소프트웨어 개발 보안 구축

1. 소프트웨어 개발 보안 설계

1) SW 개발 보안

▼ 개념

- 소스 코드 등에 존재하는 보안 취약점 제거
- 보안을 고려하여 기능을 설계 및 구현
- 개발 과정에서 지켜야 할 일련의 보안 활동

▼ 생명 주기

1. 요구 사항 명세

2. 설계

위협원 도출을 위한 위협 모델링

보안 설계 검토 및 보안 설계서 작성

3. 구현

표준 코딩 정의서 및 SW 개발 보안 가이드 준수

소스 코드 보안 약점 진단 및 개선

4. 테스트

모의 침투 테스트, 동적 분석을 통한 보안 취약점 진단 및 개선

5. 유지보수

▼ 구성 요소

- 기밀성, 무결성, 가용성을 지키고 서버 취약점을 사전에 방지하여 위협으로부터 위험을 최소화하는 구축 방법

▼ SW 개발 보안 3대 요소

- 기밀성 (Confidentiality) - 인가되지 않은 개인 혹은 시스템 접근에 따른 정보 공개 및 노출을 차단
- 가용성 (Availability) - 권한을 가진 사용자나 애플리케이션이 원하는 서비스를 지속 사용할 수 있도록 보장
- 무결성 (Integrity) - 정당한 방법을 따르지 않고선 데이터가 변경될 수 없으며, 데이터의 정확성 및 완전성과 고의/악의로 변경되거나 훼손 또는 파괴되지 않음을 보장

▼ SW 개발 보안 용어

- 자산 (Assets) - 조직의 데이터 또는 조직의 소유자가 가치를 부여한 대상 (서버의 하드웨어, 기업의 중요 데이터)
- 위협 (Threat) - 조직이나 기업의 자산에 악영향을 끼칠 수 있는 사건이나 행위 (해킹, 삭제, 자산의 불법적인 유출, 위/변조, 파손)
- 취약점 (Vulnerability) - 위협이 발생하기 위한 사전 조건에 따른 상황 (평문 전송, 입력 값 미검증, 비밀번호 공유)
- 위험 (Risk) - 위협이 취약점을 이용하여 조직의 자산 손실 피해를 가져올 가능성

2) 공격 기법

1. DOS 공격

▼ 개념

- 해당 시스템의 자원을 부족하게 하여 원래 의도된 용도로 사용하지 못하게 하는 공격

- 특정 서버에 수많은 접속 시도를 만들어 다른 이용자가 정상적으로 서비스 이용을 하지 못하게 하거나, 서버의 TCP 연결을 소진시키는 등의 공격

▼ 종류

- 지역 시스템 공격

실제 대상 시스템에 접근하여 서버 하드웨어에 직접 과부하를 주는 공격 (메모리 고갈, 프로세스 서비스 거부, 디스크 서비스 거부 등)

- 원격 네트워크 공격

공격자가 목표 시스템에 접근하지 않고 원격지에서 인터넷 등을 이용한 공격

서비스를 제공 받지 못하거나 실제 시스템에 영향

2. DDoS 공격

▼ 개념

- 여러 대의 공격자를 분산 배치하여 동시에 동작하게 함으로써 특정 사이트를 공격하는 기법
- 해커들이 취약한 인터넷 시스템에 대한 액세스가 이뤄지면, 침입한 시스템에 소프트웨어를 설치하고 이를 실행시켜 원격에서 공격을 개시
- 악성 코드 감염 시켜 공격 명령, 서버에 서비스 거부 공격

▼ 구성 요소

- 핸들러 - 마스터 시스템의 역할을 수행하는 프로그램
- 에이전트 - 공격 대상에게 직접 공격을 가하는 시스템
- 마스터 - 공격자에게서 직접 명령을 받는 시스템, 여러 대의 에이전트를 관리하는 역할
- 공격자 - 공격을 주도하는 해커의 컴퓨터
- 데몬 프로그램 - 에이전트 시스템의 역할을 수행하는 프로그램

