7

자바 시큐어 코딩

동국대학교 ▌손윤식* · 오세만**

1. 서 론

오늘날의 소프트웨어는 크게 네트워크 연결이 필요하지 않은 소프트웨어(stand-alone S/W)와 인터넷환경에서 실행되는 소프트웨어로 구분할 수 있다. 전자는 프로그램의 정확성 및 효율성을 요구하지만, 후자는 그 이외에 안전성을 요구한다. 즉, 인터넷환경에서 실행되는 소프트웨어는 프로그램 및 데이터를 네트워크를 통해 교환하므로 임의의 침입자에게 악의적인 공격을 받을 가능성이 있기 때문이다. 예를 들어, SQL 삽입공격, 악의적인 입출력 데이터, 운영체제 명령어 삽입공격 등은 최근의 해킹 문제에 있어서 주요한 공격 방법에 해당된다. 이러한 공격에 대한 취약성은 소프트웨어 보안 침해사고의 직접적인 원인이 되고 있으며 심각한 경제적 손실을 발생시킨다.

침해 사고의 예방을 위한 보안시스템은 네트워크 방화벽, 사용자 인증시스템 등이 대부분이지만 가트너의 보고서[참조사이트 4]에 의하면 소프트웨어 보안 침해사고의 75%는 취약성을 내포하는 응용프로그램에 의해 발생되고 있다. 따라서 외부환경에 대한 보안시스템을 견고히 하는 것보다 프로그래머가 견고한 프로그램 코드를 작성하는 것이 보안 수준을 향상시킬 수 있는 본질적이고 가장 효과적인 방법이다.

현재 많은 회사와 연구기관에서 안전한 소프트웨어를 개발하기 위한 코딩 규약과 취약성 분석 도구에 대한 개발이 이루어지고 있다. 이러한 연구와 개발이 프로그래밍 언어적인 측면에서 활발하게 진행되고 있으며, 본 고에서는 자바언어를 이용하여 안전하게 프로그램을 작성하기 위한 연구 및 동향에 대해에 소개하고자 한다. 본 고의 구성은 다음과 같다. 먼저, 자바의 언어적인 특징과 시큐어 코딩 관점에서 자바의 특징을 소개하고, 다음으로, 시큐어 코딩의 필요성과 특징들을 자바 관점에서 살펴본다. 또한, 가장 위험한 프로그래밍 오류를 규정하고 있는 Top 25에 관하여 소개하고, Top 25의 사례를 자바 중심으로 살펴본다.

* 학생회원
** 종신회원

2. 시큐어 코딩 관점에서 본 자바 소개

2.1 자바란?

자바는 Sun사에서 개발한 범용 객체지향 프로그래밍 언어로 인터넷 환경에 적합하게 설계되었으며, 신뢰성을 높이기 위한 언어구조 및 보안기능을 제공하는 특징을 가진다. 또한, 플랫폼 독립적인 언어로 기존의 프로그램이 네트워크를 통해 데이터를 전송하고 이를 처리하는데 그쳤던 반면, 자바는 프로그램의 전송이 가능하고 JVM 상에서 운영체제와 독립적으로 실행될 수 있다.

인터넷의 발달로 인해 자바는 현재 가장 널리 쓰이는 프로그래밍 언어로 자리잡고 있다. 2009년 TIOBE사의 조사에 따르면, 자바의 사용률은 표 1에서 보여주듯이 19.34%이다[참조사이트 9].

2.2 보안 강화를 위한 자바 기능

자바는 언어적인 측면과 실행 시스템 측면에서 보안 강화를 위한 기능을 제공한다. 먼저 언어적인 측면에서 C/C++ 등 다른 언어에 비해 안전하게 설계되었고, 다음과 같은 특징으로 요약된다.

- 무한연산(Infinite Arithmetic)과 같이 잘 정의된 수치적 연산을 제공하며,
- 명시적으로 포인터를 사용하지 않고
- 플랫폼 독립적인 실행이 가능하다.

또한, 잘 정의된 예외 검사 및 처리기능을 언어 수준에서 지원하며, 예외 클래스를 제공하여 예외를 객

표 1 세계에서 각 프로그래밍 언어의 사용률(2009.4 기준)

No.	Programming Language	Ratings Apr. 2009
1	Java	19.34%
2	C	15.47%
3	C++	10.74%
4	PHP	9.89%
5	(Visual) Basic	9.10%

체로 처리하고 있다. 그리고, 배열 범위 검사와 널 포인터 검사와 같은 미리 정의된 다양한 예외를 시스템 레벨에서 제공하고 있다.

