소프트웨어 보안을 위한 시큐어 코딩
Secure Coding for Software Security

김정숙(삼육대학교)

차 례

1. 서론
2. 안전(secure)과 보안(security)
3. 시큐어 코딩 표준 가이드
4. 소프트웨어 보안 진단도구
5. 결론

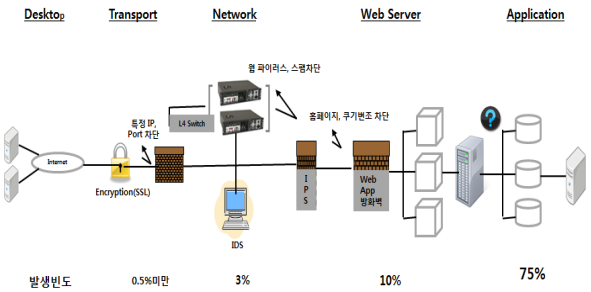
1. 서론

오늘날의 소프트웨어는 인터넷 환경에서 데이터를 교환하기 때문에 해커에 의해 악의적인 공격을 받을 가능성이 항상 존재한다. 이러한 보안 약점은 심각한 경제적 손실을 발생시키는 소프트웨어 보안 침해사고의 직접적인 원인이 될 수 있다. 최근, 이러한 보안약점을 해소하기 위해 외부환경에 대한 보안시스템을 강화하는 것보다 프로그래머가 견고한 소프트웨어를 개발하는 것이 보안 수준을 향상시킬 수 있는 본질적이고 가장 효과적인 방법이라는 인식이 늘고 있다[1][2][3].

이러한 이유로 코딩단계에서부터 소프트웨어에 대한 취약점을 해소하기 위한 코딩안내서를 제시하는 것이 추세이다. 따라서 소프트웨어 개발단계에서부터 취약점을 고려하고 차단하면, 운영단계에서 취약점을 인지하고 수정하는 노력에 비해 막대한 비용을 절감할 수 있을 뿐만 아니라 해커로부터 안전한 소프트웨어를 개발하는데 큰 기여를 할 수 있을 것이다. 국외의 CWE(Common Weakness Enumeration)[4]와 CERT(Computer Emergency Response Team)[5]등의 기관에서는 취약점 목록을 도출하고 안전한 코딩 가이드를 제시하기 위한 연구를 진행하고 있다. 또한, 국내·외 주요 소프트웨어 개발 기업들도 내부적인 코딩 가이드를 통해 보다 양질의 소프트웨어 개발을 위한 노력을 기울이고 있다.

2. 안전과 보안

현재까지도 우리는 소프트웨어 보안 침해사고의 예방을 위한 보안시스템은 네트워크 방화벽, 사용자 인증 시스템 등이 대부분이지만, 가트너의 보고서에 의하면 그림 1과 같이 소프트웨어 보안침해사고의 75%는 취약점을 내포하는 응용프로그램에 의해 발생되었다고 알려져 있다.



▶▶ 그림 1. 보안 침해사고 발생 빈도

또한, 개발 완료 후, 취약점을 보완하기 위한 비용이 매우 크기 때문에 개발 단계에서부터 프로그램의 안전성을 고려해야 한다. 따라서 외부 환경에 대한 보안시스템을 견고히 하는 것보다 프로그래머가 견고한 소프트웨어를 개발하는 것이 보안 수준을 향상시킬 수 있는 본질적이고 가장 효과적인 방법이라고 할 수 있다.

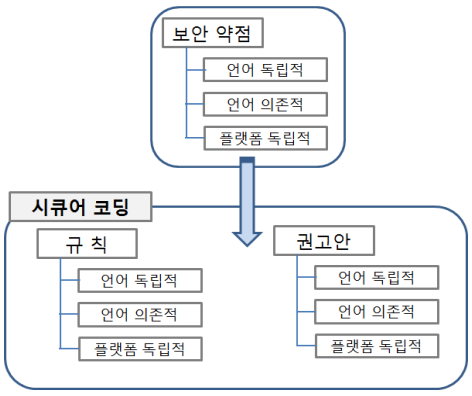
프로그래머는 자신의 프로그램 내의 취약점이 완전히 제거되어 안전한(secure) 프로그램이기를 원한다. 하지만 취약점 항목에 대한 전문적인 지식을 습득하기 어렵고 취약점을 어떻게 수정해야 하는지 인지하는데 어려움이 있기 때문에 소스코드 레벨에서 취약점을 자동 분석할 수 있는 도구가 필요하다.

