논문 13-38C-05-02

한국통신학회논문지 '13-05 Vol.38C No.05
http://dx.doi.org/10.7840/kics.2013.38C.5.420

안전한 소프트웨어 개발을 위한 정적분석 도구 시험코드 개발

방 지 호◆, 하 란*

Validation Test Codes Development of Static Analysis Tool for Secure Software

Jiho Bang◆, Rhan Ha*

요 약

최근 안전한 소프트웨어 개발을 위해 소프트웨어의 소스코드를 분석하여 보안취약점의 원인이 되는 소프트웨어 보안약점을 식별해 주는 정적분석 도구가 많이 활용되고 있다. 최적의 정적분석 도구를 선택하기 위해서는 도구가 보유한 보안약점 규칙 및 분석기능이 중요한 요소가 된다. 따라서, 본 논문은 정적분석 도구가 보유한 규칙 및 분석 성능을 평가하기 위해 개발한 시험코드를 제시하고자 한다. 시험코드는 SQL 삽입 등 43개 보안약점이 존재하는 소스코드로 정적분석 도구가 보유한 보안약점 규칙과 이를 기반으로 한 도구의 분석기능의 적절성을 평가하기 위해 사용될 수 있다.

Key Words : Static Analysis Tool, Weakness, CWE, Test Code, Source Code

ABSTRACT

Recently, for secure software development, static analysis tools have been used mostly to analyze the source code of the software and identify software weaknesses caused of vulnerabilities. In order to select the optimal static analysis tool, both weaknesses rules and analysis capabilities of the tool are important factors. Therefore, in this paper we propose the test codes developed for evaluating the rules and analysis capabilities of the tools. The test codes to involve 43 weaknesses such as SQL injection etc. can be used to evaluate the adequacy of the rules and analysis capabilities of the tools.

Ⅰ. 서 론

최근 안전한 소프트웨어(이하, SW) 개발을 위해 SW 개발단계에서 소스코드에 대한 정적분석을 통해 SW 결함, 오류 등 SW 보안약점(Weakness)을 식별해 주는 자동화된 정적분석 도구(이하, 진단도구)를 많이 활용하고 있다[1]. SW 보안약점은 SW 개발이후 운영단계에서 SW의 보안기능 우회나 관리자 권한상승 등으로 인해 개인정보 등의 중요자산 유출과 같은 사이버공격을 유발할 수 있는 보안취약점(Vulnerability)의 원인이기 때문에 사전에 식별하여 제거하는 것이 효과적이다.

정적분석 기반의 진단도구는 SW 기능 동작 없이 수백만 라인의 소스코드를 짧은 시간에 분석해서 SQL 삽입, 크로스사이트스크립트(이하, XSS) 등과 같은 보안취약점의 원인인 다양한 보안약점을 찾아주는 장점이 있다. 그러나, 진단도구는 보안약점과 관련된 패턴 등 진단규칙을 기반으로 보안약점을 점

※ This work was supported by the National Research Foundation of Korea grant funded by the Korean government, Ministry of Education, Science and Technology (No.2013-023635).
◆ 주저자 : 홍익대학교 컴퓨터공학과 실시간시스템 연구실, jhbang@kisa.or.kr, 정회원
* 홍익대학교 컴퓨터공학과 실시간시스템 연구실, rhanha@cs.hongik.ac.kr
논문번호 : KICS2013-03-110, 접수일자 : 2013년 3월 4일, 최종논문접수일자 : 2013년 5월 9일

검할 수 있기 때문에 진단규칙 보유 수준과 이를 기반으로 한 진단능력 등이 진단도구 성능에 중요한 영향을 미친다.

진단규칙 및 진단능력 등 진단도구의 성능을 평가하기 위해서는 보안약점이 내포된 시험코드가 필수적이다. 시험코드는 실제 운영 가능한 공개용 오픈소스(예, OpenSSL 등)를 활용하는 방법[2,3]과 인위적으로 시험코드를 개발하는 방법[4-6]이 있다. 공개용 오픈소스를 활용하는 경우, 제품에 대한 보안취약점 정보를 공개하는 CVE(Common Vulnerabilities and Exposures)[7] 등을 기반으로 공개용 오픈소스를 선정하여 관련 소스코드 보안약점 정보를 식별해서 시험코드로 사용할 수 있다. 그러나, 공개된 정보 이외의 보안약점을 알 수 없기 때문에 이를 보완하기 위해 소스코드 리뷰를 통해 추가적인 보안약점을 식별해야 한다. 이는 오픈소스 규모 및 복잡도 등의 제약사항 때문에 많은 시간 및 노력이 요구된다. 인위적으로 보안약점이 존재하도록 시험코드를 개발하여 활용하는 경우, 다양한 보안약점을 구현할 수 있기 때문에 진단도구의 진단규칙 및 진단능력을 평가하는데 효과적인 반면 공개용 오픈소스보다 소스코드 규모가 작기 때문에 다양한 구현 유형을 반영하여 이를 보완하는 것이 필요하다.

