2

소프트웨어 보안취약점 데이터베이스 구축 사례

한양대학교 ▌김 유 경*
한국기술교육대학교 ▌신 승 철**
한국항공대학교 ▌안 준 선**
한양대학교 ▌이 욱 세**
동덕여자대학교 ▌이 은 영*
성균관대학교 ▌한 환 수**

1. 서 론

컴퓨팅 시스템의 보안문제의 해결에 있어서 소프트웨어 소스코드 수준의 보안성 강화를 통한 근원적인 접근법에 관심이 증가하고 있다. 최근 들어 응용프로그램의 취약점을 사용한 공격의 빈도가 사이버 위협의 주요 원인이 되고 있는 것으로 보고되고 있으며[1], 거의 모든 종류의 행정서비스와 정보서비스가 불특정 다수가 직접 접근할 수 있는 인터넷을 통해 제공되는 상황에서 소프트웨어 소스코드 수준의 보안성 강화가 중요한 관심사로 등장하고 있다. 또한 2009년 7월 7일 국내에서 발생한 대규모 DDoS(분산서비스거부) 공격과 같은 소프트웨어 시스템에 대한 공격 피해사례가 전 세계적으로도 증가하는 추세이며, 이러한 사례의 근원적인 원인도 대부분 소프트웨어 소스코드의 보안 취약성에서 찾을 수 있는 것으로 밝혀지고 있다.

소프트웨어 보안 분야는 정보보호, 프로그래밍 언어, 소프트웨어 공학 분야가 비슷하거나 서로 변환 가능한 목적과 수단을 공유하면서 새로이 형성되고 있다. 소프트웨어의 보안 취약점은 접근제어 시스템과 같은 보안용 소프트웨어가 구현이 미흡하여 본래의 보안 기능을 완전하게 달성하지 못하는 것과 같은 보안 프로그램의 기능적, 논리적 허점이나, 웹 브라우저와 같은 응용 소프트웨어가 버퍼 넘침(Buffer overflow)과 같은 소스코드상의 약점을 보유함으로써 공격의 빌미를 제공하게 되는 문제를 통틀어 포함한다. 이러한 소프트웨어 보안 문제를 해결하기 위하여 최근 들어 프로그래밍언어 분야의 프로그램 분석 및 검증 기술 또는 소프트웨어 공학 분야의 개발 방법론이나 정형 기법 등을 사용하고자 하는 연구과 활발히 진행되고 있다.

소프트웨어 보안 문제를 해결하는 것이 쉽지 않은 것은 이를 해결하는 몇 개의 원리적인 방법이 있는 것이 아니라 개개의 사안마다 접근 방법이 다르다는 데에서 그 원인을 찾을 수 있다. 따라서 현재 기술 수준에서 유일한 방법은 개개의 취약점과 공격 방법에 대한 방지책을 일일이 마련하는 것이다. 소프트웨어 보안 전문가들을 중심으로 이런 공격의 패턴과 방지방법을 연구하고 있고, 미국 등지에서는 정부차원의 활동도 활발히 이루어지고 있다.

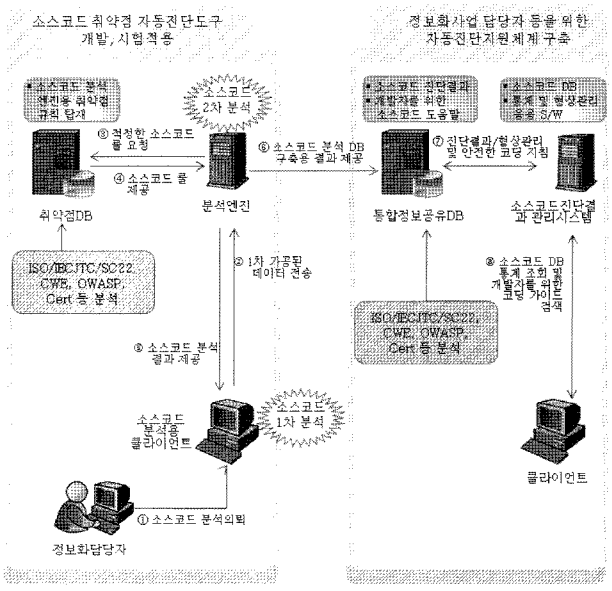

그림 1 소스코드 취약점 자동진단 체계

* 정회원
** 종신회원

국내에서도 행정안전부를 중심으로 전자정부 소프트웨어 개발 시에 보안 취약점에 대한 검사를 의무화하는 정책을 추진하고 있다. 이런 정책의 시행을 위해서는 현재까지 알려진 보안 취약점을 데이터베이스화하고, 취약점마다 취약점을 진단하는 규칙들을 고안하여 이 규칙들을 데이터베이스화한 다음, 이를 바탕으로 소프트웨어를 자동 검사하는 도구를 개발해야 한다. 뿐만 아니라 보안취약점 데이터베이스의 지속적인 갱신 관리체계, 의무화를 위한 법률, 소프트웨어 개발 시에 취약점이 발생하지 않도록 개발하는 방법(안전한 코딩 표준; secure coding standard)과 그의 교육 방안 등도 마련되어야 한다.

그림 1은 소프트웨어(소스코드)의 취약점을 자동으로 진단하는 시스템의 일반적인 구조를 보여준다. 소스코드의 분석을 의뢰하면 취약점 데이터베이스 내의 모든 취약점에 대하여 분석 엔진이 진단 규칙을 이용하여 자동검사를 수행하고, 그 분석 결과를 제공함과 함께 발견된 취약점을 제거하기 위한 코딩 표준을 안내함으로써 보안취약점이 없는 소프트웨어 개발을 유도할 수 있게 된다.

본 논문은 2009년 행전안전부의 주관 및 지원 하에 한국인터넷진흥원을 중심으로 (주)지티원, (주)파수닷컴, 한국정보보호학회 소프트웨어보안연구회에서 개발한 소스코드 보안취약점 자동진단 시스템의 일부인 보안취약점 데이터베이스의 구축과 관련하여 구축된 데이터베이스의 전체 구조와 관련 취약점의 개요에 대하여 제시한다. 논문의 나머지 부분은 다음과 같이 구성하였다. 2장 관련동향에서는 전 세계적으로 다양하게 수행되고 있는 보안취약점 관련 연구 동향을 설명한다. 3장에서는 이러한 기존 결과들에 기반하여 본 연구에서 구축한 보안 취약점 데이터베이스의 전체 구성과 데이터베이스에 포함된 취약점들을 분류별로 설명하고 각 취약점들의 중요도 및 영향도 평가 방법을 제시한다. 4장은 결론이다.

2. 관련 동향

2.1 CWE

CWE(Common Weakness Enumeration)는 미국 국토보안부(U.S. Department of Homeland Security) 내 국가사이버보안국(National Cyber Security Division)의 지원으로 MITRE에서 소프트웨어 취약점(weakness)을 다양한 관점에서 분류하여 모아 놓은 것이다. 국제적으로 공공 부문에 무료로 제공되고 있으며, 이를 통해 보다 효율적으로 소프트웨어 취약점에 대해 토의, 서술할 수 있도록 하고 있다. 또한 소스 코드에서 소프트웨어 취약점을 찾는 보안 도구 및 서비스의 사용을 활성화 시키고, 소프트웨어의 구조 및 설계에 있어서도 소프트웨어의 취약점에 대한 충분한 이해를 바탕으로 관리할 수 있도록 유도하고 있다.

2.2 CWSS

CWSS(Common Weakness Scoring System)는 취약점의 위험성에 대해 성문화하여 전달하기 위한 점수체계이다. 한 애플리케이션에서 소개되어야 할 아키텍처, 설계, 코드 또는 구현상에 존재하는 다양한 취약점들 및 소프트웨어의 보안상 중요 문제가 될 수 있는 취약점들에 대해 상대적인 중요도를 결정하기 위한 체계가 CWSS이다.

