행정안전부 소프트웨어 보안 취약점 진단기준과 Java 웹 어플리케이션 대상 오픈소스 보안 결함 검출기 검출대상의 총체적 비교†

(Systematic and Comprehensive Comparisons of the MOIS Security Vulnerability Inspection Criteria and Open-Source Security Bug Detectors for Java Web Applications)

이 재 훈 ‡ 최 한 솔 § 홍 신 ¶
(Jaehun Lee) (Hansol Choe) (Shin Hong)

요 약 경쟁적이며 급진적으로 오늘날 소프트웨어 개발 산업 현장에 시큐어 코딩 방법론을 효과적으로 적용하기 위해서는 보안 취약점 결함을 자동으로 검출하는 결함 검출기의 효과적이고 효율적인 적용이 필수적이다. 본 논문은 Java 웹 어플리케이션을 대상으로 하여 우리 행정안전부가 정의한 42개의 보안 취약점 진단 기준과 총 323개의 오픈소스 보안 취약점 결함 검출기의 검출 대상 결함 패턴을 비교하여, 동일한 결함 패턴을 대상으로 하는 것이 무엇인지를 명시화한 결과를 소개한다. 조사 결과를 바탕으로, 본 논문에서는 현재 행정안전부 보안 취약점 진단 기준 방법론의 한계점, 오픈소스 보안 결함 검출 프레임워크 간의 결함 검출 범위의 비교, 그리고 시큐어 코딩 가이드라인에 기반 한 개발 보안 방법론의 발전 과제를 논의한다.

키워드 : 보안취약점, 시큐어 코딩, 정적 분석, 오픈소스 소프트웨어

Abstract To enhance effective and efficient applications of automated security vulnerability checkers in highly competitive and fast-evolving IT industry, this paper studies a comprehensive set of security bug checkers in open-source static analysis frameworks and how they can be utilized for source code inspections according to the security vulnerability inspection guidelines by MOIS. This paper clarifies the relationship between all 42 inspection criteria in the MOIS guideline and total 323 security bug checkers in 4 popular open-source static analysis frameworks for Java web applications. Based on the result, this paper also discuss the current challenges and issues in the MOIS guideline, the comparison among the four security bug checker frameworks, and also the ideas to improve the security inspection methodologies using the MOIS guideline and open-source static security bug checkers.

Keywords : Security vulnerability, Secure coding, Static analyzer, Open-source software

1. 서 론

오늘날 정부, 금융 기관 등 주요 사회 기반의 정보 시스템에서 모바일/웹 어플리케이션 형태의 사용자 인터페이스를 제공함에 따라 모바일/웹 어플리케이션 개발 중 발생하는 사소한 소프트웨어 결함이 시스템 전체의 치명적인 보안 문제로 이어지는 사례가 늘고 있다[1]. 이와 같은 모바일/웹 어플리케이션 개발에서의 보안 취약점 발생에 대한 효과적인 대응하기 위해 소프트웨어 개발 및 검토 과정에서 프로그래밍 요소의 안전한 사용 규칙을 규제/강제하는 시큐어 코딩 기반 개발보안 방법론이 실제 소프트웨어 산업 현장에서 적용되고 있는 추세이다. 우리 정부의 행정안전부는 2013년 소프트웨어 보안 취약점 진단 기준[2]을 발표하고 이에 기반 한 보안 전문가를 양성함으로써, 전자정부 소프트웨어의 코드를 보안 전문가가 수검토하여 잠제적인 보안취약점을 진단하는 방법론을 정립해 오고 있다.

시큐어 코딩 기반 개발보안 방법론을 개발 주기가 빠른 일반적인 소프트웨어 산업 현장에 효과적으로 적용하기 위해서는 수기적 검토를 대신, 자동화된 정

† 본 연구는 과학기술정보통신부의 소프트웨어중심대학 지원사업 (2017-0-00130)과 한국연구재단을 통한 신진연구자지원과제 (NRF-2017R1C1B1008159)의 지원을 받음.
‡ 학생회원 : 포항공과대학교 컴퓨터공학과 (본 논문을 작성할 당시 한동대학교 전산전자공학부 소속)
GSoJC234@gmail.com
§ 학생회원 : 한동대학교 전산전자공학부
hansolchoe@handong.edu
¶ 종신회원 : 한동대학교 전산전자공학부 (교신저자)
hongshin@handong.edu
논문접수 : 2019년 3월 9일
심사완료 : 2019년 5월 8일

적분석(static analysis) 기반 보안취약점 결함 검출기(bug checker) 사용이 필수적이다. 하지만 보안취약점 검출을 자동적이고 체계적으로 제공하는 상용 도구의 경우, 사용에 많은 비용이 소요되어 일반적인 모바일/웹 어플리케이션 개발상황에서 활용이 비교적 제한적인 실정이다.

상용도구가 가지는 접근성을 해결하는 방법으로 오픈소스 보안취약점 자동검출 도구를 활용한 방법론을 생각해 볼 수 있다. FindBugs[3], PMD[4]와 같은 오픈소스 보안취약점 자동검출 도구를 우리 정부가 정립한 보안 취약점 진단 가이드라인을 중심으로 한 시큐어 코딩 방법론에 효과적으로 사용하기 위해서는 다음 두 가지 기술적 난점이 해결되어야 한다:

(1) 현재 우리 정부의 소프트웨어 보안 취약점 진단이 보안 취약점으로 진단하는 대상과 오픈소스 결함 검출기 간의 검출 대상 연관 관계가 명시적으로 파악되어있지 않음

(2) 검출 대상이 유사한 다수의 오픈소스 결함검출기의 검출 대상 사이의 연관 관계가 명시적으로 파악되어 있지 않음

이러한 문제로 인해, 현재 현업에서는 특정한 종류의 보안 취약점 진단을 위해 어떠한 오픈소스 결함 검출기를 사용해야 하는지 판단하기 어렵다. 또한 복수 개의 오픈소스 결함 검출기를 사용할 경우, 검출대상이 달라 검출 범위를 효과를 증대하는지 혹은 검출대상이 중복되어 실제적인 효과가 없는 지를 판별하기 어려운 실정이다.

본 논문은, 오늘날 웹 어플리케이션 개발에 널리 사용되는 Java 프로그램에 대하여 우리 정부가 정의한 42개 보안 취약점 진단 기준이 보안 취약점으로 정의하는 결함과 오픈소스로 개발된 총 323개의 결함 검출기가 결함으로 검출하는 대상을 간의 대응 관계를 명시적으로 조사한 연구 결과를 소개한다. 또한, 조사연구를 바탕으로 현재 개발된 보안 취약점 진단 기준 기반 방법론의 약점을 검토하고, 이를 보완하기 위한 방법을 논의한다. 본 연구는 궁극적으로 소프트웨어 산업 현장에서 자동화 도구를 활용한 개발보안 증진을 기대한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 개관한 후, 3장에서 본 연구의 조사 대상인 우리 정부의 보안 취약점 진단 기준과 오픈소스 결함 검출기를 소개한 후, 상호 간의 대응 관계를 밝힌 결과를 소개한다. 4장에서는, 3장에서 소개한 조사연구 결과를 기반으로 개발보안 방법론에 오픈소스 도구의 활용 방안과 개선을 논의한 후, 5장에서는 본 연구의 결론을 소개함으로써 논문을 마무리한다.