취약점을 분석하는 기법은 취약 항목별로 적합한 분석

방법이 존재하며, 크게 정적 분석 기법과 동적 분석 기법으로 구분된다. 정적 분석 기법은 프로그램의 실행없이 분석하는 기술로 토큰, 추상구문트리(AST), 제어흐름 그래프(CFG), 자료흐름 그래프(DFG) 등을 이용한다. 동적 분석 기법은 프로그램을 실행하면서 단계적으로 분석하는 기술로, 실행시간에 활용 가능한 특정 코드를 삽입하거나 라이브러리 매핑 등의 방법을 통해 분석한다.

표 1. 취약점 분석 도구

도구	분석 방법
MOPS[7]	. 버클리 대학교에서 개발한 모델 검사기 . 보안취약요소를 프로퍼티로 정의하고, 유한오토마타를 이용하여 정형화 . 적은 분석비용으로 모델링된 취약점을 모두 검사가능 . 자료흐름분석을 하지 않음으로 인한 한계가 존재
Fortify 360[8]	. Fortify에서 개발한 취약점 탐지 도구 . C/C++, Java 등 12개의 언어 지원 . 정적&동적 분석 기법 모두 사용한 취약점 분석 도구 . 발견된 취약자료는 통계자료와 함께 사용자께 제공
Coverity Prevent [9]	. 소스코드에 대한 정적 분석 도구 . 전체 코드에서 발견된 취약점을 목록으로 표시 . 각각의 목록은 취약점이 발생한 소스코드 내의 위치와 취약점이 발생된 원인을 포함함.
Findburgs[10]	. 메릴랜드 대학에서 개발한 정적 분석 도구 . JRE에서 동작되며, 이클립스 플러그인으로 개발환경에서 사용 가능하고 Java 프로그램 분석 도구 . GNU LGPL하에서 무료로 사용할 수 있는 공개 SW . 미리 작성되어 있는 버그패턴을 기반으로 Java p/g을 바이트코드 레벨에서 분석하기 때문에 소스 불필요 . 버그패턴은 확장이 가능하므로 보안약점이 추가될시 쉽게 해당항목을 검사 가능 . 현재 가지고 있는 버그패턴은 401개
PMD[11]	. 공개 프로젝트로 진행된 보안약점 분석 도구 . 이클립스나 JBuilder, Netbeans등 다양한 개발도구에서 플러그인으로 사용 가능 . Java 프로그램 분석 도구 . 개선된 버전에서 jps, xsl, ecmascript, xml 분석 가능 . BSD 라이센스를 가지고 있는 공개 SW . 미리 정해진 규칙을 이용하여 프로그램 소스코드 기반으로 분석하여 possible bugs, dead code, sub-optimal code, overcomplicated expression, duplicate code 등을 검사하며 확장 가능 . 현재 가지고 있는 규칙은 271개



▶▶ 그림 2. 보안약점과 시큐어 코딩과의 상관관계

도출된 보안 약점은 기준에 따라 언어 독립적인 것과 언어 의존적인 형태 및 플랫폼 의존적인 취약점 등으로 구분될 수 있다. 그림 2는 보안 약점과 시큐어 코딩 규칙(SCR: Secure Coding Rule), 시큐어 코딩 권고(SCG: Secure Coding Guideline)의 상관관계를 표현한 것이다.

도출된 보안 약점을 기반으로 프로그래머가 반드시 지켜야 할 시큐어 코딩 규칙을 제시할 수 있으며, 또한 안전한 프로그램을 작성하기 위한 시큐어 코딩 권고안을 유도할 수 있다. 따라서, 만약 프로그래머가 시큐어 코딩 규칙과 시큐어 코딩 권고안을 만족하는 프로그램을 개발하면, 보안 약점을 대부분 제거할 수 있을 것이다.