다음으로, 자바 가상기계에서는 네트워크 상의 클래스파일을 실행하기 전에 보안검사를 수행한다. 예를 들어, 바이트코드 검증기(Bytecode Verifier)와 클래스 로더(Class Loader)를 통하여 다음과 같은 검사를 수행한다.

· 바이트코드 검증기

바이트코드 명령어에 대한 분석 수행하는데, 주로 접근 제한 규칙 준수 여부와 타입 일치 여부에 대한 검사를 수행한다.

· 클래스 로더

클래스파일을 실행하기 위해 JVM으로 로딩할 때, 네트워크상에 존재하는 클래스파일과 로컬 클래스파일에 대한 분석을 수행한다.

다음 그림 1은 자바 프로그램의 실행 흐름을 나타내며, 위에서 설명한 바이트코드 검증기와 클래스 로더의 위치와 기능을 보여준다.

자바의 실행 시스템에서 보안 강화를 위해 제공한 보안 관리자(Security Manager)는 리소스를 제어하는 모듈이다. 이것은 개인 파일, 소켓, 그리고 중요한 리소스 접근에 대한 보안 메커니즘을 제공한다(그림 2).

위에서 설명한 내용과 같이 자바가 언어적인 측면과 실행 시스템 측면에서 기존의 언어에 비해서 보안 강화 기능을 제공하고 있지만, 이러한 기능을 잘못 사용하거나 불완전하게 구현할 경우에 소프트웨어의 취

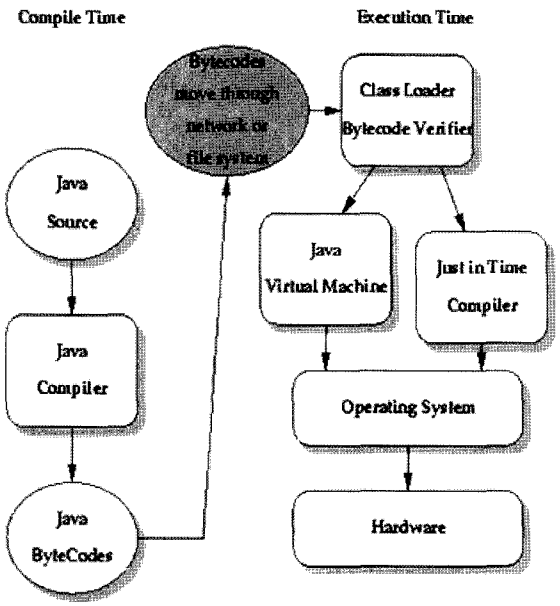

그림 1 자바 프로그램의 실행 흐름

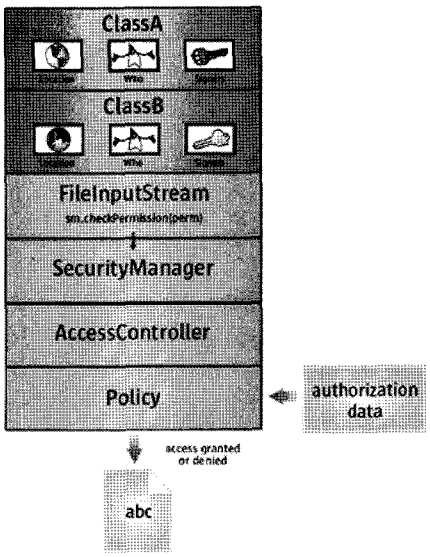

그림 2 보안 관리자

약성을 야기할 수 있으며, 해커로부터 공격 대상이 될 수 있다.

3. 자바 관점에서 본 시큐어 코딩 소개

3.1 시큐어 코딩의 기본 개념

컴퓨팅 기술의 발전과 함께 프로그래밍의 패러다임은 변화하였다. 초창기 컴퓨팅 환경에서 중요시된 알고리즘의 표현능력을 높이기 위한 방법으로 제시된 순차적 프로그래밍 기법은 프로그램의 복잡도와 크기가 날로 커짐에 따라 구조적 프로그래밍 방식으로 변화하였고, 객체 사이의 메시지 전달을 통한 프로그래밍 방식인 객체지향 프로그래밍 방식으로 발전하였다. 그림 3은 이러한 프로그래밍 패러다임의 발전 과정을 나타내고 있다.