진단도구 평가기준 및 방법에 대한 기존 연구들은 공개용 오픈소스를 선정하여 각각의 진단도구가 어떤 보안약점을 식별하였는지 등에 대해 분석[2,3] 하거나, 특정 보안약점이 존재하도록 시험코드를 개발하여 진단도구의 진단능력을 평가[4-6]하고 있다. 특정 보안약점을 이용하여 진단도구의 성능을 시험하는 경우, C/C++ 언어 기반의 진단도구는 '버퍼오버플로우' 보안약점을 중심으로 하고 있으며, JAVA 언어 기반의 진단도구는 'SQL 삽입' 및 'XSS' 보안약점을 중심으로 시험코드를 개발하여 활용하고 있다.

최근 국내 SW의 안전한 개발을 위해 SW 보안약점 43개 항목(지침[8] 별표3)이 내포되지 않도록 개발단계에서 진단하여 제거하도록 하는 SW 개발보안 제도가 의무화 되었다. 이에 따라, 국내 전자정부 SW 개발을 위해 사용되는 진단도구는 SW 보안약점 항목에 대한 진단규칙 및 진단능력을 보유하는 것이 필요하며, 이를 평가할 수 있는 환경 구축이 필요하다. 기존 연구들은 특정 보안약점에 초점을 맞추고 있기 때문에 국내 환경에 적용하는데 한계가 있다.

따라서, 본 논문은 안전한 SW 개발을 위해 사용할 수 있는 진단도구 개발 및 평가를 위해 개발한 시험코드 개발방법을 제시하여 진단도구 평가기관 및 개발업체 등에서 활용할 수 있게 하였다.

본 논문은 2장에서는 진단도구 시험환경 관련 국외 동향을 설명하고, 3장에서는 진단도구 평가를 위해 개발한 시험코드를 설명한다. 그리고, 4장에서는 시험코드 구현결과를 설명하고, 5장에서 결론을 맺는다.

Ⅱ. 관련 동향

SW 보안약점을 구현한 시험코드는 소스코드를 대상으로 하는 정적분석 기반의 진단도구의 성능 평가를 위해 개발되어 사용되고 있다. 기존 연구들은 진단도구에 구현된 진단규칙과 관련된 시험코드만 제한적으로 개발[4,5]하거나, 특정 보안약점의 진단규칙에 독립적인 범용성의 목적을 가지고 개발[6]하는 경우로 구분된다. 또한, 비교대상 진단도구의 진단결과를 비교하기 위해 공개용 오픈소스를 사용하는 경우, CVE를 통해 공개된 보안취약점 정보를 활용[3]하는 경우와 진단도구의 진단결과에 의존[2]하여 결과를 분석하는 경우로 구분된다.

보안약점을 인위적으로 구현하여 시험코드로 사용하는 경우, 특정 진단규칙의 성능을 점검하기 위해 관련 보안약점만 제한적으로 개발하는 것은 진단규칙에 종속적일 수 있다. 따라서, 특정 진단도구의 진단규칙을 고려하지 않고, 보안약점을 기반으로 다양한 존재 유형을 분석하여 체계적으로 개발하는 것이 진단도구 성능평가시 보다 객관적인 지표로 사용될 수 있다. NSA(美 국가안보국, National Security Agency)의 CAS(Center for Assured Software)에서 개발된 Juliet 코드[6]가 대표적인 예라고 할 수 있다.

공개용 소스코드를 시험코드로 사용하는 경우, 공개용 소스코드에 존재하는 보안약점 정보 없이 진단도구의 진단결과에 의존하여 성능평가를 하는 것보다 CVE에 공개된 보안취약점을 기반으로 공개용 소스코드에 존재하는 보안약점 정보를 확인한 이후 진단도구 성능을 평가하는 것이 보다 정확한 성능평가를 수행할 수 있다. 그러나, 공개용 소스코드는 CVE 등을 통해 알려진 보안취약점 정보만 활용 가능하기 때문에 활용할 수 있는 보안약점의 수가 제한적인 단점이 있다. CVE를 활용하여 시험코드로 공개용 소스코드를 선택하는 경우는 NIST(美 국립표준기술연구소, National Institute of Standards and Technology) SATE(Static Analysis Tool Exposition)[3]가 대표적이라고 할 수 있다.

