
}
   ......
```

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

```
public static void main(String[] args) throws Exception {
     if (args.length <= 0) {
         System.err.println("가격을 검색할 식품의 이름을 입력하세요.");  return;
     }
    String name = args[0];
    DocumentBuilder docBuilder =
  DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = docBuilder.parse("http://www.w3schools.com/xml/simple.xml");
    Xpath xpath = XPathFactory.newInstance().newXPath();
// 프로그램의 커맨드 옵션으로 입력되는 외부값 name을 사용하여 쿼리문을 직접 작성하여 수행하므로 안전하지 않다.
    NodeList nodes = (NodeList) xpath.evaluate("//food[name='" + name +
    "']/price", doc, XPathConstants.NODESET);
    for (int i = 0; i < nodes.getLength(); i++) {
            System.out.println(nodes.item(i).getTextContent());
    }
}
```

외부 입력값을 XPath 구문 생성 및 실행에 사용하는 경우 XPath 구문을 조작할 수 있는 문자열을 제거하고 사용할 수 있도록 한다.

안전한 코드의 예 JAVA

```
public static void main(String[] args) throws Exception {
    if (args.length <= 0) {
       System.err.println("가격을 검색할 식품의 이름을 입력하세요.");
       return;
    }
/*프로그램의 커맨드 옵션으로 입력되는 외부값 name에서 XPath 구문을 조작할 수 있는 문자를
  제거하는 검증을 수행하여 안전하다.*/
    String name = args[0];
    if (name != null) {
        name = name.replaceAll("[()₩₩-'₩₩[₩₩]:,*/]", "");
    }
 DocumentBuilder docBuilder =
  DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document doc = docBuilder.parse("http://www.w3schools.com/xml/simple.xml");
    XPath xpath = XPathFactory.newInstance().newXPath();
    NodeList nodes = (NodeList) xpath.evaluate("//food[name='" + name + "']/price",
doc, XPathConstants.NODESET);
    for (int i = 0; i < nodes.getLength(); i++) {
        System.out.println(nodes.item(i).getTextContent());
    }
}
```

다음의 예제는 외부입력을 사용하여 XPath 식에 바로 사용한다. 이는 공격자의 입력에 따라 XQuery의 구조를 바꾸어 예기치 않은 공격이 발생할 수 있다.

안전하지 않은 코드의 예 C#

```
string acctID = Request["acctID"];
string query = null;  if(acctID != null)
{
StringBuffer sb = new StringBuffer("/accounts/account[acctID='");  sb.Append(acctID);
```

안전하지 않은 코드의 예 C#

```
sb.Append("']/email/text()");  query = sb.ToString();

}

XPathDocument docNav = new XPathDocument(myXml);  XPathNavigator nav =
docNav.CreateNavigator();
//외부 입력값을 검증없이 사용하고 있습니다.
nav.Evaluate(query);
```

다음의 예제는 XPathExpression을 사용하여 미리 쿼리 골격을 생성함으로써 외부입력으로 인해 쿼리 구조가 바뀌는 것을 막을 수 있다.

안전한 코드의 예 C#

```
string xpath = "/accounts/account[@acctID=$acctID]/email/text()";
XPathExpression expr = DynamicContext.Compile(xpath);
DynamicContext ctx = new DynamicContext();
ctx.AddVariable("acctID", AccountIDTextBox.Text);
expr.SetContext(ctx);
XPathNodeIterator data = nav.Select(expr);
```

라. 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+Injections

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.

외부 입력값을 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 특수문자를 제거한다. 만약 특수문자를 사용해야 하는 경우에는 특수문자(= + 〈 〉 # ; ₩ 등)가 실행명령이 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

```
private void searchRecord(String userSN, String userPassword) throws
    NamingException {
      Hashtable<String, String> env = new Hashtable<String, String>();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
      try {
            DirContext dctx = new InitialDirContext(env);
            SearchControls sc = new SearchControls();
            String[] attributeFilter = { "cn", "mail" };
```

안전하지 않은 코드의 예 JAVA

```
 sc.setReturningAttributes(attributeFilter);
          sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
          String base = "dc=example,dc=com";
/*userSN과 userPassword 값에 LDAP필터를 조작할 수 있는 공격 문자열에 대한 검증이 없어
  안전하지 않다.*/
          String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";
          NamingEnumeration<?> results = dctx.search(base, filter, sc);
          while (results.hasMore()) {
               SearchResult sr = (SearchResult) results.next();
               Attributes attrs = sr.getAttributes();
               Attribute attr = attrs.get("cn");
  .....
          }
          dctx.close();
   } catch (NamingException e) { … }
}
```

검색을 위한 필터 문자열로 사용되는 외부의 입력에서 위험한 문자열을 제거하여 위험성을 부분적으로 감소시킬 수 있다.

안전한 코드의 예 JAVA

```
private void searchRecord(String userSN, String userPassword) throws
  NamingException {
   Hashtable<String, String> env = new Hashtable<String, String>();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
   try {
         DirContext dctx = new InitialDirContext(env);
         SearchControls sc = new SearchControls();
         String[] attributeFilter = {"cn", "mail" };
         sc.setReturningAttributes(attributeFilter);
sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
         String base = "dc=example,dc=com";
```

안전한 코드의 예 JAVA

```
// userSN과 userPassword 값에서 LDAP 필터를 조작할 수 있는 문자열을 제거하고 사용
            if (!userSN.matches("[₩₩w₩₩s]*") || !userPassword.matches("[₩₩w]*")) {
                throw new IllegalArgumentException("Invalid input");
            }
            String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";
            NamingEnumeration<?> results = dctx.search(base, filter, sc);
            while (results.hasMore()) {
                SearchResult sr = (SearchResult) results.next();
                Attributes attrs = sr.getAttributes();
        Attribute attr = attrs.get("cn");
   ......
        }
       dctx.close();
    } catch (NamingException e) { … }
}
```

다음은 인증하지 않은 익명 바인딩으로 LDAP 쿼리를 실행하는 C# 코드 예제이다.

안전하지 않은 코드의 예 C#

```
static void SearchRecord(string userSN, string userPW)
{
try {
   DirectoryEntry oDE;
   oDE = new DirectoryEntry(GetStrPath());
   // 인증을 하지않은 익명 바인딩으로 LDAP 쿼리를 실행
   foreach(DirectoryEntry objChildDE om oDE.Children) {
...
}
            } catch (NamingException e) { ... }
}
```

익명 바인딩을 사용하지 않고 인증을 진행해야 한다.

안전한 코드의 예 C#

```
void LDAPInjection() {
  char *filter = getenv("Filter");
  int error_code;
  LDAP *ld = NULL;
  LDAPMessage *result;
  // 외부에서 불러온 filter를 검증없이 사용
  error_code = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter,
  NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
}
```

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

```
void LDAPInjection() {
  char *filter = getenv("Filter");
  int error_code;
  LDAP *ld = NULL;
  LDAPMessage *result;
  // 외부에서 불러온 filter를 검증없이 사용
  error_code = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter,
  NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
}
```

공격 가능성이 있는 문자열을 필터하여 문제를 해결할 수 있다.

안전한 코드의 예 C

```
void LDAPInjection() {
char *filter = getenv("Filter");
int error_code;  int i;
LDAP *ld = NULL;
LDAPMessage *result;
// 정보를 알고 싶은 사용자의 이름을 고정 값으로 사용
  for(i = 0; *(filter + i) != 0; i++) {
// 공격 가능한 문자열 검사
  switch(*(filter +  i)) {
     case '*':
     case '(':
     case ')':
  ...
          return;
  }
}
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. 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, 게시물삭제, 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 작성 시 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 전달 된 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.
session.setAttribute("SESSION_CSRF_TOKEN", UUID.randomUUID().toString());
// 입력화면에 임의의 토큰을 HIDDEN 필드항목의 값으로 설정해 서버로 전달되도록 한다.
〈input type="hidden" name="param_csrf_token" value="$ {SESSION_CSRF_TOKEN }"/〉

// 요청 파라미터와 세션에 저장된 토큰을 비교해서 일치하는 경우에만 요청을 처리한다.
String pToken = request.getParameter("param_csrf_token");
String sToken = (String)session.getAttribute("SESSION_CSRF_TOKEN");
if (pToken != null && pToken.equals(sToken) {
   // 일치하는 토큰이 존재하는 경우 -〉 정상 처리
      ......
} else {
  // 토큰이 없거나 값이 일치하지 않는 경우 -〉 오류 메시지 출력
      ......
}
```

AntiForgeryToken() 등을 이용하여 크로스사이트 요청 위조를 방지해야 한다.

안전한 코드의 예 C#

```
@using  (Html.BeginForm("PostTest","Home",FormMethod.Post,null))
{
        // AntiForgeryToken()  을 이용해 크로스사이트 요청 위조를 방지
        @Html.AntiForgeryToken()
        〈input type="submit" value="Html PsBk Click" /〉
}
```

라. 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

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        // 사용자 입력값(url)을 검증없이 사용하여 안전하지 않다.
        URL url = new URL(req.getParameter("url"));
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
}
```

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

```
public class Connect {
        // key, value 형식으로 URL의 리스트를 작성한다.
        private Map<String, URL> urlMap;
        protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
                // 사용자에게 urlMap의 key를 입력받아 urlMap에서 URL값을 참조한다.
                URL url = urlMap.get(req.getParameter("url"));
                // urlMap에서 참조한 값으로 Connection을 만들어 접속한다.
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        }
}
```

라. 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 수행할 수 있다.

나. 보안대책

요청 파라미터의 값을 HTTP응답헤더(예를 들어, Set-Cookie 등)에 포함시킬 경우 CR, LF와 같은 개행문자를 제거한다.

다. 코드예제

외부 입력값을 사용하여 반환되는 쿠키의 값을 설정하고 있다. 그런데, 공격자가 Wiley Hacker ₩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.
String lastLogin = request.getParameter("last_login");
if (lastLogin == null || "".equals(lastLogin)) {
    return;
}
// 쿠키는 Set-Cookie 응답헤더로 전달되므로 개행문자열 포함 여부 검증이 필요
Cookie c = new Cookie("LASTLOGIN", lastLogin);
c.setMaxAge(1000);
c.setSecure(true);
response.addCookie(c);
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

```
String lastLogin = request.getParameter("last_login");
if (lastLogin == null || "".equals(lastLogin)) {
    return;
}
// 외부 입력값에서 개행문자(₩r₩n)를 제거한 후 쿠키의 값으로 설정
lastLogin = lastLogin.replaceAll("[₩₩r₩₩n]", "");
Cookie c = new Cookie("LASTLOGIN", lastLogin);
c.setMaxAge(1000);
c.setSecure(true);
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 검증없이 사용하는 것은 안전하지 않다.
string usrInput = Request.QueryString["ID"];
Response.AddHeader("foo", "bar" + usrInput);
```

개행문자를 모두 제거한 이후에 사용해야 한다.

안전한 코드의 예 C#

```
ring usrInput = Request.QueryString["ID"];
// 개행문자를 제거 한 이후에 사용해야 한다.
ring validatedInput = usrInput.Replace("₩n", "").Replace("₩r","");
sponse.AddHeader("foo", "bar" + validatedInput);
```

라. 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 정수형 크기는 고정되어 있는데 저장 할 수 있는 범위를 넘어서, 크기보다 큰 값을 저장하려 할 때 실제 저장되는 값이 의도치 않게 아주 작은 수이거나 음수가 되어 프로그램이 예기치 않게 동작될 수 있다. 특히 반복문 제어, 메모리 할당, 메모리 복사 등을 위한 조건으로 사용자가 제공하는 입력값을 사용하고 그 과정에서 정수형 오버플로우가 발생하는 경우 보안상 문제를 유발 할 수 있다.

나. 보안대책

언어/플랫폼별 정수타입의 범위를 확인하여 사용한다. 정수형 변수를 연산에 사용하는 경우, 결과 값의 범위를 체크하는 모듈을 사용한다. 특히 외부입력값을 동적 메모리 할당에 사용하는 경우, 변수 값이 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

```
String msg_str = "";
String tmp = request.getParameter("slf_msg_param_num");
tmp = StringUtil.isNullTrim(tmp);
if (tmp.equals("0")) {
    msg_str = PropertyUtil.getValue(msg_id);
} else {
    // 외부 입력값을 정수형으로 사용할 때 입력값의 크기를 검증하지 않고 사용
    int param_ct = Integer.parseInt(tmp);
    String[] strArr = new String[param_ct];
```

동적 메모리 할당을 위해 외부 입력값을 배열의 크기로 사용하는 경우 그 값이 음수가 아닌지 검사하는 작업이 필요하다.

안전한 코드의 예 JAVA

```
String msg_str = "";
String tmp = request.getParameter("slf_msg_param_num");
tmp = StringUtil.isNullTrim(tmp);
if (tmp.equals("0")) {
    msg_str = PropertyUtil.getValue(msg_id);
} else {
// 외부 입력값을 정수형으로 사용할 때 입력값의 크기를 검증하고 사용
try {
    int param_ct = Integer.parseInt(tmp);
    if (param_ct < 0) {
        throw new Exception();
    }
    String[] strArr = new String[param_ct];
} catch(Exception e) {
    msg_str = "잘못된 입력(접근) 입니다.";
}
```

외부 입력값으로 배열의 접근할 경우, 입력값이 너무 클 때 음수가 되어 시스템에 문제가 발생 할 수 있다.

안전하지 않은 코드의 예 C#

```
public static void Main(string[] args)
{
  // 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
  int usrNum = Int32.Parse(args[0]);
  string[] array = {"one", "two", "three", "four" };
  string num = array[usrNum];
}
```

checked 구문을 이용하여 오버플로우 발생 확인 및 처리를 해야한다.

안전한 코드의 예 C#

```
public static void Main(string[] args)
{
  // checked 구문을 사용하여 오버플로우의 발생 여부 및 크기 확인
  try {
    int usrNum = checked(Int32.Parse(args[0]));
    string[] array = {"one", "two", "three", "four" };
    if(usrNum < 3)string num = array[usrNum];
  }
  catch (System.OverflowException e) { … }
}
```

안전하지 않은 코드의 예 C

```
id main(int argc, char* argv[])

// 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
int usr_num = 0;
char* num_array[] = {“one”, “two”, “three”, “four” };
char* num = NULL;
usr_num = atoi(argv[1]);
num = num_array[usr_num];
}
```

안전한 코드의 예 C

```
id main(int argc, char* argv[])

    // 외부 입력값을 사용할 때, 입력 값의 크기가 너무 클 경우 오버플로우 발생
int usr_num = 0;
 char* num_array[] = {“one”, “two”, “three”, “four” };
 char* num = NULL;
 usr_num = atoi(argv[1]);
 if (usr_num >= 0 && usr_num < 4) {
   num = num_array[usr_num];
}
  }
```

라. 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 외부 입력값에 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 가정 하지만 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 암호 화, 무결성 체크를 수행하고 이와 같은 메커니즘이 없는 경우엔 외부사용자에 의한 입력값을 신뢰해서는 안 된다.

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 구조 로 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

```
〈input type="hidden" name="price" value="1000"/〉
〈br/〉품목 : HDTV
〈br/〉수량 : 〈input type="hidden" name="quantity" /〉개
〈br/〉〈input type="submit" value="구입" /〉
.....
```

안전하지 않은 코드의 예 JAVA

```
try {
 // 서버가 보유하고 있는 가격(단가) 정보를 사용자 화면에서 받아서 처리
      price = request.getParameter("price");
      quantity = request.getParameter("quantity");
      total = Integer.parseInt(quantity) * Float.parseFloat(price);
} catch (Exception e) {
.....
```

사용자 권한, 인증 여부 등 보안결정에 사용하는 값은 사용자 입력값을 사용하지 않고 서버 내부의 값을 활용한다. 또한 사용자 입력에 의존해야하는 값을 제외하고는 반드시 서버가 보유하고 있는 정보를 이용하여 처리한다.

안전한 코드의 예 JAVA

```
<input type="hidden" name="price" value="1000"/>
<br/>품목 : HDTV
<br/>수량 : <input type="hidden" name="quantity" />개
<br/><input type="submit" value="구입" />
.....
try {
      item = request.getParameter("item");
    // 가격이 아니라 item 항목을 가져와서 서버가 보유하고 있는 가격정보를  이용하여 전체 가격을
        계산
    price = productService.getPrice(item);
    quantity = request.getParameter("quantity");
    total = Integer.parseInt(quantity) * price;
} catch (Exception e) {
    .....
}
.....
```

평문으로 사용자의 인증정보를 쿠키에 저장하고 있는 C# 예제코드이다.

안전하지 않은 코드의 예 C#

```
HttpCookie cookie = new HttpCookie("Authentificated", "1");
//평문으로 사용자의 인증정보를 쿠키에 저장한다.
Response.Cookies.Add(cookie);
```

중요한 정보를 쿠키에 저장시에는 암호화해서 사용하고, 되도록이면 해당정보는 서버의 세션에 저장하도록 한다.

안전한 코드의 예 C#

```
// 사용자의 인증정보를 세션에 저장한다.
Session["Authentificated"] = "1";
```

아래 C 코드는 외부에서 가져온 서버 정보를 기반으로 연결을 진행한다. 사용자가 환경변수를 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

```
void SecurityDecision() {
    int sockfd = socket(PF_INET, SOCK_STREAM, 0);
    char* server_info = getenv("server_addr");
    // 외부에서 가져온 서버 정보를 그대로 사용한다.
    if( connect( sockfd, (struct sockaddr *)server_addr, sizeof(struct socketaddr) ) < 0 ) {
        return;
    }
   /* 라이선스 검증 코드 */
}
```

인증 서버의 정보를 환경 변수가 아닌 고정된 정보를 이용하여 연결을 진행한다.

안전한 코드의 예 C#

```
void SecurityDecision() {
  int sockfd = socket(PF_INET, SOCK_STREAM, 0);
  struct sockaddr_in server_addr;
  memset( &server_ info, 0, sizeof(server_info));
  server_info.sin_family = AF_INET;
  server_info.sin_port = htons(5555);
  server_info.sin_addr.s_addr = inet_addr("127.0.0.1");
  // 고정된 서버 주소를 사용하여 연결을 진행한다.
  if( connect( sockfd, (struct sockaddr *)server_addr, sizeof(struct socketaddr) ) < 0 ) {
    return;
  }
 /* 라이선스 검증 코드 */
}
```

라. 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.

```
void foo() {
    int num = 10;
    char message[40];
    gets(message);5:
}
```

정상적인 프로그램의 실행은 다음과 같은 메모리 구조를 가지며, 함수 foo()의 스택 공간 끝에는 복귀 주소가 보관된다.

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.

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 저장 시 널(Null) 문자로 종료하지 않으면 의도하지 않은 결과를 가져오게 되므로 널(Null) 문자를 버퍼 범위 내에 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 때문에 문자열의 참조 시 잘못된 결과를 가져올 수 있다.