2. 관련 연구

2.1. 보안 취약점 진단 기준 정립과 효과적인 적용 방법에 대한 연구

국내 IT산업 환경에서 전자정부 시스템 등 높은 신뢰성이 요구되는 소프트웨어 개발과 검수에서 활용할 수 있는 보안 취약점 진단 기준을 수립하기 위해, 시큐어 코딩 가이드라인을 기반으로 한 효과적인 모바일/웹 어플리케이션 개발보안 방법론에 대한 연구가 진행되었다. 2012년에는 기존에 제시되어 온 보안 취약점 목록을 바탕으로 국내환경에 적합한 소프트웨어 보안 취약점 최소 기준을 제시하는 연구가 발표되었다[5]. 또한 2015년에는 우리 정부의 소프트웨어 개발 보안을 위한 보안 취약점 표준 목록을 제안한 연구[6]가 발표되었다.

또한, 시큐어 코딩에 기반한 보안 취약점 진단 방법을 우리 IT 산업 현장에 효과적으로 적용하여 소프트웨어 보안 문제에 대응하기 위한 정책적/제도적 방안이 연구 되어 왔다. 보안 취약점의 심각성을 객관적으로 평가할 수 있는 정량 평가기준에 대한 연구[11], 국내 IT산업 환경에 적합한 진단도구 요구사항과 진단도구의 신뢰성을 보증할 수 있는 평가 방법론에 관한 연구[12], 미국의 개발보안 방법론 사례 연구[13], 시큐어 코딩중심으로 신뢰할 수 있는 안정적인 정보보호와 정보보안활동을 강화하는 방법 제안[14]과 보안 취약점 점검도구 확산과정에서 발생하는 문제를 정부에서 추진한 표준프레임워크 사업과 비교 점검하여 개선안을 제시[15]하는 등의 연구가 발표되었다.

2.2. 자동 결함검출기의 활용에 대한 연구

정확하고 효율적인 보안 취약점 진단을 위한 자동화 도구로 오픈소스 보안 결함 검출기를 활용하는 방안이 연구되어 왔다. 보안 취약점을 검출 대상으로 하는 오픈소스 결함 검출기의 동향에 대한 조사[7]로 한국인터넷진흥원은 오픈소스 도구인 FindBugs, FindSecurityBugs, PMD를 활용하여 행정자치부 47개 보안 취약점 진단 기준에 활용하는 방안을 소개하고, 진단방법과 89개 결함검출기 간의 연간 관계를 조사한 결과를 소개한 자료를 2016년에 발표했다[8]. 방지호 등(2013)의 연구결과에서는 보안 취약점 진단 기준과 연관된 결함검출기의 검출 성능을 평가하기 위해 테스트 입력으로 사용할 검증 대상 코드 집합을 개발한 결과를 발표하였다[9][10].

본 논문은, 앞서 소개한 관련 연구에 이어서 다음의 부분에서 새로운 결과를 보고 한다:

- 본 논문에서는 보안 취약점과 관련된 총 323개의 오픈소스 결함검출기의 검출 대상을 총체적이며, 구체적으로 조사/비교한 결과를 수행하였다(3.1절, 3.2절)
- 본 논문에서는 보안 취약점 진단방법과 결함검출

기 간의 검출 대상의 대응 관계는 물론, 결함검출기 간의 연관 관계도 조사하였다. 이 연구 결과를 소프트웨어 산업현장에서 오픈소스 결함검출기를 활용한 개발보안 방법론 적용에 활용할 수 있도록, 조사 결과 데이터를 체계적으로 조직화 하여 공개하였다(3.3절)

- 조사연구 결과를 바탕으로, 현재의 방법론과 도구가 가지는 한계점을 파악한 결과를 소개하며, 이를 바탕으로 구체적인 개선 방향을 제시하였다(4장)

3. 보안 취약점 진단 기준과 오픈소스 결함검출기의 연관 관계 조사

3.1. 조사의 목표와 대상

본 연구는 행정안전부가 발표한 보안 취약점 진단 기준[2]이 제시하는 총 47개 Java 프로그램 보안 취약점 진단 방법과 2018년 6월 현재 4종의 오픈소스 프로젝트를 통해 공개된 총 1310개의 보안 취약점 대상 결함검출기에 대하여, 다음 3가지 연관 관계를 명료화 하는 것을 목표로 하였다:

1. 특정 보안 취약점 진단 방법(이후 진단 방법)이 검출하고자 하는 보안 취약점 결함을 특정한 오픈소스 결함 검출기를 통해 검출할 수 있는가?
2. 특정 보안 취약점 결함검출기가 검출하는 대상은 어떠한 보안 취약점 진단 방법의 진단 대상에 속하는가?
3. 두 개의 보안 취약점 결함 검출기가 검출하고자 하는 대상이 같은가?

본 연구는 2013년 행정안전부(당시 안전행정부)가 발표한 보안 취약점 진단 기준[2]의 7개 카테고리의 총 47개의 보안 취약점 진단 방법 중 (1) Java 프로그램과 연관성이 없는 1개의 진단 방법과 (2) 보안 취약점에 한정되기 보다는 일반적인 결함 유형(예: Null Pointer Dereference)에 대해 정의한 4개의 진단 방법을 제외한 42개 진단 방법을 조사 대상으로 선정하였다.

또한, 본 연구는 Java 프로그램을 대상으로 한 오픈소스 프로젝트 형태의 보안 취약점 결함검출기를 총체적으로 조사하기 위하여, OWASP(오픈소스 웹 애플리케이션 보안 프로젝트)에 소개 된 총 41개의 결함검출기 프레임워크의 오픈소스 프로젝트 중 (1) Java언어를 지원하고, (2) 오픈소스 프로젝트로 개발되고 무료로 공개되었으며, (3) 결함검출기의 검출 대상에 대한 설명이 구체적으로 갖추어져 져있는 FindBugs[2]를 포함하는 FindSecurityBugs[15] (이후 FB+FSB), PMD [4], LAPSE+ [16], SonarQube [17]를 일차적 조사대상 프레임워크로 선정한 후, 선정된 4개의 프레임워크 내에 있는 총 1310개의 결함검출기의 결함검출 기능을 검토하여 보안 취약점과 연관성이 높은 총 323개의 결함검출기를 최종적 조사 대상으로 선정하였다.

표1(Table 10). Studied Bug Checkers

Platform (Version, Release Date)	#. Target Checkers (#. Total Checkers)
FindBugs (3.0.1) + FindSecurityBugs (1.7.1, 2017.11.20)	219 (549)
PMD (6.0.0, 2018.2.25)	34 (297)
LAPSE+ (2.8.1, 2012.2.11)	12 (12)
SonarQube (7.0, 2018.2.15)	58 (452)

표1(Table 29). Studied Bug Checkers

표 1(Table 1)은 본 연구에서 조사대상으로 선정한 4종의 오픈소스 도구가 제공하는 오픈소스 보안 취약점 결함 검출기의 개수이다. 참고를 위해, 각 프로젝트에 포함된 결함 검출기의 전체의 개수는 괄호 안에 표기하였다.

현존하는 오픈소스 보안 결함 검출기를 총체적으로 조사 범위에 포함시키기 위해서, 본 연구에서는 조사 대상으로 선정한 4개 프로젝트가 제공하는 결함 검출기 관련 문서, 결함 검출기 카테고리(분류 기준)을 면밀하게 검토하였다. 각 문서에는 특정 결함 검출기의 검출 대상, 작동 방식에 대한 문서를 제공하고 있으며, 각 도구별로 1개에서 10개의 카테고리로 결함검출기를 분류한 정보를 제공한다.

본 연구에서는 (1) 문서 에 보안과 연관성이 명시적으로 나타나 있는 결함검출기와 (2) 선정된 결함검출기와 같은 카테고리에 속한 결함검출기를 모두 선별하여, 포괄적인 분석이 될 수 있게 하였다.