다음절에서 대표적인 시험코드 개발 및 활용 사례

인 NIST의 SATE 및 Juliet 코드에 대해 살펴본다.

2.1. NIST SATE[3]

SW 보안성을 향상시키기 위해 NIST는 국토안보부(DHS, Department of Homeland Security)의 지원하에 2004년부터 SAMATE(SW Assurance Metrics And Tool Evaluation) 프로젝트를 시작하였다. SW 보안성을 지원하는 도구 평가 방법, 도구와 개발보안 기법의 차이 및 효과 분석 등을 수행하는 SAMATE 프로젝트는 정적분석 기반의 상용 진단도구를 대상으로 2008년부터 매년 상용 진단도구 개발업체의 참여하에 SATE를 개최하고 있다. SATE는 참여를 희망하는 상용 진단도구 개발업체를 대상으로 NIST에서 제시하는 공개용 오픈소스를 시험코드로 하여 각 진단도구 특징 및 분석 성능을 시험한다. 2010년에는 공개용 오픈소스 외에 CVE에서 리포팅한 정보를 기반으로 취약한 공개용 오픈소스와 보안패치된 공개용 오픈소스를 한 쌍의 시험코드로 사용하였다. 2012년에는 단순 공개용 오픈소스는 시험코드에서 제외하고, 대신 Juliet 코드를 시험코드에 포함하였다. 2012년 이전까지는 진단도구의 성능보다 SATE에 참여한 진단도구의 특성 분석 및 비교에 초점을 두었으나, 2012년에는 3차례의 SATE 진행 경험을 토대로 SATE에 참여한 진단도구의 성능(정탐, 오탐 등) 관점으로 SATE 진행 결과를 리포트 하였다.

2.2. Juliet 코드[6,9]

NIST는 SAMATE 프로젝트의 일환으로, SRD(SAMATE Reference Database)를 구축하였다. SRD는 잘 알려진 보안약점을 소스코드 형태로 제공하여 진단도구 개발 및 평가 등에 도움을 주는데 목적이 있다. SRD는 파일단위(예, *.java)의 시험코드나 컴파일 가능한 라이브러리 등의 환경을 같이 제공하는 독립적인 시험코드를 제공한다. 독립적인 시험코드는 Juliet 코드가 대표적으로, NSA의 CAS에서 개발하여 2010년 12월에 1.0 버전을 공개한 이후, 2012년 7~9월경에 2.0 버전을 공개하였다. Juliet 코드는 JAVA 및 C/C++ 프로그래밍 언어를 대상으로 하고 있으며, C/C++의 경우 C++에만 해당되는 보안약점을 제외하고 대부분 C 언어 위주로 개발되었다. Juliet 코드는 코드 구조의 복잡도를 고려하지 않고 보안약점을 구현한 기본코드와 데이터 및 제어흐름 등 코드 복잡도(표1)를 고려하여 기본코드에 반영하여 개발한 복잡도 반영 코드로 구성된다.

표 1. 코드 복잡도
Table 1. Code Complexity

Type	Description	JAVA	C/C++
Basic	Basic Structure	1	1
Control flow	Weaknesses implemented within various condition or loop statement for testing an analysis capability of tool	19	20
Data flow	For testing an analysis capability of tool for data flow such as multiple variable assignment etc.	18	26

Ⅲ. 시험코드 개발

본 장에서는 SW 개발을 위해 필수적으로 제거해야 하는 SW 보안약점이 존재하도록 개발한 시험코드(JAVA, C/C++)를 JAVA 언어 기반의 시험코드 중심으로 설명한다.

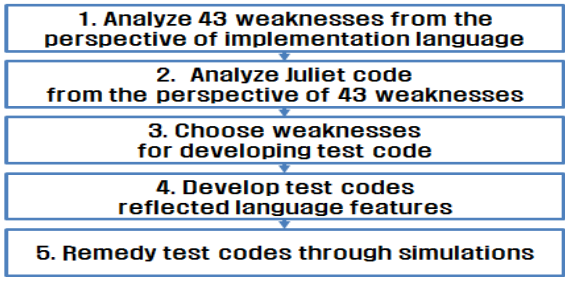
3.1. 시험코드 구조

본 논문에서 설명하는 시험코드는 Juliet 코드[6,9] 개발구조를 적용하여 개발하였으나, Juliet 코드와 다르게 진단도구 분석결과를 보다 쉽게 평가할 수 있도록 보안약점이 내재된 Bad 코드와 보안약점이 제거된 Good 코드로 구분하여 별도 파일 단위(예, *_bad.java, *.good.java)로 개발하였다. Juliet 코드는 대부분 Bad 코드와 Good 코드를 하나의 파일(예, *.java)에 동시에 포함되도록 구현하였다. Bad 코드는 제3.2절에서 설명한 내용을 기반으로 구현하였으며, Good 코드는 Bad 코드에 존재하는 보안약점을 제거하는 방식으로 구현하였다. 시험코드 단순성을 극복하기 위해 Juliet 코드에서 적용한 코드 복잡도(표 10)를 Bad 코드 및 Good 코드에 적용하여 다양한 유형의 시험코드를 개발하였다.