한편으로 프로그래머의 근원적인 프로그래밍 철학 관점에서는 그림 4와 같이 정확한 입력에 대한 정확한 출력과 효율성을 요구하는 정확한 프로그래밍에

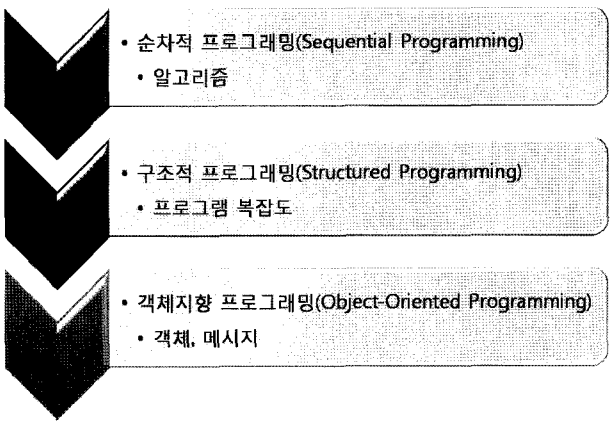

그림 3 프로그래밍 패러다임의 발전 과정

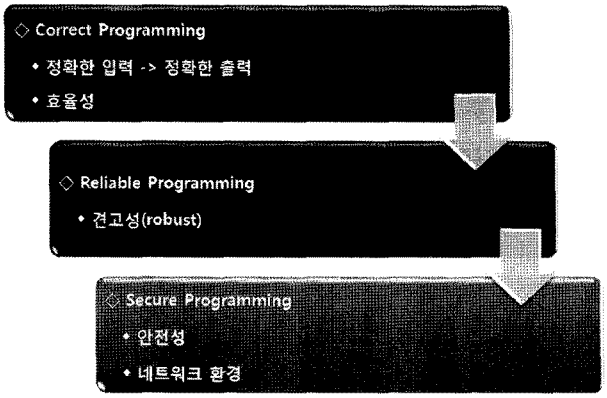

그림 4 프로그래밍 철학의 변화

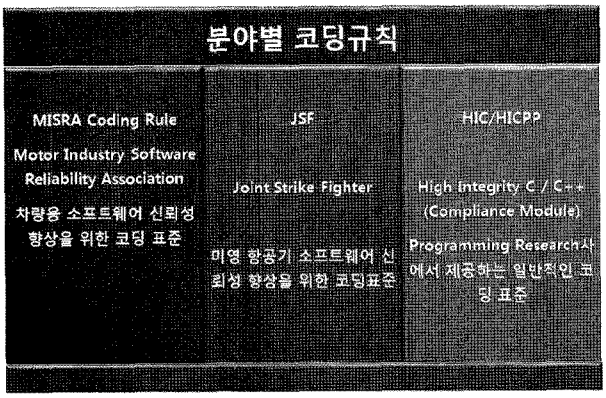

그림 5 산업체의 분야별 코딩 규칙

서 프로그램의 견고성을 중시하는 신뢰성 있는 프로그래밍으로 변화했으며, 최근에는 네트워크 환경의 발달과 함께 프로그램의 안전성을 중시하는 시큐어 프로그래밍으로 프로그래밍의 철학이 바뀌고 있다.

시큐어 프로그래밍은 "프로그래밍은 어떻게 해야 하는가?"에 대한 내용을 다루고 있으며, 프로그램의 안전성을 최우선시 하고 있다. 이러한 특징은 시큐어 코딩을 프로그래밍 패러다임 관점이 아니라 프로그래밍 철학 측면에서 보아야 더 적합하게 이해할 수 있게 한다.

시큐어 코딩은 시큐어 프로그래밍을 위한 방법론 중의 하나로 코딩 규칙(Coding Rules)과 취약성 보완 규칙(Weakness Guideline)으로 구성된다. 코딩 규칙은 프로그램을 표준화하여 작성하기 위한 핵심 코딩 규칙으로 프로그램의 일반성과 프로그래밍 원리를 반영하고 있으며, 취약성 보완 규칙은 프로그램의 취약점을 보완하기 위한 지침으로, 해킹 방어관점에서 정한 규칙이다.

프로그래밍시에 시큐어 코딩을 적용하기 위해, 프로그래머가 모든 상황을 전적으로 이해하고 취약점에 대한 다양한 경우를 전부 헤아리는 것은 많은 어려움이 있으므로, 도구를 통하여 시큐어 코딩을 효과적으로 적용할 수 있다. 이러한 도구로 규칙 검사기(Rule Checker)와 취약점 분석기(Weakness Analyzer) 등이 있다.

코딩 규칙 중, 표준 코딩 규칙은 특정한 기준에 맞는 프로그램을 작성하기 위한 필수 규칙으로, 최소 규칙으로 구성되며, 각 언어와 분야 별로 존재한다. 예로, 전자정부용 자바 표준 코딩 규칙은 30개로 언어 독립적인 부분 10개와 언어의존적인 부분 20개로 구성된다. 표준 코딩 규칙은 프로그래머가 따라야 하는 규약을 정의한 것으로, 작성된 프로그램이 표준 코딩 규칙을 만족하는가는 규칙 검사기를 이용하여 검증한다. 그림 5는 자동차, 항공기, 및 일반 산업체에서 사용하는 코딩 규칙의 대표적인 예를 정리한 것이다.