안전하지 않은 코드의 예 C

```
typedef struct _charvoid {
        char x[16];
        void * y;
        void * z;
} charvoid
void badCode() {
        charvoid cv_struct
        cv_struct.y = (void *) SRC_STR;
        printLine((char *) cv_struct.y);

        /* sizeof(cv_struct)의 사용으로 포인터 y에 덮어쓰기 발생 */
        memcpy(cv_struct.x, SRC_STR, sizeof(cv_struct));
        printLine((char *) cv_struct.x);
        printLine((char *) cv_struct.y);15:
}
```

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

```
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);
}
```

라. 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 발생할 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. 포맷문자열을 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 validDate가 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
import java.util.Calendar
.....
public static void main(String[] args) {
```

안전하지 않은 코드의 예 C#

```
        Calendar validDate = Calendar.getInstance();
        validDate.set(2014, Calendar.OCTOBER, 14);

        System.out.printf( args[0] + " did not match! HINT: It was issued on %1$terd
    of some month", validate);
}
```

사용자로부터 입력 받은 문자열을 포맷 문자열에 직접 포함시키지 않고, %s 포맷 문자열을 사용함으로써 정보유출을 방지한다.

안전한 코드의 예 JAVA

```
// 외부 입력값이 포맷 문자열 출력에 사용되지 않도록 수정
import java.util.Calendar
                :
public static void main(String[] args) {
        Calendar validDate = Calendar.getInstance();
        validDate.set(2014, Calendar.OCTOBER, 14);
      System.out.printf("%s did not match! HINT: It was issued on %2$terd of some
    month", args[0], validate);
}
```

이 예제의 msg는 신뢰할 수 없는 사용자 입력을 포함하고 있고 fprintf() 호출에서 포맷문자열 인자로 전달되기 때문에 포맷스트링 삽입에 취약하다.

안전하지 않은 코드의 예 C

```
void incorrect_password(const char *user) {
        static const char msg_format[] = "%s cannot be authenticated.₩n";
        size_t len = strlen(user) + sizeof(msg_format);
        char *msg = (char *)malloc(len);
        if (msg == NULL) {
```

안전하지 않은 코드의 예 C

```
            /* 오류 처리 */
        }
        int ret = snprintf(msg, len, msg_format, user);
            if (ret < 0 || ret >= len) {
            /* 오류 처리 */
        }
//
    fprintf(stderr, msg);
    free(msg);
    msg = NULL;
}
```

이 예제는 fprintf() 대신에 fputs()를 사용하여, msg를 포맷문자열처럼 취급하지 않고 그대로 stderr로 출력한다.

안전한 코드의 예 C

```
void incorrect_password(const char *user) {
        static const char msg_format[] = "%s cannot be authenticated.₩n";
        size_t len = strlen(user) + sizeof(msg_format);
        char *msg = (char *)malloc(len);
        if (msg == NULL) {
                /* 오류 처리 */
        }
       int ret = snprintf(msg, len, msg_format, user);
       if (ret < 0 || ret >= len) {
      /* 오류 처리 */
      }
if (fputs(msg, stderr) == EOF) {
     /* 오류 처리 */
        }
      free(msg);
      msg  =  NULL;
}
```

라. 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

제2절 보안기능

보안기능(인증, 접근제어, 기밀성, 암호화, 권한관리 등)을 부적절하게 구현 시 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(OpenSSL 이나 ESAPI의 보안기능 등)를 사용하는 것이 중요하다.

다. 코드예제

회원정보 수정 시 수정을 요청한 사용자와 로그인한 사용자의 일치 여부를 확인하지 않고 처리하고 있다.

안전하지 않은 코드의 예 JAVA

```
@RequestMapping(value = "/modify.do", method = RequestMethod.POST)
public ModelAndView memberModifyProcess(@ModelAttribute("MemberModel")
MemberModel memberModel, BindingResult result, HttpServletRequest request,
HttpSession session) {
   ModelAndView mav = new ModelAndView();
   //1. 로그인한 사용자를 불러온다.
  String userId = (String) session.getAttribute("userId");
```

안전하지 않은 코드의 예 JAVA

```
  String passwd = request.getParameter("oldUserPw");
...
  //2. 실제 수정하는 사용자와 일치 여부를 확인하지 않고, 회원정보를 수정하여  안전하지 않다.
  if (service.modifyMember(memberModel)) {
     mav.setViewName("redirect:/board/list.do");
     session.setAttribute("userName", memberModel.getUserName());
     return mav;
  } else {
    mav.addObject("errCode", 2);
    mav.setViewName("/board/member_modify");
    return mav;
  }
}
```

로그인한 사용자와 요청한 사용자의 일치 여부를 확인한 후 회원정보를 수정하도록 한다.

안전한 코드의 예 JAVA

```
@RequestMapping(value = "/modify.do", method = RequestMethod.POST)
public ModelAndView memberModifyProcess(@ModelAttribute("MemberModel")
    MemberModel memberModel, BindingResult result, HttpServletRequest request,
     HttpSession session) {
      ModelAndView mav = new ModelAndView();

      //1. 로그인한 사용자를 불러온다.
     String userId = (String) session.getAttribute("userId");
     String passwd = request.getParameter("oldUserPw");

    //2. 회원정보를 실제 수정하는 사용자와 로그인 사용자와 동일한지 확인한다.
    String requestUser = memberModel.getUserId();
    if (userId != null && requestUser != null && !userId.equals(requestUser)) {
        mav.addObject("errCode", 1);
        mav.addObject("member", memberModel);
        mav.setViewName("/board/member_modify");
        return mav;
    }
...
```

안전한 코드의 예 JAVA

```
//3. 동일한 경우에만 회원정보를 수정해야 안전하다.
if (service.modifyMember(memberModel)) {
...
```

사용자 자격인증 없이 로그인 기능을 수행하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
protected void LoginButton_Click(object sender, EventArgs e) {
// 사용자의 자격인증 과정이 없이 로그인 기능을 수행한다.
    FormsAuthentication.RedirectFromLoginPage(UserName.Text,
 RememberMe.Checked);
}
```

사용자의 자격인증 후 로그인 기능을 수행해야 한다.

안전한 코드의 예 C#

```
protected void LoginButton_Click(object sender, EventArgs e) {
// 사용자의 자격인증 과정을 수행한다.
    if(Membership.ValidateUser(UserName.Text, Password.Text)) {
        FormsAuthentication.RedirectFromLoginPage(UserName.Text,
RememberMe.Checked);
    }
}
```

라. 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 실행경로로 정보를 유출할 수 있다.

나. 보안대책

응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격 노출면[6] (Attack Surface)을 최소화하고 사용자의 권한에 따른 ACL(Access Control List)을 관리한다. 프레임워크를 사용해서 취약점을 피할 수도 있는데 예를 들면, JAAS Authorization Framework나 OWASP ESAPI Access Control 등을 인증 프레임워크로 사용 가능하다.

다. 코드예제

아래의 코드는 사용자 입력값에 따라 삭제작업을 수행하고 있으며, 사용자의 권한 확인을 위한 별도의 통제가 적용되지 않고 있다.

안전하지 않은 코드의 예 JAVA

```
private BoardDao boardDao;
String action = request.getParameter("action");
String contentId = request.getParameter("contentId");
// 요청을 하는 사용자의 delete 작원 권한 확인 없이 수행하고 있어 안전하지 않다.
    if (action != null && action.equals("delete")) {
      boardDao.delete(contentId);
  }
```

6 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

```
private BoardDao boardDao;
String action = request.getParameter("action");
String contentId = request.getParameter("contentId");
    // 세션에 저장된 사용자 정보를 얻어온다.
    User user= (User)session.getAttribute("user");
   // 사용자정보에서 해당 사용자가 delete작업의 권한이 있는지 확인한 뒤 삭제  작업을 수행한다.
   if (action != null && action.equals("delete") &&
  checkAccessControlList(user,action)) {
        boardDao.delete(contenId);
   }
}
```

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
public class AdministrationController : Controller
{
...
}
```

운영자 권한 검사 후에 개별 액션에 접근해야 한다.

안전한 코드의 예 C#

```
// 운영자 권한 검사 후 개별 액션에 접근
[Authorize(Roles  = "Administrator")]
public class AdministrationController : Controller
{
...
}
```

사용자 자격인증 없이 LDAP 검색을 시도하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
#define FIND_DN "uid=han,ou=staff,dc=example,dc=com"
int searchData2LDAP(LDAP *ld, char *username) {
  unsigned long rc;
  char filter[20];
  LDAPMessage *result;
  snprintf(filter, sizeof(filter),"(name=%s)",username);
  // 사용자의 인증 없이 LDAP 검색을 시도한다.
  rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0,
    NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
  return rc;
}
```

사용자의 자격인증 및 로그인 정보와 일치하는지 검사 후 LDAP 검색을 진행해야 한다.

안전한 코드의 예 C

```
#define FIND_DN "uid=han,ou=staff,dc=example,dc=com"
   int searchData2LDAP(LDAP *ld, char *username, char *password) {
   unsigned long rc;
char filter[20];
   LDAPMessage *result
  // username 을 인증한다.
  if ( ldap_simple_bind_s(ld, username, password) != LDAP_SUCCESS ) {
    printf("authorization error");
    return(FAIL);
  }
  // username 이 로그인 정보와 일치하는지 검사한다.
  if ( strcmp(username,getLoginName()) != 0 ) {
  printf("Login error");
  return(FAIL);
}
  snprintf(filter, sizeof(filter), "(name=%s)", username);
```

안전한 코드의 예 C

```
  rc = ldap_search_ext_s(ld, FIND_DN, LDAP_SCOPE_BASE, filter, NULL, 0,
    NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result);
  return rc;
}
```

라. 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 해당 자원을 사용하게 된다.

나. 보안대책

설정파일, 실행파일, 라이브러리 등은 SW 관리자에 의해서만 읽고 쓰기가 가능하도록 설정하고 설정파일과 같이 중요한 자원을 사용하는 경우, 허가 받지 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 true 일 경우 소유자만 실행 권한을 가지며, false 일 경우 모든 사용자가 실행 권한을 가진다.
- setReadable(p1, p2) : 첫 번째 파라미터의 true/false 값에 따라 읽기가능 여부를 결정한다. 두 번째 파라미터가 true 일 경우 소유자만 읽기권한을 가지며, false 일 경우 모든 사용자가 읽기 권한을 가진다.
- setWritable(p1, p2) : 첫 번째 파라미터의 true/false 값에 따라 쓰기가능 여부를 결정한다. 두 번째 파라미터가 true 일 경우 소유자만 쓰기권한을 가지며, false 일 경우 모든 사용자가 쓰기 권한을 가진다.

안전하지 않은 코드의 예 JAVA

```
File file = new File("/home/setup/system.ini");
// 모든 사용자에게 실행 권한을 허용하여 안전하지 않다.
file.setExecutable(true, false);
// 모든 사용자에게 읽기 권한을 허용하여 안전하지 않다.
file.setReadable(true, false);
// 모든 사용자에게 쓰기 권한을 허용하여 안전하지 않다.
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

```
File file = new File("/home/setup/system.ini");
// 소유자에게 실행 권한을 금지하였다.
file.setExecutable(false);
// 소유자에게 읽기 권한을 허용하였다.
file.setReadable(true);
// 소유자에게 쓰기 권한을 금지하였다.
file.setWritable(false);
```

아래의 C# 코드는 모든 사용자에게 파일의 권한을 부여하고 있다.

안전하지 않은 코드의 예 C#

```
public static void AddDirectorySecurity(string FileName)
{
    // 디렉토리 정보 객체 생성
    DirectoryInfo dInfo = new DirectoryInfo(FileName);
    DirectorySecurity dSecurity = dInfo.GetAccessControl();

    // 모든 사용자에게 권한 부여.
    dSecurity.AddAccessRule(new FileSystemAccessRule("everyone",
    FileSystemRights.FullControl,
    InheritanceFlags.ObjectInherit | InheritanceFlags.ContainerInherit,
    PropagationFlags.NoPropagateInherit, AccessControlType.Allow));
    dInfo.SetAccessControl(dSecurity);
}
```

적절한 권한을 파일에 설정해야 한다.

안전한 코드의 예 C#

```
public static void AddDirectorySecurity(string FileName, string Account,
      FileSystemRights Rights, AccessControlType ControlType)
{
    // 디렉토리 정보 객체 생성
    DirectoryInfo dInfo = new DirectoryInfo(FileName);
    DirectorySecurity dSecurity = dInfo.GetAccessControl();

    // FileSystemAccessRule 에 권한 설정
    dSecurity.AddAccessRule(new FileSystemAccessRule(Account,
                                                     Rights,
                                                     ControlType));
        dInfo.SetAccessControl(dSecurity);
}
```

모든 사용자에게 권한을 부여하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
// 모든 사용자가 읽기/쓰기 권한을 갖게 된다.
umask(0);
FILE *out = fopen("file_name", "w");
if(out) {
  fprintf(out, "secure code₩n");
  fclose(out);
```

umask()를 사용하여 권한 설정을 할 때, 올바른 권한을 설정해야 합니다.

안전한 코드의 예 C

```
// 유저 외에는 아무런 권한을 주지 않는다.
umask(077);
FILE *out = fopen("file_name", "w");
if(out) {
  fprintf(out, "secure code₩n");
  fclose(out);
```

라. 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 패스워드를 보호하기 위하여 간단한 인코딩 함수를 이용하여 패스워드를 감추는 방법을 사용하기도 한다. 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수로는 패스워드를 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 몇 십 억년이 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, RC5등의 암호알고리즘을 대 신하여, 3DES, AES, SEED 등의 안전한 암호알고리즘으로 대체하여 사용한다. 또한, 업무관련 내용, 개인정보 등에 대한 암호 알고리즘 적용 시, 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
해쉬함수		• SHA-224/256/384/512
메시지 인증코드	해쉬함수기반	• HMAC
	블록기반	• CMAC, GMAC

분류		보호함수 목록
난수발생기	해쉬함수/HMAC 기반	• HASH_DRBG, HMAC_DRBG
	블록기반	• CTR_DRBG
공개키 암호		• RSAES – (공개키 길이) 2048, 3072 – RSA-OAEP에서 사용되는 해쉬함수 : SHA-224/256
전자서명		• RSA-PSS, KCDSA, ECDSA, EC-KCDSA
키 설정 방식		• DH, ECDH
보호함수		보호함수 파라미터
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

다. 코드예제

다음 예제는 취약한 DES 알고리즘으로 암호화하고 있다.

안전하지 않은 코드의 예 JAVA

```
import java.security.*;
import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
public class CryptoUtils {
    public byte[] encrypt(byte[] msg, Key k) {
        byte[] rslt = null;
        try {
// 키 길이가 짧아 취약함 암호와 알고리즘인 DES를 사용하여 안전하지 않다.
            Cipher c = Cipher.getInstance("DES");
            c.init(Cipher.ENCRYPT_MODE, k);
            rslt = c.update(msg);
        }
```

아래 코드처럼 안전하다고 알려진 AES 알고리즘 등을 적용해야 한다.

안전한 코드의 예 JAVA

```
import java.security.*;
import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;

public class CryptoUtils {
    public byte[] encrypt(byte[] msg, Key k) {
        byte[] rslt = null;
        try {
// 키 길이가 길어 강력한 알고리즘인 AES를 사용하여 안전하다.
            Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
            c.init(Cipher.ENCRYPT_MODE, k);
            rslt = c.update(msg);
        }
```

다음 예제는 취약한 DES 알고리즘을 사용하고 있다.

안전하지 않은 코드의 예 C#

```
static string Enc(string input) {
// 키 길이가 짧아 취약함 암호와 알고리즘인 DES를 사용하여 안전하지 않다.
var des = new DESCryptoServiceProvider();
...
}
```

아래 코드처럼 안전하다고 알려진 AES 알고리즘 등을 적용해야 한다.

안전한 코드의 예 C#

```
 static string Enc(string input) {
// 키 길이가 길어 강력한 알고리즘인 AES를 사용하여 안전하다.
    var des = new AesCryptoServiceProvider();
    ...
  }
```

다음 예제는 취약한 DES 알고리즘을 사용하고 있는 C 코드이다.

안전하지 않은 코드의 예 C

```
EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
// 취약한 DES 알고리즘을 사용한다.
EVP_EncryptInit(&ctx, EVP_des_ecb(), NULL, NULL);
```

안전하다고 알려진 AES 알고리즘을 사용하도록 한다.

안전한 코드의 예 C

```
EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
// 안전한 AES 알고리즘을 사용한다.
EVP_EncryptInit(&ctx, EVP_aes_128_cbc(), key, iv);
```

라. 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

사용자 또는 시스템의 중요정보가 포함된 데이터를 평문으로 송·수신 또는 저장할 때 인가되지 않은 사용자에게 민감한 정보가 노출될 수 있는 보안약점이다.

나. 보안대책

개인정보(주민등록번호, 여권번호 등), 금융정보(카드번호, 계좌번호 등), 패스워드 등 중요정보를 통신채널로 전송하거나 저장할 때는 반드시 암호화 과정을 거쳐야 한다. 필요한 경우 SSL 또는 HTTPS 등과 같은 암호채널을 사용해야 하며, HTTPS와 같은 보안 채널을 사용하여 브라우저 쿠키에 중요 데이터를 저장하는 경우, 쿠키객체에 보안속성을 반드시 설정하여 중요정보의 노출을 방지한다. 중요정보를 읽거나 쓸 경우에 권한인증 등으로 적합한 사용자가 중요정보에 접근하도록 해야 한다.

다. 코드예제

- 중요정보 평문저장

아래 예제는 인증을 통과한 사용자의 패스워드 정보가 평문으로 DB에 저장된다.

안전하지 않은 코드의 예 JAVA

```
String id = request.getParameter("id");
// 외부값에 의해 패스워드 정보를 얻고 있다.
String pwd = request.getParameter("pwd");
.....
String sql = " insert into customer(id, pwd, name, ssn, zipcode, addr)"
             + " values (?, ?, ?, ?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, id);
stmt.setString(2, pwd);
.....
// 입력받은 패스워드가 평문으로 DB에 저장되어 안전하지 않다.
stmt.executeUpdate();
```

다음 예제는 패스워드 등 중요 데이터를 해쉬값으로 변환하여 저장하고 있다.