3.2. 시험코드 개발

전자정부 SW 개발시 고려해야 하는 43개 SW 보안약점 항목은 CWE(Common Weakness Enumeration)[10]의 보안약점 중 최근 국내 전자정부 SW에서 많이 발생하는 보안약점을 기반으로 선정되었다[11]. 시험코드 개발은 다음 그림 1과 같이 개발언어(예, JAVA) 관점으로 43개 보안약점을 분석하고, CWE 기반으로 개발된 NSA CAS의 Juliet 코드[6]를 분석하여 43개 보안약점 중 Juliet 코드에 구현되지 않은 보안약점을 중심으로 시험코드를 개발하였다. 개발된 시험코드는 시범검증(제4장 참고)을 통해 보완되었다.

그림 1. 시험코드 개발절차
Fig. 1. Test Codes Development Process



3.2.1. 입력데이터 검증 및 표현

Juliet 코드[6,9]에서 '입력데이터 검증 및 표현' 유형에 해당되는 시험코드의 경우, JAVA는 'SQL 삽입' 등 8개 항목에 대한 시험코드가 존재(표2)하여 활용이 가능하다. '디렉토리 경로 조작' 보안약점은 CWE 항목이 22번으로, 해당되는 Juliet 코드는 없지만 CWE 22와 계층구조를 갖는 CWE 23(상대 디렉토리 경로 조작)과 CWE 36(절대 디렉토리 경로 조작)을 통해 발생되는 보안약점이기 때문에 Juliet 코드에 존재하는 해당 코드를 활용할 수 있다. 그리고, CSRF(크로스사이트 요청위조)의 경우 해당 코드의 유효성 문제로 인해 Juliet 1.1로 버전이 업데이트되면서 삭제[6]되었기 때문에 개발이 필요하다.

따라서, JAVA 언어 관련 시험코드는 '자원삽입' 등 5개 보안약점에 대한 시험코드 추가 개발이 필요하다.

표 2. 입력데이터 검증 및 표현
Table 2. Input data validation and expression

Weakness	CWE No.	Juliet[6]	
		JAVA	C/C++
SQL Injection	89	O	-
Resource Injection	99	-	-
XSS	80	O	-
OS Command Injection	78	O	O
Unrestricted Upload of File with Dangerous Type	434	-	-
URL Redirection to Untrusted Site	601	O	-
XQuery Injection	652	-	-
XPath Injection	643	O	-
LDAP Injection	90	O	-
CSRF	352	-	-
Path Traversal	22	△	△
HTTP Response Splitting	113	O	-
Integer Overflow	190	O	O
Reliance on Untrusted Inputs in a Security Decision	807	-	-

'자원삽입' 보안약점은 소켓, 파일 등과 같은 시스템 자원에 접근하는 경로 등의 정보를 네트워크 등을 통해 유입되는 데이터를 적절한 검증 없이 사용할 경우 발생된다. 관련 시험코드는 소켓 및 파일 등의 시스템 자원에 대한 접근 경로를 다양한 데이터 유입 유형(표3) 중 DB, Config, Network 등의 유형을 통해 유입되는 경로 데이터를 검증 없이 사용하도록 구현(표4)하였다.

표 3. 데이터 유입 유형[12]
Table 3. Data inflow type

Type	Description
Basic	Data defined in SW internal
DB	Data from DBMS, filesystem
Config	Data from console, environment, property etc.
Web	Data from servlet, cookies, parameter, URL etc.
Network	Data through TCP protocol

표 4. '자원삽입' 보안약점 관련 예시
Table 4. Example related to 'Resource Injection' weakness

Feature	내용
Inflow Type	· DB, Config, Network etc.
Resource Type	· file, socket etc.
Method	· File(), ServerSocket() etc.

Code	data = buffread.readLine();
	…
	int port = Integer.parseInt(data);
	sock = new ServerSocket(port + 3000);

'위험한 형식 파일 업로드' 보안약점은 업로드 대상 파일 확장자 등 파일 형식에 대한 검증 없이 업로드를 허용하는 경우 발생된다. 관련 시험코드는 서블릿 웹 환경에서 파일 형식에 대한 검증 없이 업로드(예, getOriginalFilename()) 하도록 구현하였다.