▼ 대응 방안

- 차단 정책 업데이트 - 공격 규모를 확인하여 가용성이 침해될 수 있는 지점을 확인 및 데이터 기반 차단 정책 업데이트
- 좀비 PC IP 확보 - 공격자는 대부분 Source IP를 위조하므로 IP 위변조 여부를 확인하는 절차 필요
- 보안 솔루션 운영 - 방화벽, 침입 탐지 시스템 등의 보안 솔루션 운영
- 홈페이지 보안 관리 - 홈페이지에 대한 모의 해킹 등을 수행하여 보안 유지
- 시스템 패치 - 시스템에 존재하는 취약점을 패치

3. 자원 고갈 공격

▼ 개념

- 서버 간 핸드셰이크를 통해 통신이 연결되는 정상 트래픽과 달리 DoS 공격은 정상 접속을 시도하는 오픈된 소켓에 트래픽을 집중
- 공격이 임계치에 도달하면 사용자들은 네트워크에 전혀 접속할 수 없음

▼ 종류

▼ SYN 플러딩

- TCP 프로토콜의 구조적 문제를 이용한 공격
- 서버의 동시 가용 사용자 수를 SYN 패킷만 보내 점유하여 다른 사용자가 서버를 사용 불가능하게 하는 공격

- **공격자는 ACK를 발송하지 않고 계속 새로운 연결 요청**을 하게 되어 서버는 자원 할당을 해지하지 않고 자원만 소비하여 자원이 고갈

▼ 대응 방안

- TCP 연결 타임아웃을 짧게 가져가서 연결 요청 대기 시간을 줄임
- Backlog Queue를 늘림
- Syncookie 기능 활성화
- Anti-DDoS, 방화벽, 침입 차단 시스템 등 보안 장비를 통해 침입 탐지 및 차단 수행
- 패치 및 업데이트

▼ UDP 플러딩

- 대량의 UDP 패킷을 만들어 임의의 포트 번호로 전송하여 응답 메시지를 생성하게 하여 지속해서 자원 고갈시킴
- ICMP 패킷은 변조되어 공격자에게 전달되지 않아 대기함

▼ 스머프

- 출발지 주소를 공격 대상의 IP로 설정하여 네트워크 전체에게 ICMP Echo 패킷을 직접 브로드캐스팅하여 마비시키는 공격
- 바운스 사이트라고 불리는 제3의 사이트를 이용해 공격

▼ 대응 방안

- 증폭 네트워크로 사용되는 것을 막기 위해 다른 네트워크로 부터 자신의 네트워크로 들어오는 직접 브로드캐스트 패킷을 허용하지 않도록 라우터 설정
- 브로드캐스트 주소로 전송된 ICMP Echo Request 메시지에 대해 응답하지 않도록 시스템 설정

▼ PoD (Ping of Death)

- 큰 사이즈의 패킷을 의도적으로 목표 시스템으로 발생시켜 시스템이 서비스 할 수 없는 상태로 만드는 공격
- Ping → 비정상 패킷 잘게 쪼개져 전송

▼ 대응 방안

- 보통 ICMP 패킷은 분할하지 않으므로 패킷 중 분할이 일어난 패킷을 공격으로 의심하여 탐지하도록 설정
- 현재 시스템 대부분은 반복적으로 들어오는 일정 수 이상의 ICMP 패킷을 무시하도록 설정되어 있지만, 취약점을 가지고 있다면 패치 필요

4. 애플리케이션 공격

▼ 종류

- HTTP GET 플러딩 (Cache Control Attack 공격)

HTTP 캐시 옵션을 조작하여 캐싱 서버가 아닌 웹 서버가 직접 처리하도록 유도, 웹 서버 자원을 소진시키는 서비스 거부 공격

- Slowloris (Slow HTTP Header DoS)