취약성 보완 규칙은 다른 표현으로 시큐어 코딩 가이드라고도 하는데, 시큐어 코딩 가이드는 프로그래머가 범하면 안되는 프로그램상의 취약점을 정리한 일종의 지침서로, 전자 정부용 자바 시큐어 코딩 가이드를 예로 들면 총 131개 항목으로 구성되어 있으며, CERT, CWE, Sun, Cigital의 코딩 가이드를 분석하여 작성되어 있다. 시큐어 코딩 가이드에서 언급하는 프로그램의 취약성은 도구를 이용하여 검사하는데, 이러한 도구는 일반적으로 크게 패턴을 기반으로 한 규칙기반 검사기(Rule based Analyzer)와 전용 분석기로 구분된다.

시큐어 코딩에서 시큐어 코딩 가이드는 적은 수의 필수 표준 코딩 규칙과는 다르게 프로그램에서의 취약성 발견이 지속적으로 발생하기 때문에 그 내용이 점차 확대되어야 하는데, 그림 6에 이러한 특성이 잘 반영되어 있다.

3.2 시큐어 코딩의 필요성

프로그램의 보안을 위한 요소(Security Features)와 안전을 위한 요소(Secure Features)가 일치하지 않는

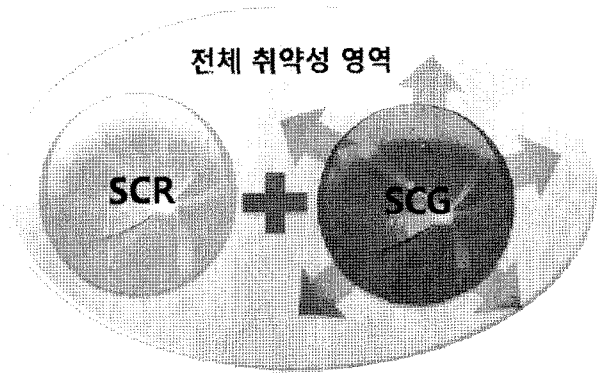

그림 6 클로져 속성

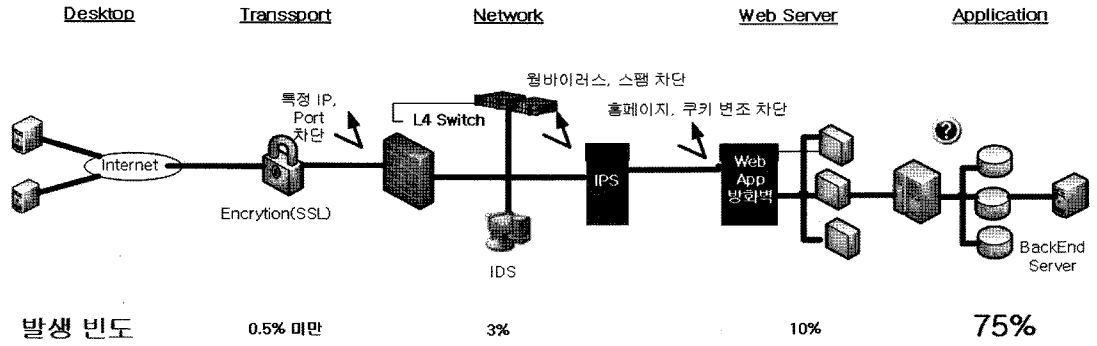

그림 7 인터넷기반 컴퓨터 정보 시스템의 취약성

특징을 가진다. 프로그램의 보안을 위한 요소는 프로그램이 올바른 기능과 시스템 보안을 관리하기 위한 요소이며, 프로그램의 안전을 위한 요소는 프로그램이 안전하게 되기 위한 조건으로 모든 특징 및 기능이 반드시 안전해야 하기 때문이다. 따라서, 프로그램에서의 보안 요소가 아닌 특징들이 프로그램을 위험하게 만들 수 있는 경우가 발생한다.

그림 7은 컴퓨터 정보 시스템의 취약성을 나타내고 있으며, 75~78%가 응용프로그램으로부터 기인하기 때문에, 응용 프로그램을 안전하게 개발하기 위한 효과적인 방법론 필요하다는 것을 알 수 있다[참조사이트 4]. 또, 개발 완료 후, 취약성을 보완하기 위한 비용이 매우 크므로 개발 단계에서부터 프로그램의 안전성을 고려해야 하는데, 이러한 관점에서 보면 개발 단계에서부터 안전한 프로그램을 작성하기 위해 정의된 시큐어 코딩은 매우 효과적이라고 볼 수 있다.