3.2. 조사 방법

본 연구에서는 개발보안 관련 연구 및 소프트웨어 개발에 경험이 있는 1명의 대학원생과 1명의 학부생, 그리고 1명의 교수가 보안 취약점 진단 방법과 결함검출기의 설명, 관련 결함사례(예: 관련된 CVE, CWE 사례)를 정성적으로 분석하여, 검출 대상 결함이 같은 경우를 찾았다. 우선 모든 보안 취약점 점검 기준과 모든 결함검출기의 설명을 검토한 후, 각 보안 취약점 점검 기준을 중심으로 이와 연관된 결함검출기 집단을 정의하였다. 그 후, 같은 집단에 속한 두 검출기의 명세, 연관 결함 사례, 소스코드를 검토하여, 두 결함 검출기가 같은 대상을 결함으로 검출하는 경우를 찾았다. 이 때, 보안 취약점 진단 방법과 결함검

출기는 기본적으로 다대다 연관관계를 가질 수 있으며, 결함검출기 간에도 다대다 연관관계를 가질 수 있음을 전재로 조사를 수행하였다.

조사 과정에서, 대부분의 경우, 3명의 판별이 동일하여 연관성 판별이 명확하였다. 다만, 행정안전부의 보안 취약점 진단 기준에서 제시한 진단 방법에서 보안 취약점에 대한 설명은 일반적이나, 사례로 제시한 예제가 특정 상황에 국한되어 검출대상의 연관 여부에 합의가 되지 않는 9개의 경우가 있었다. 이 경우, 보안 취약점 진단 기준의 일부 설명에서 편협하게 한정된 부분을 확장하는 방향으로 해석하여, 최종적인 연관성 판별을 결정하였다(4.2절 참조).

조사 과정을, 보안 취약점 진단 기준 중 하나인 '경로 조작 및 자원 삽입'의 사례를 예로 들어 자세히 설명하겠다. '경로 조작 및 자원 삽입'은 사용자가 웹 어플리케이션의 입력에 악의적인 문자열을 삽입할 경우, 웹 서버가 의도하지 않은 파일을 접근하게 되는 결함을 뜻한다. 행정안전부의 보안 취약점의 점검 기준에서는 CWE-22, CWE-99를 관련 결함 사례로 제시하며, 해당 결함을 아래와 같이 간단히 기술하고 있다[8]:

"검증되지 않은 외부입력값을 통해 파일 및 서버 등 시스템 자원에 대한 접근 혹은 식별을 허용할 경우, 입력 값 조작을 통해 시스템이 보호하는 자원에 임의로 접근할 수 있는 보안 약점".

본 조사에서는 CWE-22, CWE-99를 언급하거나 혹은 보안 취약점 점검 기준에서 언급된 '경로(path)', '자원 삽입(injection)'의 키워드가 등장하는 모든 결함 검출기를 일차적으로 나열하였으며, 각 결함 검출기의 동작을 검토하여 FindBugs에서 10개, LAPSE+에서 1개로 총 11개의 관련 결함 검출기 집단을 정의하였다.

이들 11개 검출기는 정적 오염 분석(taint analysis)을 바탕으로 보안 결함을 검출하며, 각 검출기 별로 어떠한 함수 호출을 오염 발생 지점(source)과 오염 도달 지점(sink)로 지정하느냐에 따라 검출 대상이 달라짐을 확인할 수 있었다. 각 결함 검출기의 명세, 소스코드를 분석하여 오염 발생 지점, 오염 도달 지점을 분석한 결과, FindBugs의 'PATH_TRAVERSAL_IN'(표2의 19 행)와 LAPSE+의 'Path Traversal'(표 2의 29행) 검출기는 JDK의 File, RandomAccessFile, FileReader 객체의 메소드를 동일한 대상을 오염 도달 지점으로 정의하므로, 두 결함 검출기의 검출 대상이 같음을 확인할 수 있었다(표2의 19행과 29행의 2열에 나타난 결함 검출기 번호가 같음). 반면, 'PATH_TRAVERSAL_OUT'은 오염 도달 지점으로 FileWriter 객체의 메소드 등 다른 결함 검출기와 구별되는 결함을 검출하므로, 관련 결함 검출기 집단 중 중복되는 검출기가 없는 것으로 판별하였다(표2의 20행 결함 검출기 번호가 고유함).

3.3 조사 결과

본 연구에서는 앞서 선장한 42개의 보안 취약점 진단방법과 323개의 오픈소스 결함검출기 중 동일한 종류의 결함을 검출하는 경우를 파악했다.

표2(Table 2)는 총 42개의 보안 취약점 진단 기준 중 동일한 보안 결함을 검출하는 오픈소스 결함 검출기가 1건 이상 있는 32개 경우에 대하여, 각 진단 기준이 어떠한 결함 검출기와 동일한 범주의 결함을 검출하는 지를 대응한 결과다. 표2는 총 238행을 3단 병행으로 표현한 것으로, 병합된 열로 표현된 행은 보안 취약 진단 기준의 이름을 표시하고 있으며, 그 아래 3열은 각각 결함 검출기 프레임워크의 이름(F는 FindBugs, L은 LARSE+, P는 PMD, S는 SonarQube를 지칭함), 결함 패턴 고유 번호, 결함 검출기 명칭을 지칭하고 있다. 결함 패턴 고유 번호는 검출기의 검출 대상 결함 패턴이 상이한 경우, 고유한 번호를 부여하였다(1~211). 서로 다른 결함 검출기이지만 동일한 고유 번호를 부여 받은 경우는, 해당 결함 검출기가 동일한 결함 패턴을 검출하는 것으로 판단되는 경우로, 고유 번호 위에 별표(*)로 표시하였다. 3열에 표시한 결함검출기 이름이 길 경우 말줄임표로 간단히 표시하였다. 표2의 데이터는, 관련 URL을 포함하여, 공개 소프트웨어 저장소에 공개하여 일반에 활용할 수 있도록 하였다.

다음 10개 보안 취약점 진단 기준은 총 323개의 오픈소스 결함 검출기 중에 대응되는 검출기가 1건도 발견되지 않았고, 따라서 표2에 표시되지 않았다(자세한 논의는 4.1절 참고):적절한 인증 없는 중요기능 허용, 부적절한 인가, 중요한 자원에 잘못된 권한 설정, 중요 정보 평문 저장, 취약한 비밀번호 사용, 주석문에 포함된 시스템 주요정보, 솔트 없이 일방향 해시함수 사용, 반복된 인증시도 제한기능 부재, DNS lookup에 의존한 보안결정.

총 323개의 조사대상 결함 검출기 중 1개 이상의 보안 취약점에 대응되는 경우는 총 285건이었으며, 이 중 동일한 결함 검출 대상을 가지는 결함 검출기를 고려할 경우, 이들은 총 211종의 고유한 결함 패턴을 검출하였다. 총 323개의 조사대상 결함 검출기 중 38개는 42개의 보안 취약점 진단 기준 중 어디에도 대응되지 않아 표2에 표시되지 않았다(자세한 논의는 4.2절과 4.3절 참고).

4. 관찰 및 논의

본 장에서는 3장에서 소개한 결과를 관찰하여 행정안전부가 발표한 보안 취약점 진단 기준의 제한점, 효과적인 활용 방안, 그리고 발전을 위한 개선 방향을 논의한다.