안전한 코드의 예 JAVA

```
String id  = request.getParameter("id");
// 외부값에 의해 패스워드 정보를 얻고 있다.
String pwd = request.getParameter("pwd");
// 패스워드를 솔트값을 포함하여 SHA-256 해쉬로 변경하여 안전하게 저장한다.
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.reset();
md.update(salt);
byte[] hashInBytes = md.digest(pwd.getBytes());
StringBuilder sb = new StringBuilder();
        for (byte b : hashInBytes) {
            sb.append(String.format("%02x", b));
        }
pwd = sb.toString();
.....
String sql = " insert into customer(id, pwd, name, ssn, zipcode, addr)"
              + " values (?, ?, ?, ?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, id);
stmt.setString(2, pwd);
.....
stmt.executeUpdate();
```

다음은 사용자의 패스워드를 평문으로 저장해 놓고 출력하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
namespace Security
{
   public class FindPassword : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         var userId = "tmp";
         MembershipUser user = Membership.GetUser(userId);
```

안전하지 않은 코드의 예 C#

```
            if (user != null)
            {
                var password = user.GetPassword();
                Response.Write(password);
            }
            else
            {
                Response.Write("the given userId is not valid");
            }
        }
    }
}
```

패스워드는 암호화 하여 저장해야 하며, 출력하지 않도록 한다.

안전한 코드의 예 C#

```
namespace Security
{
    public class FindPassword : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            var userId = "tmp";
            MembershipUser user = Membership.GetUser(userId);
            if (user != null)
            {
                var encrypetedPassword = user.GetPassword();
                SecureFindPasswordFunction();
            }
            else
            {
                Response.Write("the given userId is not valid");
            }
        }
    }
}
```

• 중요정보 평문전송

아래 예제는 패스워드를 암호화하지 않고 네트워크로 전송하고 있다. 이 경우 패킷 스니핑으로 패스워드가 노출될 수 있다.

안전하지 않은 코드의 예 JAVA

```
try {
      Socket s = new Socket("taranis", 4444);
      PrintWriter o = new PrintWriter(s.getOutputStream(), true);
// 패스워드를 평문으로 전송하여 안전하지 않다.
     String password = getPassword();
     o.write(password);
} catch (FileNotFoundException e) {
     ......
```

아래 예제는 패스워드를 네트워크로 서버로 전송하기 전에 AES 등의 안전한 암호알고리즘으로 암호화한 안전한 프로그램이다.

안전한 코드의 예 JAVA

```
// 패스워드를 암호화 하여 전송
try {
      Socket s = new Socket("taranis", 4444);
      PrintStream o = new PrintStream(s.getOutputStream(), true);
// 패스워드를 강력한 AES암호화 알고리즘으로 전송하여 사용한다.
      Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");

      String password = getPassword();
      byte[] encPassword = c.update(password.getBytes());
    o.write(encPassword, 0, encPassword.length);
} catch (FileNotFoundException e) {
         ......
```

아래 C# 예제 또한 패스워드를 암호화하지 않고 네트워크로 전송하고 있다. 이 경우 패킷 스니핑으로 패스워드가 노출될 수 있다.

안전하지 않은 코드의 예 C#

```
public void EmailPassword_OnClick(object sender, EventArgs args)
{
  MembershipUser u = Membership.GetUser(UsernameTextBox.Text, false);
  String password;

   if (u != null)
  {
    try
    {
      password = u.GetPassword(); // sensitive data created
    }
    catch (Exception e)
    {
      Msg.Text = "An exception occurred retrieving your password: " +
    Server.HtmlEncode(e.Message);
      return;
    }
    MailMessage Message = new MailMessage();
    Message.Body = "Your password is: " + Server.HtmlEncode(password);
//패스워드가 포함된 메시지를 네트워크로 전송하고 있다.
    SmtpMail.Send(Message);
    Msg.Text = "Password sent via e-mail.";
  }
  else
  {
    Msg.Text = "User name is not valid. Please check the value and try again.";
  }
}
```

패스워드를 네트워크로 전송할 때에는 암호화 하는 것이 바람직하다.

안전한 코드의 예 C#

```
public void EmailPassword_OnClick(object sender, EventArgs args)
{
  MembershipUser u = Membership.GetUser(UsernameTextBox.Text, false);
  String password;

  if (u != null)
  {
    try
    {
      password = u.GetPassword();
      byte[] data = System.Text.Encoding.ASCII.GetBytes(password);
      data = new
      System.Security.Cryptography.SHA256Managed().ComputeHash(data);
      String hashedPassword = System.Text.Encoding.ASCII.GetString(data);
    }
    catch (Exception e)
    {
      Msg.Text = "An exception occurred retrieving your password: " +
    Server.HtmlEncode(e.Message);
      return;
    }
    MailMessage Message = new MailMessage();
    Message.Body = "Your password is: " + Server.HtmlEncode(hasedPassword);
    SmtpMail.Send(Message);
    Msg.Text = "Password sent via e-mail.";
  }
  else
  {
    Msg.Text = "User name is not valid. Please check the value and try again.";
  }
}
```

파일에서 읽어온 패스워드를 바로 사용하는 C예제 코드이다.

안전하지 않은 코드의 예 C

```
int dbaccess() {
   FILE *fp; char *server = "DBserver";
   char passwd[20];
   char user[20];
   SQLHENV henv;
   SQLHDBC hdbc;
   fp = fopen("config", "r");
   fgets(user, sizeof(user), fp);
// 패스워드를 파일에서 읽어온다.
   fgets(passwd, sizeof(passwd), fp);
   fclose(fp);
   SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
   SQLConnect(hdbc,
                          (SQLCHAR*) server,
                          (SQLSMALLINT) strlen(server),
                          (SQLCHAR*) user,
                          (SQLSMALLINT) strlen(user),
                          // 패스워드 암호화 없이 직접 연결한다.
                          (SQLCHAR*) passwd,
                          (SQLSMALLINT) strlen(passwd) );
   return 0;
```

외부에서 입력된 패스워드는 검증 과정을 거쳐 사용해야 한다.

안전한 코드의 예 C

```
int dbaccess() {
   FILE *fp; char *server = "DBserver";
   char passwd[20];
   char user[20];
   char *encPasswd;
   char *key;
   SQLHENV henv;
    SQLHDBC hdbc;
   // AES-CBC로 암호화 모드를 설정한다.
   HCkCrypt2 crypt = CkCrypt2_putCryptAlgorithm(crypt,"aes");
   CkCrypt2_putCipherMode(crypt,"cbc");
   // 외부에서 암호화 키를 불러와 설정한다.
   key = getenv("encrypt_key");
   CkCrypt2_SetEncodedKey(crypt,key,"hex");
   fp = fopen("config", "r");
   fgets(user, sizeof(user), fp);
   // 패스워드를 파일에서 읽어온다.
   fgets(passwd, sizeof(passwd), fp);
   fclose(fp);
   // 패스워드 암호화를 진행한다.
   encPasswd = CkCrypt2_encryptStringENC(crypt, passwd);
   SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
   SQLConnect(hdbc,
                              (SQLCHAR*) server,
                              (SQLSMALLINT) strlen(server),
                              (SQLCHAR*) user,
                              (SQLSMALLINT) strlen(user),
                            // 암호화된 패스워드를 사용한다.
                              (SQLCHAR*) encPasswd,
                              (SQLSMALLINT) strlen(verifiedPwd) );
   return 0;
}
```

라. 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 Insecure Transport, OWASP, https://www.owasp.org/index.php/Insecure_Transport

6. 하드코드된 중요정보

가. 개요

프로그램 코드 내부에 하드코드된 패스워드 또는 암호화키를 포함하여 내부 인증에 사용하거나 암호화를 수행하면 중요정보(관리자 정보, 암호화된 정보 등)가 유출될 수 있는 보안약점이다.

나. 보안대책

패스워드는 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 패스워드를 소스코드 내부에 상수 형태로 하드코딩 하는 경우, 접속 정보가 노출될 수 있어 위험하다.

안전하지 않은 코드의 예 JAVA

```
public class MemberDAO {
    private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
    private static final String URL  = "jdbc:oracle:thin:@192.168.0.3:1521:ORCL";
    private static final String USER = "SCOTT"; // DB ID;
// DB 패스워드가 소스코드에 평문으로 저장되어 있다.
    private static final String PASS = "SCOTT"; // DB PW;
.....
    public Connection getConn() {
        Connection con = null;
        try {
            Class.forName(DRIVER);
            con = DriverManager.getConnection(URL, USER, PASS);
  .....
```

패스워드는 안전한 암호화 방식으로 암호화하여 별도의 분리된 공간(파일)에 저장해야 하며, 암호화된 패스워드를 사용하기 위해서는 복호화 과정을 거쳐야 한다.

안전한 코드의 예 JAVA

```
public class MemberDAO {
      private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
      private static final String URL  = "jdbc:oracle:thin:@192.168.0.3:1521:ORCL";
      private static final String USER = "SCOTT"; // DB ID
  ......
      public Connection getConn() {
         Connection con = null;
         try {
              Class.forName(DRIVER);
// 암호화된 패스워드를 프로퍼티에서 읽어들여 복호화해서 사용해야한다.
              String PASS = props.getProperty("EncryptedPswd");
              byte[] decryptedPswd = cipher.doFinal(PASS.getBytes());
              PASS = new String(decryptedPswd);
              con = DriverManager.getConnection(URL, USER, PASS);
  ......
```

Credential 객체를 생성하기 위한 패스워드를 소스코드 내부에 상수 형태로 하드코딩 하는 경우, 접속 정보가 노출될 수 있어 위험하다.

안전하지 않은 코드의 예 C#

```
string UserName = "username";
string Password = "password";
// 평문으로 저장된 패스워드를 이용하여 NetworkCredential 생성
NetworkCredential myCred = new NetworkCredential(UserName, Password);
```

암호화된 패스워드로 Credential 객체를 생성한다.

안전한 코드의 예 C#

```
string UserName = "username";
string Password = "password";
SecureString SecurelyStoredPassword = new SecureString();

foreach (char c in Password)
{
    SecurelyStoredPassword.AppendChar(c);
}

// 암호화된 패스워드를 사용하여 NetworkCredential 생성
NetworkCredential secure_myCred = new NetworkCredential(UserName,
    SecurelyStoredPassword);
```

하드코드된 패스워드를 바로 사용하는 C예제 코드이다.

안전하지 않은 코드의 예 C

```
int dbaccess(char *server, char *user) {
  SQLHENV henv;
  SQLHDBC hdbc;
  char *password = “password”;
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
// 하드코드된 패스워드를 사용
  SQLConnect(hdbc,(SQLCHAR*)server,strlen(server),user,strlen(user),
    password, strlen(password));
  return 0;
```

패스워드는 코드 상에서 보이지 않게 사용해야 한다. 아래의 예제는 환경 변수를 이용하여 패스워드를 사용한다.

안전한 코드의 예 C

```
int dbaccess(char *server, char *user, char *passwd) {
  SQLHENV henv;
  SQLHDBC hdbc;
  // 패스워드를 외부에서 불러와서 사용
  char *password = getenv("password");
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
  SQLConnect(hdbc, (SQLCHAR*) server, strlen(server), user, strlen(user),
    password, strlen(password));
  SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
  SQLFreeHandle(SQL_HANDLE_ENV, henv);
  return 0;
}
```

- 하드코드된 암호화 키

소스코드 내부에 암호화 키를 상수 형태로 하드코딩하여 사용하면 악의적인 공격자에게 암호화 키가 노출될 위협이 있다.

안전하지 않은 코드의 예 JAVA

```
import javax.crypto.KeyGenerator;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Cipher;
    ……
public String encriptString(String usr) {
// 암호화 키를 소스코드 내부에 사용하는 것은 안전하지 않다.
  String key = "22df3023sf~2;asn!@#/>as";
  if (key != null) {
        byte[] bToEncrypt = usr.getBytes("UTF-8");
        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

```
import javax.crypto.KeyGenerator;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Cipher;
    ......
public String encriptString(String usr) {
// 암호화 키는 외부 파일에서 암호화 된 형태로 저장하고, 사용시 복호화 한다.
  String key = getPassword("./password.ini");
  key = decrypt(key);
  if (key != null) {
      byte[] bToEncrypt = usr.getBytes("UTF-8");
      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.
byte[] key = new byte[] { 0x43, 0x87, 0x23, 0x72 };
byte[] iv = new byte[] { 0x43, 0x87, 0x23, 0x72 };
FileStream fStream = File.Open(fileName, FileMode.OpenOrCreate);

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.

byte[] key = GetKey(./password.ini);
byte[] iv = GetIV(./password.ini);
FileStream fStream = File.Open(fileName, FileMode.OpenOrCreate);

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 있다.

안전하지 않은 코드의 예 C

```
typedef int SQLSMALLINT;
int dbaccess(char *user, char *passwd) {
  char *server = "DBserver";
  char *cpasswd;
  SQLHENV henv;
  SQLHDBC hdbc;
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
  // 암호화된 비밀번호와 솔트를 사용한다. 코드에 접근 권한이 있는 사용자는 해당 비밀번호와
     솔트를 획득할 수 있다.
    cpasswd = crypt(passwd, "salt");
  if (strcmp(cpasswd, "68af404b513073582b6c63e6b") != 0) {
    // USE_OF_HARDCODED_CRYPTOGRAPHIC_KEY
    printf("Incorrect password \n");
    return -1;
  }
```

암호화되어 저장된 암호를 외부에서 불러오고 이를 비교하는 코드가 작성되어야 한다.

안전한 코드의 예 C

```
extern char *salt;
typedef int SQLSMALLINT;
int dbaccess(char *user, char *passwd) {
  char *server = "DBserver";
  char *cpasswd;
  // 외부에 있는 암호화된 비밀번호와 솔트를 불러온다.
  char* storedpasswd = getenv("password");
  char* salt = getenv("salt");
  SQLHENV henv;
  SQLHDBC hdbc;
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
  // 외부에서 불러온 솔트 값을 사용해 비밀번호를 암호화한다.
  cpasswd = crypt(passwd, salt);
  // 암호화된 비밀번호와 외부에서 불러온 값을 비교한다.
  if (strcmp(cpasswd, storedpasswd) != 0) {
     printf("Incorrect password \n");
     SQLFreeHandle(SQL_HANDLE_DBC, &hdbc);
     SQLFreeHandle(SQL_HANDLE_ENV, &henv);
      return -1;
   }
}
```

라. 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 패스워드를 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

```
public static final String ALGORITHM = "RSA";
public static final String PRIVATE_KEY_FILE = "C:/keys/private.key";
public static final String PUBLIC_KEY_FILE = "C:/keys/public.key";
public static void generateKey() {
    try {
        final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
// RSA 키 길이를 1024 비트로 짧게 설정하는 경우 안전하지 않다.
        keyGen.initialize(1024);
        final KeyPair key = keyGen.generateKeyPair();
        File privateKeyFile = new File(PRIVATE_KEY_FILE);
            File publicKeyFile = new File(PUBLIC_KEY_FILE);
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 JAVA

```
public static final String ALGORITHM = "RSA";
public static final String PRIVATE_KEY_FILE = "C:/keys/private.key";
public static final String PUBLIC_KEY_FILE = "C:/keys/public.key";
public static void generateKey() {
    try {
        final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
        keyGen.initialize(2048);
        final KeyPair key = keyGen.generateKeyPair();
        File privateKeyFile = new File(PRIVATE_KEY_FILE);
        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#

```
static string UseRSA(string input) {
// RSA 키 길이를 1024 비트로 짧게 설정하는 경우 안전하지 않다.
    var rsa = new RSACryptoServiceProvider(1024);
    ...
  }
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 C#

```
static string UseRSA(string input) {
// RSA 키 길이를 2048 비트 이상으로 길게 설정한다.
    var rsa = new RSACryptoServiceProvider(2048);
    ...
  }
```

다음의 예제는 보안성이 강한 RSA 알고리즘을 사용함에도 불구하고, 키 사이즈를 작게 설정함으로써 프로그램의 보안약점을 야기한 경우이다.

안전하지 않은 코드의 예 C

```
EVP_PKEY *RSAKey() {
  EVP_PKEY *pkey;
  RSA *rsa;
// RSA 키 길이를 512 비트로 짧게 설정하는 경우 안전하지 않다.
  rsa = RSA_generate_key(512, 35, NULL, NULL);
  if(rsa == NULL) {
    printf("Error \n");
    return NULL;
  }
 pkey = EVP_PKEY_new();
 EVP_PKEY_assign_RSA(pkey, rsa);
 return pkey;
```

공개키 암호화에 사용하는 키의 길이는 적어도 2048비트 이상으로 설정한다.

안전한 코드의 예 C

```
EVP_PKEY *RSAKey() {
  EVP_PKEY *pkey;
  RSA *rsa;
// 2048비트 이상으로 설정한 후에 사용해야 한다.
  rsa = RSA_generate_key(2048, 35, NULL, NULL);
  if(rsa == NULL) {
    printf("Error \n");
    return NULL;
  }
  pkey = EVP_PKEY_new();
  EVP_PKEY_assign_RSA(pkey, rsa);
  return pkey;
}
```

라. 참고자료

① CWE-326 Inadequate Encryption Strength, MITRE, http://cwe.mitre.org/data/definitions/326.html

8. 적절하지 않은 난수값 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 발생 한다. 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, C 에서는 rand()함수 사용 시 매번 변경되는 기본 시드(Seed)값이 없으므로, srand()로 매번 변경 되는 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

```
import java.util.Random;
...
public Static int getRandomValue(int maxValue) {
// 고정된 시드값을 사용하여 동일한 난수값이 생성되어 안전하지 않다.
    Random random = new Random(100);
    return random.nextInt(maxValue);
```

안전하지 않은 코드의 예 JAVA

```
}
public Static String getAuthKey() {
// 매번 변경되는 시드값을 사용하여 다른 난수값이 생성되나 보안결정을 위한 난수로는 안전하지 않다.
    Random random = new Random();
    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 사용 시에는 java.security.SecureRandom 클래스를 사용하는 것이 보다 안전하다.

안전한 코드의 예 JAVA

```
import java.util.Random;
import java.security.SecureRandom;
...
public Static int getRandomValue(int maxValue) {
// setSeed로 매번 변경되는 시드값을 설정 하거나, 기본값인 현재 시간 기반으로 매번 변경되는
   시드값을 사용하도록 한다.
   Random random = new Random();
   return random.nextInt(maxValue);
}
public Static String getAuthKey() {
// 보안결정을 위한 난수로는 예측이 거의 불가능하게 암호학적으로 보호된 SecureRandom을
   사용한다.
try {
    SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");
  MessageDigest digest = MessageDigest.getInstance("SHA-256");
  secureRandom.setSeed(secureRandom.generateSeed(128));
  String authKey = new String(digest.digest((secureRandom.nextLong() +
  "").getBytes()));
...
} catch (NoSuchAlgorithmException e) {
```

보안결정을 위한 난수로는 안전하지 않은 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
static int GenerateDigit()
{
  // 매번 변경되는 시드값을 사용하여 다른 난수값이 생성되나 보안결정을 위한 난수로는 안전하지 않다.
  Random rng = new Random();
  return rng.Next(10);
}
```

암호학적으로 보호된 SecureRandom 을 사용한다.

안전한 코드의 예 C#

```
static int GenerateDigitGood()
{
// 보안결정을 위한 난수로는 예측이 거의 불가능하게 암호학적으로 보호된
    SecureRandom을 사용한다.
      byte[] b = new byte[4];
      new System.Security.Cryptography.RNGCryptoServiceProvider().GetBytes(b);
      return (b[0] & 0x7f) << 24 | b[1] << 16 | b[2] << 8 | b[3];
}
```

Seeding 없이 사용하는 rand() 함수는 암호화에 사용되기 힘듭니다.