또한, 그림 8과 같이 해킹을 방지하기 위한 노력과 지출이 실제로 발생되는 부분이 상충되어 있다. 25%에 해당하는 취약성을 해결하기 위해 전체 소요 금액의 90%가 집중되어 있는 것을 볼 수 있는데, 이는 프로그램과 시스템의 취약성을 해결하기 위한 방법론에 변화가 있어야 한다는 것을 보여준다.

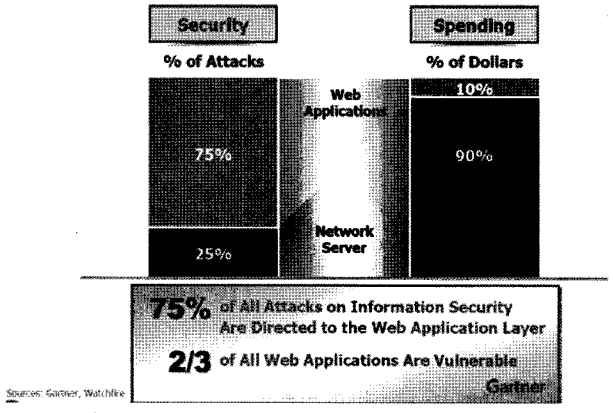

그림 8 보안문제 발생 빈도와 사용 금액

3.3 자바 취약점에 관한 최신 기술 동향

자바 취약점에 대해서는 최근에 CWE, CERT와 같은 단체에서 많은 연구와 활동이 이루어지고 있으며, 이를 간략히 소개하고자 한다.

먼저, CWE(Common Weakness Enumeration)는 미국 토안보부(The U.S. Department of Homeland Security)에서 관리하고 있으며, 소프트웨어의 취약성을 사전식으로 분류하여 프로그래머가 쉽게 접근할 수 있도록 구성되어 있다[참조사이트 2]. 이것은 수집된 취약점 항목을 뷰, 카테고리, 취약점, 복합요소를 기준으로 분류하여 살펴볼 수 있는 특징을 가지고 있으며, 취약점 항목에 대한 갱신은 현재에도 지속적으로 이루어지고 있다.

다음으로, 카네기 멜론 대학교의 소프트웨어 공학 연구소에서 관리하는 CERT(Computer Emergency Response Team)는 시큐어 코딩에서도 활발한 활동을 하고 있는데, 특히 코딩 규칙 및 가이드(Coding Rules/Guide)의 표준화 작업을 수행하고 있다[참조사이트 8]. CERT는 Secure Coding Standard를 프로그래밍 언어별 특징을 기준으로 분류하여 안전한 코딩을 언어가 갖는 특징 별로 구분하여, 각 언어의 사용자 및 학습자가 수월하게 접근하도록 하고 있다. CERT에서 제공하는 Secure Coding Standard for 자바는 2009년 10월 30일 기준으로 17개의 카테고리로 구성되어 있으며, 각 항목은 지속적으로 갱신되고 있다.

SANS(SysAdmin, Audit, Network, Security)는 1989년에 설립된 산학협동연구소로, 이 곳에서 제시한 "SANS TOP 25 Most Dangerous Programming Errors"는 취약점을 프로그래밍 개념을 기준으로 분류하였으며, Insecure Interaction Between Components, Risky Resource Management, Porous Defenses의 3개 카테고리로 구성되어 있다. 각 카테고리는 다시 세부 항목을 포함하고 있으며, 항목의 개수는 각각 9개, 9개, 7개이다[참조사이트 6].

표 2 자바 언어 관점에서의 취약성 항목 분류

분류기준	CERT	Sun	Cigital	CWE	total
Declaration	28		3	13	44
Statement	19	2	5	16	42
Class	32	13	1	11	57
Exception	11		3	14	28
Thread	11	1	7	20	39
Package(API)	17	4	4	37	62
Environment	20		32	60	112
total	138	20	55	171	384

Seven Pernicious Kingdom은 Katrina Tsipenyuk, Brian Chess(Fortify), Gary McGraw(Cigital)가 제안한 분류 방법으로, 취약점을 프로그래밍 개념을 기준으로 분류하였으며, Input validation and representation, API abuse, Security features, Time and state, Errors, Code quality, Encapsulation, Environment의 7(+1)개의 카테고리로 구성되어 있다[참조사이트 7].

자바 언어를 개발한 Sun Microsystems도 자바언어를 위한 코딩 지침을 제공하고 있으며, 총 6개의 분류로 구성되어 있으며[참조사이트 5], Fortify는 SAST (Static Application Security Testing) 관련 회사로, CERT rule pack 기반의 Fortify 360과 같은 취약점 분석 도구를 개발하였으며[참조사이트 3], Cigital은 소프트웨어 보안 및 품질에 대한 컨설팅 회사로 CERT rule pack을 확장한 자바 룰팩을 제공하고 있다[참조사이트 1].