표3(Table 3)은 3장의 결과에 대한 관찰과 논의를 위해, 표2(Table 2)의 내용을 행정안전부 결함 검출 기준을 중심으로 요약한 데이터다. 표3은 상하로 크게 두 부분으로 나누어지는데 1~44행까지는 보안 취약점 진단 방법과 결함검출기 사이의 연관관계 있는 경우에 대한 결과이며, 45~50행은 연관된 행정안전부 보안 취약점 진단 기준이 없는 오픈소스 결함 검출기에 대한 결과이다. 표3의 1~44행 부분에는 7열이 있는데, 첫 번째 열과 두 번째 열은 보안 취약점 점검방법의 커테고리와 구체적인 방법 이름을 보여준다. 세 번째 열에서 여섯 번째 열은 각각 FB+FSB(F), PMD(P), LAPSE+(L), SonarQube(S)에 해당 검토방법과 연관된 결함검출기의 개수 나타낸다. 이 때 괄호 안의 숫자는 해당 보안 취약점 점검 방법에 직접해당 되지는 않으나, 관련성이 높은 결함검출기의 개수이다(4.2절 참고). 마지막 열은, 앞선 4개의 열에서 보고하는 결함검출기 중 중복된 것을 제외한, 결함검출 의도가 다른 결함검출기의 숫자이다. 표3의 45~50번째 행은 연관된 보안 취약점 진단방법이 없는 결함검출기에 대한 정보를 보고한다. 이 영역은 총 6개의 행으로 나누어졌으며, 첫 번째 행은 본 연구에서 명명한 결함 패턴 이름이며, 나머지 행은 앞서 설명한 바와 같이, 4개의 결함검출기 프레임워크에서 해당 결함패턴에 대해 존재하는 결함 검출기의 숫자를 나타낸다.

4.1 자동 진단을 지원하는 결함 검출기가 없는 보안 취약점 진단 방법

표3에서 마지막 열이 0인 9개 취약점 진단 방법은 현존 하는 오픈소스 결함 검출기 중 대응되는 경구가 없는 경우로, 오픈소스 도구를 통한 자동적인 진단을 지원하는 방법이 없는 것으로 관찰 되었다. 이에 해당하는 보안 취약점 진단방법은 다음과 같다:

- 적절한 인증 없는 중요기능 허용 (표3의 17행)
- 부적절한 인가 (표3의 18행)
- 중요한 자원에 잘못된 권한 설정 (표3의 19행)
- 중요 정보 평문 저장 (표3의 21행)
- 취약한 비밀번호 사용 (표3의 26행)
- 주석문에 포함된 시스템 주요정보(표3의 29행)
- 솔트 없이 일방향 해시함수 사용(표3의 30행)
- 반복된 인증시도 제한기능 부재 (표3의 32행)
- DNS lookup에 의존한 보안결정(표3의 43행)

해당 보안 취약점 진단 방법에 대한 결함 검출기가 존재하지 않는 이유를 파악하기 위해, 해당 보안 취약점 진단 기준을 정성적으로 분석하였다.

분석 결과, 'DNS lookup에 의한 보안결정'의 경우, 해당 결함 검출 기준을 바탕으로 패턴 기반 결함 검출기 개발이 가능하나, 아직 이루어지지 않은 상황으로 파악되었다. 객관적이고 효율적인 보안 취약점 진단을 위해, 'DNS lookup에 의한 보안결정'을 FindSecurityBugs와 같은 오픈소스 정적 분석기 프레임워크 내에 결함 검출기로 개발하여 활용성이 높은 도구로 확보하는 것이 유용할 것으로 보인다.

나머지 8개 보안 취약점 진단 기준의 경우, 다음 2 가지 원인으로 인해 연관된 결함 검출기의 개발이 어려웠던 것으로 파악되었다:

① 동적 정보가 필수적인 보안 취약점 진단 기준. 아래의 3 가지 진단과정 중 코드 정보 이외에 어플리케이션 별 기능을 고려한 동작(동적) 정보가 필수적으로 요청되어, 결함검출기 개발이 원천적으로 어려운 것으로 판별 된다:

- 적절한 인증 없는 중요기능 허용
- 부적절한 인가
- 반복된 인증시도 제한기능 부재

예를 들어, '적절한 인증 없는 중요기능 허용'의 경우, 프로그램의 코드만을 가지고, 사용자 인증이 된 상태인지, 안 된 상태인지 일반적으로 알 수 없으므로, 코드 정보 외에 프로그램 실행을 고려한 분석이 필요하다. 이와 같은 진단 방법의 경우로, 코드 정보만을 활용하는 정적분석 기반 결함검출기의 설계가 원천적으로 어려우므로, 시큐어 코딩 가이드라인만으로는 정확한 검침이 어려운 상황이다. 해당 보안 취약점을 정확하게 진단하기 위해서는, 코드 정보에 기반한 진단 기준 이외에 프로그램 동적 정보나 보안 테스팅 정보를 활용하는 개발보안 방법론의 적용이 요청되는 주제로 보인다(4.6절).

② 취약점 여부 판별 기준이 모호한 진단 기준. 아래의 5개 진단 기준은 보안 취약점 판별 조건이 모호하게 작성되었거나, 특정 세부사항에 한정되어 편협하게 개발되어, 객관적인 검출 기준을 마련하기 어렵고, 따라서 관련된 결함 검출기를 특정하기 어려운 경우로 판별 되었다:

- 중요한 자원에 대한 잘못된 권한 설정
- 중요정보 평문 저장
- 취약한 비밀번호 저장
- 주석문 안에 포함된 시스템 주요정보
- 솔트 없이 일방향 해쉬 함수 사용

표2(Table 11). Mapping from MOIS Inspection Criteria to Corresponding Security Bug Checkers
(F: FindBugs+FindSecurityBugs, P: PMD, L: LARSE+,S: SonarQube)