'XQuery' 보안약점은 XML 데이터에 대한 질의문 생성시 네트워크 등을 통해 유입되는 데이터를 적절한 검증 없이 질의문으로 사용할 경우 발생된다. 관련 시험코드는 다양한 데이터 유입 유형(표3) 중 DB, Config, Web, Network 등의 유형을 통해 유입되는 데이터를 검증 없이 XQuery로 사용하도록 구현하였다.

'CSRF(크로스사이트 요청 위조, Cross-Site Request

Forgery)' 보안약점은 금융결재 등과 같은 웹 서비스 상의 중요기능을 적절한 검증 없이 허용하여 웹 서버로 관련 데이터가 전송될 경우 발생된다. Juliet 1.0 코드는 웹서버 측면에서 해당 보안약점을 구현하여 제기된 유효성 문제로 Juliet 1.1 코드에서 삭제[6]되었다. 따라서, CSRF 보안약점 관련 시험코드는 HTML 형태로 사용자 권한 검증 없이 웹 서버로 Form 데이터를 전송하도록 구현하였다.

'보호메커니즘을 우회할 수 있는 입력값 변조' 보안약점은 보안메커니즘 동작과 관련된 보안결정(예, 인증, 권한부여 등)시 네트워크 등을 통해 유입되는 데이터를 적절한 검증 없이 사용할 경우 발생된다. 관련 시험코드는 인증 여부 플래그 및 사용자 역할(Role) 변경시 다양한 데이터 유입 유형(표3) 중 Web 등의 유형을 통해 유입되는 데이터를 검증 없이 사용하여 보호메커니즘을 우회할 수 있도록 구현하였다. 보호메커니즘 등은 다양하기 때문에 해당 시험코드 사용시 진단도구에 보호메커니즘 관련 중요 데이터 정보 등을 반영하여 평가하는 것이 필요하다.

'입력데이터 검증 및 표현' 유형의 보안약점은 특성상 입력데이터 유형의 다양성이 중요하기 때문에 위에서 설명한 구현 내용을 기반으로 다양한 데이터 유입 유형(표3)을 반영할 수 있도록 구현하였다.

3.2.2. 보안기능

Juliet 코드[6,9]에서 '보안기능' 유형에 해당되는 시험코드의 경우, JAVA는 '취약한 암호화 알고리즘 사용' 등 9개 항목에 대한 시험코드가 존재한다(표5). '사용자 중요정보 평문 저장(또는 전송)' 보안약점은 CWE 항목이 311번으로, 해당되는 Juliet 코드는 없지만 CWE 311와 계층구조를 갖는 CWE 319(중요정보 평문전송)을 통해 발생되는 보안약점이기 때문에 Juliet 코드에 존재하는 해당 코드를 활용할 수 있다.

따라서, JAVA 언어 관련 시험코드는 '적절한 인증 없는 중요기능 허용' 등 7개 보안약점에 대한 시험코드 추가 개발이 필요하다.

'적절한 인증 없는 중요기능 허용' 보안약점은 개인정보 같은 중요정보를 적절한 인증 없이 열람 또는 변경할 수 있도록 허용할 경우 발생된다. 관련 시험코드는 적절한 인증 과정 없이 은행 계좌이체 등과 같은 중요기능을 수행 하도록 구현하였다.

'부적절한 인가' 보안약점은 적절한 접근제어 부재로 외부에서 서버에 존재하는 임의 사용자 정보 등의 정보에 접근할 경우 발생된다. 관련 시험코드는 사용자의 권한을 점검하지 않고 사용자 정보를 전송하도록 구현하였다.

'중요한 자원에 대한 잘못된 권한설정' 보안약점은 설정파일 등과 같은 중요 자원에 대한 권한설정이 부적절할 경우 발생된다. 관련 시험코드는 설정파일 등과 같은 중요자원에 대한 권한을 부적절하게 부여(예, umask(0))하여 권한 없는 사용자도 쉽게 접근할 수 있도록 구현하였다.

'충분하지 않은 키 길이 사용' 보안약점은 데이터 암호화를 위해 사용되는 암호알고리즘의 키 길이가 충분하지 않을 경우 발생된다. 관련 시험코드는 암호화알고리즘의 안전성을 위해 요구되는 최소 키 길이(예, RSA 2,048bit)를 만족하지 못하도록 구현하였다.

'취약한 패스워드 허용' 보안약점은 비밀번호를 쉽게 유추할 수 있을 정도로 비밀번호 조합규칙이 부적절할 경우 발생된다. 관련 시험코드는 비밀번호 생성 규칙에 대한 적용하지 않는 경우와 단순한 규칙(예, 비밀번호 7자리 이하, 영문자로만 생성 가능 등)만 적용하여 비밀번호를 생성하도록 구현하였다.