안전하지 않은 코드의 예 C

```
void foo() {
   int i;
   for(i=0; i<20; i++)
// 프로그램을 여러 번 실행 했을 때 얻는 결과 값이 같다. 범위가 작기 때문에 암호화에 사용되기
    힘들다.
    printf("%d", rand());
```

srandom(), random() 함수를 사용하면 보안적으로 안전한 난수를 얻을 수 있다.

안전한 코드의 예 C

```
void foo() {
  srandom(time(NULL));
  int i;
  for(i=0; i<20; i++)
    printf("%ld", random());
}
```

라. 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 「패스워드 선택 및 이용 안내서」의 안전한 패스워드 설정규칙을 적용해야 한다.

나. 보안대책

비밀번호 생성 시 강한 조건 검증을 수행한다. 비밀번호(패스워드)는 숫자와 영문자, 특수문자 등을 혼합하여 정해진 자릿수를 사용하여 생성되도록 하고, 주기적으로 변경하도록 해야 한다.

다. 코드예제

가입자가 입력한 비밀번호에 대한 복잡도 검증 없이 가입 승인 처리를 수행하고 있다.

안전하지 않은 코드의 예 JAVA

```
String id = request.getParameter("id");
String pass = request.getParameter("pass");
UserVo userVO = new UserVo(id, pass);
.....
// 비밀번호의 자릿수, 특수문자 포함 여부 등 복잡도를 체크하지 않고 등록
String result = registerDAO.register(userVO);
```

사용자 계정을 보호하기 위해 가입 시, 비밀번호 복잡도 검증 후 가입 승인처리를 수행한다.

안전한 코드의 예 JAVA

```
String id = request.getParameter("id");
String pass = request.getParameter("pass");
// 비밀번호에 자릿수, 특수문자 포함 여부 등의 복잡도를 체크하고 등록하게 한다.
Pattern pattern = Pattern.compile("((?=.*[a-zA-Z])(?=.*[0-9@#$%]). {9, })");
Matcher matcher = pattern.matcher(pass);
```

안전한 코드의 예 JAVA

```
if (!matcher.matches()) {
        return "비밀번호 조합규칙 오류";
}
UserVo userVO = new UserVo(id, pass);
.....
String result = registerDAO.register(userVO);
```

빈 비밀번호를 허용하는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
// 빈 비밀번호를 허용
NetworkCredential myCred = new NetworkCredential(UserName, "");
```

빈 비밀번호를 허용하지 않도록 한다.

안전한 코드의 예 C#

```
// 빈 비밀번호를 사용하지 않음
NetworkCredential secure_myCred = new NetworkCredential(UserName, Password);
```

비밀번호에 대한 검증 없이 사용하는 C코드 예제이다.

안전하지 않은 코드의 예 C

```
bool authentication(char* id, char* pwd)
{
  MYSQL *connectInstance;
  connectInstance = mysql_init(NULL);
// 패스워드 값에 대한 검증없이 사용한다.
   mysql_real_connect(connectInstance, "192.168.100.211", id, pwd,
     "database", 0, NULL, 0);
    ...
```

비밀번호에 대한 적절한 검증이 필요하다.

안전한 코드의 예 C

```
bool authentication(char* id, char* pwd)
{
  MYSQL *connectInstance;
  connectInstance = mysql_init(NULL);
// 패스워드에 대한 적절한 검증을 수행해야 한다.
   if( checkValidationId( id ) == true && checkValidationPwd( pwd ) == true )
  {
    mysql_real_connect(connectInstance, "192.168.100.211", id, pwd,
        "database", 0, NULL, 0);
 }
 ...
}
```

라. 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

```
File f = new File(downloadedFilePath);
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.

안전한 코드의 예 JAVA

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f, true);
Enumeration<JarEntry> ens = jf.entries();
while (ens.hasMoreElements()) {
        JarEntry en = ens.nextElement();
        if (!en.isDirectory()) {
                if (en.toString().equals(path)) {
                byte[] data = readAll(jar.getInputStream(en), en.getSize());
                CodeSigner[] signers = en.getCodeSigners();
                ...
                }
        }
}
jf.close();
```

라. 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

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

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
foo=SSL_get_verify_result(ssl);
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

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {
 /* 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. */
 }
```

아래 예제는 인증서 DN 일치여부와 유효기간 등을 검증한다.

안전한 코드의 예 JAVA

```
private boolean verifySignature(X509Certificate toVerify, X509Certificate signingCert) {
        /* 검증하려는 호스트 인증서(toVerify)와 CA인증서(signing Cert)의 DN(Distinguished
           Name)이 일치하는지 여부를 확인한다.*/
        if (!toVerify.getIssuerDN().equals(signingCert.getSubjectDN())) return false;
        try {
                // 호스트 인증서가 CA인증서로 서명 되었는지 확인한다.
                toVerify.verify(signingCert.getPublicKey());
                // 호스트 인증서가 유효기간이 만료되었는지 확인한다.
                toVerify.checkValidity();
                return true;
        } catch (GeneralSecurityException verifyFailed) {
                return false;
        }
}
```

또한, 유효기간이 남아 있는 인증서의 해지여부를 확인하기 위해서는 CRL(Certificate revocation lists)으로 인증서 해지목록 또는 OCSP(Online Certificate Status Protocol)으로 실시간 인증서 상태확인이 필요하다. CRL 리스트는 해당 인증서를 참조하여 인증기관에서 다운받을 수

있으며, 다운받은 CRL으로 해지된 인증서를 확인할 수 있다.

라. 참고자료

① CWE-295: Improper Certificate Validation, MITRE, http://cwe.mitre.org/data/definitions/295.html

12. 사용자 하드디스크에 저장되는 쿠키를 통한 정보노출

가. 개요

대부분의 웹 응용프로그램에서 쿠키는 메모리에 상주하며, 브라우저의 실행이 종료되면 사라진다. 프로그래머가 원하는 경우, 브라우저 세션에 관계없이 지속적으로 저장되도록 설정할 수 있으며, 이것은 디스크에 기록되고, 다음 브라우저 세션이 시작되었을 때 메모리에 로드된다. 개인정보, 인증 정보 등이 이와 같은 영속적인 쿠키(Persistent Cookie)에 저장된다면, 공격자는 쿠키에 접근할 수 있는 보다 많은 기회를 가지게 되며, 이는 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

```
Cookie loginCookie = new Cookie("rememberme", "YES");
// 쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있어 안전하지 않다.
loginCookie.setMaxAge(60*60*24*365);
response.addCookie(loginCookie);
```

쿠키의 만료시간은 해당 기능에 맞춰 최소로 설정하고 영속적인 쿠키에는 중요 정보가 포함되지 않도록 한다.

안전한 코드의 예 JAVA

```
Cookie loginCookie = new Cookie("rememberme", "YES");
// 쿠키의 만료시간은 해당 기능에 맞춰 최소로 사용한다.
loginCookie.setMaxAge(60*60*24);
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#

```
HttpCookie cookie = Request.Cookies.Get(“ExampleCookie”);
// 쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있어 안전하지 않다.
cookie.Expires = DateTime.Now.AddMinutes(60.0*24.0*365.0);
Response.Cookies.Add(cookie);
```

쿠키의 만료 시간을 10분으로 설정하고 있는 C# 코드의 예제이다.

안전한 코드의 예 C#

```
HttpCookie cookie = Request.Cookies.Get(“ExampleCookie”);
// 쿠키의 만료시간은 해당 기능에 맞춰 최소로 사용한다.
cookie.Expires = DateTime.Now.AddMinutes(10d);
Response.Cookies.Add(cookie);
```

라. 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

패스워드를 주석문에 넣어두면 시스템 보안이 훼손될 수 있다. 소프트웨어 개발자가 편의를 위해서 주석문에 패스워드를 적어둔 경우, 소프트웨어가 완성된 후에는 그것을 제거하는 것이 매우 어렵게 된다. 또한, 공격자가 소스코드에 접근할 수 있다면, 아주 쉽게 시스템에 침입할 수 있다.

나. 보안대책

주석에는 ID, 패스워드 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, 패스워드의 중요한 정보를 노출시켜 안전하지 않다.
// DB연결 root / a1q2w3r3f2!@
con = DriverManager.getConnection(URL, USER, PASS);
```

프로그램 개발 시에 주석문 등에 남겨놓은 사용자 계정이나 패스워드 등의 정보는 개발 완료 시에 확실하게 삭제하여야 한다.

안전한 코드의 예 JAVA

```
// ID, 패스워드등의 중요 정보는 주석에 포함해서는 안된다.
con = DriverManager.getConnection(URL, USER, PASS);
```

주석에 패스워드를 포함하고 있는 C# 코드이다.

안전하지 않은 코드의 예 C#

```
// 주석문으로 DB연결 ID, 패스워드의 중요한 정보를 노출시켜 안전하지 않다.
// DB연결 root / a1q2w3r3f2!@
conn = customGetConnection(USER, PASS);
```

프로그램 개발 시에 주석문 등에 남겨놓은 사용자 계정이나 패스워드 등의 정보는 개발 완료 시에 확실하게 삭제하여야 한다.

안전한 코드의 예 C#

```
// ID, 패스워드등의 중요 정보는 주석에 포함해서는 안된다.
conn = customGetConnection(USER, PASS);
```

주석에 패스워드를 포함하고 있는 C 예제 코드이다.

안전하지 않은 코드의 예 C

```
/* password is "admin" */
/* passwd is "admin" */
int verfiyAuth(char *ipasswd, char *orgpasswd) {
  char *admin = "admin";
  if(strncmp(ipasswd, oprgpasswd, sizeof(ipasswd)) != 0) {
     printf("Authentication Fail! \n");
  }
  return admin;
```

불필요한 주석은 삭제해야 한다.

안전한 코드의 예 C

```
int verfiyAuth(char *ipasswd, char *orgpasswd) {
  char *admin = "admin";
  if(strncmp(ipasswd, oprgpasswd, sizeof(ipasswd)) != 0) {
     printf("Authentication Fail! \n");
  }
  return admin;
}
```

라. 참고자료

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

14. 솔트 없이 일방향 해쉬함수 사용

가. 개요

패스워드를 저장 시 일방향 해쉬함수의 성질을 이용하여 패스워드의 해쉬값을 저장한다. 만약 패스워드를 솔트(Salt)없이 해쉬하여 저장한다면, 공격자는 레인보우 테이블과 같이 해쉬값을 미리 계산 하여 패스워드를 찾을 수 있게 된다.

나. 보안대책

패스워드를 저장 시 패스워드와 솔트를 해쉬함수의 입력으로 하여 얻은 해쉬값을 저장한다.

다. 코드예제

다음의 예제는 패스워드 저장 시 솔트 없이 패스워드에 대한 해쉬값을 얻는 과정을 보여준다.

안전하지 않은 코드의 예 JAVA

```
public String getPasswordHash(String password) throws Exception {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
// 해쉬에 솔트를 적용하지 않아 안전하지 않다.
    md.update(password.getBytes());
    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();
}
```

패스워드만을 해쉬함수의 입력으로 사용하기에 레인보우 테이블을 이용한 사전 공격이 가능하며, 이를 방지하기 위해 패스워드와 솔트를 함께 해쉬함수에 적용하여 사용한다.

안전한 코드의 예 JAVA

```
public String getPasswordHash(String password, byte[] salt) throws Exception {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(password.getBytes());
// 해쉬  사용 시에는 원문을 찾을 수 없도록 솔트를 사용하여야 한다.
    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()
}
```

다음의 예제는 패스워드 저장 시 솔트 없이 패스워드에 대한 해쉬값을 얻는 과정을 보여준다.

안전하지 않은 코드의 예 C#

```
static void HashWithoutSalt()
{
    // 해쉬에 솔트를 적용하지 않아 안전하지 않다.
    var bytes = new byte[100];
    (new Random()).NextBytes(bytes);
    var source = bytes;
    var sha256 = new SHA256CryptoServiceProvider();
    sha256.ComputeHash(source);
}
```

패스워드만을 해쉬함수의 입력으로 사용하기에 레인보우 테이블을 이용한 사전 공격이 가능하며, 이를 방지하기 위해 패스워드와 솔트를 함께 해쉬함수에 적용하여 사용한다.

안전한 코드의 예 C#

```
static void HashWithSalt(int saltLength)
{
    // 해쉬에 솔트를 적용하여 원문을 찾을 수 없게 한다.
    var bytes = new byte[100];
    (new Random()).NextBytes(bytes);
    var source = bytes;
    var sha256 = new SHA256CryptoServiceProvider();
    byte[] saltBytes = GenerateRandomCryptographicBytes(saltLength);
    List<byte> sourceWithSaltBytes = new List<byte>();
    sourceWithSaltBytes.AddRange(source);
    sourceWithSaltBytes.AddRange(sourceWithSaltBytes);
    sha256.ComputeHash(sourceWithSaltBytes.ToArray());
}
```

솔트 값 없이 해쉬를 생성하는 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
void GenerateHash(char* data)
{
  char[512] hashedData = {0 };
//솔트 값 부분이 NULL 로 되어있어 들어가지 않는다.
   MD5HashAlgorithm( data, hashedData, NULL );
...
```

솔트 값을 인자로 넘겨줘야 한다.

안전한 코드의 예 C

```
void GenerateHash(char* data, char* salt)
{
  char hashedData[512] = {0 };
  MD5HashAlgorithm( data, hashedData, salt );
  ...
}
```

라. 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_cryptographically_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 전송 시의 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 전송 시 신뢰할 수 있는 암호 기법을 이용하여 코드를 암호화한다. 또한 다운로드한 코드는 작업 수행을 위해 필요한 최소한의 권한으로 실행하도록 한다.

다. 코드예제

이 예제는 URLClassLoader()으로 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

```
URL[] classURLs = new URL[] { new URL("file:subdir/") };
URLClassLoader loader = new URLClassLoader(classURLs);
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.
String jarFile = "./download/util.jar";
byte[] loadFile = FileManager.getBytes(jarFile);
loadFile = encrypt(loadFile, privateKey);
// jarFileName으로 암호화된 파일을 생성한다.
FileManager.createFile(loadFile, jarFileName);
// 클라이언트에서는 파일을 다운로드 받을 경우 Public Key로 복호화한다.
URL[] classURLs = new URL[] { new URL("http://filesave.com/download/util.jar") };
URLConnection conn = classURLs.openConnection();
InputStream is = conn.getInputStream();
// 입력 스트림을 jarFile명으로 파일을 출력한다.
 FileOutputStream fos = new FileOutputStream(new File(jarFile));
 While (is.read(buf) != -1) {
      ......
}
byte[] loadFile = FileManager.getBytes(jarFile);
loadFile = decrypt(loadFile, publicKey);
// 복호화된 파일을 생성한다.
FileManager.createFile(loadFile, jarFile);
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("MyClass", true, loader);
```

파일 무결성 검사를 하지 않고 파일을 다운로드하는 C#코드 예제이다.

안전하지 않은 코드의 예 C#

```
public override bool DownloadFile()
    {
        var url = "https://www.somewhere.untrusted.com";
        var desDir = "D:/DestinationPath";
        string fileName = Path.GetFileName(url);
        string descFilePath = Path.Combine(desDir, fileName);
        try
        {
            WebRequest myre = WebRequest.Create(url);
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        try
        {
            byte[] fileData;
// 파일 무결성 검사 없이 다운로드
            using (WebClient client = new WebClient())
            {
                fileData = client.DownloadData(url);
            }
            using (FileStream fs = new FileStream(descFilePath, FileMode.OpenOrCreate))
            {
                fs.Write(fileData, 0, fileData.Length);
            }
            return true;
        }
        catch (Exception ex)
        {
 throw new Exception(ex.Message);
        }
    }
```

해쉬값등을 이용하여 파일 무결성 검사 후 다운로드를 해야 한다.

안전한 코드의 예 C#

```
public override bool DownloadFile()
        {
          var url = "https://www.somewhere.untrusted.com";
          var desDir = "D:/DestinationPath";
          string fileName = Path.GetFileName(url);
          string descFilePath = Path.Combine(desDir, fileName);
          try
          {
             WebRequest myre = WebRequest.Create(url);
           }
           catch (Exception ex)
          {
              throw new Exception(ex.Message);
          }
          try
         {
              byte[] fileData;
              using (WebClient client = new WebClient())
              {
                  fileData = client.DownloadData(url);
              }
// 해쉬 값 등을 사용하여 다운로드 받은 파일 무결성 검사
                 CheckIntegrity(fileData);
 using (FileStream fs = new FileStream(descFilePath,
FileMode.OpenOrCreate))
              {
                   fs.Write(fileData, 0, fileData.Length);
              }
              return true;
 }
           catch (Exception ex)
          {
              throw new Exception(ex.Message);
           }
        }
```

리턴 값을 이용하여 무결성 검사를 하지 않은 C코드의 예제이다.

안전하지 않은 코드의 예 C

```
void foo() {
  /* ... */
  hFile = CreateFile((LPCWSTR)data,GENERIC_WRITE, 0, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
  InternetQueryDataAvailable(m_hURL, &dwSize,0,0);
  InternetReadFile(m_hURL, lpBuffer, dwSize, &dwRead);
  WriteFile(hFile, lpBuffer, dwRead, &dwWritten, NULL);
  /* ... */
```

리턴 값을 이용하여 무결성을 확인한 후 사용해야 한다.

안전한 코드의 예 C

```
void foo() {
  /* ... */
  hFile = CreateFile((LPCWSTR)data,GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);
  InternetQueryDataAvailable(m_hURL, &dwSize,0,0);
  bool result = InternetReadFile(m_hURL, lpBuffer, dwSize, &dwRead);
  if, lp( result == true) {
WriteFile(hFileBuffer, dwRead, &dwWritten, NULL);
  }
 /* ... */
}
```

라. 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 성공할법한 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

```
private static final String SERVER_IP = "127.0.0.1";
private static final int SERVER_PORT = 8080;
private static final int FAIL = -1;
public void login() {
  String username = null;
  String password = null;
  Socket socket = null;
  int result = FAIL;
  try {
        socket = new Socket(SERVER_IP, SERVER_PORT);
        //인증 실패에 대해 제한을 두지 않아 안전하지 않다.
        while (result == FAIL) {
        ...
        result = verifyUser(username, password);
}
```

다음 예제는 사용자 인증시도 횟수를 기록하는 MAX_ATTEMPTS 변수를 정의하고, 이를 인증시도 횟수를 제한하는 카운터로 사용함으로써 무차별 공격에 대응하는 코드이다.

안전한 코드의 예 JAVA

```
private static final String SERVER_IP = "127.0.0.1";
private static final int SERVER_PORT = 8080;
private static final int FAIL = -1;
private static final int MAX_ATTEMPTS = 5;
public void login() {
    String username = null;
    String password = null;
    Socket socket = null;
    int result = FAIL;
    int count = 0;
     try {
          socket = new Socket(SERVER_IP, SERVER_PORT);
          // 인증 실패 및 시도 횟수에 제한을 두어 안전하다.
          while (result == FAIL && count < MAX_ATTEMPTS) {
          ...
          result = verifyUser(username, password);
          count++;
}
```

안전하지 않은 코드의 예 C

```
int validateUser(char *host, int port) {
    int socket = openSocketConnection(host, port);
    if (socket < 0) {
         printf("Unable to open socket connection");
         return(FAIL);
    }
    int isValidUser = 0;
    char nm[NAME_SIZE];
    char pw[PSWD_SIZE];
    // 인증시도 횟수를 제한하고 있지 않다.
    while (isValidUser==0) {
```

안전하지 않은 코드의 예 C

```
        if (getNextMsg(socket, nm, NAME_SIZE) > 0) {
            if (getNextMsg(socket, pw, PSWD_SIZE) > 0) {
                isValidUser = AuthenticateUser(nm, pw);
            }
        }
    }
    return(SUCCESS);
}
```

안전한 코드의 예 C

```
#define MAX_ATTEMPTS 5

int validateUser(char *host, int port) {
        .....
        // 연속적인 사용자 인증 시도에 대한 횟수를 제한
        int count = 0;
        while ((isValidUser==0) && (count<MAX_ATTEMPTS)) {
            if (getNextMsg(socket, nm, NAME_SIZE) > 0) {
                if (getNextMsg(socket, pw, PSWD_SIZE) > 0) {
                    isValidUser = AuthenticateUser(nm, pw);
                }
            }
            count++;
        }
        if (isValidUser) {
            return(SUCCESS);
        } else {
            return(FAIL);
        }
}
```

다음 예제는 로그인 정보를 잘못 입력하였을 경우 다시 입력을 시도하는 데 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
override protected void OnLoginError(EventArgs e)
{
  //do nothing
}
```

로그인 시도에 대한 횟수를 제한한다.