3. 시큐어코딩 표준 가이드

3.1 시큐어 코딩

프로그래밍 패러다임은 순차 프로그래밍으로부터 구조적 프로그래밍을 거쳐 객체지향 프로그래밍으로 발전하였다. 이와 함께, 프로그래밍 철학은 정확한 입력이 주어졌을 경우 정확한 결과를 도출할 수 있는 정확한 프로그래밍에서 외부 요인으로부터 견고한 신뢰성 높은 프로그래밍으로 발전하였다. 최근에는 대부분의 응용 프로그램이 네트워크로 연결된 환경에서 동작함에 따라, 보안 침해사고를 예방하기 위해 해커로부터 안전한 프로그램을 작성하는 시큐어 코딩(Secure Coding)[2][3]의 중요성이 크게 부각되고 있다.

최근에는 보안 침해사고 발생빈도가 응용프로그램에 의한 경우가 가장 높다는 문제점을 인식하고 개발단계에서부터 해커로부터 안전한 코드를 작성하는 시큐어 코딩에 대한 연구가 활발히 이루어지고 있다. 특히, 프로그래밍 언어에서 발생할 수 있는 취약점을 분석하는 기관인 CWE에서는 소스코드 작성 단계에서 발생할 수 있는 다양한 취약점을 언어별로 분석하여 명시하고 있다. 또한, CERT는 안전한 소스 코드를 작성하기 위한 시큐어 코딩 규칙을 정의하고 있다. Cigital[6]에서는 Katrina Tsipenyuk, Brian Chess, Gray McGraw가 제안한 Seven Pernicious Kingdoms[16] 분류 방법에 따라 취약점을 제거하기 위한 61개의 규약을 분류하였다. Cigital에서 제시한 코딩 규약은 XML 형태로 정의되어 있으며, 취약점 분석기 등의 프로그램에서 입력으로 사용할 수 있다. 소프트웨어의 결함으로 인해 치명적인 문제가 발생할 수 있는 항공기, 자동차 등의 산업에서는 이

미 JSF(Joint Strike Fighter), MISRA Coding Rule 등의 코딩 규약을 도입하여 양질의 소프트웨어 개발을 위한 지속적인 노력을 기울이고 있다.

현재 국외에서는 이러한 소프트웨어의 보안을 위한 취약점 목록, 안전한 코딩 규칙, 분석도구, 체계를 확립하기 위해 많은 투자를 아끼지 않고 있으며 이와 같은 기술은 향후 IT, 소프트웨어 개발, 보안 산업에 핵심기술로 자리잡게 될 것이다.

표 2. 안전한 코딩의 분류 사례

분류방법	분류 방법
사전식	. 취약점을 사전식으로 분류 . 안전한 코딩 및 취약성을 사전식으로 분류하여, 검색과 활용을 수월하도록 함. . 안전한 코딩 사전식 분류 사례: CWE(Common Weakness Enumeration) . http://cwe.mitre.org
개념별	. 취약점을 프로그래밍 개념을 기준으로 분류 . 해당 개념을 프로그램에 안전하게 구현하거나 활용하는 용도로 사용 . 안전한 코딩의 개념별 분류 사례: CWE/SANS Top 25 Most Dangerous Programming Errors http://www.sans.org/top25-programming-errors/ 3개의 카테고리로 구성 - Insecure Interaction Between Components, - Risky Resource Management, - Porous Defenses . Seven Pernicious Kingdom http://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf 7(+1)개의 카테고리로 구성 - Input validation and representation - API abuse - Security features - Time and state - Errors - Code quality - Encapsulation - Environment
언어별	. 언어별 특징을 기준으로 분류 . 각 언어의 사용자 및 학습자가 수월하게 접근토록 함 . 안전한 코딩의 언어별 분류 사례: The CERT Sun Microsystems Secure Coding Standard for Java http://www.securecoding.cert.org/confluence/display/java/ . 자바 관련 CERT 분류 14개의 카테고리로 구성 00. Security(SEC) 01. Declarations and Initialization(DCL) 02. Expressions(EXP) 03. Scope(CSP) 04. Integers(INT) 05. Floating Point(FLP) 06. Object Orientation(OB) 07.Input Output(FIO) 08. Concurrency(CON) 09.Methods(MET) 11. Serialization(SER) 49.Micelianeous(MSC) 99. The Void