표 5. 보안기능
Table 5. Security features

Weakness	CWE No.	Juliet[6]	
		JAVA	C/C++
Missing Authentication for Critical Function	306	-	-
Improper Authorization	285	-	-
Incorrect Permission Assignment for Critical Resource	732	-	-
Use of Broken or Risky Crypto. Algorithm	327	O	O
Missing Encryption of Sensitive Data	311	△	△
Use of Hard-coded Password	259	O	O
Insufficient Key Size	310	-	-
Use of Insufficiently Random Values	330	O	-
Plaintext Storage of Password	256	O	O
Use of Hard-coded Crypto. Key	321	O	O
Weak Password Requirements	521	-	-
Info. Through Persistent Cookies	539	-	-
Sensitive Cookie in HTTPS Session without Secure Attribute	614	O	-
Info. Exposure Though Comments	615	O	-
Use of an One-Way Hash without a Salt	759	O	-
Download of Code Without Integrity Check	494	-	-

‘사용자 하드디스크에 저장되는 쿠키를 통한 정보노출’ 보안약점은 세션 ID, 사용자 권한 등 쿠키에 포함된 중요정보가 사용자 하드디스크에 저장되거나 쿠키의 유효기간이 긴 경우 발생된다. 관련 시험코드는 쿠키의 유효기간을 길게 설정(예, 1주일 이상) 하도록 구현하였다.

‘무결성 검사 없는 코드 다운로드’ 보안약점은 업데이터 서버 등과 같은 원격서버로부터 업데이트 파일 등을 무결성 검사 없이 다운로드 받을 경우 발생된다. 관련 시험코드는 특정 주소(URL) 또는 디렉토리에서 클래스 및 리소스를 무결성 검사 없이 다운로드(예, URLClassLoader()) 하도록 구현하였다.

‘보안기능’ 유형의 보안약점은 다른 보안약점 유형과 달리 SW 설계 및 구현결과를 반영되어야 구체화되기 때문에 ‘보안기능’ 유형의 보안약점을 진단하기 위해서는 SW 설계 및 구현 내용을 파악하여 사용자 진단규칙을 설정하거나 기존 진단규칙을 수정하는 것이 필요하다. 따라서, ‘보안기능’ 유형의 보안약점과 관련된 시험코드는 진단도구의 기본 진단규칙 보유여부를 점검하는데 활용하는 것이 좋다.

3.2.3. 시간 및 상태 등 5개 유형의 보안약점

Juliet 코드[6,9]에서 ‘시간 및 상태’ 등 5가지 보안약점 유형에 해당되는 시험코드의 경우, JAVA는 ‘제어문을 사용하지 않는 재귀함수’ 등 9개 항목에 대한 시험코드가 존재(표6)하여 활용이 가능하다.

따라서, JAVA 언어 관련 시험코드는 ‘경쟁조건: 검사시점과 사용시점(TOCTOU)’ 등 6개 보안약점에 대한 시험코드 추가 개발이 필요하다.

‘경쟁조건: 검사시점과 사용시점(TOCTOU)’ 보안약점은 멀티 프로세스 또는 쓰레드 환경에서 파일, 소켓 등의 자원을 다른 프로세스 또는 쓰레드가 사용(또는 점유)하는지 여부를 점검하지 않고 사용할 경우 발생된다. 관련 시험코드는 서로 다른 쓰레드(예, A, B)가 동일한 파일에 접근하고 있을 때, A 쓰레드는 해당 파일을 삭제하고, B 쓰레드는 해당 파일의 존재여부를 점검하지 않고 작업하도록 구현하였다.

‘적절하지 않은 예외처리’ 보안약점은 메소드(함수) 수행 결과에 대한 적절한 검증 및 예외상황 조건을 적절하게 점검하지 않는 경우 발생된다. 관련 시험코드는 TCP 통신을 통해 전달받은 파일명을 기반으로 해당 파일에 접근해서 데이터를 읽을 때, 해당 파일의 널(Null) 여부를 검사하지 않고, 부적절한 예외처리(예, Exception e)를 하도록 구현하였다.

표 6. 시간 및 상태 등 5개 유형의 보안약점
Table 6. 5 types weakness, such as Time and Status etc.