현재 취약점을 위한 평가방법으로 보안 취약점에 대한 평가 체계인 CVSS가 존재하며, 일반적인 취약점을 위한 유사한 점수 체계인 CWSS를 만들기 위한 연구가 CERT 등을 주축으로 진행되고 있다. CVSS에서는 "기본", "임시", 그리고 "환경" 유형으로 세분화하여 취약점의 심각성을 기술하고 있다. CWSS는 이보다 좀 더 광범위 한 점수 체계로서 취약점을 존재하게 만드는 요소, 침투가능하게 만드는 요소, 그리고 소프트웨어 시스템의 여러 부분에 크고 작은 영향을 주도록 만드는 요소들을 망라하게 될 것이다.

2.3 CVE

CVE(Common Vulnerabilities and Exposures)는 시간에 따라 감지된 보안취약점을 정리해 둔 목록이다. CWE와 다른 것은 CWE는 일반적인 취약점의 분류체계라고 한다면 CVE는 발견된 보안 취약점의 히스토리 기록이라고 볼 수 있다. 모든 CVE의 항목은 "CVE-년도

그림 2 CVSS 메트릭 그룹

-순번"의 식별자를 갖는다. 기본적으로 MITRE를 주축으로 해서 각종 소프트웨어 개발 회사(주로 OS)와 CERT/CC같은 기관에서 감지된 보안취약점을 보고하면, CVE 조정위원회를 통해서 목록이 관리된다. 1999년부터 시작해서 공인된 것만 매년 1,000 여개의 목록이 추가되고 있다.

2.4 CVSS

CVSS(Common Vulnerabilities Scoring System)는 보안 취약점들을 평가하고 확인할 수 있도록 제공된 오픈 프레임워크로서, 3가지 메트릭 그룹으로 구성되어 있다. 기본 메트릭 그룹은 시간과 사용자 환경에 의해 변하지 않는 취약점에 대한 본질적이고 근본적인 특징을 나타낸다. 더욱 정확하게 자신들의 환경에 존재하는 위험이 반영된 정황 정보를 제공받기 위해서 임시 메트릭과 환경 메트릭 그룹을 사용한다. 임시 메트릭 그룹은 시간의 흐름에 따라 변경되는 취약점의 특징을 나타낸다. 환경 메트릭 그룹은 특정 사용자 환경에 유일하고 관련된 취약점의 특징을 나타낸다.

기본 메트릭에 의해 지정된 값은 Base Formula에 의해 계산되고, 0과 10 사이의 점수로 계산된 결과는 취약점의 심각도를 의미한다. 이 결과는 다음 단계인 임시 메트릭 계산식에 입력되고, 임시 메트릭에 의해 지정된 값은 Temporal Formula에 의해 계산된다. 계산결과는 취약점의 심각도를 의미하며, 0과 10 사이의 값이다. 이 결과는 다음 단계인 환경 계산식에 입력되고, 환경 메트릭에 의해 지정된 값은 Environmental Formula에 의해 계산된다. 계산결과는 최종적으로 계산된 취약점의 종합점수이며, 제거해야 할 취약점의 우선순위가 된다.

2.5 CAPEC

CAPEC(Common Attack Pattern Enumeration and Classification)은 취약점을 분류하는 것이 아니라 보안취약점을 뚫는 공격패턴을 분류한다. 미국 국토안전부에서 지원을 하여 Cigital 회사 주도로 목록이 관리된다. CAPEC에는 공격 패턴의 형태와 실제 사례, 소프트웨어의 공격패턴에 대한 취약점 시험 및 검출 방법, 공격에 대응하는 방법, 중요도 등을 정리하고 있다. 이러한 정보를 통하여 시스템의 보안취약점을 제거하는데 도움이 될 수 있다. CAPEC 목록은 버전 1.0에서 시작해서, 2010년 1월 현재 버전 1.4까지 목록이 있으며, 버전 1.4 목록에는 287개의 공격패턴이 기록되어 있다.

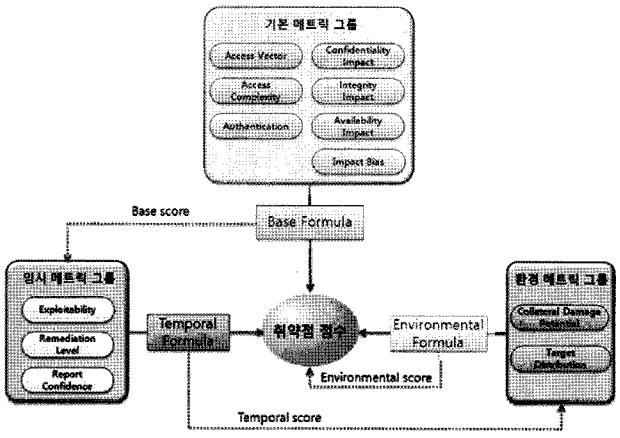

그림 3 CVSS 메트릭의 구성

2.6 SANS Top 25

MITRE와 SANS 협회에서는 매년 'CWE/SANS Top 25', 혹은 'SANS Top 25'라고 불리는 리스트를 발표하고 있다. 이 리스트는 가장 위험한 프로그래밍 오류(Most Dangerous Programming Errors)라는 부제를 가지고 있으며, 소프트웨어 취약점을 야기하는 가장 중요한 프로그래밍 오류들의 명시화하는 것을 목표로 하고 있다.

SANS Top 25 리스트는 SANS 협회와 MITRE, 그리고 미국과 유럽의 소프트웨어 보안 전문가들로 이루어진 자문회의에서 결정된다. 이 리스트는 SANS의 'Top 20 attack vector'와 MITRE의 CWE 리스트를 기반으로 만들어지며, 각각의 취약점들은 해당하는 CWE 번호와 대응되어 있다. SANS Top 25 리스트의 가장 큰 목적은 교육은 통하여 프로그래머들이 소스 레벨에서 취약점을 야기하는 코드를 만들지 않도록 하는 것이다. 따라서 이 리스트는 가장 핵심적인 오류들을 항상 포함하고 있도록 매년 지속적인 업데이트가 진행되고 있으며, 프로그래머들을 위한 교육에서 널리 사용되고 있다.

SANS Top 25 리스트의 내용은 크게 3가지 범위로 나누어진다. 첫 번째가 컴포넌트 사이의 안전하지 않은 상호작용에 관련된 내용이며, 두 번째가 위험한 자원 관리, 마지막으로 세 번째가 적절하지 못한 보안 방어법의 사용에 관련된 내용이다.

첫 번째 범주에 속하는 오류들은 잘못된 입력 값 검증이나 적절하지 못한 인코딩 기법을 사용한 출력 값의 생성, 에러 메시지를 통한 정보의 유출 등이 포함된다. 또한 프로그램 내부에서 SQL 질의 구조체를 잘못 관리하여 SQL 삽입 공격을 가능하게 하는 취약점이 발생하거나, 웹 페이지 구조체를 잘못 관리하여 크로스 사이트 스크립트 공격을 가능하게 하는 취약점이 발생하는 경우도 첫 번째 범주에 포함된다.

두 번째 범주는 프로그램이 자원 관리는 제대로 하지 못하여 소프트웨어에 취약점이 발생하는 경우를 포함한다. 메모리 버퍼의 범위를 제대로 관리하지 못하는 프로그램 오류나 중요한 내부 상태 데이터나 파일 경로 등을 외부에서 직접 조정하도록 허용할 때 발생하는 프로그램 오류들이 여기에 해당한다. 사용되는 코드의 흐름을 제대로 제어하지 못해서 발생하는 오류(code injection)와 다운로드 받은 코드의 무결성을 제대로 체크하지 않는 오류도 두 번째 범주에 해당된다. 또한 두 번째 범주는 자원을 사용 후 제대로 닫거나 릴리즈하지 않는 오류, 객체나 변수의 초기화를 제대로 수행하는 않는 오류, 그리고 잘못된 산술연산으로 인하여 발생하는 오류도 포함한다.