HTTP GET 메소드를 사용하여 헤더의 최종 끝을 알리는 개행 문자열 전송하지 않고 일부만 전송하여 대상 웹 서버와 연결 상태를 장시간 지속시키고 연결 자원을 모두 소진 시키는 서비스 거부 공격

- RUDY (Slow HTTP POST DoS)

요청 헤더의 Content-length를 비정상적으로 크게 설정하여 메시지 바디 부분을 매우 소량으로 보내 계속 연결 상태를 유지시키는 공격

▼ 대응 방안

- 동시 연결에 대한 임계치 설정을 통해 차단

- 연결 타임아웃 설정을 통해 차단
- 읽기 타임아웃 설정을 통해 차단

5. 네트워크 서비스 공격

▼ 종류

- 네트워크 스캐너, 스니퍼

네트워크 하드웨어 및 소프트웨어 구성의 취약점 파악을 위해 공격자가 사용하는 공격 도구

- 패스워드 크래킹

사전 크래킹, 무차별 크래킹 방법 사용해 네트워크 패스워드 탐색 (John the Ripper)

- IP 스푸핑

서버에 대한 인증되지 않은 액세스 권한을 입수하는 데 사용하는 기법

침입자가 패킷 헤더 수정을 통해 인증된 호스트의 IP 어드레스를 위조

타깃 서버로 메시지를 발송한 이후 패킷은 해당 포트에서 유입되는 것 처럼 표시

- 트로이 목마

악성 루틴이 숨어 있는 프로그램으로서 겉보기에는 정상적인 프로그램 처럼 보이지만 실행하면 악성 코드 실행

▼ 대응 방안

- 방화벽, 침입 차단 시스템 등 네트워크 보안 장비를 통해 방어
- 네트워크 접속 차단 시스템을 통해 방어
- 내부 호스트 및 시스템의 악성 코드 감염 방지를 위한 백신 설치 등을 통해 방어

6. 취약점 공격

▼ 종류

- 랜드 어택

출발지 IP와 목적지 IP를 같은 패킷 주소로 만들어 보냄으로써 수신자가 자기 자신에게 응답을 보내게 하여 시스템의 가용성을 침해하는 공격기법

대응 방안: 수신되는 패킷 중 출발지 주소와 목적지 주소가 동일한 패킷들 차단

- 봉크 / 보잉크

프로토콜의 오류 제어를 이용한 공격 기법으로 시스템의 패킷 재전송과 재조립이 과부하를 유발

봉크: 같은 시퀀스 번호를 계속 보냄

보잉크: 일정한 간격으로 시퀀스 번호에 빈 공간 생성

방안: 과부하가 걸리거나 반복되는 패킷 재전송 요구를 하지 않고 버림

- 티어 드롭

IP 패킷의 재조합 과정에서 잘못된 Fragment Offset 정보로 인해 수신 시스템이 문제를 발생하도록 만드는 DoS 공격

공격자는 IP Fragment Offset 값을 서로 중첩되도록 조작하여 전송, 이를 수신한 시스템이 재조합하는 과정에서 오류 발생, 시스템의 기능을 마비시키는 공격

3) 암호화 알고리즘

▼ 개념

- 데이터의 무결성 및 기밀성 확보를 위해 정보를 쉽게 해독할 수 없는 형태로 변환하는 기법

▼ 방식

▼ 대칭 키 암호 방식

- 암호화와 복호화에 같은 암호키를 쓰는 알고리즘
- 대칭 키는 블록 암호화와 스트림 암호화 알고리즘으로 나뉨

▼ 대칭 키 암호 방식

- 블록 암호 방식

긴 평문을 암호화하기 위해 고정길이의 블록을 암호화하는 블록 암호 알고리즘을 반복하는 방법

DES : 56bit 키를 이용, 64bit의 평문 블록을 64bit 암호문 블록으로 만드는 블록 암호 방식의 미국 표준 암호화 알고리즘

AES : DES를 대체

SEED : 고속 블록 단위의 128bit 대칭 키 암호화 알고리즘

- 스트림 암호 방식

매우 긴 주기의 난수열을 발생시켜 평문과 더불어 암호문을 생성하는 방식

RC4

▼ 비대칭 키 암호 방식

- 공개 키를 이용해 암호화하고 공개 키에 해당하는 개인 키를 이용해 복호화
- 공개 키는 누구나 알 수 있지만 그에 대응하는 개인키는 키의 소유자만 알 수 있음