Weakness		CWE No.	Juliet[6]	
			JAVA	C/C++
Time & Status	TOCTOU Race Condition	367	-	O
	Uncontrolled Recursion	674	O	O
Errors	Information Exposure through an error message	209	O	-
	Detection of Error Condition Without Action	390	O	O
	Improper Check for Unsual or Exceptional Conditions	754	-	-
Code Quality	NULL Pointer Dereference	476	O	O
	Improper Resource Shutdown or Release	404	O	O
Encapsulation	Exposure of Data Element to Wrong Session	488	-	-
	Leftover Debug Code	489	O	O
	Info. Leak of System Data	497	O	-
	Private Array-Typed Field Returned From A Public Method	495	-	-
	Public Data Assigned to Private Array-Typed Field	496	-	-
API Abuse	Reliance on DNS Lookups in a Security Decision	247	-	O

‘잘못된 세션에 의한 데이터 정보노출’ 보안약점은 서로 다른 세션에서 데이터가 공유되는 경우 발생한다. 관련 시험코드는 싱글톤 패턴 사용 환경에서 지역변수가 아닌 멤버필드를 사용하여 권한 없는 사용자가 해당 변수에 저장된 데이터에 접근할 수 있도록 구현하였다.

‘Public 메소드부터 반환된 Private 배열’ 보안약점은 외부에 공개되어 임의로 변경되지 않아야 하는 Private 배열이 Public 메소드의 반환값으로 설정될 경우 발생된다. 관련 시험코드는 Private 속성으로 선언된 배열이 Public 메소드를 통해 반환되도록 구현하였다.

‘Private 배열에 Public 데이터 할당’ 보안약점은 Public 속성으로 선언된 데이터 또는 메소드의 매개변수가 Private 배열에 저장될 경우 발생된다. 관련 시험코드는 Public 속성으로 선언된 메소드의 매개변수 값을 Private 속성으로 선언된 배열에 저장하도록

구현하였다.

'DNS lookup에 의존한 보안결정' 보안약점은 DNS 스푸핑 등으로 도메인(예, www.test.com) 정보가 변조될 수 있기 때문에 DNS 서버를 통해 얻은 도메인을 기반으로 신뢰된 호스트 결정시 발생될 수 있다. 관련 시험코드는 DNS 서버를 통해 얻은 도메인을 화이트리스트에 포함된 도메인과 비교를 통해 해당 도메인을 기반으로 TCP 통신을 하도록 구현하였다.

'시간 및 상태' 등 5개 유형의 보안약점은 '입력데이터 검증 및 표현'과 '보안기능' 유형의 보안약점보다 관련된 메소드(함수) 및 데이터 유입유형이 제한적이기 때문에 상대적으로 적은 수의 시험코드가 개발되었다.

Ⅳ. 구현결과

진단도구의 진단규칙 및 진단성능을 평가하기 위해 본 논문에서 제시한 시험코드는 제3장에서 설명한 내용을 기반으로 43개 보안약점을 JAVA 및 C/C++ 프로그래밍 언어 기반으로 구현하였다. JAVA 언어 관련 시험코드는 308개의 기본코드로 구성되어 있으며, 다양한 데이터 및 제어 흐름 등을 기본코드에 반영하여 약 9,000개의 시험코드를 개발하였다. 그리고, C/C++ 언어 관련 시험코드는 427개의 기본코드를 포함하여 약 15,000개의 시험코드를 개발하였다.

본 논문에서 제시한 시험코드는 국내·외 상용 진단도구 13종(JAVA 7종, C/C++ 6종)에 대해 시범검증시 적용되었다. 먼저 기본코드를 기반으로 진단도구 진단규칙 및 관련 진단기능을 점검하였으며, 기본코드를 기반으로 진단도구가 정확히 진단한 보안약점 항목에 대해서는 복잡도가 적용된 시험코드를 기반으로 추가 분석을 수행하였다. SW 개발보안 적용 의무화에 따라, 시험결과는 각 도구업체에 영향을 미칠 수 있기 때문에 본 논문에서는 시험코드 관점으로 결과를 간략하게 설명한다.

보안약점 발생 환경 및 조건의 유사성으로 시범검증시 일부 보안약점을 잘 구분하지 못하는 경우가 있었는데, '자원삽입'과 '디렉토리 경로조작' 보안약점이 대표적이다. '자원삽입' 보안약점은 시스템 자원에 대한 식별자로 외부 입력값을 검증 없이 사용하는 경우 시스템 자원에 임의로 접근할 수 있는 보안약점이 존재하는데 시스템 자원이 설정 파일일 경우 외부 입력값으로 해당 파일명 이외에 해당 파일에 대한 경로도 같이 입력될 수 있다. 이와 같은 이유로 시스템 자원을 파일인 경우로 구현한 '자원삽입' 보안약점 시험코드에 대해 일부 진단도구는 '디렉토리 경로조작' 보안약점으로 식별하는 경우가 발생하였다. 그 외에도, 'HTTP 응답분할'과 '신뢰되지 않은 URL 주소로 자동 접속 연결' 보안약점 등에 대해 구분을 하지 못하는 경우도 있었다.