마지막으로 세 번째 범주에는 프로그램에서 사용되는 보안과 관련된 기법들이 오용, 혹은 과용되거나 적절하게 사용되지 못했을 때 발생하는 오류들이다. 여기에는 오류가 있는 암호화 알고리즘의 사용이나 문자열 상태로 프로그램 코드에 포함된 패스워드, 적절하지 못한 난수 발생기의 사용 등이 포함된다. 또한 세 번째 범주에는 부적절한 인증 기법의 사용이나 중요한 자원에 대한 권한 설정이 적절하지 못한 오류가 포함되어 있으며, 서버 측에서 처리해야 할 보안상의 문제를 클라이언트 측에서 다루는 시스템 디자인상의 오류도 언급되어 있다.

현재 가장 업데이트된 SANS Top 25 리스트 버전은 2009년도 버전이며, 각 범주별 분류와 대응되는 CWE 번호와 설명은 모두 웹페이지(http://cwe.mitre.org/top25/)에 공개되어 있다.

2.7 OWASP Top 10

OWASP Top 10 리스트는 웹 프로그래밍와 관련하여 가장 많이 발생하는 취약점을 정리한 리스트로 Open Web Application Security Project(OWASP)에 의해서 유지, 발표되는 리스트이다. 이 리스트는 MITRE에서 발표하는 CVE(Common Vulnerability Enumeration)을 기반으로 만들어지며, OWASP에서 진행되는 여러 가지 취약점 관련 프로젝트의 일부이다.

현재 최신 버전은 2007년도 버전으로 웹페이지(http://www.owasp.org/index.php)에서 공개되어 있으며, 미국 IBM 및 한국의 삼성 SDS를 포함한 다수의 회사와 학교들이 실무용, 혹은 교육용으로 OWASP Top 10 리스트를 사용하고 있는 것으로 알려져 있다.

OWASP Top 10 리스트의 장점은 웹 프로그래밍을 중심으로 취약점이 정리되어 있으며, 리스트의 길이가 짧아서 개발 시 적용이나 교육이 비교적 용이하다는 점과 리스트의 순서가 중요도(빈도와 심각성)을 반영하고 있다는 점을 들 수 있다. OWASP Top 10 리스트의 내용을 살펴보면 다음과 같으며, 잘 알려진 공격 패턴을 다수 포함하고 있는 것을 알 수 있다.

i. 크로스 사이트 스크립팅(Cross Site Scripting)
ii. 질의문 삽입 오류(Injection Flaws)
iii. 악의적인 파일 수행(Remote File Inclusion; RFI)
iv. 직접적인 객체 참조 오류(Insecure Direct Object Reference)
v. 크로스 사이트 요구 변조(Cross Site Request Forgery; CSRF)
vi. 정보의 유출과 잘못된 오류 처리(Information Leakage and Improper Error Handling)
vii. 잘못된 인증과 세션 관리(Broken Authentication and Session Management)
viii. 안전하지 않은 암호화 및 저장(Insecure Cryptographic Storage)
ix. 안전하지 않은 통신(Insecure Communications)
x. 제한되지 않은 URL 접근(Failure to Restrict URL Access)

3. 보안 취약점 데이터베이스 구축

본 장에서는 본 연구에서 개발한 소스코드 보안취약점 자동진단 시스템의 일부인 보안취약점 데이터베이스의 구축 사례에 관하여 설명한다.

3.1 전체 구성

취약점 데이터베이스는 기본 취약점 관련 정보인 취약점 테이블, 언어별 관련 취약점 정보와 검출 규칙을 정리한 취약점 규칙 테이블 및 취약점과 관련한 공격 패턴을 정리한 공격패턴 테이블로 구성된다. 본 취약점 데이터베이스는 CWE, CVE, CAPEC 등의 국외 관련 연구 내용과 행안부 주요 소프트웨어의 특성 및 업계 관련 제품의 성능 등을 참고하여 구축되었으며 xml 형식으로 정리되었다.

취약점 테이블은 취약점 식별자, 설명, 중요도, 관련 언어, 분류, 관련 공격 패턴, 참고문헌 등의 정보로 이루어져 있으며 약 212개의 취약점에 대하여 정리하고 있다. 취약점 규칙 테이블은 현재 C 및 Java와 관련된 취약점 각각에 대하여 취약점 규칙 식별자, 취약점 예제 및 수정 예제와 관련설명, 취약점 검출 규칙 및 규칙 설명 등으로 이루어져 해당 언어의 프로그램 개발 시 참고할 수 있다. 또한 취약점 검출 규칙은 규칙 명세 언어(Rule Description Language,

RDL)로 기술되어, 프로그램 취약점 자동 분석 엔진의 입력으로 사용된다. 공격 패턴 테이블은 공격 패턴 식별자, 공격 패턴 제목, 관련 공격 도구, 공격 패턴에 대한 취약점 테스트 방법 및 취약점 회비 방법 등의 정보를 담고 있다. 취약점 테이블의 각각의 취약점에 대해서는 해당하는 언어별 취약점 규칙이 1:n으로 연결되며, 취약점과 공격 패턴에 대해서는 n:n의 관계로 연관된다.

3.2 보안 취약점 분류

본 절에서는 구축된 취약점 데이터베이스의 취약점들을 원인 및 특성에 따라 분류하여 각각의 개념과 관련 취약점을 설명한다.

3.2.1 검증되지 않은 값의 사용(Data Handling)

본 취약점 분류는 검증되지 않은 부적절한 값이 보안에 민감한 작업에 사용됨으로써 발생하는 취약점을 말한다. 부적절한 값이란 외부의 불특정 입력이나, 범위를 벗어나는 주소값 및 첨자(index) 값, 각각의 자료형(types)의 범위를 벗어나는 값 등이 되며, 민감한 작업이란 메모리 접근, 명령어 수행, 경로값을 사용한 파일(file) 접근, 데이터베이스 접근, 동적 웹페이지의 생성, 형식 문자열(format string)을 사용하는 입출력문 등을 들 수 있다.

본 취약점 분류에 속하는 대표적인 취약점을 설명하면 다음과 같다.

- SQL 삽입 공격(SQL Injection Attack, KCWE-89)

이 프로그램은 대부분 웹 응용 프로그램에서 많이 발생한다. 외부에서 전달되는 문자열 값이 SQL 질의문의 생성에 사용될 때 적절한 문자열 형태만이 사용되도록 입력값 검증(input validation)이 동반되어야 한다. 부적절한 문자열 값이 질의문 생성에 사용될 경우 질의문의 의미를 변경하여 의도하지 않은 연산이나 허용되지 않은 접근이 일어날 수 있게 된다. 다음은 외부의 Name 입력을 받아 쿼리를 생성하는 프로그램 일부이다.

```
String name = getExternalInput("Name");
String query = "SELECT * FROM userTable WHERE
    Name = '"+name+"'";
ResultSet rs = stmt.executeQuery(query);
```

만약 공격자가 Name의 값으로 "name'; DELETE FROM userTable; --'"을 전달하게 되면 다음과 같은 질의문을 수행하여 테이블을 삭제할 수 있게 된다.

```
SELECT * FROM userTable WHERE Name ='name';
DELETE FROM userTable; --'
```

- 사이트 교차접속 스크립트 공격 취약점(Cross Site Scripting, KCWE-79)

외부에서 입력되는 검증되지 않은 입력이 동적 웹페이지의 생성에 사용될 경우, 전송된 동적 웹페이지를 열람하는 접속자의 권한으로 부적절한 스크립트가 수행되어 정보 유출과 같은 피해를 입힐 수 있게 된다. 이러한 공격이 이루어지는 절차는 다음과 같다.