3.2 표준 가이드

안전한 코딩의 기본 개념은 표준 코딩 규칙과 코딩 가이드를 조화롭게 적용함으로써 전체 취약점 영역을 커버하는 것이다. 즉, 설계자가 의도한 행동과 코딩자의 실제 행동이 일치해야 한다는 것이다.

컴퓨터 정보 시스템의 취약성은 75~78%가 응용프로그램으로부터 기인하기 때문에 응용 프로그램을 안전하게 개발하기 위한 효과적인 방법론은 반드시 필요하다. 또 개발 완료 후, 취약성을 인지하고 수정·보완하는 운용비용이 매우 크기 때문에 개발 단계에서부터 프로그램의 안전성을 고려하면 막대한 비용을 절감할 수 있을 뿐만 아니라 해커로부터 안전한 소프트웨어를 개발하는데 큰 기여를 할 수 있다. 이러한 안전한 코딩의 필요성으로 인하여 분야별 코딩 규칙이 만들어졌다.

MISRA(Motor Industry Software Reliability Association) 코딩 규칙은 차량용 소프트웨어 신뢰성 향상을 위한 코딩 표준을 제안한 것이다.

JSF(Joint Strike Fighter) 코딩 규칙은 미영 항공기 소프트웨어 신뢰성 향상을 위한 코딩 표준이다.

HIC/HICPP(High Integrity C/C++: Compliance Module)는 Programming Research사에서 제공하는 일반적인 코딩 표준이다.

4. 소프트웨어 보안 진단도구[12]

보안 약점이 내포된 소프트웨어는 해커의 공격 목표가 되어 심각한 보안 위협이 초래된다. 더구나 사이버 심해사고의 약 75%가 응용 프로그램(sw)의 취약점을 악용한 것이라는 가트너 보고가 있다. 또한 IBM의 보고서에 의하면, 운영단계에서의 취약점 제거 비용은 개발단계보다 60~100배의 비용 소요가 이루어진다고 한다. 이는 정보시스템 운영 이전인 개발단계부터 보안성 고려 및 잔존 취약점 제거는 반드시 필요하다는 것을 뒷받침하고 있다. 이에 우리 정부에서는 사전 예방체계 강화를 위하여 정보시스템 개발단계부터 보안성 고려 및 SW 보안 약점 진단·제거를 강화하기 위해 다음과 같은 과정을 통한 SW 개발보안(시큐어 코딩) 강화체계의 도입하고 있다.

정보시스템 구축·운영과정에서 SW 보안 약점으로 분석된 유형별 사례는 표 3과 같다.

KISA에서는 자체 진단 개발도구 SSEN을 개발하여 그림 4와 같은 절차로 Java, C, C++등 11개 언어에 대한 보안약점 진단과 웹기반의 진단결과 관리 서비스를 지원하고 있다.