진단도구 성능 개선 등의 목적으로 개발한 시험코드를 기반으로 시범검증을 수행하고 분석한 결과, 해당 보안약점을 탐지하지 못하거나 잘못 탐지하는 경우는 시험코드의 구현 결함 보다 시범검증 대상 진단도구의 진단규칙이 부재하거나 정교화되지 않아서 발생하는 문제였다.

Ⅴ. 결 론

본 논문에서는 진단도구 성능 평가 및 개선을 위해 '정보시스템 구축·운영 지침'으로 의무화된 43개 SW 보안약점을 기반으로 개발한 시험코드 개발 방법을 제시하였다. 그리고, 개발한 시험코드를 기반으로 상용 진단도구 시범검증에 적용하여 해당 진단도구가 개선될 수 있도록 유도하였다.

본 논문에서 설명한 시험코드는 진단도구 시범검증 및 정보보호제품 평가·인증을 위해 사용될 예정이며, 전자정부 표준프레임워크 환경을 반영하여 지속적으로 시험코드를 개선할 예정이다. 향후, 안전한 SW 개발을 위한 공개용 진단도구 대상 보안약점 진단규칙을 제안할 예정이다.

참 고 문 헌

[1] P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in *Proc. IEEE Int. Conf. Inform. Theory Inform. Security (ICITIS) 2010*, pp. 521-524, Beijing, China, Dec. 2010.

[2] R. K. McLean, "Comparing static security analysis tools using open source software," in *Proc. 6th IEEE Int. Conf. SW Security Reliability Companion (SERE-C)*, pp. 68-74, Gaithersburg, U.S.A., June 2012.

[3] NIST, "Report on the Static Analysis Tool Exposition(SATE) IV," *NIST Special Publication 500-297*, Jan. 2013.

[4] T. Hofer, "Evaluation static source code analysis tools," M.S. Thesis, School Compt.

Commun. Sci., École Polytechnique Fédérale de Lausanne, Mar. 2010.

[5] M. Johns and M. Jodeit, "Scanstud: a methodology for systematic, fine-grained evaluation of static analysis tools," in *Proc. IEEE 4th ICSTW*, pp. 523~530, Berlin, Germany, Mar. 2011.

[6] NIST and NSA CAS, *Juliet Test Suite for Java and C/C++*, Retrieved Sep., 2012, from http://samate.nist.gov/SRD/testsuite.php.

[7] MITRE, *Common Vulnerabilities and Exposures*, Retrieved June, 20, 2012, from http://cve.mitre.org.

[8] MOPAS, "Guidelines on building and operating Information Systems," *MOPAS Notification No.2012-25*, June 2012.

[9] T. Boland and P. E. Black, "Juliet 1.1 C/C++ and JAVA test suite," *IEEE Computer Soc.*, pp.88-90, Oct. 2012.

[10] MITRE, *Comon Weakness Enumeration V2.4,* Retrieved Feb., 21, 2013, from http://cwe.mitre.org.

[11] J. Bang, R. Ha, J. Park, and P. Kang, "Minimum standard of weakness in development of reliable e-GOV software," in *Proc. KICS Int. Conf. Commun. 2012 (KICS ICC 2012)*, vol. 48, pp.127-128, Jeju Island, Korea, June 2012.

[12] J. Bang and R. Ha, "Evaluation Methodology of Diagnostic Tool for Security Weakness of e-GOV Software," *J. KICS*, vol. 38C, no. 04, pp. 335-343, Apr. 2013.

방 지 호 (Jiho Bang)

1996년 2월 홍익대학교 컴퓨터공학과 졸업
2001년 8월 홍익대학교 컴퓨터공학과 석사
2004년 3월~현재 홍익대학교 컴퓨터공학과 박사과정
2001년 7월~2009년 7월 한국정보보호진흥원 선임연구원
2009년 7월~현재 한국인터넷진흥원 책임연구원
<관심분야> 모바일/SW보안, 모바일컴퓨팅, 실시간 시스템 등

하 란 (Rhan Ha)

1987년 2월 서울대학교 컴퓨터공학과 졸업
1989년 2월 서울대학교 컴퓨터공학과 석사
1995년 4월 University of Illinois at Uubana-Champaign 전산학 박사
1989년 3월~1990년 7월 한국통신 전임연구원
1995년 9월~현재 홍익대학교 컴퓨터공학과 교수
<관심분야> 모바일컴퓨팅, 실시간시스템, SW보안 등