i. 동적 웹페이지 생성에 외부의 불특정한 입력을 사용
ii. 악의적인 자바 스크립트 전달하는 URL을 피해자가 수행
iii. 생성된 도억 웹페이지가 피해자의 브라우저에서 수행(악성 스크립트가 실행)
iv. 웹서버와 관련된 정보의 유출과 같은 피해자에 대한 침해 발생

다음은 취약점을 가지는 서블릿(Servlet) 프로그램의 부분이다. 외부에서 전달되는 param 값이 악성 스크립트에 대한 접근을 포함할 경우 생성되는 웹페이지가 피해자에게 보이면서 악성 스크립트도 같이 수행되어 피해자의 정보 유출 등이 일어날 수 있게 된다.

```
PrintWriter out = response.getWriter();
    ...
String param = request.getParameter("name");
out.println("Hello "+param+" !");
```

- 메모리 버퍼 범위 조건 위반(KCWE-119)

프로그램이 메모리 첨자나 포인터값 등이 할당된 메모리 범위를 넘어가는 경우를 허용할 경우 공격자가 적절하지 않은 값을 전달함으로써, 의도되지 않은 메모리 구역을 접근하여 중요한 정보를 접근하거나 프로그램을 강제로 종료시키거나 혹은 임의의 코드를 실행시킬 수 있는 취약점을 갖게 된다.

다음 프로그램의 경우 4보다 큰 값이 index의 값으로 전달되면 의도하지 않았던 메모리 구역을 프로그램이 접근하게 된다.

```
int main (int argc, char **argv) {
    char *items[] = {"boat", "car","truck", "train"};
    int index = GetUntrustedOffset();
    printf("You selected %s₩n", items[index-1]);
}
```

3.2.2 API 사용 오류(API Abuse)

API 사용 오류란 개발환경에서 제공하는 시스템 라이브러리 중 자체에 보안취약점이 있는 것을 사용하거나, 라이브러리에 대한 이해 부족으로 잘못 사용하여 보안취약점을 발생시키는 것을 말한다.

API 사용 오류로 분류된 보안취약점 중 본 데이터베이스에는 약 34개의 오류를 정의하고 있다. 중요도가 높은 것들은 모두 포함하고 있고, 언어별로는 C 언어 관련 12개와 Java 관련 26개를 정의하고 있다.

대표적인 API 사용 오류는 다음과 같다.

- 위험하다고 알려진 함수 사용(KCWE-242)

이미 그 사용이 위험할 수 있다고 알려진 라이브러리 함수를 사용하는 것은 위험하다. 특정 라이브러리 함수들은 보안 취약점을 전혀 고려하지 않고 개발되어서, 사용자체가 취약점이 될 수 있다. 예를 들면, gets() 함수는 입력 크기 제한사항을 점검하지 않기 때문에, 입력 버퍼를 넘치게 할 수 있다. 즉, 이러한 함수들은 사용만 하면 언제든지 공격 대상이 될 수 있다. 다음 C 코드를 보면 gets를 사용하고 있다.

```
char buf[BUFSIZE];
gets(buf);
```

buf는 BUFSIZE의 배열인데 gets는 그 크기와 상관없이 입력을 buf에 저장하기 때문에 사용자가 BUFSIZE 이상의 문자열을 입력하면 버퍼 넘침을 유발할 수 있다. 버퍼 넘침은 해킹에서 자주 사용하는 요소이다.

- 함수 결과 검사 부재(Unchecked Return Value, KCWE-252)

함수의 결과로 정상 수행인지 오류가 있는지 알려주는 경우, 프로그래머가 함수의 결과를 항상 주시해야 한다. 이를 검사하지 않고 진행하는 경우 위험하다. 분류를 해보면 크게 두 가지로 나눌 수 있다. 하나는 함수의 반환값을 실패했을 때 널 포인터로 표시하는 경우, 반환값을 검사 없이 사용하면 널 포인터 접근 오류가 발생한다. 또 다른 하나는 버퍼에 결과를 쓰는 함수의 경우, 일부 함수는 그 실패를 버퍼가 아닌 반환값으로 전달한다. 반환값을 검사하지 않고 버퍼를 사용하면 위험하다.

다음 C 코드를 보면 malloc이 실패하면 buf는 널 포인터 값을 갖는다. 결과적으로 strncpy에서 널 포인터 접근을 하게 된다.

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

두 번째 경우에 해당하는 다음 C 코드를 보면, fgets가 충분히 값을 읽어오지 못하면 buf가 널 문자로 끝나지 않는 문자열을 갖게 된다. 그러면 strcpy에서 오류가 발생한다.

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

- super.finalize()를 호출하지 않는 finalize() 메소드(KCWE-568)

Java 언어 명세에 따르면 finalize() 메소드를 호출해도 자동으로 super.finalize()를 호출하지 않는다. 하지만 일반적으로 현재 객체가 finalize() 될 때 상위클래스의 finalize()도 호출되는 것이 안전하다. 아래와 같이 호출하지 않으면 상위클래스에서 정의된 끝내는 루틴이 작동되지 않아 오류가 발생할 수 있다.

```
protected void finalize() {
    discardNative();
}
```

3.2.3 보안 관련 오류(Security Features)

보안성과 관련된 오류란 패스워드 관리를 담당하는 프로그램이나 암호화 등을 이용하는 과정에서 발생한 프로그램 상의 오류가 시스템 전체를 취약하게 만드는 것을 말한다. 여기에는 사용자 암호를 정하지 않고 사용하는 경우나, 프로그램 내부에서 문자열 상수로 패스워드를 저장하고 그대로 사용하는 경우, 혹은 이미 취약점이 알려진 암호화 기법을 사용하는 경우 등이 포함된다.

본 데이터베이스에서는 약 34개의 보안성과 관련된 오류를 정의하고 있다. 이 중에서 C언어와 관련된 오류는 23개, Java와 관련된 언어는 33개이다.

보안성과 관련된 오류들은 오류가 발생하면 사용자 아이디나 패스워드 등이 노출되면서 시스템 전체의 기밀성이나 무결성이 침해되는 경우가 발생할 수 있다. 이 때문에 패스워드를 상수의 형태로 노출하거나 잘못된 암호화 알고리즘을 사용하는 오류는 발생빈도는 낮더라도 발생하는 경우, 침해의 결과는 치명적일 수 있다.

대표적인 보안성 관련 오류는 다음과 같다.

- 코드에 고정된 패스워드(Hard-coded Password, KCWE-259)

소프트웨어가 코드 내부에 고정된 패스워드를 포함한 경우, 이를 내부 인증에 사용하거나 외부 컴포넌

트와 통신을 하는 것은 위험하다. 코드 내부에 고정된 패스워드는 시스템 관리자가 찾아낼 수 없는 방식으로 심각한 인증실패를 야기할 수 있다. 일단 취약성이 발견되더라고 수정이 매우 어렵기 때문에, 시스템 관리자는 소프트웨어 시스템 전체를 중지시켜야하는 경우가 발생하기도 한다.

- 취약한 암호화 알고리즘의 사용(KCWE-327)

보안적으로 취약하거나 위험한 암호화 알고리즘을 사용하는 경우 발생하는 오류이다. 표준화되지 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 십억 년이 걸리던 알고리즘이 며칠이나 몇 시간 내에 해독되기도 한다. MD4, MD5, SHA1, DES 알고리즘이 여기에 해당된다.

- 주석문 안에 포함된 패스워드(KCWE-9302)

패스워드를 주석문에 넣어두면 쉽게 찾아서 고치기 어려운 형태로 시스템 보안이 훼손될 수 있다. 만약 소프트웨어 개발자가 보기 위해서 주석문에 패스워드를 넣는 경우, 일단 소프트웨어가 완성된 후에는 이를 제거하는 것은 매우 어렵게 된다. 또한 공격자가 소스코드에 접근할 수 있거나, 혹은 역어셈블러를 사용하여 주석문의 내용을 볼 수 있다면 아주 쉽게 시스템 침입이 가능해진다.