▶▶ 그림 3. SW 개발보안 과정도

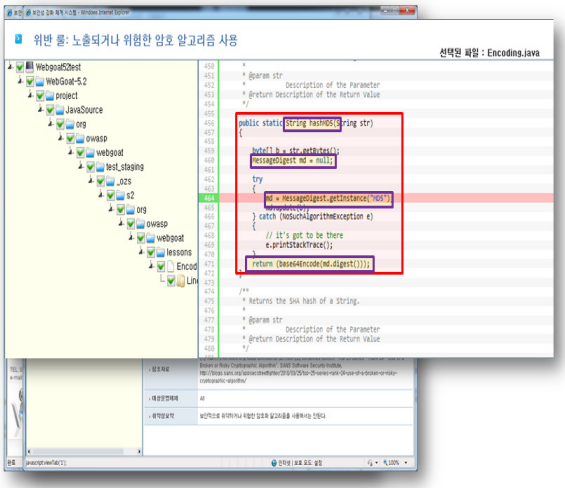
표 3. 시스템 구축·운영과정에서의 SW 보안 약점

유형	주요내용	개수 (43)
입력 데이터 검증 및 표현	프로그램 입력 값에 대한 부적절한 검증 등으로 인해 발생할 수 있는 보안약점 (예) SQL 삽입, 자원 삽입, 크로스사이트스크립트 등	14
보안기능	인증, 접근제어, 권한 관리 등을 적절하지 않게 구현시 발생할 수 있는 보안약점 (예) 부적절한 인가, 중요정보 평문저장, 하드코드된 패스워드 등	16
시간 및 상태	멀티프로세스 동작환경에서 부적절한 시간 및 상태 관리로 발생할 수 있는 보안약점 (예) 경쟁조건(TOCTOU), 제어문을 사용하지 않는 재귀함수 등	2
에러처리	불충분한 에러 처리로 중요정보가 에러정보에 포함되어 발생할 수 있는 보안약점 (예) 오류상황 대응 부재, 오류메시지를 통한 정보노출 등	3
코드오류	개발자가 범할 수 있는 코딩오류로 인해 유발되는 보안약점 (예) 널 포인터 역참조, 부적절한 자원 해제 등	2
캡슐화	불충분한 캡슐화로 인가되지 않은 사용자에게 데이터가 노출될 수 있는 보안약점 (예) 제거되지 않고 남은 디버그 코드, 시스템 데이터 정보노출 등	5
API 오용	부적절하거나, 보안에 취약한 API 사용으로 발생할 수 있는 보안약점 (예) DNS lookup에 의존한 보안결정 등	1

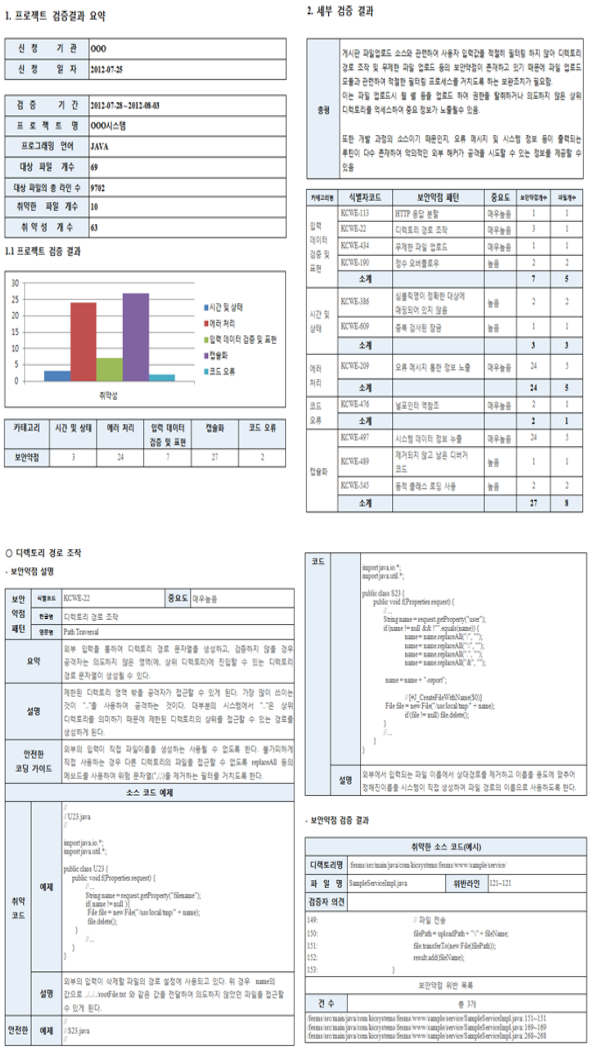
※ 국외사례(CWE, SANS Top25, OWASP Top10 등) 참조

▶▶ 그림 4. 소스코드 보안약점 진단 과정도

그림 5, 그림 6은 SSEN의 진단 및 보고서 예시이다.