CERT, Sun, Cigital, CWE에서 다루고 있는 자바의 취약항목을 자바의 언어적인 관점에서 분류하면 표 2와 같다.

4. Top 25

4.1 Top25 기본 개념

TOP 25란 SANS와 MITRE가 주관한 프로젝트로, 2009년 1월 12에 SANS, MITRE, CWE, CERT, 미 국토안보부, Microsoft, Symantec 등 30개 이상의 단체가 함께 가장 위험한 프로그래밍 에러 25개를 선정한 것을 의미한다. 'The Top 25 Errors'는 보안 버그로 이어지며 사이버 스파이 행위 및 사이버 범죄를 가능케 하는 프로그래밍 에러를 선정해 벤더들이 소프트웨어가 판매되거나 설치되기 전에 에러를 제거하고자 하는 목적으로 진행되었다.

TOP 25의 발표시 Mason Brown SANS 소장는 "이제 (프로그래밍 에러들을) 수정할 때가 왔다", "우선, 모든 프로그래머들이 TOP 25에서 (언급한 취약성을) 벗어난 코드를 작성하는 법을 알아야만 하며, 모든 프로그래밍 팀은 문제를 발견하고 수정, 또는 피할 수 있는 과정을 갖추고 이러한 에러들로부터 벗어난 그들의 코드를 인증하기 위한 툴을 갖춰야만 한다"와 같이 소개하며[참조사이트 6], 이러한 취약성의 분류를 소프트웨어 개발 측면에서 매우 중요하게 언급하였다. TOP 25는 크게 3개의 카테고리로 분류하였으며 그 내용을 요약하면 다음과 같다.

- Insecure Interaction Between Components: 9개
  - "컴포넌트의 불안전한 상호작용"
  - 컴포넌트들 사이에 불안전한 데이터 송수신
  - 서로 다른 컴포넌트, 모듈, 프로그램, 프로세스, 스레드, 시스템 사이에서 발생
- Risky Resource Management: 9개
  - "위험한 리소스 관리"
  - 소프트웨어 상에서 중요한 시스템 자원의 부적절한 관리
  - 리소스의 생성, 사용, 전달, 제거 등에서 발생
- Porous Defenses: 7개
  - "미흡한 방어"
  - 잘못 이용되고, 악용되거나 단순히 무시되는 방어적 프로그래밍 기술

Insecure Interaction Between Components: 9개

- 1. Improper Input Validation <- CWE-20
- 2. Improper Encoding or Escaping of Output <- CWE-116
- 3. Failure to Preserve SQL Query Structure ('SQL Injection') <- CWE-89
- 4. Failure to Preserve Web Page Structure ('Cross-site Scripting') <- CWE-79
- 5. Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection') <- CWE-78
- 6. Cleartext Transmission of Sensitive Information <- CWE-319
- 7. Cross-Site Request Forgery (CSRF) <- CWE-352
- 8. Race Condition <- CWE-362
- 9. Error Message Information Leak <- CWE-209

Risky Resource Management: 9개

- 10. Failure to Constrain Operations within the Bounds of a Memory Buffer <- CWE-119
- 11. External Control of Critical State Data <- CWE-642
- 12. External Control of File Name or Path <- CWE-73
- 13. Untrusted Search Path <- CWE-426
- 14. Failure to Control Generation of Code ('Code Injection') <- CWE-94
- 15. Download of Code Without Integrity Check <- CWE-494
- 16. Improper Resource Shutdown or Release <- CWE-404
- 17. Improper Initialization <- CWE-665
- 18. Incorrect Calculation <- CWE-682

Porous Defenses: 7개

- 19. Improper Access Control (Authorization) <- CWE-285
- 20. Use of a Broken or Risky Cryptographic Algorithm <- CWE-327
- 21. Hard-Coded Password <- CWE-259
- 22. Incorrect Permission Assignment for Critical Resource <- CWE-732
- 23. Use of Insufficiently Random Values <- CWE-330
- 24. Execution with Unnecessary Privileges <- CWE-250
- 25. Client-Side Enforcement of Server-Side Security <- CWE-602

그림 9 Top25와 CWE의 항목별 매핑

4.2 Top25와 CWE, CERT

Top 25는 기존에 존재하는 CWE의 취약항목에서 가장 위험한 에러 25개를 선정한 것으로 CWE와 항목별 매핑 관계는 그림 9와 같다. 이를 통해 소프트웨어에서 문제가 되는 취약성이 어떠한 특징을 가지고 있는가와 그 해결 방법을 쉽게 파악할 수 있다.