- 적절하지 않은 난수 값의 사용(KCWE-330)

예측 불가능한 숫자가 필요한 상황에서 완전하지 않은 난수를 사용하는 것은 시스템에 취약성을 야기한다. 공격자는 숫자를 예상하여 시스템을 공격하는 것이 가능하다.

- 보안 속성이 결여된 HTTPS 세션으로 보내지는 민감한 내용의 쿠키(KCWE-614)

HTTPS 세션을 통하여 보안에 민감한 쿠키를 전송할 때, 보안 속성을 세팅하지 않는 경우이다. 보안과 관련된 민감한 정보가 쿠키에 담겨있는 경우, 보안 속성을 세팅하지 않고 보내는 쿠키는 공격자에게 단순한 텍스트의 형태로 노출될 수 있다.

- 익명의 LDAP 바인딩(KCWE-9311)

익명 바인딩을 이용하여 LDAP 쿼리를 수행하는 것은 인증 과정이 없기 때문에 효율적일 수는 있지만, LDAP 환경을 공격당하기 쉬운 상태로 만들 수 있다.

3.2.4 Time and State

시간과 상태에 대한 오류란 프로그램의 동작과정에서 시간적 개념을 포함한 개념(프로세스, 혹은 쓰레드 등)이나 시스템 상태에 대한 정보(자원 잠금이나 세션 정보)에 관련된 오류를 말한다. 이러한 오류에 속하는 것들로는 데드락(dead lock)이나 시간을 이용한 비밀 채널(covert timing channel), 세션 고착 등을 들 수 있다.

이와 관련하여 본 데이터베이스에서는 약 20개의 오류를 정의하고 있으며, 이 중에서 C언어와 관련된 오류는 12개, Java와 관련된 오류는 16개를 차지한다. 이런 유형의 오류로 인해서 야기되는 취약점은 소프트웨어 시스템의 가용성(availability)를 크게 약화시킬 수 있으며, 세션과 관련된 취약점은 소스트웨어 시스템의 기밀성과 무결성에도 심각한 영향을 미칠 수 있다. 그렇지만 시간이나 상태에 대한 오류를 포함하고 있는 프로그램은 오류가 실제로 프로그램을 실행시킬 때까지는 쉽게 관찰할 수 없다는 특징을 가지고 있다. 따라서 프로그램 소스분석에서는 위험 요소를 최대한 배제하는 보수적인(conservative) 방법을 채택하고 있다.

대표적인 시간과 상태에 관련된 오류는 다음과 같다.

- 외부에서 제한 없이 접근 가능한 잠금(KCWE-412)

비록 소프트웨어가 잠금(lock)의 존재를 올바르게 확인한다고 해도, 의도했던 제어권보다 더 밖의 어떤 공격자에 의해 잠금이 제어되거나 영향을 받을 수 있으면 위험하다. 이 취약성에 의해 소프트웨어가 관련된 자원을 사용하지 못하거나 잠금의 존재에 따라 제어되는 행동들을 제대로 수행하지 못할 수 있다. 잠금은 배타적인 잠금(exclusive lock)이나 상호 배제를 포함할 수도 있고 잠금으로 간주되는 공유 자원을 조작할 수도 있다. 만일 이런 잠금이 예상치 못하게 계속 붙잡혀있게 되면, 일종의 서비스 거부(DoS) 공격이 가능해지게 된다.

- 중복 검사된 잠금(Double-Checked Locking, KCWE-609)

중복 검사된 잠금(double-checked locking)은 프로그램의 효율성을 높이기 위해 사용하지만, 의도한 대로 동작하지 않는다. 동기화 비용을 줄이기 위해, 프로그래머는 하나의 객체만 할당될 수 있도록 코드를 작성하지만, 자바에서는 객체 참조 주소를 할당하고 생성자를 호출하므로 의도한 객체가 완전하게 초기화되지 않은 상태에서 사용되는 경우가 발생할 수 있다.

- 세션 고착(Session Fixation, KCWE-384)

기존 세션을 무효화하지 않은 채로 사용자를 인증하거나 새로운 사용자 세션을 만들면, 공격자가 인증된 세션과 그 정보를 탈취할 수 있게 된다. 이런 공격 시나리오는 흔히 다음과 같은 경우에 가능하다: (1) 웹 응용프로그램이 기존 세션을 무효화하지 않은 채로 사용자를 인증하여, 사용자와 이미 연관된 세션을 그대로 계속 사용하는 경우, (2) 공격자가 이미 알려진 특정 세션 식별자를 강제할 수 있어서, 일단 사용자가 인증을 받으면, 공격자도 그 인증된 세션에 접근할 수 있는 경우, 그리고 (3) 응용프로그램이나 컨테이너가 쉽게 예측 가능한 세션 식별자를 사용하는 경우가 여기에 해당한다.

세션 고착 취약성을 이용하는 일반적인 방식은, 공격자가 웹 응용프로그램에 새로운 세션을 만들고 관련된 세션 식별자를 기록하는 것이다. 그러면 공격자는 피해자로 하여금 그 식별자를 이용해 서버에 인증하게 만들고, 활성화된 세션을 통해 사용자의 계정에 접근할 수 있게 된다.

3.2.5 에러 처리(Error Handling)

프로그램 수행 중에 발생할 수 있는 에러를 부적절하게 처리할 경우 발생할 수 있는 소프트웨어 취약점을 의미한다. 이러한 소프트웨어의 취약점을 공격자가 인지할 경우, 다양한 조작된 입력 값을 주어 프로그램이 에러를 내도록 하여 다른 공격에서 필요한 프로그램의 내부 작동 구조나 내부 정보를 습득할 수 있으며 제대로 에러 상황을 처리하지 않아 프로그램을 크래쉬 시킬 수도 있다. 본 데이터베이스에서는 약 10여개의 Java관련 취약점을 정의하고 있다. 여기에 속하는 대표적인 취약점은 다음과 같다.

- 액션 없는 오류 조건 탐지(KCWE-390)

프로그램의 수행 도중 에러 조건을 검사하여 적절하게 예외상황을 탐지하나 이를 처리하는 코드는 비어 있는 경우이다. 즉, Java의 try 블록에서 적절한 exception들을 throw하지만 catch 블록에서는 아무런 조치를 취하지 않아 프로그램이 컴파일은 되나 에러 상황에 대한 적절한 조치가 취해지지 않은 경우이다.

- 미검사된 에러 조건(KCWE-391)

예외 및 기타 오류 조건을 검사하지 않고 무시하도록 프로그램을 작성하는 경우이다. 이 경우, 프로그램은 마치 예외상황이 발생하지 않은 것처럼 계속 실행되어 프로그램의 비정상적인 실행 과정에 대해 아무런 흔적이 없이 오작동을 일으키거나 프로그램이 중단되는 경우를 초래할 수 있다.

- NullPointerException을 통한 Null 포인터 사용 포착(KCWE-395)

Null 포인터를 참조를 막기 위해서 NullPointerException을 광범위하게 포착(catch)하도록 프로그램을 작성하는 경우이다. Null 포인터 참조를 방지하기 위해서는 미리 포인터가 Null인지 검사하고 참조하도록 하는 것이 올바른 방법이다. Null 포인터가 참조되는 것은 프로그램에 문제가 존재하는 경우이며 이 문제를 바로 잡는 대신 NullPointerException을 포착하도록 하는 것은 올바른 방법으로 코드를 수정하는 것이 아니다.

- 포괄적 예외를 사용한 catch 선언(KCWE-396)

예외를 포착(catch)할 때 각 예외상황을 구별하지 않고 Exception과 같이 광범위한 예외상황을 한가지로 포착하게 되면 예외상황마다 특별하게 처리해야 하는 경우를 무시하게 될 수 있다. 예를 들어, 프로그램에 새로운 예외상황이 개발과정 후반부에 추가될 경우, 포괄적인 예외로 처리한 프로그램은 이러한 예외상황을 제대로 포착하지 못하고 지나치게 될 수 있다.