안전한 코드의 예 C#

```
override protected void OnLoginError(EventArgs e)
        {
// 연속적인 사용자 인증 시도에 대한 횟수를 제한
            if(ViewState["LoginErrors"] == null)
                ViewState["LoginErrors"] = 0;
            int ErrorCount = (int)ViewState["LoginErrors"] + 1;
            ViewState["LoginErrors"] = ErrorCount;

            if((ErrorCount > 3) && Login1.PasswordRecoveryUrl !=
string.Empty)
                Response.Redirect(Login1.PasswordRecoveryUrl);
        }
```

라. 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

제3절 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: 검사시점과 사용시점(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[7]이 발생할 수 있다.

안전하지 않은 코드의 예 JAVA

```
class FileMgmtThread extends Thread {
    private String manageType = "";
    public FileMgmtThread (String type) {
      manageType = type;
    }
  // 멀티쓰레드 환경에서 공유자원에 여러프로세스가 사용하여 동시에 접근할 가능성이 있어 안전
     하지 않다.
    public void run() {
      try {
        if (manageType.equals("READ")) {
          File f = new File("Test_367.txt");
          if (f.exists()) {
            BufferedReader br
             = new BufferedReader(new FileReader(f));
            br.close();
          }
        } else if (manageType.equals("DELETE")) {
          File f = new File("Test_367.txt");
          if (f.exists()) {
            f.delete();
          } else { … }
        }
      } catch (IOException e) { … }
    }
  }
public class CWE367 {
  public static void main (String[] args) {
    FileMgmtThread fileAccessThread = new FileMgmtThread("READ");
```

7 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

```
        FileMgmtThread fileDeleteThread = new FileMgmtThread("DELETE");
// 파일의 읽기와 삭제가 동시에 수행되어 안전하지 않다.
        fileAccessThread.start();
        fileDeleteThread.start();
    }
}
```

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

```
class FileMgmtThread extends Thread {
    private static final String SYNC = "SYNC";
    private String manageType = "";
    public FileMgmtThread (String type) {
        manageType = type;
    }
    public void run() {
// 멀티쓰레드 환경에서 synchronized를 사용하여 동시에 접근할 수 없도록 사용해야한다.
        synchronized(SYNC) {
            try {
                if (manageType.equals("READ")) {
                    File f = new File("Test_367.txt");
                    if (f.exists()) {
                        BufferedReader br
                            = new BufferedReader(new FileReader(f));
                        br.close();
                    }
                } else if (manageType.equals("DELETE")) {
                    File f = new File("Test_367.txt");
                    if (f.exists()) {
                      f.delete();
                    } else { … }
                }
```

안전한 코드의 예 JAVA

```
            } catch (IOException e) { … }
          }
        }
    }
    public class CWE367 {
      public static void main (String[] args) {
        FileMgmtThread fileAccessThread = new FileMgmtThread("READ");
        FileMgmtThread fileDeleteThread = new FileMgmtThread("DELETE");
        fileAccessThread.start();
        fileDeleteThread.start();
        }
     }
```

다음의 C# 코드도 파일에 동시에 접근하는 레이스 컨디션이 발생할 수 있다.

안전하지 않은 코드의 예 C#

```
// 멀티쓰레드 환경에서 동시에 접근할 수 없도록 사용해야한다.
public void ReadFile(String f)
{
            if(File.Exists(f))
  {
    File.ReadAllLines(f);
    }
}
```

아래와 같은 코드를 추가하여 레이스 컨디션을 방지해야 한다.

안전한 코드의 예 C#

```
// 멀티쓰레드 환경에서 동시에 접근할 수 없도록 사용해야한다.
[MethodImpl(MethodImplOptions.Synchronized)]
public void ReadFile(String f)
{
```

안전한 코드의 예 C#

```
        if(File.Exists(f))
{
    File.ReadAllLines(f);
    }.
}
```

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

```
static volatile double account;
void deposit(int amount) {

// lock 없이 공유 자원에 접근
  account += amount;
}
void withdraw(int amount) {
  account -= amount;
}
```

아래 C 코드는 mutex_lock()으로 공유 자원에 대한 동시 접근을 제한한 것이다.

안전한 코드의 예 C

```
static volatile double account;
static mtx_t account_lock;

void deposit(int amount) {
   // mutex_lock, mutex_unlock을 이용해 공유 자원 접근을 제한한다.
   mutex_lock(&account_lock);
   account += amount;
   mutex_unlock(&account_lock);
}

void withdraw(int amount) {
   mutex_lock(&account_lock);
   account -= amount;
   mutex_unlock(&account_lock);
}
```

라. 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 재귀함수

가. 개요

재귀의 순환횟수를 제어하지 못하여 할당된 메모리나 프로그램 스택 등의 자원을 과다하게 사용하면 위험하다. 대부분의 경우, 귀납 조건(Base Case)이 없는 재귀 함수는 무한 루프에 빠져들게 되고 자원고갈을 유발함으로써 시스템의 정상적인 서비스를 제공할 수 없게 한다.

나. 보안대책

모든 재귀 호출 시, 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

```
#include <stdio.h>
int factorial(int i)
{
// 재귀함수 탈출 조건을 설정하지 않아 무한루프가 된다.
    return i * factorial(i - 1);
}
int main()
{
    int num = 5;
    int result = factorial(num);
    printf("%d! : %d\n", num, result);
    return 0;
}
```

재귀 함수를 구현할 때는 아래와 같이 재귀문을 빠져 나오는 조건인 귀납조건(Base case)을 반드시 구현해야 한다.

안전한 코드의 예 C

```
#include <stdio.h>

int factorial(int I)
{
// 재귀함수 사용 시에는 아래와 같이 탈출 조건을 사용해야 한다.
   if (i <= 1) {
      return 1;
   }
   return i * factorial(i - 1);
}

int main()
{
   int num = 5;
   int result = factorial(num);
   printf("%d! : %d\n", num, result);
   return 0;
}
```

라. 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

제4절 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 데이터 또는 시스템의 내부데이터 등 민감한 정보를 포함하는 오류 메시지를 생성하여 외부에 제공하는 경우, 공격자의 악성 행위를 도울 수 있다. 예외 발생 시 예외 이름이나 스택 트레이스를 출력하는 경우, 프로그램 내부구조를 쉽게 파악할 수 있기 때문이다.

나. 보안대책

오류 메시지는 정해진 사용자에게 유용한 최소한의 정보만 포함하도록 한다. 소스코드에서 예외 상황은 내부적으로 처리하고 사용자에게 시스템 내부 정보 등 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

```
try {
        rd = new BufferedReader(new FileReader(new File(filename)));
} catch(IOException e) {
        // 에러 메시지로 스택 정보가 노출됨
        e.printStackTrace();
}
```

안전하지 않은 코드의 예 JAVA

```
 catch(IOException e) {
        // 오류발생시 화면에 출력된 시스템 정보로 다른 공격의 빌미를 제공한다.
        System.err.print(e.getMessage());
}
```

아래 코드와 같이 예외 이름이나 오류추적 정보를 출력하지 않도록 한다.

안전한 코드의 예 JAVA

```
try {
        rd = new BufferedReader(new FileReader(new File(filename)));
} catch(IOException e) {
        // 에러 코드와 정보를 별도로 정의하고 최소 정보만 로깅
        logger.error("ERROR-01: 파일 열기 에러");
}
```

다음 예제는 오류메시지에 예외 이름이나 오류추적 정보를 출력하여 프로그램 내부 정보가 유출되는 C#코드이다.

안전하지 않은 코드의 예 C#

```
try
{
   //do something
}
catch (CustomException e)
{
   Console.WriteLine(e);
}
```

예외 관련한 최소한의 정보만 출력하도록 한다.

안전한 코드의 예 C#

```
try
{
   //do something
}
catch (CustomException e)
{
   _log.Debug("ERROR-01 : error information");
}
```

라. 참고자료

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

② CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere, MITRE, http://cwe.mitre.org/data/definitions/497.html

③ 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

④ Error Handling, OWASP, https://www.owasp.org/index.php/Error_Handling

2. 오류 상황 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

```
protected Element createContent(WebSession s) {
            ......
    try {
        username = s.getParser().getRawParameter(USERNAME);
        password = s.getParser().getRawParameter(PASSWORD);
        if (!"webgoat".equals(username) || !password.equals("webgoat")) {
            s.setMessage("Invalid username and password entered.");
            return (makeLogin(s));
        }
    } catch (NullPointerException e) {
    // 요청 파라미터에 PASSWORD가 존재하지 않을 경우 Null Pointer Exception이 발생하고
        해당 오류에 대한 대응이 존재하지 않아 인증이 된 것으로 처리
    }
```

예외를 포착(catch)한 후, 각각의 예외 사항(Exception)에 대하여 적절하게 처리해야 한다.

안전한 코드의 예 JAVA

```
protected Element createContent(WebSession s) {
            ......
    try {
        username = s.getParser().getRawParameter(USERNAME);
        password = s.getParser().getRawParameter(PASSWORD);
        if (!"webgoat".equals(username) || !password.equals("webgoat")) {
            s.setMessage("Invalid username and password entered.");
            return (makeLogin(s));
        }
    } catch (NullPointerException e) {
    // 예외 사항에 대해 적절한 조치를 수행하여야 한다.
        s.setMessage(e.getMessage());
        return (makeLogin(s));
    }
```

다음의 C# 코드도 예외상황에 대한 조치가 없다.

안전하지 않은 코드의 예 C#

```
try {
     InvokeMtd();
   } catch (CustomException e) {
     // 예외 상황에 대한 대응 부재
}
```

각각의 예외 상황에 대해 적절한 조치를 수행해야 한다.

안전한 코드의 예 C#

```
try {
        InvokeMtd();
    } catch (CustomException e) {
// 예외 상황에 대해 적절한 조치를 수행하여야 한다.
        logger.Debug("log message");
}
```

라. 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 예외 상황에 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

```
try {
        ...
    reader = new BufferedReader(new InputStreamReader(url.openStream()));
    String line = reader.readLine();
    SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
    Date date = format.parse(line);
// 예외처리를 세분화 할 수 있음에도 광범위하게 사용하여 예기치 않은 문제가 발생 할 수 있다.
} catch (Exception e) {
    System.err.println("Exception : " + e.getMessage());
}
```

발생 가능한 예외를 세분화하고 발생 가능한 순서에 따라 예외를 처리하고 있다.

안전한 코드의 예 JAVA

```
try {
        ...
        reader = new BufferedReader(new InputStreamReader(url.openStream()));
        String line = reader.readLine();
      SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
      Date date = format.parse(line);
      // 발생할 수 있는 오류의 종류와 순서에 맞춰서 예외처리 한다.
} catch (MalformedURLException e) {
        System.err.println("MalformedURLException : " + e.getMessage());
} catch (IOException e) {
        System.err.println("IOException : " + e.getMessage());
} catch (ParseException e) {
        System.err.println("ParseException : " + e.getMessage());
}
```

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#

```
try {
      InvokeMtd();
// 예외처리를 세분화할 수 있음에도 광범위하게 사용하여 예기치 않은 문제가 발생할 수 있다.
      } catch (Exception e) {

}
```

발생 가능한 예외를 세분화하고 발생 가능한 순서에 따라 예외를 처리하고 있다.

안전한 코드의 예 C#

```
ry {
        InvokeMtd();
// 발생할 수 있는 오류의 종류와 순서에 맞춰서 예외처리 한다.
    } catch (IOException e) {
    logger.Debug("IOException log here");
} catch (SQLException e){
    logger.Debug("SQLException log here");
}
```

라. 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

제5절 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

```
public static int cardinality (Object obj, final Collection col) {
    int count = 0;
    if (col == null) {
        return count;
    }
    Iterator it = col.iterator();
    while (it.hasNext()) {
```

안전하지 않은 코드의 예 JAVA

```
        Object elt = it.next();
// obj가 null이고 elt가 null이 아닐 경우, Null.equals 가 되어 널(Null) 포인터 역참조가 발생한다.
        if ((null == obj && null == elt) || obj.equals(elt)) {
            count++;
        }
    }
    return count;
}
```

obj가 null인지 검사 후 참조해야 한다.

안전한 코드의 예 JAVA

```
public static int cardinality (Object obj, final Collection col) {
    int count = 0;
    if (col == null) {
        return count;
    }
    Iterator it = col.iterator();
    while (it.hasNext()) {
        Object elt = it.next();
// obj를 참조하는 equals가 null이
        if ((null == obj && null == elt) || (null != obj && obj.equals(elt))) {
            count++;
        }
    }
    return count;
}
```

다음 예제의 경우 request.getParameter에 의해 null이 들어오게 되면 널(Null) 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 JAVA

```
String url = reuqest.getParamter("url");
// url 에 null이 들어오면 널(Null) 포인터 역참조가 발생한다.
if ( url.equals("") )
```

null을 가질 수 있는 참조 변수를 사용해 객체의 속성이나 메소드를 사용하는 경우 null 검사를 수행하고 사용한다.

안전한코드의 예 JAVA

```
String url = reuqest.getParamter("url");
// null값을 가지는 참조 변수를 사용할 경우, null 검사를 수행하고 사용한다.
if ( url != null || url.equals("") )
```

다음 예제의 경우 Request 객체에서 QeuryString을 사용하여 url의 파라미터 중 name 에 해당하는 값을 가져오는 코드이다. url의 파라미터에 name이 없으면 QueryString[“name”]은 null을 리턴 하게 되고, username은 null 값을 가지게 되어 널(Null) 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 C#

```
protected void Page_Load(object sender, EventArgs e) {
  // url 파라미터에 name 이 없으면 username은 null 값을 가지게 된다.
  string username = Request.QueryString[“name”];
  // null 값을 가지는 username을 참조하여 널(Null) 포인터 역참조가 발생한다.
  if (username.Length 〉 20) {
    // length error
  }
}
```

null을 가질 수 있는 참조 변수를 사용해 객체의 속성이나 메소드를 사용하는 경우 null 검사를 수행하고 사용한다.

안전한 코드의 예 C#

```
protected void Page_Load(object sender, EventArgs e) {
   // url 파라미터에 name 이 없으면 username은 null 값을 가지게 된다.
    string username = Request.QueryString["name"];
   // null 값을 가지는 username을 참조하기 전에 null 검사를 수행하므로 안전하다.
     if ( username != null && username 〉 20) {
       // length error
     }
}
```

아래 C 코드는 null 값을 반환할 수 있는 함수 IntegerAddressReturn()을 호출한다. P가 null 인 상태에서 p 값을 참조하면 널 포인터 역참조가 발생한다.

안전하지 않은 코드의 예 C

```
void NullPointerDereference(int count) {
  // IntegerAddressReturn()이 0을 return 하면 p는 null 값을 가지게 된다.
  int *p = IntegerAddressReturn();
  // null 값을 가지는 p 값을 참조하여 널(Null) 포인터 역참조가 발생한다.
 *p = count;
}
```

아래 C 코드는 null 값을 가질 수 있는 p를 참조하기 전에 null 검사를 진행하므로 안전하다.

안전한 코드의 예 C

```
void NullPointerDereference(int count) {
// IntegerAddressReturn()이 0을 return 하면 p는 null 값을 가지게 된다.
int *p = IntegerAddressReturn();
// 참조하기전에 null 검사를 수행하므로 안전하다.
If(p != 0) *p = count;
```

라. 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

```
InputStream in = null;
OutputStream out = null;
    try {
     in = new FileInputStream(inputFile);
     out = new FileOutputStream(outputFile);
     ...
     FileCopyUtils.copy(fis, os);
     // 자원반환 실행 전에 오류가 발생할 경우 자원이 반환되지 않으며, 할당된 모든 자원을 반환해야
        한다.
     in.close();
     out.close();
} catch (IOException e) {
      logger.error(e);
}
```

예외상황이 발생하여 함수가 종료될 때, 예외의 발생 여부와 상관없이 항상 실행되는 finally 블록에서 할당받은 모든 자원을 반드시 반환하도록 한다.

안전한 코드의 예 JAVA

```
InputStream in = null;
OutputStream out = null;
   try {
     in = new FileInputStream(inputFile);
     out = new FileOutputStream(outputFile);
     ...
     FileCopyUtils.copy(fis, os);
} catch (IOException e) {
       logger.error(e);
// 항상 수행되는 finally 블록에서 할당받은 모든 자원에 대해 각각 null검사를 수행 후 예외처리를
   하여 자원을 해제하여야 한다.
   } finally {
      if (in != null) {
         try {
               in.close();
         } catch (IOException e) {
             logger.error(e);
         }
      }
      if (out != null) {
          try {
               out.close();
          } catch (IOException e) {
             logger.error(e);
          }
      }
}
```

파일스트림이 해제되지 않는 C# 코드 예제이다.

안전하지 않은 코드의 예 C#

```
public void FileStreamTest()
{
// fsSource에 자원이 할당되었으나 해제되지 않는다.
   FileStream fsSource = new FileStream(pathSource, FileMode.Open, FileAccess.Read);
   byte[] bytes = new byte[fsSource.Length];
   int numBytesToRead = (int)fsSource.Length;
   int numBytesRead = 0;
   while(numBytesToRead > 0)
   {
     int n = fsSource.Read(bytes, numBytesRead, numBytesToRead);
     if(n==0)
      break;
      numBytesToRead += n;
      numBytesToRead -= n;
   }
   using(FileStream fsNew = new FileStream(pathNew, FileMode.Create, FileAccess.Write))
{ /* OK */
       fsNew.Write(bytes, 0, numBytesToRead);
   }
}
```

using 구문을 이용하여 쉽게 자원을 해제할 수 있다.