		SQL 삽입
F	1	CUSTOM_INJECTION
F	2*	SQL_INJECTION
F	3	SQL_INJECTION_TURBINE
F	4	SQL_INJECTION_HIBERNATE
F	5	SQL_INJECTION_JDO
F	6	SQL_INJECTION_JPA
F	7	SQL_INJECTION_SPRING_JDBC
F	8	SQL_INJECTION_JDBC
F	9	SCALA_SQL_INJECTION_SLICK
F	10	SCALA_SQL_INJECTION_ANORM
F	11	SQL_INJECTION_ANDROID
F	12	AWS_QUERY_INJECTION
F	13	SQL_NONCONSTANT_STRING…
F	14*	SQL_PREPARED_STATEMENT…
L	2*	SQL Injection
S	14*	SQL binding mechanims should …
		경로조작 및 자원삽입
F	15*	PATH_TRAVERSAL_IN
F	16	PATH_TRAVERSAL_OUT
F	17	SCALA_PATH_TRAVERSAL_IN
F	18	STRUTS_FILE_DISCLOSURE
F	19	SPRING_FILE_DISCLOSURE
F	20	REQUESTDISPATCHER_FILE…
F	21	EXTERNAL_CONFIG_CONTROL
F	22	BEAN_PROPERTY_INJECTION
F	23	PT_ABSOLUTE_PATH_TRAVERSAL
F	24	PT_RELATIVE_PATH_TRAVERSAL
L	15*	Path Traversal
		크로스사이트 스크립트
F	25*	XSS_REQUEST_WRAPPER
F	26	JSP_JSTL_OUT
F	27	XSS_JSP_PRINT
F	28	XSS_SERVLET
F	29	ANDROID_GEOLOCATION
F	30	A.W.V_JAVASCRIPT
F	31	A.W.V_JAVASCRIPT_INTERFACE
F	32	HTTPONLY_COOKIE
F	33	SCALA_XSS_TWIRL
F	34	SCALA_XSS_MVC_API
F	35	XRP_TO_JSP_WRITER
F	36	X.R.P_TO_SEND_ERROR
F	37	X.R.P_TO_SERVLET_WRITER
L	25*	Cross-Site Scripting(XSS)
		운영체제 명령어 삽입
F	38*	COMMAND_INJECTION
F	39	SCALA_COMMAND_INJECTION
L	38*	Command Injection
S	38*	Values passed to OS command …
		위험한 형식 파일 업로드
F	40	WEAK_FILENAMEUTILS
F	41	FILE_UPLOAD_FILENAME
6. 신뢰되지 않는 URL 주소로 자동접속 연결		
F	42*	UNVALIDATED_REDIRECT
F	43	PLAY_UNVALIDATED_REDIRECT
F	44	SPRING_UNVALIDATED_REDIRECT
L	42*	URL Tampering
		XQuery 삽입
F	45	XMLStreamReader
F	46	XXE_SAXPARSER
F	47	XXE_XMLREADER
F	48*	XXE_DOCUMENT
F	49	XXE_DTD_TRANSFORM_FACTORY
F	50	XXE_XSLT_TRANSFORM_FACTORY
F	51	XML_DECODER
F	52	JSP_XSLT
F	53	MALICIOUS_XSLT
L	48*	XML Injection
		XPath 삽입
F	54*	XPATH_INJECTION
L	54*	Xpath Injection
		LDAP 삽입
F	55*	LDAP_INJECTION
F	56	LDAP_ANONYMOUS
F	57	LDAP_ENTRY_POISONING
L	55*	LDAP Injection
S	55*	Values passed to LDAP queries …
		크로스사이트 요청 위조
F	58	S.C_PROTECTION_DISABLED
F	59	S.C_UNRESTRICTED_REQUEST…
F	60	SCALA_PLAY_SSRF
F	61	URLCONNECTION_SSRF_FD
P	62	NoUnsanitizedJSPExpression
		HTTP 응답분할
F	63*	HTTP_RESPONSE_SPLITTING
L	64*	HRS_REQUEST_PARAMETER…
	64*	Header Manipulation
	63*	HTTP Response Splitting
		정수형 오버프로우 [Extended]
F	65	BAD_HEXA_CONVERSION
P	66	BadComparison
S	67	Double.longBitsToDouble
		보안기능 결정에 사용되는 부적절한 입력값
F	68*	SERVLET_PARAMETER
F	69	SERVLET_CONTENT_TYPE
F	70	SERVLET_SERVER_NAME
F	71*	SERVLET_SESSION_ID
F	72	SERVLET_QUERY_STRING
F	73	SERVLET_HEADER
F	74*	SERVLET_HEADER_REFERER
F	75	SERVLET_HEADER_USER_AGENT
F	76	HTTP_PARAMETER_POLLUTION
F	77	TRUST_BOUNDARY_VIOLATION
L	78	Cookie Poisoning
L	68*	Parameter Tampering
S	71*	HttpServletRequest.getRequested…
S	74*	HTTP referers should not be…
S	79	Untrusted data should not be …
		포맷 스트링 삽입
F	80	FORMAT_STRING_MANIPULATION
		취약한 암호화 알고리즘 사용
F	81	WEAK_MESSAGE_DIGEST_MD5
F	82*	WEAK_MESSAGE_DIGEST_SHA1
F	83	SSL_CONTEXT
F	84*	CUSTOM_MESSAGE_DIGEST
F	85	HAZELCAST_SYMMETRIC…
F	86*	NULL_CIPHER
F	87*	DES_USAGE
F	88	TDES_USAGE
F	89	RSA_NO_PADDING
F	90	ECB_MODE
F	91	PADDING_ORACLE
F	92	ESAPI_ENCRYPTOR
F	93	CIPHER_INTEGRITY
S	87*	NeitherDES nor DESede…
S	94	Cryptographic RSA algorithms …
S	86*	"javax.crypto.NullCipher" …
S	84*	Only standard cryptographic…
S	95	Pseudorandom number generators…
S	82*	SHA-1 and Message-Digest hash…
		중요정보 평문 전송
F	96	DEFAULT_HTTP_CLIENT
F	97	UNENCRYPTED_SOCKET
F	98	UNENCRYPTED_SERVER_SOCKET
F	99*	INSECURE_COOKIE
F	100	INSECURE_SMTP_SSL
F	101	URL_REWRITING
S	99*	Cookies should be secure
		하드코드된 비밀번호
F	102	HARD_CODE_PASSWORD
F	103*	DMI_CONSTANT_DB_PASSWORD
F	104	DMI_EMPTY_DB_PASSWORD
S	103*	Credentials should not be…
		충분하지 않은 키 길이 사용
F	105	BLOWFISH_KEY_SIZE
F	106	RSA_KEY_SIZE
		적절하지 않은 난수값 사용
F	107	PREDICTABLE_RANDOM
F	108	PREDICTABLE_RANDOM_SCALA
S	109	"SecureRandom" seeds should …
		하드코드된 암호화 키
F	110	HARD_CODE_KEY
S	111	IP addresses should not be …
		사용자에게 저장되는 쿠키를 통한 정보노출
F	112	COOKIE_USAGE
F	113	COOKIE_PERSISTENT
F	114	HRS_REQUEST_PARAMETER_TO…
		부적절한 검사 없는 코드 다운로드
F	115	USP_INCLUDE
S	116	Classes should not be loaded ….
		경쟁조건(TOCTOU)
F	117	AT_OPERATION_SEQUENCE_ON_…
F	118	DC_DOUBLECHECK
F	119	DC_PARTIALLY_CONSTRUCTED
F	120	D.S.O_BOOLEAN
F	121	D.S.O_UNSHARED_BOXED_…
F	122	D.S.O_SHARED_CONSTANT
F	123	D.S.O_UNSHARED_BOXED_…
F	124	DM_MONITOR_WAIT_ON_…
F	125	DM_USELESS_THREAD
F	126	ESync_EMPTY_SYNC
F	127	IS2_INCONSISTENT_SYNC
F	128	IS_FIELD_NOT_GUARDED
F	129	JLM_JSR166_LOCK_MONITORENTER
F	130	JLM_JSR166_UTILCONCURRENT…
F	131	JML_JSR166_CALLING_WAIT…
F	132	LI_LAZY_INIT_STATIC
F	133	LI_LAZY_INIT_UPDATE_STATIC
F	134	ML_SYNC_ON_FIELD_TO_GUARD…
F	135	ML_SYNC_ON_UPDATED_FIELD
F	136	MWN_MISMATCHED_NOTIFY
F	137	MWN_MISMATCHED_WAIT
F	138	NN_NAKED_NOTIFY
F	139	NO_NOTIFY_NOT_NOTIFYALL
F	140	RS_READOBJECT_SYNC
F	141	RV_RETURN_VALUE_OF…
F	142	RU_INVOKE_RUN
F	143	SC_START_IN_CTOR
F	144	SP_SPIN_ON_FIELD
F	145	STCAL_INVOKE_ON_STATIC_…
F	146	STCAL_INVOKE_ON_STATIC_DATE_…
F	147	STCAL_STATIC_CALENDAR_INST…
F	148	STCAL_STATIC_SIMPLE_DATE…
F	149	SWL_SLEEP_WITH_LOCK_HELD
F	150*	ITLW_TWO_LOCK_WAIT
F	151	UG_SYNC_SET_UNSYNC_GET
F	152	UL_UNRELEASED_LOCK
F	153	UL_UNRELEASED_LOCK_EXCEPTI…
F	154	UW_UNCOND_WAIT
F	155	VO_VOLATILE_INCREMENT
F	156	VO_VOLATILE_REFERENCE…
F	157	WL_USING_GETCLASS_RATHER…
F	158	WS_WRITEOBJECT_SYNC
F	159	WA_AWAIT_NOT_IN_LOOP
F	160	WA_NOT_IN_LOOP
P	161	AvoidSynchronizedAtMethodLevel
P	162	AvoidUsingVolatile
P	163	DoubleCheckedLocking
P	164	NonThreadSafeSingleton
P	165	UnsynchronizedStaticDateFormatter
P	166	UseConcurrentHashMap
S	150*	"wait" should not be called…
S	167	Value-based classes should not be…
S	168	"getClass" should not be used for…
S	169	Getters and setters should be…
S	170	Non-thread-safe fields should not…
S	171	Blocks should be synchronized on…
S	172	".equals()" should not be used to…
S	173	Synchronization should not be…
		종료되지 않은 반복문 또는 재귀함수
F	174	IL_CONTAINER_ADDED_TO_ITSELF
F	175*	LI_INFINITE_LOOP
F	176	LI_INFINITE_RECURSIVE_LOOP
P	177	EmptyWhileStmt
S	175*	Loops should not be infinite
S	178	Double-checked locking should …
S	179	Locks should be released
		오류메시지를 통한 정보노출
S	180	Throwable.printStackTrace(…) shoul…
		오류 상황 대응 부재
P	181	AvoidInstanceofChecksInCatch…
P	182	AvoidLiteralsInIfCondition
P	183	CloneThrowsCloneNotSupported…
P	184	DoNotExtendJavaLangThrowable
P	185	EmptyCatchBlock
P	186	ReturnFromFinallyBlock
S	187	Exceptions should not be thrown…
S	188	"SingleConnectionFactory" …
S	189	"Iterator.next()" methods should…
S	190	Return values should not be ignored…
S	191	Exception should not be created…
		부적절한 예외 처리
F	192	DE_MIGHT_DROP
F	193*	DE_MIGHT_IGNORE
P	193*	AvoidCatchingNPE
P	193*	AvoidLosingExceptionInformation
P	194	UseCorrectExceptionLogging
P	195	DoNotThrowExceptionInFinally
S	193*	"InterruptedException" should not…
		잘못된 세션에 의한 데이터 정보노출
F	196*	MSF_MUTABLE_SERVLET_FIELD
P	197	StaticEJBFieldShouldBeFinal
S	198	Members of Spring components…
S	196*	Servlets should not have mutable …
		제거되지 않고 남은 디버그 코드
S	199	Web applications should not have…
		Public 메소드로부터 반환된 Private 배열
F	200*	EI_EXPOSE_REP
F	201	MS_EXPOSE_REP
S	200*	Mutable members should not be…
		Private 배열에 Public 데이터 할당
F	202	EI_EXPOSE_REP2
		취약한 API
F	203	DM_EXIT
F	204	DM_RUN_FINALIZERS_ON_EXIT
P	205	AvoidThreadGroup
P	206	DoNotUseThreads
P	207*	DontCallThreadRun
P	208	ProperCloneImplementation
P	209	UseNotifyAllInsteadOfNotify
P	210	UseProperClassLoader
S	211	"File.createTempFile" should not be…
S	207*	Thread.run() should not be called…