▼ 비대칭 키 암호 구성하는 알고리즘

- 디피-헬만

암호 키를 교환하는 방법으로 두 사람이 암호화 되지 않은 통신망을 통해 공통의 비밀 키를 공유할 수 있도록 하는 방식

- RSA

가장 널리 사용 중 이며 소인수 분해의 어려움을 이용한 방식

▼ 해시 방식

- 단방향 알고리즘으로서 임의의 데이터를 고정된 길이의 데이터로 매핑하는 함수
- 해시 함수의 결과로 원본 데이터를 유추하기 어려운 것을 이용
- 연산에 걸리는 시간이 빠른 것이 장점, 동일한 결과를 갖는 값이 발생하는 해시 충돌 문제 발생 가능

▼ 종류

- SHA

미국 국가 표준 (SHA-2로 통칭)

- MD5

MD4를 대체하기 위해 고안한 128bit 해시 암호화 알고리즘

4) 보안 설계 방법

- **정보에 대한 보안 항목 식별**

▼ 법률적 검토

- 개인정보 보호법
- 정보통신망법

- 신용 정보법

▼ 개인정보 등급 분류

- 1 등급

고유 식별정보 - 주민번호, 여권번호, 운전면허, 외국인등록번호

민감 정보 - 유전자 검사 정보, 범죄경력 정보 등 정보 주체의 사생활을 현저하게 침해할 정보

- **자산에 대한 보안 항목 식별**

▼ 정보 자산 주요 용어

- 자산 - 조직에서 보유한 가치 있는 모든 것 (정보, 소프트웨어, 물리적 자산, 인력, 서비스 등)
- 사용자
- 소유자
- 관리자 - 자산의 보관 및 운영 책임

▼ 정보 자산 분류 기준

- 소프트웨어, 하드웨어, 데이터, 문서, 시설, 지원 설비, 인력
- 자산 목록 작성

- **기능에 대한 보안 항목 식별**

▼ 입력 데이터 검증 및 표현

- 사용자, 프로그램 입력 데이터에 대한 유효성 검증 체계를 갖추고, 실패 시 처리 할 수 있도록 설계
- DBMS 조회, 결과 검증
- XML 조회, 결과 검증
- 웹 서비스 요청, 결과 검증
- 허용된 범위 내 메모리 접근 검증
- 보안 기능 입력값 검증
- 업로드, 다운로드 파일 검증