안전한 코드의 예 C#

```
public void FileStreamTest()
{
   // using 구문으로 자원을 할당하면 구문이 끝나는 지점에서 자동으로 자원이 해제된다.
    using(FileStream fsSource = new FileStream(pathSource, FileMode.Open,
FileAccess.Read)){
       byte[] bytes = new byte[fsSource.Length];
       int numBytesToRead = (int)fsSource.Length;
       int numBytesRead = 0;
```

안전한 코드의 예 C#

```
        while(numBytesToRead > 0)
        {
          int n = fsSource.Read(bytes, numBytesRead, numBytesToRead);
          if(n==0)
            break;
            numBytesToRead += n;
            numBytesToRead -= n;
        }
      }
      using(FileStream fsNew = new FileStream(pathNew, FileMode.Create,
FileAccess.Write)) { /* OK */
        fsNew.Write(bytes, 0, numBytesToRead);
      }
}
```

아래 C 코드는 파일을 연 상태에서 오류가 발생했을 때 자원 누수가 발생한다.

안전하지 않은 코드의 예 C

```
void ImproperResourceRelease(char* filename) {
        char buf[BUF_SIZE];
        FILE *f = fopen(filename, "r");
        if(!checkSomething()) {
printf("Something is wrong");
        return;
}
// checkSomething에서 false를 반환하는 경우, 파일 핸들러를 종료할 수 없다.
fclose(f);
}
```

오류가 발생한 상황에서도 정상적으로 파일 핸들러를 종료하도록 수정한다.

안전한 코드의 예 C#

```
void ImproperResourceRelease(char* filename) {
      char buf[BUF_SIZE];
      FILE *f = fopen(filename, “r”);
      if(!checkSomething()) {
        printf(“Something is wrong”);
      // checkSomthing에서 false를 반환해도 파일 핸들러를 종료하도록 수정
        fclose(f);
      return;
       }
fclose(f);
}
```

라. 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

```
int main(int argc, const char *argv[]) {
        char *temp;
        temp = (char *)malloc(BUFFER_SIZE);
        ......
        free(temp);
        // 해제한 자원을 사용하고 있어 의도하지 않은 결과가 발생하게 된다.
        stmcpy(temp, argv[1], BUFFER_SIZE-1);
}
```

다음 예제와 같이 메모리를 해제하기 전에 할당한 메모리를 사용하는 작업을 수행하고 최종적으로 메모리를 해제한다.

안전한 코드의 예 C

```
int main(int argc,const char *argv[]) {
    char *temp;
    temp = (char *)malloc(BUFFER_SIZE);
    ......
    // 할당된 자원을 최종적으로 사용하고 해제하여야 한다.
    stmcpy(temp,argv[1], BUFFER_SIZE-1);
    free(temp);
}
```

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

```
char *data;
int data_type
if (data_type==val_1) { free(data); }
        ......
// 이미 해제된 자원을 이중 해제하여 문제가 발생한다.
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

```
char *data;
int data_type
if (data_type==val_1) {
      free(data);
   // 메모리를 해제한 후 항상 포인터에 NULL을 할당하여 이중 해제하더라도 무시되게 한다.
      data = NULL;
}
          ……
if (data_type==val_2) {
      free(data);
   // 메모리를 해제한 후 항상 포인터에 NULL을 할당하여 이중 해제하더라도 무시되게 한다.
      data = NULL;
}
```

라. 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 임의값을 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 메모리에 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

```
// 변수의 초기값을 지정하지 않을 경우 공격에 사용 될 수 있어 안전하지 않다.
int x, y;
switch(position) {
      case 0: x = base_position; y = base_position beak;
      case 1: x = base_position + i; y = base_position – i break;
      default: x=1; break;
}
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

```
// 변수의 초기값은 항상 지정하여야 한다.
int x=1, y=1;
switch(position) {
      case 0: x = base_position; y = base_position beak;
      case 1: x = base_position + i; y = base_position – i break;
      default: x=1; break;
}
setCursorPosition(x,y);
```

라. 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 수신 측에 전달하게 된다.

역직렬화(Deserialization)는 반대 연산으로 바이너리 파일이나 바이트 스트림으로부터 객체 구조로복원하게 된다.

이 때, 송신자가 네트워크를 이용하여 직렬화된 정보를 수신자에게 전달하는 과정에서 공격자가 전송또는 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 있으나사실은 이전 함수에서 사용되었던 내용을 포함하고 있다. 공격자는 이러한 약점을 사용하여 메모리에저장되어 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 바이트스트림을 조작하여 역직렬화 공격이 가능한 객체를 생성할 수 있는 예제이다.

안전하지 않은 코드의 예 JAVA

```
public static void main(String[] args) throws
IOException, GeneralSecurityException, ClassNotFoundException {
        ....
        // map을 역직렬화 한다.
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
        sealedMap = (SealedObject) in.readObject();
        in.close();

        // 객체를 추출한다.
        cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, key);
        signedMap = (SignedObject) sealedMap.getObject(cipher);

        // 서명값 검증 과정에서 불일치 시 예외를 리턴하고, 일치 시 map 값을 읽는다.
        if (!signedMap.verify(kp.getPublic(), sig)) {
        throw new GeneralSecurityException("Map failed verification");
        }
        map = (SerializableMap<String, Integer>) signedMap.getObject();
}
```

다음 예제는 서명 값을 검증하여 메시지 위변조를 방지할 수 있는 코드이다.

안전한 코드의 예 JAVA

```
public static void main(String[] args) throws
IOException, GeneralSecurityException, ClassNotFoundException {
        ....
        // map을 역직렬화 한다.
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
        sealedMap = (SealedObject) in.readObject();
        in.close();

        // 객체를 추출한다.
        cipher = Cipher.getInstance("AES");
```

안전한 코드의 예 JAVA

```
        cipher.init(Cipher.DECRYPT_MODE, key);
        signedMap = (SignedObject) sealedMap.getObject(cipher);
        // 서명값 검증 과정에서 불일치 시 예외를 리턴하고, 일치 시 map 값을 읽는다.
            if (!signedMap.verify(kp.getPublic(), sig)) {
                throw new GeneralSecurityException("Map failed verification");
            }
            map = (SerializableMap<String, Integer>) signedMap.getObject();
}
```

다음 예제는 바이트 배열의 입력 값을 검증 없이 readObject()로 역직렬화하여 악의적인 코드가 실행될 수 있다.

안전하지 않은 코드의 예 JAVA

```
class DeserializeExample {
        public static Object deserialize(byte[] buffer)
        throws IOException, ClassNotFoundException {
            Object ret = null;
            try (ByteArrayInputStream bais = new ByteArrayInputStream(buffer)) {
                try (ObjectInputStream ois = new ObjectInputStream(bais)) {
                    ret = ois.readObject();
                }
            }
        return ret;
        }
}
```

이를 안전한 코드로 변환하기 위해서는 ObjectInputStream을 상속 받아Whitelisted Object InputStream 객체를 구현하여 사용한다. WhitelistedObjectInputStream에서는 readObject()를 실행 시, resolveClass 함수를 호출하여 설정한 화이트리스트와 비교하여 리스트에 없는 데이터일 경우 예외를 발생시킨다.

안전한 코드의 예 JAVA

```
public class WhitelistedObjectInputStream extends ObjectInputStream {
        public Set<String> whitelist;
        // WhilelistedObjectInputStream을 생성할 때 화이트리스트를 입력받는다.
        public WhitelistedObjectInputStream(InputStream inputStream, Set<String> wl)
        throws IOException {
                super(inputStream);
                whitelist = wl;
        }

@Override
protected Class<?> resolveClass(ObjectStreamClass cls) throws IOException,
ClassNotFoundException {
        // ObjectStreamClass의 클래스명이 화이트리스트에 있는지 확인한다.
        if (!whitelist.contains(cls.getName())) {
        throw new InvalidClassException("Unexpected serialized class", cls.getName());
                }
                return super.resolveClass(cls);
        }
}

@RequestMapping(value = "/upload", method = RequestMethod.POST)
public Student upload(@RequestParam("file") MultipartFile multipartFile)
throws ClassNotFoundException, IOException {
Student student = null;
        File targetFile = new File("/temp/" + multipartFile.getOriginalFilename());
           // 역직렬화 대상 클래스 이름의 화이트리스트 생성한다.
           Set<String> whitelist = new HashSet<String>(Arrays.asList(
           new String[] {
                   "Student"
           }));
           try (InputStream fileStream = multipartFile.getInputStream()) {
                   try (WhitelistedObjectInputStream ois =
                   new WhitelistedObjectInputStream(fileStream, whitelist)) {
                     // 화이트리스트에 없는 역직렬화 데이터의 경우 예외 발생시킨다.
                           student = (Student) ois.readObject();
                   }
           }
           return student;
}
```

라. 참고자료

① CWE-502 Deserialization of Untrusted Data, MITRE, https://cwe.mitre.org/data/definitions/502.html

제6절 캡슐화

중요한 데이터 또는 기능성을 불충분하게 캡슐화하거나 잘못 사용함으로써 발생하는 보안약점으로 정보노출, 권한문제 등이 발생할 수 있다.

1. 잘못된 세션에 의한 데이터 정보노출

가. 개요

다중 스레드 환경에서는 싱글톤(Singleton)[8] 객체 필드에 경쟁조건(Race Condition)이 발생할 수 있다. 따라서, 다중 스레드 환경인 Java의 서블릿(Servlet) 등에서는 정보를 저장하는 멤버 변수가 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 HttpServlet 클래스의 하위클래스에서 멤버 필드를 선언하지 않도록 하고, 필요한 경우 지역 변수를 선언하여 사용한다.

다. 코드예제

JSP 선언부(〈%! 소스코드 %〉)에 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.

8 싱글톤 패턴 : GOF 32가지 패턴 중 하나. 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

```
<%@page import="javax.xml.namespace.*"%>
<%@page import="gov.mogaha.ntis.web.frs.gis.cmm.util.*" %>
<%!
    // JSP에서 String 필드들이 멤버 변수로 선언됨
    String username = "/";
    String imagePath = commonPath + "img/";
    String imagePath_gis = imagePath + "gis/cmm/btn/";
.....
%>
```

JSP의 서블릿(<% 소스코드 %>)에 정의한 변수는 _jspService 메소드의 지역변수로 선언되므로 공유가 발생하지 않아 안전하다.

안전한 코드의 예 JAVA

```
<%@page import="javax.xml.namespace.*"%>
<%@page import="gov.mogaha.ntis.web.frs.gis.cmm.util.*" %>
<%
    // JSP에서 String 필드들이 로컬 변수로 선언됨
    String commonPath = "/";
    String imagePath = commonPath + "img/";
    String imagePath_gis = imagePath + "gis/cmm/btn/";
    .....
%>
```

Controller에 멤버 변수를 사용하면 공유가 발생하여 동기화 오류가 발생할 수 있다.

안전하지 않은 코드의 예 JAVA

```
@Controller
public class TrendForecastController {
    // Controller에서 int 필드가 멤버 변수로 선언되어 스레드간에 공유됨
```

안전하지 않은 코드의 예 JAVA

```
    private int currentPage = 1;
    public void doSomething(HttpServletRequest request) {
        currentPage = Integer.parseInt(request.getParameter("page"));
    }
.....
```

Controller에 멤버 변수를 사용하지 않고 지역변수로 사용한다.

안전한 코드의 예 JAVA

```
@Controller
public class TrendForecastController {
    public void doSomething(HttpServletRequest request) {
    // 지역변수로 사용하여 스레드간 공유되지 못하도록 한다.
    int currentPage = Integer.parseInt(request.getParameter("page"));
    }
.....
```

다중 스레드 환경에서 IHttpHandler 클래스에 정보를 저장하는 필드가 포함되서는 안된다.

안전하지 않은 코드의 예 C#

```
class DataLeakBetweenSessions : IHttpHandler
  {
      // 다중 스레드 환경에서 IHttpHandler 를 구현하는 클래스에 정보를 저장하는 필드가 포함되면
         안된다.
      private String id;
      public void ProcessRequest(HttpContext ctx)
      {
var json = new JSONResonse()
        {
            Success = ctx.Request.QueryString["name"] != null,
            Name = ctx.Request.QueryString["name"]
```

안전하지 않은 코드의 예 C#

```
        };
          ctx.Response.ContentType = "application/json";
          ctx.Response.Write(JsonConvert.SerializeObject(json));
      }
       public bool IsReusable
      {
          get { return false; }
      }
}
```

해당 내용을 지역변수나 세션변수를 이용하여 처리하여야 한다.

안전한 코드의 예 C#

```
class DataLeakBetweenSessions : IHttpHandler
    {
        public void ProcessRequest(HttpContext ctx)
        {
// 지역변수나 세션변수를 선언해서 사용해야한다.
         ctx.Session["id"] = ctx.Request.QueryString["id"];
                ctx.Response.ContentType = "application/json";
                ctx.Response.Write(JsonConvert.SerializeObject(json));
            }
            public bool IsReusable
            {
                get { return false; }
            }
        }

            var json = new JSONResonse()
            {
                Success = ctx.Request.QueryString["name"] != null,
                Name = ctx.Request.QueryString["name"]
            };
```

라. 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 웹응용프로그램을 제작할 때 디버그용도의 코드를 main()에 개발한 후 이를 삭제하지 않는 경우가 많다.디버깅이 끝나면 main() 메소드를 삭제해야 한다.

다. 코드예제

다음의 예제는 main() 메소드 내에 화면에 출력하는 디버깅 코드를 포함하고 있다. J2EE의 경우 main() 메소드 사용이 필요 없으며, 개발자들이 콘솔 응용프로그램으로 화면에 디버깅코드를 사용하는 경우가 일반적이다.

안전하지 않은 코드의 예 JAVA

```
class Base64 {
    public static void main(String[] args) {
      if (debug) {
         byte[] a = { (byte) 0xfc, (byte) 0x0f, (byte) 0xc0 };
         byte[] b = { (byte) 0x03, (byte) 0xf0, (byte) 0x3f };
         ......
       }
    }
      public void otherMethod() { … }
}
```

이에 따라 J2EE와 같은 응용프로그램에서 main() 메소드는 삭제한다. J2EE의 main() 메소드의 경우 디버깅 코드인 경우가 일반적이다.

안전한 코드의 예 JAVA

```
class Base64 {
    public void otherMethod() { … }
}
```

디버그용 코드가 남아있는 C# 코드의 예제이다.

안전하지 않은 코드의 예 C#

```
class Example {
    public void Log() {
    // Console.WriteLine 등의 메소드를 사용한 디버그용 코드가 남아있다.
   Console.WriteLine("sensitive info");
  }
}
```

디버그용 코드를 삭제 하여야 한다.

안전한 코드의 예 C#

```
class Example {
    public void Log() {
    // 디버그용 코드를 삭제해야한다.
    //Console.WriteLine("sensitive info");
  }
}
```

아래는 디버그용 코드가 남아있는 C 코드의 예제이다. 공격자가 출력되는 콜 스택으로 프로그램 구조를 유추할 수 있다.

안전하지 않은 코드의 예 C

```
void LeftoverDebugCode() {
        int i, ntprs;
        char **strings;
        nptrs = backtrace(buffer, 100);
        strings = backtrace_symbols(buffer, nptrs);
        ...
        // 디버그 모드일 시 콜스택을 출력한다.
        if(debug) {
        for(i=0; i < nptr; i++) printf("%s\n", strings[j]);
        }
}
```

릴리즈 시에는 디버그용 코드를 삭제 하여야 한다.

안전한 코드의 예 C

```
void LeftoverDebugCode() {
        ··· // 디버그 코드를 삭제하고 동작 코드만 남긴다.
}
```

라. 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

```
// private 인 배열을 public인 메소드가 return한다.
private Color[] colors;
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)

```
private Color[] colors;
// 메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
Color[] newColors = getUserColors();
    ......
}
public Color[] getUserColors(Color[] userColors) {
 // 배열을 복사한다.
Color[] colors = new Color [userColors.length];
for (int i = 0; i < colors.length; i++)
// clone()메소드를 이용하여 배열의 원소도 복사한다.
colors[i] = this.colors[i].clone();
return colors;
}
```

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)

```
// private 인 배열을 public인 메소드가 return한다.
private String[] colors;
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)

```
private String[] colors;
// 메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
String[] newColors = getColors();
.....
}
public String[] getColors() {
String[] ret = null;
  if ( this.colors != null ) {
        ret = new String[colors.length];
        for (int i = 0; i < colors.length; i++) { ret[i] = this.colors[i]; }
}
return ret;
}
```

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#

```
// private 인 collection을 public인 메소드가 return한다.
private List<Color> colors;
public List<Color> getUserColors() { return colors; }
```

메소드를 private 으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만들어야 한다.

안전한 코드의 예 C#

```
private List<Color> colors;
// 메소드를 private으로 하거나, 복제본 반환, 수정하는 public 메소드를 별도로 만든다.

public List<Color> getUserColors() {
 // 배열을 복사한다.
List< ICloneable> newList = new List< ICloneable>(colors.Count);
//Clone()메소드를 이용하여 collection의 원소도 복사한다.
colors.ForEach((item) =>
{
  newList.Add((ICloneable)item.Clone());
});
return newList;
}
```

라. 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)

```
// userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면,
   사실상 public 필드가 된다.
private UserRole[] userRoles;
public void setUserRoles(UserRole[] userRoles) {
this.userRoles = userRoles;
}
```

인자로 들어온 배열의 복사본을 생성하고 clone() 메소드로 복사된 원소를 저장하도록 하여 private변수에 할당하면 private으로 할당된 배열과 원소에 대한 의도하지 않은 수정을 방지할 수 있다.

안전한 코드의 예 JAVA (배열의 원소가 일반객체일 경우)

```
// 객체가 클래스의 private member를 수정하지 않도록 한다.
private UserRole[] userRoles;
public void setUserRoles(UserRole[] userRoles) {
this.userRoles = new UserRole[userRoles.length];
for (int i = 0; i < userRoles.length; ++i)
this.userRoles[i] = userRoles[i].clone();
}
```

아래의 코드는 멤버 변수 userRoles는 private로 선언되었지만 public으로 선언된 setUserRoles 메소드로 인자가 할당되어 배열의 원소를 외부에서 변경할 수 있다. 이 경우 의도하지 않은 배열에 대한수정이 발생할 수 있다.