일반적으로, Top 25의 각 항목을 설명하기 위해 CWE의 취약성 항목을 이용하고 있으며, 다음 절인 Top25의 사례에서는 이와 같은 매핑 관계를 이용하여 사례를 설명하고 있다.

다음으로, 그림 10은 TOP 25의 3개 카테고리와 CERT의 17개 카테고리간의 매핑 관계를 보여주고 있으며, 정확한 매핑이 이루어지지 않아 중복된 카테고리를 갖고 있는 것을 알 수 있다.

4.3 Top 25 사례

Top 25에서 다루는 내용을 Porous Defenses 카테고리의 한 항목을 통해 확인해 보면 다음과 같다. 21번째 항목인 Hard-Coded Password는 CWE-259번과 동일한 내용을 다루며, CERT MSC03-J. Never hardcode sensitive information 항목과도 연관을 가진다. 이 항목은 프로그램 상에 직접 명시되어 있는 패스워드와 같은 민감한 정보가 외부에 노출되기 쉬우므로 해커들에 의해 악용될 가능성이 높고, 취약성을 발견했을 때 수정이 매우 어렵기 때문에 소스 코드상에 민감한 정보를 Hard-Coding 하지 말 것을 요구하고 있다.

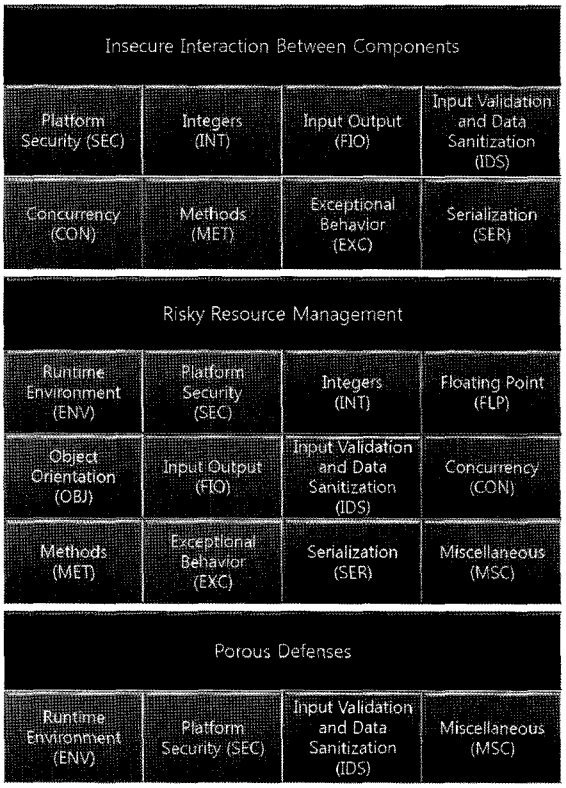

그림 10 Top25 카테고리와 CERT 카테고리 매핑

표 3은 소스상에 민감한 정보를 직접 입력한 소스의 예이며, 이러한 소스는 표 4와 같이 간단한 역어셈블러를 통해 프로그램 내의 민감한 정보를 확인할 수 있다.

표 3 취약한 소스

```
class Hardcoded {
    String password = new String("guest");
    public static void main(String[] args) {
      //..
    }
}
```

표 4 취약한 소스파일로부터 얻어진 클래스 파일을 역어셈블한 결과

```
javap -c : Deassembling class file
Compiled from "Hardcoded.java"
class Hardcoded extends java.lang.Object {
java.lang.String password;
Hardcoded();
  Code:
    0:   aload_0
    //Method java/lang/Object."<init>":()V
    1:   invokespecial #1;
    4:   aload_0
    5:   new         #2; //class java/lang/String
    8:   dup
    9:   ldc         #3; //String guest
    //Method java/lang/String."<init>":(Ljava/lang/String;)V
    11:  invokespecial #4;
    //Field password:Ljava/lang/String;
    14:  putfield    #5;
    17:  return
public static void main(java.lang.String[]);
  Code:
    0:   return
}
```

표 5는 표 3에서 발생하는 문제점을 제거한 소스의 예로 패스워드 정보를 외부의 암호화된 파일에 유지하고, 프로그램 내에서 해당 정보를 읽고, 복호화해서 사용하는 과정을 보여주고 있다.