- 보안 기능

인증 관리, 권한 관리, 암호화, 중요 정보 처리

- 에러 처리 및 세션 통제

세션 통제: 다른 세션 간 데이터 공유 금지, 세션 ID 노출 금지, 세션 종료 처리 등, 세션 하이재킹 방지

예외 처리: 오류 메시지에 중요 정보가 포함된 출력 방지 방법 설계, 에러 및 오류의 부적절한 처리 방지 방법 설계

5**) SW 개발 보안 적용 사례**

- MS-SDL
- Seven Touchpoints

객관적인 위험 분석 및 테스트를 거쳐 안전한 소프트웨어를 만드는 방법 정의, 위험요소 추적하고 모니터링

- CLASP

2. 소프트웨어 개발 보안 구현

(1) 시큐어 코딩 가이드

- 보안 취약점을 사전에 제거하고, 외부 공격으로부터 안전한 소프트웨어 개발하는 기법

▼ 가이드

▼ 입력 데이터 검증 및 표현

- 내용

프로그램 입력 값에 대한 검증 누락, 부적절한 검증, 잘못된 형식 지정

- 방안

사용자, 프로그램 입력 데이터에 대한 유효성 검증 체계 수립하고 실패 시 처리 설계 및 구현

▼ 주요 공격

- SQL 삽입

사용자의 입력 값 등 외부 입력 값이 SQL 쿼리에 삽입되어 공격

→ PreparedStatement 객체 등을 이용, DB에 컴파일 된 쿼리문(상수)을 전달

개념: 웹 애플리케이션에서 입력 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 창 및 URL에 SQL문을 삽입하여 DB로 부터 정보를 열람, 조작할 수 있는 취약점 공격 기법

```
// 안전하지 않은 코드 -> 외부로 부터 입력 받은 카테고리의 값을 아무런 검증 과정을 거치지 않고 SQL 쿼리를 생성하는 데 사용

String category = request.getParameter("category");

// 외부로 부터 입력 받은 값을 검증 없이 사용할 경우 안전하지 않음
String sql = "SELECT * FROM board WHERE b_category=" + category + "";

Connection conn = db.getConnection();
// 외부로부터 입력 받은 값이 처리 없이 쿼리로 수행되어 안전하지 않음
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

```
// 안전 코드 -> 매개 변수를 받는 PreparedStatement 객체를 상수 문자열로 생성하고 파라미터 부분을 setString 등의 메서드로 설정하여 외부의 입력

String category = request.getParameter("category");

// 외부로부터 입력 받은 값은 안전하지 않을 수 있어 PreparedStatement 사용을 위해 ?문자로 바인딩 변수 사용
String sql = "SELECT * FROM board WHERE b_category=?";

Connection conn = db.getConnection();

// PreparedStatement 사용
PreparedStatement pstmt = conn.prepareStatement(sql);

// PreparedStatement 객체를 상수 문자열로 생성하고 파라미터 부분을 setString 메서드로 설정하여 안전
pstmt.setString(1, category);
ResultSet rs = pstmt.executeQuery();
```

- 크로스 사이트 스크립트 (XSS)

검증되지 않은 외부 입력 값에 의해 브라우저에서 악의적인 코드가 실행

→ 입출력 값에 문자열 치환 함수를 사용

개념 :

웹 페이지에 악의적인 스크립트를 포함해 사용자 측에서 실행되게 유도할 수 있는 공격 기법

검증되지 않은 외부 입력이 동적 웹 페이지 생성에 사용될 경우 전송된 동적 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 수행되어 정보 유출 등의 공격을 유발

```
<-- 외부 입력 값을 검증 없이 화면에 출력될 경우 공격 스크립트가 포함된 URL을 생성할 수 있어 위험(Reflected XSS)-->
<% String keyword = request.getParameter("keyword"); %>

<-- 게시판의 입력 form을 통해 외부 값이 DB에 저장하고 검증 없이 화면에 출력 시 공격 스크립트가 실행되어 위험(Stored XSS) -->
검색 결과: ${m.content}
<script type="text/javascript">

<-- 서버를 거치지 않는 공격 스크립트가 포함된 URL을 새엉할 수 있어 위험(DOM 기반 XSS)
document.write("keyword:" + <%=keyword%>);
</script>
```

```
<-- 입력값에 대하여 스크립트 공격 가능성이 있는 문자열을 치환 -->
<% String keyword = request.getParameter("keyword"); %>
keyword = keyword.replaceAll("&", "&amp;");
keyword = keyword.replaceAll("<", "&lt;");
keyword = keyword.replaceAll(">", "&gt;");
...
<-- JSP 에서 출력값에 JSTL의 <c:out>을 사용하여 처리 -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri = "http://java.sun.cpm/jsp/jstl/functions" prefix="fn" %>

검색 결과 <c:out value="${m.content}"/>

<script type="text/javascript">
document.write("keyword:"