안전하지 않은 코드의 예 JAVA (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
// userRoles 필드는 private이지만, public인 setUserRoles()로 외부의 배열이 할당되면,
   사실상 public 필드가 된다.
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
this.userRoles = userRoles;
}
```

인자로 들어온 배열의 복사본을 생성하여 private변수에 할당하면 private으로 할당된 배열에 대한 의도하지 않은 수정을 방지할 수 있다.

안전한 코드의 예 (배열의 원소가 String 타입 등과 같이 수정이 되지 않는 경우)

```
// 객체가 클래스의 private member를 수정하지 않도록 한다.
private String[] userRoles;

public void setUserRoles(String[] userRoles) {
  this.userRoles = new String[userRoles.length];
  for (int i = 0; i < userRoles.length; ++i)
  this.userRoles[i] = userRoles[i];
}
```

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#

```
class Program
    {
// userRoles 필드는 private이지만, public인 setUserRoles()으로 외부의 배열이 할당되면,
   사실상 public 필드가 된다.
        private String[] userRoles;
        public void SetUserRoles(String[] userRoles)
        {
          this.userRoles = userRoles;
         }

    }
```

객체가 클래스의 private member를 수정하지 않도록 하여야 한다.

안전한 코드의 예 C#

```
class Program
    {
// 객체가 클래스의 private member를 수정하지 않도록 한다.
        private String[] userRoles;
        public void SetUserRoles(String[] userRoles)
        {
            int length = userRoles.Length;
            this.userRoles = new String[length];
            for(int i = 0; i < length; i++) { \
              this.userRoles[i] = userRoles[i];
            }
        }
    }
```

라. 참고자료

① CWE-496 Public Data Assigned to Private Array-Typed Field, MITRE, http://cwe.mitre.org/data/definitions/496.html

제7절 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 접근 통제 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 캐쉬 등을 조작해서 쉽게 이러한 보안 설정을 우회할 수 있다.

안전하지 않은 코드의 예 JAVA

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    boolean trusted = false;
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
// 도메인은 공격자에 의해 실행되는 서버의 DNS가 변경될 수 있으므로 안전하지 않다.
```

안전하지 않은 코드의 예 JAVA

```
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    do_something_for_Trust_System();
}
```

그러므로, 다음의 예제와 같이 DNS lookup에 의한 호스트 이름 비교를 하지 않고, IP 주소를 직접 비교하도록 수정한다.

안전한 코드의 예 JAVA

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    String ip = req.getRemoteAddr();
    if (ip == null || "".equals(ip)) return ;
// 이용하려는 실제 서버의 IP 주소를 사용하여 DNS변조에 방어한다.
    String trustedAddr = "127.0.0.1";
    if (ip.equals(trustedAddr)) {
        do_something_for_Trust_System();
    }
```

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#

```
bool trusted;
string remoteIpAddress = Request.ServerVariables["REMOTE_HOST"];
IPAddress hostIPAddress = IPAddress.Parse(remoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
string hostName = hostInfo.HostName;
if (hostName.EndsWith("trust.com"))
{
trusted = true;
}
```

IP주소를 직접 비교하도록 한다.

안전한 코드의 예 C#

```
bool trusted;
string remoteIpAddress = Request.ServerVariables["REMOTE_HOST"];

if ( remoteIpAddress.Equals(trustedAddr))
{
   trusted = true;
   Do_something_for_Trust_System();
}
```

아래 C 코드는 요청의 신뢰성을 호스트의 이름으로 판별하고 있다. 공격자는 DNS 캐시를 조작하여 보안 정책을 우회할 수 있다.

안전하지 않은 코드의 예 C

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);

hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
// 요청의 신뢰성을 호스트의 이름으로 판별하고 있다.
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {

    trusted = true;
} else {
  trusted = false;
}
```

다음의 C 코드는 IP 주소를 직접 비교하여 DNS 캐시 조작에 대해 안전하다.

안전한 코드의 예 C

```
 struct hostent *hp;struct in_addr myaddr;
  char* tHost = "127.0.0.1";
       myaddr.s_addr=inet_addr(ip_addr_string);

  hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
// 호스트의 이름이 아니라 IP로 직접 비교한다.
  if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
      trusted = true;
      } else {
      trusted = false;
}
```

라. 참고자료

① CWE-350 Reliance on Reverse DNS Resolution for a Security-Critical Action, MITRE, http://cwe.mitre.org/data/definitions/350.html

② 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 gets(), strcat(), strcpy(), strncat(), strncpy(), sprintf() 등이 있다. 또한 보안상 문제가 없다 하더라도 잘못된 방식으로 함수를 사용할 때도 역시 보안 문제를 발생시킬 수 있다.

나. 보안대책

보안 문제로 인해 금지된 함수는 이를 대체할 수 있는 안전한 함수를 사용한다. 그 예로, 위에 언급된 API 대신에 gets_s()/fgets(), 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 있다면7) 반드시 준수한다.

다. 코드예제

아래 예제에서 gets() 함수는 크기와 상관없이 입력값을 버퍼에 저장하기 때문에 버퍼 오버플로우를 유발할 수 있다.

안전하지 않은 코드의 예 C

```
#include <stdio.h>

void requestString()
{
char str[100];
  // gets() 함수는 문자열 길이를 제한 할 수 없어 안전하지 않다.
  gets(str);
}
```

아래 예제와 같이 gets_s() 또는 fgets()함수를 사용하여 입력값의 크기를 제한하여 사용해야 한다.

안전한 코드의 예 C

```
#include <stdio.h>

void requestString()
{
  char str[100];
  // gets_s() 함수는 문자열 길이 제한이 가능하다.
  gets_s(str, sizeof(str));
}
```

다음 예제는 J2EE 응용 프로그램에서 프레임워크 메소드 호출 대신에 소켓을 직접 사용하고 있어, 프레임워크에서 제공하는 보안기능을 제공받지 못한다.

안전하지 않은 코드의 예 JAVA

```
public class S246 extends javax.servlet.http.HttpServlet {
    private Socket socket
    protected void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException {
        try {
// 프레임워크의 메소드 호출 대신 소켓을 직접 사용하고 있어 프레임워크에서 제공하는 보안기능을
   제공 받지 못해 안전하지 않다.
socket = new Socket("kisa.or.kr", 8080);
        } catch (UnknownHostException e) {
                              ……
```

타겟이 WAS로 작성될 경우 아래의 코드처럼 보안기능을 제공하는 프레임워크 메소드인 URL Connection 을 이용하여야한다.

안전한 코드의 예 JAVA

```
public class S246 extends javax.servlet.http.HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException {
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;
        try {
            URL url = new URL("http://127.0.0.1:8080/DataServlet");
        // 보안기능을 제공하는 프레임워크의 메소드를 사용하여야한다.
            URLConnection urlConn = url.openConnection();
            urlConn.setDoOutput(true);
                .....
```

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

```
public class U382 extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            do_something(logger);
} catch (IOException ase) {
            logger.info("ERROR");
    // J2EE 프로그램에서 System.exit()을 사용하여 서비스가 종료 될 수 있다.
            System.exit(1);
        }
```

서비스 종료를 막기 위해, J2EE 프로그램에서는 System.exit() 메소드를 사용하지 않는다.

안전한 코드의 예 JAVA

```
public class U382 extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            do_something(logger);
        } catch (IOException ase) {
            logger.info("ERROR");
        // 서비스 종료를 막기 위해 J2EE에서는 System.exit()를 사용하지 않는다.
        }
```

C# 코드에서 Application.Exit() 을 사용하면 일부 이벤트가 처리되지 않습니다.

안전하지 않은 코드의 예 C#

```
try {
    ...
} catch (Exception e) {
  ...
// Application.Exit() 은 즉시 프로그램을 종료하기 때문에, Form.Closed 혹은 Form.Closing
    이벤트가 처리되지 않는다.
Application.Exit();
}
```

이벤트 처리를 위해 Application.Exit()을 사용하지 않습니다.

안전한 코드의 예 C#

```
try {
   ...
} catch (Exception e) {
   ...
// Application.Exit() 을 사용하지 않으면, 이벤트를 처리하지 못하고 프로그램이 종료되는 것을
   방지할 수 있다.
   this.Close();
}
```

라. 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

제5장 부록

제5장

부록

제1절 설계단계 보안설계 기준

1. 입력데이터 검증 및 표현

사용자와 프로그램의 입력 데이터에 대한 유효성검증 체계를 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 크로스 사이트 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	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 경계값을 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	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	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 패스워드를 설정할 때 한국인터넷진흥원의 『암호이용안내서』 의 패스워드 생성규칙을 적용해야 한다. ② 네트워크로 패스워드를 전송하는 경우 반드시 패스워드를 암호화하거나 암호화된 통신 채널을 이용해야 한다. ③ 패스워드 저장 시, 솔트가 적용된 안전한 해쉬함수를 사용해야 하며, 해쉬함수 실행은 서버에서 해야 한다. ④ 패스워드 재설정/변경 시 안전하게 변경할 수 있는 규칙을 정의해서 적용해야 한다. ⑤ 패스워드 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 솔트값을 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	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.

제2절 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 공격이가능한 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 의도한 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 결과가 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 가능한보안약점
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 시주요정보도 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 발생할 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 역직렬화하여 발생하는 보안약점 * 직렬화: 객체를 전송 가능한 데이터형식으로 변환 * 역직렬화: 직렬화된 데이터를 원래 객체로 복원

6. 캡슐화

번호	보안약점	설명
1	잘못된 세션에 의한 데이터 정보노출	잘못된 세션에 의해 인가되지 않은 사용자에게 중요정보가 노출 가능한 보안약점
2	제거되지 않고 남은 디버그 코드	디버깅을 위한 코드를 제거하지 않아 인가되지 않은 사용자에게 중요정보가 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
1	DNS lookup에 의존한 보안결정	도메인명 확인(DNS lookup)으로 보안결정을 수행할 때 악의적으로 변조된 DNS 정보로 예기치 않은 보안위협에 노출되는 보안약점
2	취약한 API 사용	취약한 함수를 사용해서 예기치 않은 보안위협에 노출되는 보안약점

제3절 설계단계 보안설계 적용계획서

개발 중인 애플리케이션에서 발생할 수 있는 보안위협의 식별으로 조직 환경에 적합한 보안요구 항목별 적용 계획서를 작성한다. 이 계획서는 보안담당자의 기술 지원으로 소프트웨어 분석·설계자가 새로 개발되는 애플리케이션의 업무 현황 파악 문서 및 보안 정책 검토자료를 기반으로 보안요구 항목에 부합하는 효과적인 보안대책이 적용될 수 있도록 작성한다.

1. 입력데이터 검증 및 표현(예시)

요구사항분류	입력데이터 검증 및 표현	요구사항번호	SR1-1
요구사항이름	DBMS 조회 및 결과 검증		
요구사항내용	① 애플리케이션에서 DB연결을 수행할 때 최소권한의 계정을 사용해야 한다.		
보안요구항목	1. 애플리케이션에서 DB연결을 위해 사용되는 계정 “APP1002_1010” 생성 2. APP1002_1010 계정은 “APP1002” 데이터베이스에 대해 조회, 수정, 삭제, 업데이트 권한만 설정 3. 서버의 DB 커넥션풀 설정으로 DB 연결설정 정의		
비고			

2. 보안기능(예시)

요구사항분류	보안기능	요구사항번호	SR2-2
요구사항이름	인증수행 제한		
요구사항내용	① 로그인 기능 구현 시, 인증시도 횟수를 제한하고 초과된 인증시도에 대해 인증 제한 정책을 적용해야 한다.		
보안요구항목	1. 로그인 시도 횟수를 5회로 제한 – 시도 횟수가 초과되면 계정을 잠그고, 새로운 패스워드를 설정하도록 함 – 새로운 패스워드 설정을 위해 본인인증 수행(i-pin 인증, 휴대폰 인증 중 선택해서 사용 가능) – 본인 인증이 성공하면 회원가입 시 등록된 이메일 주소를 이용하여 패스워드 재설정 링크를 전송함 – 링크로 요청된 패스워드 재설정인지 확인하고 패스워드를 재설정하도록 함 2. 시도 횟수 추적을 위해 DB에 LOGIN_HISTORY 정보가 관리되도록 DB설계		
비고	공통 보안 모듈로 작성함		

3. 에러처리(예시)

요구사항분류	에러처리	요구사항번호	SR3-1
요구사항이름	예외처리		
요구사항내용	① 에러가 발생한 경우 상세한 에러 정보가 사용자에게 노출되지 않게 해야 한다.		
보안요구항목	1. 디폴트 에러페이지를 생성하고 서버 설정으로 프로그램에서 처리되지 않은 에러에 대해 적용되도록 설정 2. ERROR-CODE 북을 작성 - 에러 코드북 작성 시 메시지 내용에 중요정보(개인정보, 시스템정보, 민감정보 등)가 포함되지 않도록 함 3. 에러 발생 시 지정된 ERROR-CODE북의 에러코드를 이용해 메시지가 출력되도록 표준코딩 정의서 작성		
비고	표준코딩정의서 작성항목 있음		

4. 세션통제(예시)

요구사항분류	세션통제	요구사항번호	SR4-1
요구사항이름	세션통제		
요구사항내용	① 세션ID가 안전하게 관리되도록 해야 한다.		
보안요구항목	1. “Session Server”를 구축하여 안전하게 세션 정보를 관리 - 세션서버 설정으로 세션객체의 유효주기를 설정하고, 컨텍스트 간 세션 객체 공유가 설정되지 않도록 함 2. 세션ID가 안전하게 전달되도록 서버 설정 - 세션 전달 모드를 Cookie모드로 설정 - 세션쿠키의 속성값을 HttpOnly로 설정 3. 장기간 접속하여 사용하는 경우에 세션ID 노출을 최소화하기 위해 세션ID 재할당 정책 설정 4. 성공적으로 로그인한 후 반드시 세션ID가 재 할당되도록 표준코딩정의서 작성		
비고	표준코딩정의서 작성항목 있음		

제4절 설계단계 보안설계 적용 산출물

전자정부 프레임워크 공통모듈의 화면 설계서에 보안설계 기준을 적용한 예이다. 예시는 DB에 데이터쓰기, 읽기, 수정하기, 삭제하기와 같은 각 기능별 적용해야할 보안항목들과 파일업로드 기능, 관리자페이지, 웹에디터와 같은 기능을 구현할 때 고려해야 하는 보안 항목들을 정리하였다.

1. DB에 데이터를 입력하는 기능(예시)

ID	SP-CREATE-01		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	회원가입 화면
업무기능	사용자 회원가입 신청	관련 TABLE	
단위시스템	사용자 회원가입		
화면			

처리개요
사용자의 회원정보를 입력받는 화면을 구성한다. 사용자에게 입력받는 회원정보는 아이디/패스워드를 포함한 사용자의 개인정보이다. 사용자 입력 후 가입신청 버튼을 누르면 사용자의 회원정보가 DB에 추가된다.

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	1	입력	일반회원아이디	중복아이디 검색
2	1	입력	일반회원이름	
3	1	입력	비밀번호	
4	1	입력	비밀번호확인	
5	1	입력	비밀번호힌트	비밀번호 힌트 선택
6	1	입력	비밀번호정답	
9	1	입력	전화번호	
11	1	입력	핸드폰번호	
12	1	입력	이메일 주소	
13	1	입력	소속정보	일반기업/공공기관 선택
14	1	입력	소속 명	
15	1	입력	우편번호	우편번호 검색
16	1	입력	주소	
17	1	입력	상세주소	
18	1	입력	인증서DN	인증서 등록
19	1	클릭	가입신청	입력된 회원정보로 가입신청

업무흐름

1. 사용자는 입력 창에 각각 자신의 회원정보를 입력한다.
2. 중복아이디 검색으로 입력한 아이디가 기존 회원정보에 존재하는지 확인한다.
3. 우편번호는 외부링크로 연결되어 값을 검색하고 검색된 결과가 입력된다.
4. 사용자는 첨부파일을 업로드 하여 인증서를 등록한다.
5. 회원정보 입력 후 가입신청 버튼을 누르면 시스템은 사용자의 회원정보를 데이터베이스에 추가한다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 중복아이디 검색을 위한 사용자ID에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함. • 우편번호 검색을 위한 입력값을 조작하여 SQL 삽입 공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 회원가입을 위한 입력정보에 악의적인 스크립트가 포함 될 수 있으므로 입력값 검증이 필요함.	공통필터 적용

보안설계			
구분	설계항목	중점점검항목	비고
SR1-6	웹 기반 중요 기능 수행 요청 유효성 검증	• 스크립트를 이용하여 회원가입 요청이 처리될 수 있으므로, 실제 사용자의 유효한 요청인지 여부를 확인해야 함.	공통필터 적용
SR1-10	업로드·다운로드 파일 검증	• 인증서 등록 시 허가된 유형의 파일만 업로드 되도록 해 야 함.	
SR2-3	비밀번호 관리	• 사용자가 입력한 비밀번호에 대해 안전한 패스워드 설정 정책이 적용되어야 함. • 비밀번호 힌트 및 정답은 예측 가능하지 않은 질문과 답변을 사용하도록 해야 함.	
SR2-7	중요정보 저장	• 비밀번호는 솔트가 적용된 안전한 해쉬함수를 사용해서 암호화해서 저장해야 함. • 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	공통함수 hash()를 호출하여 비밀번호
SR2-8	중요정보 전송	• 사용자가 입력한 회원정보는 암호화하여 전송하거나, 암호화된 통신 채널으로 전달해야 함.	
SR3-1	예외처리	• 회원정보 입력 및 제출과정에서 오류가 발생하는 경우 지정된 페이지로 리다이렉트 하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

2. 데이터를 조회하는 기능(예시)

ID	SP-READ-02		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	아이디/비밀번호 찾기 화면
업무기능	사용자 로그인 정보 입력	관련 TABLE	
단위시스템	사용자 로그인		
화면			

처리개요
인증정보 조회 및 재설정을 위한 정보를 입력 받는 화면을 구성한다. 사용자에게 입력 받는 정보는 이름, 이메일 등 개인정보이다. 사용자 입력 후 아이디 및 비밀번호 찾기 버튼을 누르면 시스템은 그에 해당하는 사용자 인증정보를 조회한다.

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	1	입력	이름	
2	1	입력	이메일	
3	1	클릭	아이디 찾기	아이디 조회
4	1	입력	아이디	
5	1	입력	이름	
6	1	입력	이메일	

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
7	1	선택	비밀번호 힌트	
8	1	입력	비밀번호 정답	
9	1	클릭	비밀번호 찾기	임시비밀번호 발송
10	0	출력	아이디 찾기 결과	조회된 아이디 출력
11	0	출력	비밀번호 찾기 결과	임시비밀번호 발송결과 출력

업무 흐름

1. 사용자는 이름과 이메일을 입력하여 아이디를 조회한다.
2. 시스템은 해당 정보와 매칭되는 아이디 정보를 사용자에게 제공한다.
3. 사용자는 아이디, 이름을 포함한 개인정보를 입력하여 비밀번호 조회를 시도한다.
4. 시스템은 해당 정보와 매칭되는 계정의 존재여부를 확인하고, 계정이 존재할 경우 이메일 주소로 임시비밀번호를 발송한다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 아이디/비밀번호 찾기 시 검색을 위한 사용자 이름, 이메일, ID에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 아이디/비밀번호 찾기시 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR2-3	비밀번호 관리	• 비밀번호 찾기 시 사전에 등록된 이메일 주소로만 임시 비밀번호가 전송되도록 제한해야 함. • 임시 비밀번호로 첫 로그인시 반드시 신규 비밀번호로 변경하도록 관리해야 함.	
SR2-4	중요자원 접근 통제	• URL을 변조하여 중요페이지 접근에 대한 인증을 우회할 수 없도록 해야 함. • SSI(Server-Side Includes) 변수가 조작되어 명령실행으로 서버 데이터 정보가 누출되지 않도록 해야 함.	
SR2-7	중요정보 저장	• 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	
SR2-8	중요정보 전송	• 사용자가 입력한 정보는 암호화하여 전송하거나, 암호화된 통신 채널으로 전달해야 함.	
SR3-1	예외처리	• 사용자 정보 입력 및 제출과정에서 오류가 발생하는 경우 지정 된 페이지로 리다이렉트 하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

3. 데이터를 수정/삭제하는 기능(예시)

ID	SP-UPDATE/DELETE-03		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	게시물 수정/삭제 화면
업무기능	게시물 수정/삭제	관련 TABLE	
단위시스템	게시물 수정/삭제		
화면			

처리개요
Q&A 게시글을 수정/삭제하거나 목록을 출력하는 화면을 구성한다.