예를 들어, '중요정보 평문 저장'의 경우 프로그램에서 저장하는 수많은 정보들 중, 중요한 정보가 무엇인지를 일반적으로 판별하는 기준을 세우기 어렵다. '솔트 없이 일방향 해쉬 함수 사용'의 경우, 해쉬 함수를 사용한 상황에 따라 보안 취약점을 발생시킬 수도 있고, 그렇지 않을 수도 있으므로 진단 기준이 모호하다. 이와 같은 진단 방법의 경우, 과학적이고 체계적인 진단 기준을 정의하기 어렵고, 따라서 결함검출기와 같은 자동화 기법의 도입이 어렵다. 해당 보안 취약점 진단 기준에 대응되는 자동화된 결함 검출기를 개발하기 위해서는, 실제 결함 사례를 중심으로 하여 진단기준을 구체화하는 개선이 필수적으로 보인다. 예를 들어, '중요정보 평문 저장'의 경우 중요정보를 URL, 특정 패턴의 문자열로 구체화하는 접근이 가능할 것으로 보인다.

4.2 진단 기준이 편협하여 개선이 필요한 보안 취약점 진단 방법

표3의 괄호 안에 표시한 9개의 결함 검출기는, 해당 보안 취약점 진단 기준이 특정 하는 진단 대상 결함에 대한 설명에 따를 경우 대응되지 않으나, 진단 기준의 편협한 조건을 개선할 경우 대응 관계를 찾을 수 있는 경우이다. 해당 9개 결함 검출기가 연관된 6개 보안 취약점 진단 기준의 개선 방향은 다음과 같이 논의할 수 있다:

- '크로스사이트 요청 위조'의 경우, 크로스사이트 요청 위조와 본질적으로 유사한 기존 진단 기준에서는 Server Side Request Forgery (SSRF) 결함검출기를 포함하도록 확장
- '정수형 오버플로우'의 경우, 타입 변환(예: int를 double로)에서 발생하는 버그 등 기본형 타입의 예외적 연산에 대한 확장이 필요함
- '보안기능 결정에 사용되는 부적절한 입력 값' 의 경우, 부적절한 값 조건은 물론 입력 값의 속성에 위험요소가 없는 지 검사하는 결함검출기를 포괄하도록 확장
- '취약한 암호화 알고리즘의 사용'은 암호화 알고리즘을 사용의 부적절한 사용으로 인한 결함패턴을 아우르도록 확장
- '중요정보 평문전송'의 경우, 중요정보의 다양한 종류(예: URL, IP 주소, 해쉬 키)를 포괄할 수 있도록 확장
- '하드코드된 암호화 키'의 경우, 암호화 키 외에도 하드코드가 되었을 경우 위험성이 있는 정보(예: IP 주소)를 고려하도록 확장

표3(Table 3). Associating MOIS Security Vulnerability Weakness Inspection Methods and Security Vulnerability Bug Checker (F: #. checkers in FindBugs and FindSecurity Bugs, P: #. checkers in PMD, L: #. checkers in Larse-Plus, S: #. checkers in SonarQube, Tot the sum of F, P, L and S)

MOIS Security Vulnerability Weakness Inspection Method		Bug Checkers				
Categ	**Security Vulnerability Issue**	**F**	**P**	**L**	**S**	**Tot**
입력 데이터 검증 및 표현	SQL 삽입	14	0	1	1	16
	경로 조작 및 자원 삽입	10	0	1	1	12
	크로스사이트 스크립트	13	0	1	0	14
	운영체제 명령어 삽입	2	0	1	1	4
	위험한 형식 파일 업로드	2	0	0	0	2
	신뢰되지 않는 URL 자동접속연결	3	0	1	0	4
	XQuery 삽입	9	0	1	0	10
	Xpath 삽입	1	0	1	0	2
	LDAP 삽입	3	0	1	1	5
	크로스사이트 요청 위조	4 (2)	1	0	0	5
	HTTP 응답분할	2	0	2	0	4
	정수형 오버플로우	1 (1)	1 (1)	0	1 (1)	3
	보안기능 결정에 부적절한 입력값 사용	10 (1)	0	2	3	15
	포맷 스트링 삽입	1	0	0	0	1
보안 기능	적절한 인증 없는 중요기능 허용	0	0	0	0	0
	부적절한 인가	0	0	0	0	0
	중요한 자원에 잘못된 권한 설정	0	0	0	0	0
	취약한 암호화 알고리즘 사용	13 (1)	0	0	6	19
	중요정보 평문 저장	0	0	0	0	0
	중요정보 평문 전송	6 (1)	0	0	1	7
	하드코드된 비밀번호	3	0	0	1	4
	충분하지 않은 키 길이 사용	2	0	0	0	2
	적절하지 않은 난수값 사용	2	0	0	1	3
	취약한 비밀번호 사용	0	0	0	0	0
	하드코드된 암호화 키	1	0	0	1 (1)	2
	저장된 쿠키를 통한 정보노출	3	0	0	0	3
	주석문에 포함된 시스템 주요정보	0	0	0	0	0
	솔트 없이 일방향 해시함수 사용	0	0	0	0	0
	무결성 검사 없는 코드 다운로드	1	0	0	1	2
	반복된 인증시도 제한 기능 부재	0	0	0	0	0
시간, 상태	경쟁조건: 검사 시점과 사용 시점	44	6	0	8	58
	종료되지 않는 반복문/재귀함수	3	1	0	3	7
에러 처리	오류 메시지를 통한 정보노출	0	0	0	1	1
	오류 상황 대응 부재	0	6	0	5	11
	부적절한 예외 처리	2	4	0	1	7
캡슐화	잘못된 세션에 의한 데이터 정보노출	1	1	0	2	4
	제거되지 않고 남은 디버그 코드	0	0	0	1	1
	시스템 데이터 정보노출	1	0	0	0	1
	Public 메서드의 Private 배열 반환	2	0	0	1	3
	Private 배열에 Public 데이터 할당	1	0	0	0	1
API 오용	DNS lookup에 의존한 보안결정	0	0	0	0	0
	취약한 API 사용	2	6	0	2	10
Uncovered Security Vulnerability Issues		**F**	**P**	**L**	**S**	**Tot**
Misuse of Cryptology Algorithms		4	0	0	0	4
Incorrect Use. Framework Features		13	0	0	6	19
Overconsumption. System Resource		1	0	0	2	3
Code Injection		10	0	0	0	10
Log Injection		2	0	0	0	2