▶▶ 그림 5. SSEN의 진단 예시



▶▶ 그림 6. SSEN의 진단 보고서 예시

5. 결론

최근까지 대부분의 소프트웨어 보안 문제는 방화벽, 사용자 인증과 같은 추가적인 도구에 대한 연구를 중심으로 이루어졌다. 그러나 소프트웨어 보안 침해 사고에서 이러한 영역이 차지하는 비중은 25%에 불과하다는 보고가 있다[5]. 오히려 보안 침해 사고의 75%는 취약점을 내포하는 소스 코드에 의해 발생되었으며, 본질적으로 소프트웨어를 개발하는 프로그래머가 견고한 코드를 작성하는 것이 보안 수준을 높일 수 있는 가장 효과적인 방법이 될 수 있음을 뒷받침하고 있다.

코딩 단계에서부터 소프트웨어에 대한 취약점을 고려하여 제거하면, 운용단계에서 취약점을 인지하고 수정하는 노력과 비교하여 막대한 비용을 절감할 수 있다. 또한

해커로부터 소프트웨어를 보호하여 안전하게 개발하는데 기여할 수 있을 것이다. 최근의 컴퓨팅 환경은 대형 컴퓨터에서 퍼스널 컴퓨터로, Post PC에서 스마트폰, 타블렛 PC, 디지털 TV(DTV) 등으로 변화되면서 각광받는 시대가 되었다. 이러한 상황에 직면하면서, 대부분의 소프트웨어는 인터넷 환경에서 데이터를 교환하기 때문에 데이터에 대한 신뢰성 보장이 매우 어렵다.

본 논문에서는 부가적인 보안 시스템을 통해 침해 사고를 방어하는 것보다, 안전한 소프트웨어 개발을 위한 시큐어 코딩에 대한 중요성을 인식하고 시큐어 코딩 가이드 및 보안 진단방법에 대하여 소개하였다.

참고문헌

[1] 손윤식, 오세만, "모바일 애플리케이션을 위한 보안약점 구조화 기법에 대한 연구", 멀티미디어학회논문지 제15권 제11호 pp. 1349-1357, 2012.

[2] Gary McGraw, "Software Security: Building Security In", Addison-Wesley, Boston, Masachusetts, 2006.

[3] John Viega, Gray Magraw, "Software Security: How to Avoid Security Problems the Right Way", Addison-Wesley, Boston, Masachusetts, 2006.

[4] Common Weakness Enumeration(CWE): A community Developed Dictionary of Software Weakness Types, http://cwe.mitre.or/, 2009.

[5] J. McManus and D. Mohindra, The CERT Sun Microsystems Secure Coding Standard for java, http://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=34669015/, 2009.

[6] Cigital, Cigital Java Security Rulepack, http://www.cigital.com/securitypack/ view/index.html, 2009.

[7] H. Chen and D. Wagner, "MOPS: an Infrastructure for Examining Security Properties of Software", Proc. of the 9th ACM Conference on Computer and Communications Security, pp.235-244, 2002.

[8] Fortify Software Inc., Fortify Source Code Analysis (SCA), http://www.fortify.com/products/sca, 2009.

[9] Coverity, Inc., Coverity Static Analysis, http://www.coverity.com/products/static- analysis.html, 2009.

[10] FindBugs, http://findburgs.sourceforge.net/, 2012.

[11] PMD, http://pmd.sourceforge.net/pmd-5.0.0/, 2012.

[12] 강필용, "SW 보안약점 진단도구 적용사례: SSEN", 시큐어코딩을 위한 공개분석도구 튜토리얼, 2012. 11.

저자소개

● 김 정 숙(Jung-Sook Kim) 정회원

- 1984년 2월 : 광운대학교 전자계산과(이학사)
- 1999년 2월 : 동국대학교 컴퓨터공학과(공학박사)
- 2000년 3월 : 김포대학 컴퓨터계열 교수
- 2001년 3월 ~ 현재 : 삼육대학교 컴퓨터학부 교수

<관심분야> : IT 컨버전스, 모바일 컴퓨팅, 임베디드시스템, 웹프로그래밍, 프로그래밍언어, 컴파일러 등