- 포괄적 예외를 사용한 throws 선언(KCWE-397)

함수가 발생(throws)시키는 예외상황을 포괄적으로 선언하는 경우, 호출자 함수에서는 발생 가능한 예외들을 제대로 파악하기 힘들게 되며, 예외상황에 따라 각기 다른 에러 처리가 필요한 경우에도 구분하여 에러 처리를 할 수 없게 된다.

- Finally 블록 내에서의 리턴(KCWE-584)

함수의 return을 finally 블록에서 하게 되면 이 함수의 try 블록에서 발생시킨 예외를 제대로 전달하지 못하게 되어 예외상황을 호출자 함수에서 제대로 포착할 수 없게된다. 따라서 finally 블록에서는 할당된 자원만 반환 시키도록 하고 return은 finally 블록 바깥에서 하도록 프로그램을 작성하여야 한다.

3.2.6 코드 품질이 낮음을 표시하는 요소(Indicator of Poor Code Quality)

작성된 프로그램에는 직접적으로 결점이나 보안과 관련된 취약점을 나타내지는 않으나 코드가 충분한 주의를 기울여 개발되고 관리되었는지를 표시하는 요소들이 있다. 프로그램이 좋은 개발과정과 관리과정을 거쳐서 만들어져야 프로그램이 보다 안전하게 작성될 수 있다. 즉, 프로그램 코드가 너무 복잡하여 관리하

기 힘들거나 다른 시스템에 이식하기 힘들도록 되어 있다든지 충분한 주의를 기울여 작성되지 않았음을 나타내는 요소가 있다면 이 프로그램에는 안전성을 위협할 취약점들이 코드 안에 숨겨져 있을 가능성이 높다. 이와 같이 코드의 품질을 가늠할 요소로는 타입 불일치, 자원 관리의 오류, 위험하다고 알려진 함수나 사용이 중지된(obsolete/deprecated) 함수의 사용, 불필요한 연산이나 조건문, 사용되지 않는 변수, Null 포인터 참조, 스택 변수의 주소 리턴, 등과 같은 것을 들 수 있다. 이러한 요소들을 통하여 개발이나 관리 과정에서 충분한 주의를 기울이지 않고 코드가 작성되었음을 판정하게 된다. 본 데이터베이스에는 총 40여개의 Java와 C관련 취약점을 정의하고 있으며 이 중 23개가 Java에 관련되는 취약점이고 27개가 C에 관련되는 취약점이다. 여기에 속하는 중요한 취약점들은 다음과 같다.

- 타입 불일치(KCWE-195, KCWE-9613, KCWE-9614, KCWE-9617)

서로 타입이 호환되지 않는 타입의 변수에 값을 저장하게 되면 C 언어에서는 묵시적으로 음수가 큰 양수 값으로 전환된다든지 정확도가 떨어지는 실수로 전환되는 경우가 존재한다. 이렇게 의도하지 않게 전환된 값이 저장되는 경우 프로그램의 이 후 실행에서 오류를 일으키거나 취약점을 나타낼 수 있게 된다.

- 메모리 누수(KCWE-401)

프로그램 실행 중에 동적으로 메모리를 힙 영역(heap segment)에 할당하는 경우, 마지막으로 사용하고 더 이상 사용하지 않을 때에는 할당된 메모리를 반환해 주어야 한다. 만일 프로그램이 이러한 메모리를 반환하지 않고 실행이 지속되는 경우, 프로그램에서 사용 가능한 메모리가 부족하여 프로그램의 실행이 중단될 수 있다. 이러한 취약점이 외부에 노출되게 되면 공격자가 프로그램을 쉽게 중단시킬 수 있는 취약점이 된다.

- 자원의 부적절한 반환(KCWE-404)

프로그램의 자원, 열린 파일 기술자(open file descriptors), 소켓(socket) 등은 유한한 자원이다. 이러한 자원을 할당 받아 사용한 후에 더 이상 사용하지 않는 다면 적절히 반환하여야 한다. 이러한 자원을 할당만 받고 반환하지 않는 경우, 오랫동안 수행되는 서버 프로그램의 경우, 자원이 모자라서 더 이상 새로운 입력을 처리하지 못하고 프로그램이 중단되는 상태에 빠지게 될 수 있다.

- 메모리 중복 반환(Double Free, KCWE-411)

동적으로 할당되는 메모리 영역은 인접한 주소에 동적 메모리 관리를 위한 메타 데이터를 가지고 있다. 같은 메모리 주소를 중복되게 반환(free) 하게 되면 이 메모리 관리 데이터 구조가 손상되어 이미 할당된 영역이 잘못 반환될 수 있기도 하며, 특정 주소에 원하는 값을 적도록 조작할 수도 있어 버퍼 넘침(overflow) 공격에 취약점이 될 수 있다.

- Null 포인터 참조(KCWE-476)

포인터(pointer)를 참조할 때 그 값이 Null인 경우, 프로그램이 중단되며 종료되거나 예외상황 처리 루틴에서 프로그램의 내부 구조나 작동에 대한 정보를 출력하게 된다. 이러한 취약점이 노출되어 외부 입력으로 제어가 가능하게 되면 공격자에 의해 프로그램의 중단되거나 이 후의 공격을 위한 프로그램 내부 정보를 유출시킬 수 있다.

- 더 이상 지원되지 않는 함수의 사용(Use of Obsolete Functions, KCWE-477)

소프트웨어 패키지의 버전이 증가하면서 이전에 제공되던 API 중에 더 이상 지원되지 않는 함수가 존재할 수 있다. 이러한 함수를 사용하는 것은 소프트웨어의 코드가 최신으로 유지되도록 관리되지 않았음을 의미한다. 이러한 경우 최신 시스템에서 오작동을 일으킬 소지도 있으며 간접적으로는 코드가 오랫동안 정상적으로 유지 보수되지 않아 현재에는 코드의 품질에 문제가 있음을 의미한다.

- 사용되지 않는 코드(Dead Code, Unused Field/Method, KCWE-561, KCWE-9609, KCWE-9610)

코드의 일부분이나 특정 함수 또는 메소드, 혹은 특정 클래스의 필드가 코드에서 전혀 사용되고 있지 않는 경우이다. 프로그램의 코드가 추가로 개발되고 변경되면서 이전에는 사용되던 코드의 일부분이 새 버전의 프로그램에서는 사용되지 않을 수 있다. 이러한 경우에는 불필요한 부분을 코드에서 제거해야 한다. 그러나 코드 개발 및 관리가 소홀하게 수행된 경우 불필요한 코드가 제거되지 않고 그대로 남아 있을 수 있다. 이는 코드의 관리가 적절히 수행되고 있지 않아 전반적으로 코드의 품질 수준이 낮음을 간접적으로 의미한다.

- 중복된 Null 검사(KCWE-9606)

프로그램은 잠재적으로 Null 포인터를 참조할 수 있다. 프로그램의 실행 순서 상으로 Null이 되지 않는다는 가정아래 포인터를 참조하고 나서 이후에 다시

Null 포인터인지 검사하도록 코드가 작성되어 있다면, 뒤에 나오는 Null 검사가 필요 없는 경우이거나 앞의 포인터 참조가 Null인지 확인이 안된 채로 참조를 수행하는 경우이다. 전자는 불필요한 검사 코드가 들어 있는 경우이고 후자는 잠재적으로 프로그램이 중단될 위험이 있는 취약점을 가진 경우이다. 두 경우 모두 코드의 품질이 낮음을 의미한다.