앞서 논의한 6개 보안 취약점 진단 기준의 경우, 현행 보안 진단 기준이 특정 하는 보안 취약점 결함 이외에도, 해당 취약점과 연관성이 높은 보안 취약점 결함 패턴이 추가로 존재하는 경우다. 따라서 해당 보안 취약점 진단 기준을 보다 포괄적인 보안 취약점 검출이 가능하도록 확장하는 개선이 요청된다.

4.3 현재 보안 취약점 진단 기준에는 없으나, 결함검출기가 탐지하는 보안 취약점 결함 패턴

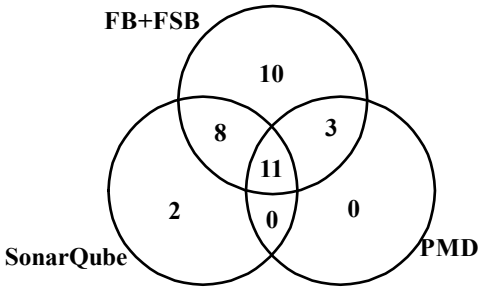
표3의 46~50행에서는 현재 행정자치부 보안 취약점 진단 기준과는 무관하나 오픈소스 결함 검출기가 개발된 총 38개의 결함 검출기를 5가지 유형으로 분류한 결과를 소개하고 있다. 이들 38개 검출기의 검출 대상을 조사한 결과, 관련된 보안 취약점이 2013년 이후에 정리되었으며, 이들 대부분이 2017년 이후에 보고된 보안 취약점에 착안하여 개발된 것을 확인할 수 있었다. 이에 해당하는 5가지 보안 결함 유형에 대한 설명은 다음과 같다:

- 'Misuse of Cryptology Algorithm'은 암호화 알고리즘의 잘못된 사용으로 인해 효과적인 암호화가 실패하는 보안 취약점 결함
- 'Incorrect Use of Framework Feature'는 Spring, Android과 같은 프레임워크의 고유한 개발 규칙 위반에 따른 보안 취약점(예: 특정 메서드의 접근 제어자를 보안에 취약하게 설정)
- 'Overconsumption of System Resource'는 시스템 자원을 무리하게 사용하여 다른 사용자들이 시스템 자원을 사용할 수 없게 되는 취약점
- 'Code Injection'은 악의적인 코드가 프로그램 내부에서 실행되는 결함으로, 예를 들어, 무분별한 deserialization으로 인해 공격자의 코드가 프로그램 내부에서 실행되도록 허용하는 취약점
- 'Log Injection'은 프로그램 실행결과 발생하는 로그를 조작하여 침입 분석과 같은 보안활동을 방해 하는 보안 취약점에 관한 패턴임

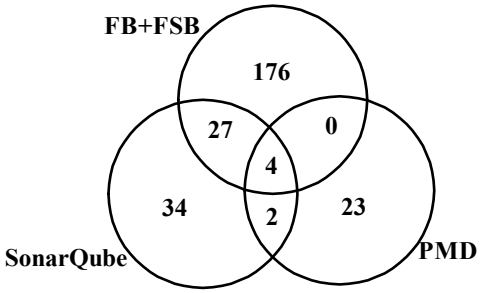
앞서 설명한 5종의 보안 결함 유형은 실제 산업현장에서 새롭게 파악된 보안 취약점으로, 이들이 보안 취약점 진단 과정에서 검출 될 수 있도록 취약점 진단 기준의 확장이 요청된다.

4.4 보안 취약점 결함검출기 프레임워크 간 비교

그림1(Figure 1)은 표2의 데이터를 바탕으로, 본 논문의 논의 대상 중 3개의 오픈소스 결함검출기 프레임워크의 소속 결함검출기의 검출 대상 범위를 보안 취약점 진단방법의 개수(그림1(a))과 결함 검출기의 개수(그림1(b))를 기준으로 비교한 결과이다. 밴다이어그램의 각 영역은, FB+FSB, PMD, SonarQube의 각 조합이 1개 이상의 결함검출기로 연관된 진단방법의 수(그림1(a) 혹은 해당 결함검출기의 수(그림1(b))를 나타낸다. 그림에 나타내지 않은 LAPSE+의 경우, 제공하는 12 결함검출기가 FB+FSB에 모두 완전히 포함되어, 그림1에는 표현하지 않았다.



(a) Number of MOIS Review Criteria



(b) Number of Static Bug Checkers

그림 1(Figure 1). Comparison of Target Coverage Among Three Open-source Static Bug Checker Frameworks

그림1(a)에서 확인할 수 있는 바와 같이, 조사 대상 오픈소스 결함 검출 프레임워크 중에서는FB+FSB가 가장 많은 수의 진단 방법과 연관된 결함검출기를 제공하고 있으며, SonarQube와 연관된 2개의 진단기준을 제외하고 대부분의 다른 프레임워크의 진단방법 범위를 포함하고 있음을 알 수 있다.

그림1(b)를 볼 때, 상이한 두 프레임워크가 같은 보안 취약점 진단방법과 연관된 결함검출기를 가지고 있다고 하더라도, 결함검출기의 기능이 일치하는 경우는 전체의 12.8%(=34/265)로, 나머지 87.2% 결함검출기는 고유한 결함 패턴을 검출 대상으로 하고 있음을 알 수 있다.

따라서, 보안 취약점 점검에 시간/자원이 한정된 상황에서는 보안 취약점 진단방법 커버리지를 고려하여 FB+FSB와 SonarQube 중 일부(FB+FSB가 검사하지 못하는 점검방법)의 우선적 적용을 생각해 볼 수 있다. 이에 더하여, 진단방법 내에도 결함검출기

별로 검출을 목표로 하는 결함 유형이 다를 수 있으므로, 본 연구에서 파악한 결함검출기 간의 상관관계를 고려하여 총 265개의 고유한 결함검출기 전체를 활용한 보안 취약점 진단을 고려해볼 수 있다.