표 5 민감한 정보를 제거한 소스 파일

```
class Password {
    public static void main(String[] args) throws IOException {
        char[] password = new char[100];
        BufferedReader br
            = new BufferedReader(new InputStreamReader(
                        new FileInputStream("password.txt")));
        // 1. Read the password into the char array,
        // return the number of bytes read
        int n = br.read(password);
        // 2. Decrypt password
        // 3. Perform operations
        // ...
        // 4. Manually clear out the password immediately after use
        for (int i = n - 1; i >= 0; i--) {
            password[i] = 0;
        }
        br.close();
    }
}
```

5. 결 론

오늘날의 컴퓨팅 환경은 소프트웨어의 정확성, 견고성 외에도 안전성이 매우 중시되고 있으며, 이는 대부분의 프로그램이 네트워크를 통해 연결되어 있기 때문이다. 시큐어 코딩은 안전한 소프트웨어 개발을 위한 방법론으로 최근에 많은 연구와 활동이 이루어지고 있는 분야이다.

시큐어 코딩은 새로운 프로그래밍 철학으로, 프로그램 설계와 작성 단계에서부터 취약성이 노출되는 것을 방지하여 프로그램의 취약성을 근본적으로 제거하고 해커로부터 안전한 소프트웨어를 개발하는 것을 목표로 하고 있다. 현재 이러한 개념에 대한 이해가 학계와 정부, 그리고 산업체에 확산되고 있으며, 컴퓨터 시스템과 네트워크의 보안 문제를 해결할 수 있는 주요한 방법으로 인식되고 있다.

본 고에서는 자바 프로그램을 안전하게 작성하기 위한 시큐어 코딩의 개념과 동향에 대해서 소개하였으며, 또한 가장 위험한 프로그래밍 오류를 규정하고 있는 Top 25를 소개하고 이를 통하여 프로그래밍시 발생할 수 있는 취약성 사례와 해결방법을 살펴보았다. 보다 많은 취약성 사례는 지면관계상 싣지 못했지만 참고문헌에 사이트 정보를 수록하여 더 많은 정보를 참조할 수 있도록 배려하였다.

앞으로 시큐어 코딩에 관련된 연구와 개발이 활발하게 진행되어 프로그래밍 활동에 시큐어 코딩이 기본이 되어 전반적인 보안 수준이 향상되고, 안전한 소프트웨어 개발이 IT산업에서의 큰 축이 되기를 바란다.

참고문헌

[1] Charlie Lai, "Java Insecurity: Accounting for Subtleties That Can Compromise Code," Software, IEEE, pp.13-19, 2008.

[2] Gary McGraw, Software Security, Addison-Wesley, February 2006.

[3] John Viega and Gary McGraw, Building Secure Software, Addison-Wesley, September 2001.

[4] JAVA™ SECURITY OVERVIEW, White Paper, Sun Microsystems, 2005.

[5] Lynn Futcher and Rossouw von Solms, "Guidelines for Secure Software Development," ACM Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, pp.56-65, 2008.

[6] The Java™ Language Specification, Third Edition, Sun Microsystems, 2005.

참조사이트

[1] Cigital Java Security Rulepack, http://www.cigital.com/securitypack/

[2] CWE(Common Weakness Enumeration), http://cwe.mitre.org/

[3] Fortify, http://www.fortify.com/products/fortify-360/

[4] Gartner, Nov 2005, http://gartner.com

[5] Secure Coding Guidelines for the Java Programming Language, Version 2.0, http://java.sun.com/security/seccodeguide.html

[6] SANS Top 25, http://www.sans.org/top25-programming-errors/

[7] SevenPerniciousKingdoms, http://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf

[8] The CERT Sun Microsystems Secure Coding Standard for Java, https://www.securecoding.cert.org/confluence/display/java/

[9] Tiobe Programming Community Index, 2009, http://www.tiobe.com/index.php/content/paperinfo/tptp/index.html

손 윤 식

2004 동국대학교 컴퓨터공학과 공학사
2006 동국대학교 컴퓨터공학과 공학석사
2009 동국대학교 컴퓨터공학과 공학박사
2010~현재 동국대학교 전문연구원
관심분야 : 프로그래밍 언어, 컴파일러, 모바일/임베디드 컴퓨팅, 로봇 소프트웨어, 소프트웨어 보안
E-mail: sonbug@dongguk.edu

오 세 만

1977 서울대 수학교육학과 졸업
1979 한국과학기술원 전산학과 석사 졸업
1985 한국과학기술원 전산학과 박사 졸업
1985~현재 동국대학교 컴퓨터공학과 교수
1993~1999 동국대학교 컴퓨터공학과 대학원 학과장
2001~2003 한국정보과학회 프로그래밍언어연구회 위원장
2004~2005 한국정보처리학회 게임연구회 위원장
관심분야 : 프로그래밍 언어, 컴파일러, 모바일 컴퓨팅, 소프트웨어 보안
E-mail: smoh@dongguk.edu