- 쓰레드 조기 종료(Premature Thread Termination, KWCE-9620)

자식 쓰레드가 생성될 때 내부적으로 쓰레드 제어를 위한 자료구조를 생성하게 된다. 이 자료구조의 반환은 일반적으로 부모 쓰레드가 책임지게 되는데 만일 부모 쓰레드가 자식 쓰레드가 끝나기 전에 종료되는 경우, 자식 쓰레드의 자료구조가 반환되지 않게 된다. 오랫동안 수행되는 서버 프로그램에 이러한 취약점이 외부에 노출되는 경우, 메모리 누수를 야기시켜 프로그램이 더 이상 진행되지 않는 상태를 만들 수 있다. 따라서 부모 쓰레드가 부득이 먼저 종료해야 하는 경우에는 자식 쓰레드를 분리(detach)하여 종료될 때 스스로 자료구조를 반환하도록 하여야 한다.

- 포맷 스트링 에러(Format String Error, KCWE-9607, KCWE-9622)

C 프로그램의 printf/scanf와 같은 함수에서 사용하는 포맷 스트링에서 인자의 개수가 다르거나 타입이 맞지 않는 경우, 정해진 범위 외의 메모리를 접근하여 보안상 민감한 정보를 유출시킬 수도 있고 잘못된 동작 혹은 프로그램의 중단을 발생시킬 수 있다.

3.2.7 불충분한 캡슐화(Insufficient Encapsulation)

소프트웨어가 중요한 데이터나 기능을 캡슐화하는 데에 실패할 가능성이 있다. 캡슐화는 데이터와 연산에 대한 강력한 울타리를 치는 것인데, 이것이 실패하면 웹 브라우저 같은 경우에 이동 코드(mobile code)가 다른 이동 코드에 의해서 의도와 다르게 사용될 수 있다. 또한 서버의 경우에도 검증된 데이터와 검증되지 않은 데이터를 구분하지 못하게 되거나, 허용되지 않는 사용자들간의 데이터 누출이 가능해진다. 소프트웨어 보안 측면에서의 캡슐화는 단순히 일반 소프트웨어 개발 방법 상의 상세한 구현 내용을 감추는 일 뿐 아니라 좀 더 넓은 의미로 사용된다. 주로 Java 언어와 관련되며 16개 취약점 중에서 일부를 아래에 설명한다.

- 클래스 이름으로 클래스를 비교하기(KCWE-486)

클래스를 비교할 때 스트링으로 된 클래스 이름을 이용하면, 의도하지 않은 클래스의 객체(코드)를 실행할 수 있다. 공격자가 프로그램이 악성 코드를 실행하게 하기 위해서 클래스 이름을 교묘하게 복제할 수 있다. 따라서 클래스 이름은 좋은 타입 식별자가 아니다. 주어진 객체를 신뢰할 것인지에 대한 근거로 클래스 이름을 사용해서는 안된다. getClass().getName().equals 대신에 getClass()와 == 연산자를 이용하도록 한다.

- 세션 간의 데이터 누출(Data Leak between Sessions, KCWE-488)

멀티쓰레딩 환경에서 세션 간의 데이터 보호를 신경쓰지 않으면 경합 조건(race condition)이 발생할 수 있다. 특히 단일 객체 필드(sigleton object field)에 대한 경합 조건이 쉽게 발생할 수 있다. 다른 세션에서 데이터를 접근할 수 없도록 해야 하는데, 멀티쓰레딩 환경에서 서블릿에 정보를 저장하는 필드를 포함하지 않도록 한다.

- 디버깅용 코드 남겨두기(KCWE-489)

프로그래머가 디버깅 목적으로 삽입된 코드는 개발 후에 삭제되어야 하지만, 남겨지는 경우에 공격자가 사용자 식별 과정을 우회하거나 의도하지 않은 정보와 제어의 누출이 가능하게 한다. 디버깅용 코드인지 아닌지 개발자의 의도를 파악하는 것은 매우 어렵다. 따라서 전형적으로 사용되는 디버깅용 코드를 식별해내도록 해야 한다.

- 공개 메소드가 비밀 배열타입 필드를 반환하기(Private Array-Typed Field Returned From a Public Method, KCWE-495)

공개 메소드가 비밀 배열의 레퍼런스를 반환하면, 의도하지 않은 수정이 가능하다. 비공개인 배열을 공개 메소드가 반환한다면, 그 배열의 레퍼런스가 외부에 공개되어 외부에서 배열의 수정이 가능해진다. 즉, 비밀 배열의 접근 제어가 공개로 변경되는 효과를 준다. 메소드를 비공개로 유지하거나 본래의 배열은 비공개로 유지하면서 복제된 배열을 반환하게 해야 한다. 배열을 수정하는 메소드는 따로 공개 메소드로 정의하여 사용한다.

- 신뢰 경계 위배(Trust Boundary Violation, KCWE-501)

안전한 데이터와 안전하지 않은 데이터를 동일한 데이터 구조에 혼합하면, 프로그래머가 안전하지 않은 데이터를 신뢰할 가능성이 있다. 신뢰 경계(trust boundary)는 프로그램 상에서 안전한 데이터를 생성

하는 구역(신뢰 구역)과 안전하지 않은 데이터를 생성하는 구역(위험 구역)을 구분 짓는다. 데이터 타당성 검사 루틴은 안전하지 않은 데이터가 신뢰 구역으로 이동하는 것을 허용한다. 신뢰 경계 위배는 신뢰 구역과 위험 구역의 경계를 흐리게 하는 것이다. 신뢰 경계를 위배하는 가장 일반적인 실수는 안전한 데이터와 안전하지 않은 데이터를 동일한 데이터 구조에 함께 담는 것이다.

- super.clone을 사용하지 않는 clone 메소드(KCWE-580)

메소드 clone()이 새로운 객체를 만들때, super.clone()을 호출하지 않아서 잘못된 타입의 객체를 생성한다. clone() 메소드를 정의할 때는 반드시 super.clone()을 호출해서 새로운 객체를 얻어야 한다. 그렇지 않으면 하위 클래스의 clone() 메소드가 잘못된 타입의 객체를 반환하게 된다.

3.3 보안 취약점의 중요도 평가

취약점의 위험성을 나타내는 중요도를 계산하여 영향력을 평가하고, 문제 해결에 대한 우선순위 결정을 지원하기 위해 보안 취약점의 중요도 평가체계를 마련하였다. 전자정부 시스템을 구성하는 소스코드에 대한 취약점의 평가 방법을 마련하는데 있어서 중요한 원칙은 합리적인 이론적 근거와 현실적인 적용 가능성으로 안전한 코딩에 관한 측정 요소의 중요도를 적절히 반영하는 것에 초점을 두고 있다. 또한 기존의 CVE, CWE, CAPEC 등의 취약점 평가와의 관련성 및 전자정부 시스템 관련 각종 보안성 측정 방법과 연계를 고려하여 설계되었다.

정보보호란 데이터 또는 시스템에 대한 고의적이거나 실수에 의한 불법적인 공개(노출), 변조, 파괴 및 지체로부터의 보호를 의미한다. 즉, 데이터 및 시스템

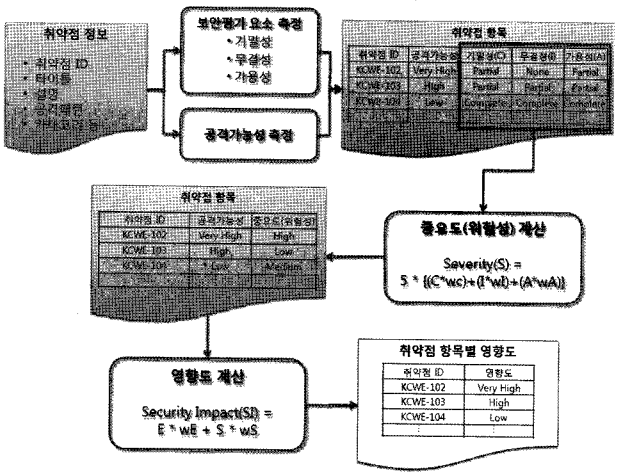

그림 4 보안 취약점의 중요도와 보안영향도 평가 절차

의 기밀성, 무결성, 가용성을 확보하는 것이며, 이들은 보안의 핵심 원칙들로서, 어떤 한 요소가 손상을 받게 되는 경우, 그 조직의 지속적인 존재 여부까지 위협할 수 있다. 따라서 보안 취약점 중요도는 기밀성, 무결성, 가용성을 기준으로 평가가 이루어진다.