자원의 제약(예: 시간, 인력)이 적은 경우에는, 두 결함검출기가 같은 결함 유형을 검출하는 연관성이 높은 결함검출기로 판단된다고 하더라도, 각 구현에 있어서 구체적인 사례에서 탐지 결과가 다를 수 있으므로(예: 오탐률), 여러 개의 결함검출기 프레임워크를 모두 수행한 후, 본 연구 결과가 파악한 연관관계를 바탕으로 상호 간 결과의 정합성을 비교할 경우, 보다 정확하고 총체적인 보안 취약점 진단 결과를 얻을 수 있을 것으로 기대된다.

4.5 개발 보안 방법론 개선을 위한 제안

3장의 조사 결과와 4장의 앞선 논의를 종합해 볼 때, 현재 개발보안 방법론의 한계를 개선하고 위한 방법으로 다음과 같은 발전 과제를 발견할 수 있다:

① **동적 정보를 활용하는 보안 테스트 가이드라인의 개발: 4.1절에서 지적하고 있는 바와 같이, 다수의 보안 취약점은 일반적인 코드 패턴보다, 어플리케이션의 동작에 기반 한 검토가 필수적이다. 이를 효과적으로 지원하기 위해서는 시큐어 코딩 기반의 방법론에 더하여, 검증대상 프로그램에 대한 테스팅을 수행하며, 동적 정보를 활영하여 보안 취약점을 체계적으로 보안 테스트 가이드라인의 개발이 필요한 것으로 보인다.**

② 신규로 발견되는 보안 취약점이 진단 기준에 지속적으로 반영될 수 있도록 하는 시스템 개발: 4.2절과 4.3절에서 지적하고 있는 바와 같이, 소프트웨어 개발 기술의 빠른 변화에 따라 새로운 보안 취약점이 지속적으로 발생하고, 이에 대한 대응책 역시 빠른 속도로 개발되고 있다. 하지만 현재와 같은 형식의 보안 취약점 진단 기준은 2013년 이후 상황에 능동적인 반영에 한계가 있다. 국내 소프트웨어 개발 환경에서 새로운 보안 취약점 점검기준의 지속적인 발전을 위해서는, 보안 취약점 보고를 수집, 연계하는 온라인 색인 시스템(예: CVE, CWE)이 필요하며, 이를 통해 국내 IT산업의 개발자가 개발 보안에서의 새로운 이슈를 용이하게 접근할 수 있도록 지원할 필요가 있는 것으로 보인다.

③ 보안 취약점 진단기준의 활용에 필요한 오픈소스 형태의 결함검출기 및 활용 방법 개발: 산업체 환경에서 보안 취약점의 효과적인 진단을 위해서는 오픈소스 기반의 결함검출기의 활용이 절대적으로 필요하다. 이러한 필요에 효과적으로 응답하기 위해서 기존에 개발된 진단기준 적용에 필요한 결함검출기를 FindSecurityBugs와 같은 오픈소스 프레임워크 상에서 추가 개발하여, 공개하고 해당 프레임워크를 활용한 자동 진단 방법론의 보완이 필요할 것으로 보인다. 또한, 4.4절의 논의와 같이, 현업에서도 복수 개의 오픈소스 결함 검출 프레임워크를 복합적으로 활용함으로써 효과적이면서도 효율적으로 시큐어 코딩 방법론을 적용하는 방법론 구축을 지원할 필요가 있다.

5. 결론

본 논문에서는 Java 프로그램에 대하여 동작하는 총 323종의 오픈소스 보안 취약점 결함검출기와 행정안전부가 2013년 발표한 보안 취약점 진단 방법 간의 대응 관계를 총체적이고 구체적으로 식별한 연구 결과를 소개한다. 조사연구 결과를 통해서 특정 결함검출기가 탐지하는 보안성 결함이 어떠한 보안 취약점 진단 방법과 대응되는지를 파악할 수 있으며, 또한 같은 보안 취약점 결함을 탐지하는 서로 다른 결함검출기 간의 관계를 구체적으로 파악하였다. 또한, 조사연구에서의 관찰을 바탕으로 오픈소스 결함검출기를 행정안전부 보안 취약점 진단 기준 방법론에 적용하는데 있어서의 문제점을 구체적으로 소개하고 개발보안 방법론 개선 방안을 제안하였다.

참 고 문 헌

[1] US-CERT, OpenSSL 'Heartbleed' Vulnerability (CVE-2014-0160), https://www.us-cert.gov/ncas/alerts/TA14-098A

[2] Ministry of the Interior and Safety, Guide of Validating Software Security Weakness for e-Government Software Validators, 2013

[3] FindBugs, https://findbugs.sourceforge.net

[4] PMD, https://pmd.github.io

[5] Jiho Bang, Rhan Ha, Jung Whan Park, Pil Young Kang, Minimum Standard of Weakness in Development of Reliable e-GOV Software, Proceedings of Symposium of the Korean Institute of Communications and Information Sciences, 2012

[6] Joonseon Ahn, Eunyoung Lee, Byeong-Mo Chang, A Study on Security Weakeness for Secure Software Development (SW 개발보안을 위한 보안약점 표준목록 연구), Journal of Korea Institute of Information Security and Cryptology

[7] Jiho Bang, Trend in Open-source Security Vulnerability Detection Tools (공개용 소스코드 보안약점 분석도구 개발 동향), Internet and Security Focus, Korea Internet & Security Agency, May

2014

[8] Ministry of the Interior and Safety, Manual on Validating Security Issues Using Open Source Tools for Software Developers and Validators (전자정부 SW 개발자, 진단원을 위한 공개SW를 활용한 소프트웨어 개발보안 진단가이드), 2016

[9] Jiho Bang, Rhan Ha, Validation Test Codes Development of Static Analysis Tool for Secure Software, Journal of the Korean Institute of Communication Sciences, 38 (5), 2013

[10] Jiho Bang, Rhan Ha, Comparing Open Source Static Security Analysis Tools based on Software Weakness, Proceedings of Korea Computing Congress, June 2013

[11] Joonseon Ahn, Ji-ho Bang, Eunyoung Lee, Quantitative Scoring Criteria on the Importance of Software Weaknesses, Journal of the Korea Institute of Information Security & Cryptology, 22 (6), Dec. 2012

[12] Jiho Bang, Rhan Ha, Evaluation Methology of Diagnostic Tool for Security Weakness of e-GOV Software, The Journal of the Korean Institute of Communication Sciences, 38(4), Apr. 2013

[13] Yanghwan Park, Minkyung Kim, Policy of Secure Coding for Secure e-Government Software Development (전자정부 소프트웨어의 보안성 강화를 위한 개발보안 제도 연구), Review of KIISC, 26(1), Feb. 2016

[14] Kilho Lee, Information Security Enhancement Focusing On Secure Coding, Proceedings of the KIISE Winter Conference, Dec. 2016

[15] Sukjin Kang, Jinyoung Choi, A Study on the Spread of Inspection Tools for the Secure Coding Culture, Proceedings of the KIISE Winter Conference, Dec. 2016

[17]FindSecurityBugs,https://find-sec-bugs.github.io

[18] LAPSE+, https://www.owasp.org/index.php/OWASP_LAPSE_PROJECT

[19] SonarQube, https://sonarqube.org

이 재 훈
2019년 한동대학교 전산전자공학부(학사)
2019년~현재 POSTECH 컴퓨터공학과 석사과정

최 한 솔
2018년 한동대학교 전산전자공학부(학사)
2018년 ~ 현재 한동대학교 전산전자공학부 석사과정

홍 신
2007년 KAIST 전산학부(학사)
2010년 KAIST 전산학부(석사)
2015년 KAIST 전산학부(박사)
2016년 ~ 현재 한동대학교 전산전자공학부 조교수