<-- 잘 만들어진 외부 라이브러리를 화용 -->
<%=Encoder.encodeForJS(Encoder.encodeForHTML(keyword))%>);
</script>
```

- 결로 조작 및 자원 삽입

외부 입력된 값의 사전 검증이 없거나 잘못 처리 될 경우

제공되는 시스템 자원에 접근 경로 등의 정보로 이용될 때 발생

→ 경로 순회 공격 위험이 있는 문자를 제거하는 필터 사용

- 운영체제 명령어 삽입

운영체제 명령어 파라미터 입력 값이 적절한 사전 검증을 거치지 않고 사용될 때 공격자가 운영체제 명령어를 조작

→ 웹 인터페이스를 통해 내부로 시스템 명령어를 전달하지 않도록 프로그램 구성

▼ 보안 기능

- 내용

인증, 접근 제어, 기밀성, 암호화, 권한 관리 등)의 부적절한 구현

- 방안

인증, 접근 통제, 권한 관리 , 비밀번호 등의 정책이 적절하게 반영되도록 설계 및 구현

▼ 약점 유형

▼ 인증 관련 보안 약점

- 적절한 인증 없는 중요 기능 허용
- 반복된 인증 시도 제한 기능 부재
- 취약 패스워드 허용

▼ 권한 관리 보안 약점

- 중요 자원에 대한 잘못된 권한 설명

- 부적절한 인가

▼ 암호화 보안 약점

- 취약 암호화 알고리즘 사용
- 충분하지 않은 키 길이 사용
- 하드 코딩 된 비밀번호
- 부적절한 난수 사용
- 솔트 없는 일방향 해시 함수 사용

▼ 중요 정보 처리 시 보안 약점

- 중요 정보 평문 저장
- 중요 정보 평문 전송

▼ 시간 및 상태

- 내용

거의 동시에 수행 지원하는 병렬 시스템 또는 하나 이상의 프로세스가 동작하는 환경에서 시간 및 상태의 부적절한 관리

- 방안

공유 자원 접근 직렬화, 병렬 실행 가능 프레임워크 사용, 블록문 내에서만 재귀함수 호출

▼ 약점 유형

▼ 경쟁 조건

- 동일 자원에 대한 검사 시점과 사용 시점이 상이
- 동기화 오류, 교착 상태를 유발하는 보안 약점

→ 동기화 구문 사용(sysnchronized(SYNC)), 한 번에 하나의 프로세스만 접근 가능하도록 함

▼ 종료되지 않은 반복문 또는 재귀 함수

- 종료 조건이 없는 재귀 함수나 반복문 사용
- 무한 루프에 빠져 자원 고갈 유발하는 보안 약점

→ 재귀 함수: 종료 조건 정의 / 반복문: 흐름 검증 수행

▼ 에러 처리

- 내용

에러 미처리, 불충분한 처리 등으로 에러 메시지에 중요 정보 포함

- 방안

에러 또는 오류 상황을 처리하지 않거나, 불충분하게 처리되어 중요 정보 유출 등 보안 약점이 발생하지 않도록 시스템 설계 및 구현

▼ 약점 유형

- 오류 메시지 정보 노출

응용 프로그램의 민감 정보가 오류 메시지를 통해 노출

→ 오류 메시지는 정해진 사용자에게 유용한 최소한의 정보만 포함

```
try{
}
// 스택 정보 노출
catch(IOException e){
  e.printStackTrace(); // -> logger.error("ERROR-01: 파일 열기 에러");
}
```

- 오류 상황 대응 부재

오류 발생 부분에 예외 처리 미구현

→ C/C++ : if, switch문, Java: try-catch문