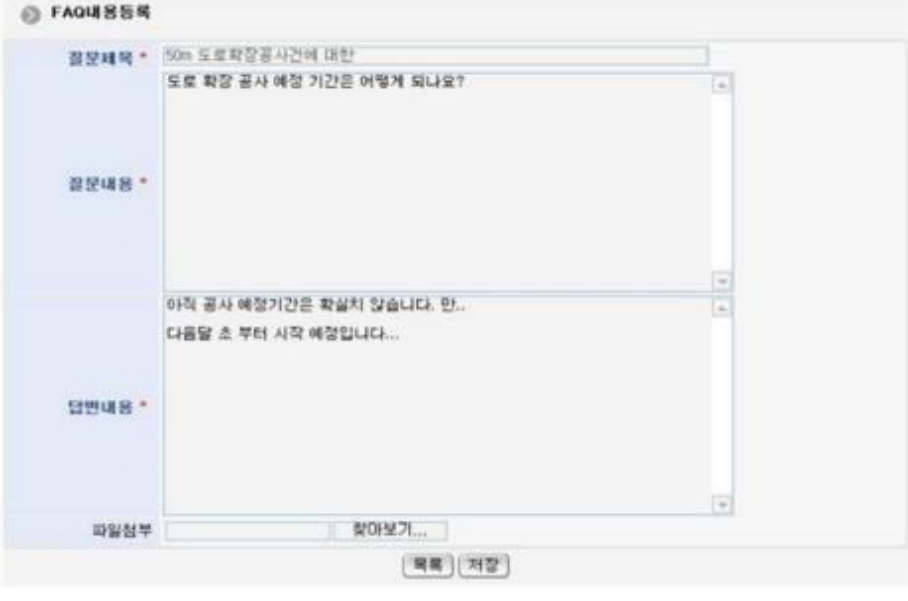
화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	1	클릭	Q&A - 글 수정	
2	1	클릭	Q&A - 글 삭제	
3	1	클릭	Q&A - 글 목록 조회	

업무 흐름
1. 사용자가 조회한 Q&A의 게시글 내용을 수정한다. 2. 사용자가 조회한 Q&A의 게시글 내용을 삭제한다. 3. Q&A의 목록 조회 화면을 출력한다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 목록 조회 시 URL 파라미터에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 게시글 수정, 목록 조회 시 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-6	웹 기반 중요기능 수행 요청 유효성 검증	• 스크립트를 이용하여 게시글 수정/삭제 요청이 처리 될 수 있으므로, 실제 사용자의 유효한 요청여부를 확인해야 함.	
SR1-9	보안기능 입력값 검증	• 쿠키·환경변수·Hidden 필드 값을 조작하여 게시글 수정/삭제 요청이 처리될 수 있으므로 실제 사용자의 유효한 요청여부를 확인해야 함.	
SR2-1	인증 대상 및 방식	• 인증이 필요한 게시판, 관리자 전용 게시판 등 중요페이지 접근시 인증 절차가 부재이거나, 부적절하게 인증 과정을 우회 할 수 있는지를 확인해야 함.	
SR2-4	중요자원 접근통제	• URL을 변조하여 중요페이지 또는 권한 없는 페이지 접근에 대한 인증을 우회할 수 없도록 해야 함. • SSI(Server-Side Includes) 변수가 조작되어 명령실행으로 서버 데이터 정보가 누출되지 않도록 해야 함.	
SR2-7	중요정보 저장	• 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야함.	
SR3-1	예외처리	• 게시물 수정/삭제시 오류가 발생할 경우 지정된 페이지로 리다이렉트 하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 중복 로그인을 허용하지 않는 경우 이전에 생성된 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료해야 함. • 로그인 상태인 사용자를 위해 해당 페이지에 로그아웃 인터페이스를 구현함. • 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야함.	

4. 파일을 업로드 하는 기능(예시)

ID	SP-UPDATE/DELETE-03		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	파일을 포함한 게시글 등록 화면
업무기능	파일을 포함한 게시글 등록	관련 TABLE	
단위시스템	파일을 포함한 게시글 등록		
화면			



처리개요
사용자에게 FAQ 내용과 첨부파일을 입력 받아 저장하거나 목록을 출력하는 화면을 구성한다.

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	1	입력	질문제목	
2	1	입력	질문내용	
3	1	입력	답변내용	
4	1	입력	파일첨부	파일 업로드
5	1	클릭	저장	FAQ 내용 등록
6	1	클릭	FAQ-글 목록조회	

업무 흐름
1. 사용자는 입력 창에 FAQ 내용을 등록한다. 2. 첨부할 파일이 있는 경우 파일첨부 기능으로 업로드 한다. 3. 내용 입력 후 저장 버튼을 누르면 시스템은 해당 내용을 데이터베이스에 추가한다. 4. FAQ의 목록 조회 화면을 출력한다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 목록 조회 시 URL 파라미터에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 게시글 등록, 목록 조회를 위한 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-6	웹 기반 중요 기능 수행 요청 유효성 검증	• 스크립트를 이용하여 게시글 등록 요청이 처리될 수 있으므로, 실제 사용자의 유효한 요청여부를 확인해야 함.	
SR1-9	보안기능 입력값 검증	• 쿠키·환경변수·Hidden 필드 값을 조작하여 게시글 등록 요청이 처리될 수 있으므로 실제 사용자의 유효한 요청여부를 확인해야 함.	
SR1-10	업로드·다운로드 파일 검증	• 첨부파일 업로드 시 사용자가 업로드 하는 파일의 크기, 확장자 및 콘텐츠 타입 등을 검증하여 웹셸 등이 업로드 되지 않도록 해야 함. • 업로드 파일의 저장경로는 외부에서 직접 접근 불가능한 경로를 사용함. • 업로드 된 파일은 실행권한을 가지지 않도록 함.	
SR2-1	인증 대상 및 방식	• 인증이 필요한 관리자 전용 게시판 등 중요페이지 접근 시 인증절차가 부재이거나, 부적절하게 인증 과정을 우회할 수 있는지를 확인해야 함.	
SR2-4	중요자원 접근통제	• URL/파라미터를 조작하여 중요페이지 또는 권한 없는 페이지 접근에 대한 인증을 우회할 수 없도록 해야 함. • SSI(Server-Side Includes) 변수가 조작되어 명령실행으로 서버 데이터 정보가 누출되지 않도록 해야 함.	
SR2-7	중요정보 저장	• 첨부파일 업로드 시 실제 저장명, 파일의 저장경로 등이 노출되지 않도록 함. • 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	
SR3-1	예외처리	• 게시물 등록 및 파일첨부 시 제출과정에서 오류가 발생하는 경우 지정된 페이지로 리다이렉트하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 중복 로그인을 허용하지 않는 경우 이전에 생성된 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료해야 함. • 로그인 상태인 사용자를 위해 해당 페이지에 로그아웃 인터페이스를 구현함. • 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

5. 파일을 다운로드 하는 기능(예시)

ID	SP-FILEDOWNLOAD-05		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	게시물 내 첨부파일 다운로드 화면
업무기능	게시물 내 첨부파일 다운로드	관련 TABLE	
단위시스템	게시물 내 첨부파일 다운로드		
화면			

처리개요
사용자가 조회하는 공지사항의 상세내용을 출력하는 화면 내 첨부파일을 누르면 다운로드 URL로 이동하여 해당 파일을 다운로드 받을 수 있도록 한다.

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	I	입력	첨부파일	첨부파일 다운로드

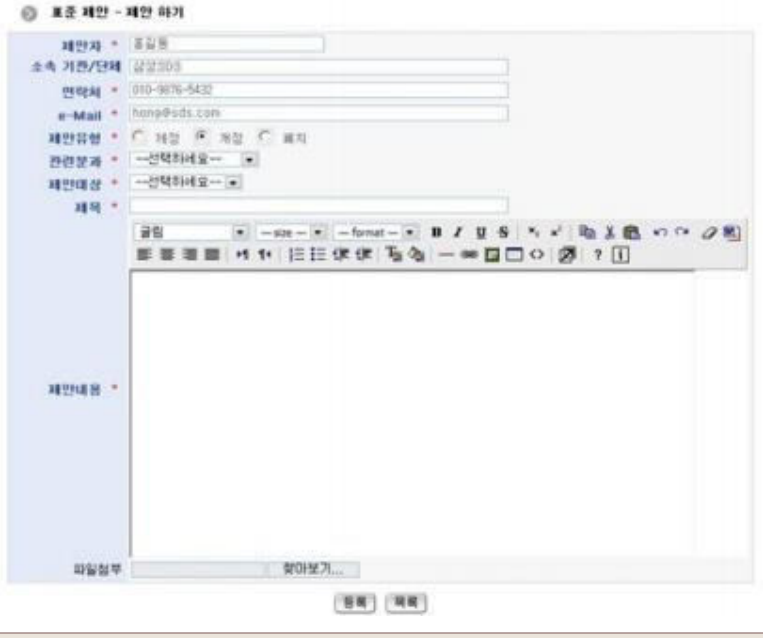
업무 흐름
사용자는 첨부파일을 클릭하여 해당 파일을 다운로드 받을 수 있다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 첨부파일 다운로드를 위한 URL 파라미터에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-4	시스템 자원 접근 및 명령어 수행 입력 값 검증	• 서버에서 명령창을 실행해두는 경우 사용자가 시스템 명령어를 입력할 가능성에 대비하여 미리 정의된 인자 값의 배열로 명령어를 선택하도록 해야 함.	

보안설계			
구분	설계항목	중점점검항목	비고
SR1-5	웹 서비스 요청 및 결과 검증	• 첨부파일 다운로드 시 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	
SR1-10	업로드·다운로드 파일검증	• 첨부파일 다운로드를 위해 요청되는 파일명에 경로를 조작하는 문자가 포함되어 있는지 점검해야 함. • 첨부파일 다운로드 시 사용자 입력 값을 검증하여 웹루트 디렉터리를 벗어난 영역에서 파일을 다운로드 할 수 없도록 해야 함. • 파일 다운로드를 처리하기 전에 파일에 대한 바이러스 및 악성코드 여부를 점검하고 다운로드 받은 파일의 무결성 검사를 실행함. • 필요시 인증된 사용자만 파일을 다운로드할 수 있도록 제한함.	
SR2-1	인증 대상 및 방식	• 인증이 필요한 게시판, 관리자 전용 게시판 등 중요페이지 접근시 인증 절차가 부재이거나, 부적절하게 인증 과정을 우회할 수 있는지를 확인해야 함.	
SR2-4	중요자원 접근통제	• URL/파라미터를 변조하여 중요페이지 또는 권한 없는 페이지 접근에 대한 인증을 우회할 수 없도록 해야 함. • SSI(Server-Side Includes) 변수가 조작되어 명령실행으로 서버 데이터 정보가 누출되지 않도록 해야 함.	
SR2-7	중요정보 저장	• 첨부파일 다운로드 시 실제 저장명, 파일의 저장경로 등이 노출되지 않도록 함. • 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	
SR3-1	예외처리	• 첨부파일 다운로드 요청 시 오류가 발생하는 경우 지정된 페이지로 리다이렉트하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 중복 로그인을 허용하지 않는 경우 이전에 생성된 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료해야 함. • 로그인 상태인 사용자를 위해 해당 페이지에 로그아웃 인터페이스를 구현함. • 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

6. 웹 에디터를 사용하는 기능(예시)

ID	SP-WEBEDITOR-06		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	웹 에디터 사용 게시물 등록 화면
업무기능	웹 에디터 사용 게시물 등록	관련 TABLE	
단위시스템	웹 에디터 사용 게시물 등록		
화면			



처리개요
사용자가 웹 에디터를 이용하여 표준제안 내용과 첨부파일을 입력하거나 목록을 출력하는 화면을 구성한다.

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
1	I	입력	제안자	
2	I	입력	소속기관/단체	
3	I	입력	연락처	
4	I	입력	e-Mail	
5	I	입력	제안유형	
6	I	입력	관련분과	

화면 입/출력 정보일람				
번호	I/O	이벤트	포인트	기능(링크 포함)
7	I	입력	제안대상	
8	I	입력	제목	
9	I	입력	제안내용	웹 에디터를 이용한 문서작성
10	I	입력	파일첨부	
11	I	클릭	등록	
12	I	클릭	목록	목록 조회

업무흐름

1. 사용자가 입력 창에 표준제안내용을 작성하고 등록버튼을 누르면 시스템은 해당 제안내용을 DB에 추가한다.
2. 표준 제안 목록 조회 화면을 출력한다.

보안설계			
구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 목록 조회를 위한 URL 파라미터에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 제안 등록, 목록 조회 시 사용자 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-6	웹 기반 중요 기능 수행 요청 유효성 검증	• 스크립트를 이용하여 제안등록 요청이 처리될 수 있으므로, 실제 사용자의 유효한 요청여부를 확인해야 함.	
SR1-9	보안기능 입력값 검증	• 쿠키·환경변수·Hidden 필드 값을 조작하여 게시글 등록 요청이 처리될 수 있으므로 실제사용자의 유효한 요청여부를 확인해야함.	
SR1-10	업로드·다운로드 파일 검증	• 첨부파일 업로드(웹 에디터 내 파일 업로드 포함)시 파일의 크기, 확장자 및 콘텐츠 타입 등을 검증하여 웹셸 등이 업로드 되지 않도록 해야 함. • 업로드 파일의 저장경로는 외부에서 직접 접근 불가능한 경로를 사용함. • 업로드된 파일은 실행권한을 가지지 않도록 해야 함.	
SR2-1	인증 대상 및 방식	• 인증이 필요한 게시판, 관리자 전용 게시판 등 중요페이지 접근 시 인증 절차가 부재이거나, 부적절하게 인증과정을 우회 할 수 있는지를 확인해야 함.	
SR2-4	중요자원 접근통제	• URL/파라미터를 변조하여 중요페이지 또는 권한 없는 페이지 접근에 대한 인증을 우회할 수 없도록 해야 함. • SSI(Server-Side Includes) 변수가 조작되어 명령실행으로 서버 데이터 정보가 누출되지 않도록 해야	

		함.	
보안설계			
구분	설계항목	중점점검항목	비고
SR2-7	중요정보 저장	• 첨부파일 업로드 시 실제저장 명, 파일의 저장경로 등이 노출되지 않도록 함. • 소스코드내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	
SR3-1	예외처리	• 게시물 등록 및 파일 첨부시 에러가 발생하는 경우 지정된 페이지로 리다이렉트 하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 중복 로그인을 허용하지 않는 경우 이전에 생성된 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료해야 함. • 로그인 상태인 사용자를 위해 해당 페이지에 로그아웃 인터페이스를 구현함. • 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

7. 관리자 페이지 기능(예시)

ID	SP-UPDATE/DELETE-03		
업무시스템	전자정부 프레임워크 공통컴포넌트	화면명	관리환경 공지사항 관리 화면
업무기능	관리환경 공지사항 관리	관련 TABLE	
단위시스템	관리환경 공지사항 관리		

화면

처리개요

관리환경에서 관리자만 접근 가능한 공지사항의 목록을 출력하는 화면을 구성한다.

화면 입/출력 정보일람

번호	I/O	이벤트	포인트	기능(링크 포함)
1	I	입력	검색 창	
2	I	클릭	검색	공지사항 검색

업무 흐름

1. 관리자는 검색창에 검색어를 입력하여 공지사항을 조회한다.
2. 시스템은 검색 및 조회된 공지사항을 화면에 출력한다.

보안설계

구분	설계항목	중점점검항목	비고
SR1-1	DBMS 조회 및 결과 검증	• 공지사항 검색을 위한 검색어에 쿼리를 조작할 수 있는 입력값으로 SQL 삽입공격이 시도될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR1-5	웹 서비스 요청 및 결과 검증	• 공지사항 검색을 위한 입력정보에 악의적인 스크립트가 포함될 수 있으므로 입력값 검증이 필요함.	공통필터 적용
SR2-1	인증 대상 및 방식	• 인증이 필요한 게시판, 관리자 전용 게시판 등 중요페이지 접근시 인증 절차가 부재이거나, 적절하지 않은 인증 과정을 우회할 수 있는지를 확인해야 함.	

보안설계			
구분	설계항목	중점점검항목	비고
SR2-4	중요자원 접근통제	• 소스접근통제 정책에 따라 관리자의 역할 및 접근권한에 기반한 올바른 접근이 이루어지도록 통제하여 인가 받은 사용자만을 허용하여야 함. • 관리자 전용 페이지의 URL은 쉽게 추측될 수 없도록 설정함. • 관리자 페이지를 원격으로 연결한 경우 암호화 통신채널을 사용해야 함.	
SR2-7	중요정보 저장	• 소스코드 내에 중요정보를 노출하는 주석문이 존재하지 않아야 함.	
SR3-1	예외처리	• 공지사항 조회 시 에러가 발생하는 경우 지정된 페이지로 리다이렉트하여 에러 정보가 노출되지 않도록 해야 함.	
SR4-1	세션통제	• 중복 로그인을 허용하지 않는 경우 이전에 생성된 로그인 세션을 종료하거나, 새로이 연결되는 세션을 종료해야 함. • 로그인 상태인 사용자를 위해 해당 페이지에 로그아웃 인터페이스를 구현함. • 세션 ID가 URL, 에러메시지, 로그 등에 노출되지 않도록 해야 함.	

제5절 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)

  해쉬 기반 메시지 인증 코드, 메시지 다이제스트 알고리듬 5(MD-5), SHA-1 등 반복적인

암호화 해쉬 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)

  해쉬알고리즘의 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 소스코드 상의 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 단 하나의 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.

● 해쉬함수

주어진 원문에서 고정된 길이의 의사난수를 생성하는 연산기법이며, 생성된 값은 '해쉬값'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.

전자정부 SW 개발·운영자를 위한

소프트웨어 개발보안 가이드

인 쇄 2021년 11월

발 행 2021년 11월

발행처 행정안전부
세종특별자치시 한누리대로 411(어진동)

한국인터넷진흥원
전라남도 나주시 진흥길 9

〈비매품〉