기밀성은 권한이 주어지지 않은 비 권한자에게 정보를 공개하고 접근하는 것을 막고, 권한자에게는 주어진 권한에 전용되는 정보 공개와 접근을 허용하는 것을 말한다. 각 취약점에 대한 공격으로 정보가 얼마나 노출되고 공격자에 의해 획득되고 제어될 지에 대한 평가 요소로서 허가 되지 않은 정보의 누출을 방지하는 수준을 측정한다. 무결성은 정보의 신뢰성을 보장할 수 있는 척도를 말한다. 즉, 허가되지 않은 정보의 수정을 방지하는 수준을 측정한다. 가용성은 공격에 대해 시스템 운영이 지속되는 수준에 대한 평가이다.

기밀성, 무결성, 가용성 평가 메트릭은 3점 척도로서, Complete=1, Partial=0.7, None=0 값으로 정의된다. 각 기준에 대한 가중치 산정은 보안 영향력의 비중을 표현하기 위해 3가지 기준이 모두 균등한 경우 0.333으로, 각 기준에 비중을 둔 경우는 비중을 둔 요소의 가중치는 0.5, 나머지 두 요소의 가중치는 0.25로 할당하였다.

보안 취약점 항목의 중요도는 보안에 영향을 주게 되는 위험성(Severity)을 나타내며, 다음과 같이 S에 의해 계산된다. 계산된 S값의 범위는 0과 5 사이의 값으로 각 취약점 항목은 값에 따라 VeryHigh, High, Medium, Low, VeryLow의 중요도를 갖게 된다.

$$S = 5*(C*w_C)+(I*w_I)+(A*w_A)$$

이렇게 계산된 중요도와 공격가능성 메트릭을 이용하여, 각 취약점들이 갖는 보안 영향력 지표 SI(Security Index)를 정의하였다. 취약점 항목별 보안 영향도는 소스코드 취약점의 보안 위험 관리에 대한 지표로 사용될 수 있다. 소스코드 취약점 항목별 보안 영향도 값이 큰 순서로 관리해야 하며, 평가 대상 프로그램에서 검출된 취약점들 가운데 영향도가 높은 취약점들이 존재한다면, 전체 프로그램의 보안 위험 수준에 위협요소가 될 가능성이 높아지게 될 것이다.

4. 결 론

컴퓨터 시스템의 보안 취약점을 근본적으로 해결하기 위해서는 해당 시스템을 구성하는 소프트웨어 소스코드의 취약점 문제가 반드시 해결되어야 한다. 최근 들어 웹을 통한 서비스의 비중이 증가하면서 응용 프로그램에 대한 불특정 다수의 직접적인 접근이 더

욱 확대되고 있어, 소스코드 취약점 문제에 대한 관심과 노력이 더욱 증가하고 있다. 이에 다양한 관점에서 소프트웨어의 취약점에 대한 최신의 정보를 구축하고 그 인식을 확대하고자 하는 노력들이 진행되고 있으며, 많은 산업체에서 소프트웨어 취약점 감소를 위한 개발 도구들을 개발하거나 채택하고 있다. 또한 국내적으로도 컴퓨터 시스템의 보안과 안정성이 중요한 산업체 및 정부 기관 등을 중심으로 소프트웨어에 대한 취약점 검사를 필수 절차화 하는 추세이다.

본 글에서는 이러한 노력의 일환으로 수행된 소프트웨어 취약점 데이터베이스 구축 사례를 소개하고 아울러 관련된 해외 연구 동향을 기술하였다. 구축된 데이터베이스의 취약점 정보는 C 및 Java 소스코드 관점에서의 주요 취약점을 망라하고 있으며, 차후 계속 확장될 예정이다. 구축된 정보는 함께 개발된 소스코드 보안취약점 자동진단 시스템의 입력 정보로 사용하여 행전안전부 발주 소프트웨어에 대한 취약점 검사에 활용할 예정이며, 아울러 관련된 교육과 정보공개를 통하여 소프트웨어 취약점에 대한 인식을 제고하고 개발자들에게 실질적인 정보를 제공할 것이다.

참고문헌

[1] Gartner, "Now is the time for security at Application Level", 2006, 12.

[2] CWSS - Common Weakness Scoring System, http://cwe.mitre.org/cwss/

[3] Common Vulnerabilities and Exposures, http://cve.mitre.org

[4] Common Vulnerability Scoring System, http://www.first.org/cvss/

[5] CAPEC : Comon Attack Pattern Enumeration and Classification, http://capec.mitre.org/

[6] 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, http://cwe.mitre.org/top25/

[7] OWASP Top 10 Project, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[8] National Vulnerability Database, http://nvd.nist.gov.

[9] Common Weakness Enumeration, http://cwe.mitre.org

김 유 경

1991 숙명여자대학교 수학과(학사)
1994 숙명여자대학교 전산학과(석사)
2001 숙명여자대학교 컴퓨터과학과(박사)
2001~2005 숙명여자대학교 정보과학부 컴퓨터과학전공 초빙교수
2005~2006 University of California Davis, 박사후연구원
2006~현재 한양대학교 공학대학 컴퓨터공학전공 연구교수
관심분야 : 소프트웨어 품질평가, 웹서비스 신뢰성 평가, SOA 모델링
E-mail : yukyong@hanyang.ac.kr

신 승 철

1987 인하대학교 전산학과(학사)
1989 인하대학교 전산학과(석사)
1996 인하대학교 전산학과(박사)
1999~2000 Kansas State University 박사후연구원
1996~2006 동양대학교 컴퓨터공학부 교수
2006~현재 한국기술교육대학교 컴퓨터공학부 교수
관심분야 : 프로그램 분석 및 검증, 소프트웨어 보안, 수리논리
E-mail : scshin@kut.ac.kr

안 준 선

1992 서울대학교 계산통계학과(학사)
1994 KAIST 전산학과(석사)
2000 KAIST 전산학과(박사)
2000~2001 KAIST 프로그램분석시스템연구단 박사후연구원
2001~현재 한국항공대학교 항공전자 및 정보통신공학부 교수
관심분야 : 프로그램 분석, 소프트웨어 보안, 유비쿼터스 컴퓨팅, 프로그램 병렬화
E-mail : jsahn@mail.hangkong.ac.kr

이 욱 세

1995 KAIST 전산학과(학사)
1997 KAIST 전산학과(석사)
2003 KAIST 전산학과(박사)
2003~2004 서울대학교 컴퓨터공학과 박사후연구원
2004~현재 한양대학교 컴퓨터공학과 교수
관심분야 : 프로그램 분석, 포인터 분석, 프로그램 검증, 타입 시스템
E-mail : oukseh@hanyang.ac.kr

이 은 영

1996 고려대학교 전산과학과(학사)
1998 고려대학교 전산과학과(석사)
2004 Princeton University,U.S.A.(박사)
2005~현재 동덕여자대학교 컴퓨터학과 조교수
관심분야 : 프로그래밍언어, 소프트웨어 보안, 컴파일러
E-mail : elee@dongduk.ac.kr

한 환 수

1993 서울대학교 컴퓨터공학과(학사)
1995 서울대학교 컴퓨터공학과(석사)
2001 University of Maryland, Computer Science(박사)
2001~2002 Intel, Senior Engineer
2003~2008 KAIST 전산학과 교수
2008~현재 성균관대학교 정보통신공학부 교수
관심분야 : 컴파일러, 컴퓨터구조, 멀티코어 컴퓨팅
E-mail : hwansoo.han@gmail.com