```
try{
}
// 대응이 없어 인증된 것으로 처리
catch(NullPointerException e){
}

try{
}
catch(NullPointerException e){
  s.setMessage(e.getMessage());
  return (makeLogin(s));
}
```

- 부적절한 예외 처리

예외 조건 적절하게 검사하지 않음

→ 함수 결과 값의 적정성 검증, 구체적인 예외 처리 수행

▼ 코드 오류

- 내용

타입 변환 오류, 메모리의 부적절한 반환 등과 같이 개발자가 범하는 코딩 오류로 인한 보안 약점

- 방안

코딩 규칙 도출 후 검증 가능한 스크립트 구성과 경고 순위의 최상향 조정 후 경고 메시지 코드 제거

▼ 약점 유형

- 널 포인터 역참조

널 값을 고려하지 않은 코드에서 발생

의도적으로 널 값을 유발해 예외 상황을 확인하고 추후 공격에 활용

→ 값을 참조하기 전에 널 값 인지 검사

```
// url에 null이 들어오면 널 포인터 역참조 발생
if(url.equals(""))

if(url != null || url.equals(""))
```

- 부적절한 자원 해제

자원 고갈로 인한 시스템 오류 유발 보안 약점

소켓, 힙 메모리 등 자원을 할당 받아 사용 후 미 반환 시 발생

→ 자원 획득 사용 후 반드시 자원 해제

- 해제된 자원 사용

해제한 메모리를 참조하여 의도치 않은 코드를 실행

→ 메모리 할당 해제 후 포인터에 널 값 저장

- 초기화 되지 않은 변수 사용

함수 내 지역 변수 초기화 하지 않고 사용할 시 발생

→ 모든 변수는 사용하기 전 초기 값을 할당

▼ 캡슐화

- 내용

기능성이 불충분한 캡슐화로 인해 인가되지 않은 사용자에게 데이터 누출

- 방안

디버그 코드 제거와 필수 정보 외의 클래스 내 프라이빗 접근자 지정

▼ 약점 유형

- 잘못된 세션에 의한 정보 노출

멀티 스레드 환경에서 서로 다른 세션 간 데이터가 공유될 수 있는 보안 약점

→ 싱글톤 패턴(전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서든지 참조 할 수 있도록 하는 패턴) 사용 시 변수 적용 범위 주의

- 제거되지 않은 디버그 코드

개발 완료 후 배포 단계에서 디버그 코드가 남아있는 경우 발생

→ 소프트웨어 배포 전 디버그 코드 확인 및 삭제

- 시스템 정보 노출

시스템 내부 데이터가 노출되어 공격의 실마리가 되는 보안 약점

→ 예외 상황 발생 시 시스템 내부 정보의 화면 노출이 없도록 개발

▼ API 오용

- 내용

의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API 사용

- 방안

개발 언어별 취약 API 확보 및 취약 API 검출 프로그램 사용

▼ 약점 유형

- DNS에 의존한 보안 결정

공격자가 DNS 정보를 변조하여 보안 결정을 우회 가능한 보안 약점

→ DNS로 확인된 정보 대신 IP 주소 사용

- 취약한 API 사용

금지되거나 안전하지 않은 함수 사용

→ 안전한 함수 사용

2. 보안 테스트와 결함 관리

(1) 보안 테스트

▼ 개념

- 보안 요구사항이 반영되어 있음을 보증하고, 취약점을 발견하고 개선하여 안전한 소프트웨어를 개발하기 위한 활동

▼ 유형

▼ 화이트박스 테스트

- 프로그램 내부 로직을 보면서 수행하는 테스트(구조 테스트)
- 정적 테스트

SW를 실행하지 않고 보안 약점 분석

SW 개발 단계에서 주로 사용

→ 취약점 초기 발견으로 수정 비용 절감

→ 컴포넌트 간 발생할 수 있는 통합된 취약점 발견에 제한적

→ 설계 구조 관점의 취약점은 식별 불가

▼ 블랙박스 테스트

- 프로그램 외부 사용자의 요구사항 명세를 보면서 수행하는 테스트(기능 테스트)
- 동적 분석

SW 실행 환경에서 보안 약점 분석

SW 시험 단계에서 주로 사용

→ 소스 코드 필요 없음

→ 정확도와 커버리지 향상

→ 구조 관점의 보안 약점 식별 불가

▼ 절차

- 준비, 실행, 개선, 관리, 종료