Journal of the Korea Academia-Industrial cooperation Society
Vol. 19, No. 12 pp. 863-871, 2018

https://doi.org/10.5762/KAIS.2018.19.12.863
ISSN 1975-4701 / eISSN 2288-4688

정적 및 동적 분석을 이용한 크로스 체크기반 취약점 분석 기법

송준호[1], 김광직[2], 고용선[2], 박재표[3*]
[1]숭실대학교 컴퓨터학과, [2]숭실대학교 IT정책경영학과, [3]숭실대학교 정보과학대학원

A Cross-check based Vulnerability Analysis Method using Static and Dynamic Analysis

Jun-Ho Song[1], Kwang-Jik Kim[2], Yong-Sun Ko[2], Jae-Pyo Park[3*]
[1]**Department of Computer Science and Engineering, Soongsil University**
[2]**Department of IT Policy Management, Soongsil University**
[3]**Graduate School of Information Science, Soongsil University**

요 약 본 논문에서는 기존의 취약점 분석 도구들의 미탐지, 오탐지, 과탐지를 발생시켜 정확한 취약점 탐지를 어렵게 하는 문제점을 해결하고 분석 대상이 되는 어플리케이션의 위험도를 평가하여 안전한 어플리케이션을 개발하거나 관리할 수 있는 정적 및 동적 분석을 이용한 크로스 체크기반의 취약점 탐지 기법을 제안한다. 또한 각각의 취약점이 가지고 있는 자체 위험도를 계산하고 정확도를 높인 취약점 탐지 기법을 바탕으로 최종적인 어플리케이션의 위험도를 평가, 제시함으로서 안전한 어플리케이션의 개발 및 운영을 돕는다. 제안하는 기법은 정적 분석 및 동적 분석 기법을 사용하는 도구들의 상호작용을 통해 각 기법의 단점들을 극복하여 취약점 탐지 정확도를 향상시킨다. 또한 기존의 취약점 위험도평가 시스템은 취약점 자체 위험도에 대해서만 평가하였으나, 제안하는 위험도 평가는 취약점 자체 위험도와 탐지 정확도를 복합적으로 반영하여 어플리케이션이 얼마나 위험에 노출되어 있는지를 평가한다. 제안하는 기법은 CWE에서 SANS top 25의 상위 10위 항목을 기준으로 기존의 분석 도구들과 탐지 가능한 목록, 탐지 정확도를 비교분석하였으며, 기존의 취약점 위험도에 대한 정량적 평가 시스템과 제안하는 어플리케이션 위험도 평가 결과를 비교 분석 및 평가하였다. 제안하는 기법으로 프로토타입 분석 툴을 구현하여 실험을 통해 어플리케이션의 취약점을 분석하였을 때, 기존의 분석 도구들의 취약점 탐지 능력보다 우수한 것으로 나타났다.

Abstract Existing vulnerability analysis tools are prone to missed detections, incorrect detections, and over-detection, which reduces accuracy. In this paper, cross-checking based on a vulnerability detection method using static and dynamic analysis is proposed, which develops and manages safe applications and can resolve and analyze these problems. Risks due to vulnerabilities are computed, and an intelligent vulnerability detection technique is used to improve accuracy and evaluate risks under the final version of the application. This helps the development and execution of safe applications. Through incorporation of tools that use static analysis and dynamic analysis techniques, our proposed technique overcomes weak points at each stage, and improves the accuracy of vulnerability detection. Existing vulnerability risk-evaluation systems only evaluate self-risks, whereas our proposed vulnerability risk-evaluation system reflects the vulnerability of self-risk and the detection accuracy in a complex fashion to evaluate relative. Our proposed technique compares and analyzes existing analysis tools, such as lists for detections and detection accuracy based on the top 10 items of SANS at CWE. Quantitative evaluation systems for existing vulnerability risks and the proposed application's vulnerability risks are compared and analyzed. We developed a prototype analysis tool using our technique to test the application's vulnerability detection ability, and to show that our proposed technique is superior to existing ones.

Keywords : Vulnerability analysis, Degree of risk, Vulnerability detection, Accuracy of detection, Application

*Corresponding Author : Jae-Pyo Park(Soongsil Univ.)
Tel: +82-2-820-0270 email: pjerry@ssu.ac.kr
Received September 14, 2018 Revised October 11, 2018
Accepted December 7, 2018 Published December 31, 2018

1. 서론

개인용 컴퓨터의 사용과 인터넷 사용이 증가함에 따라 사이버 공격으로 인한 피해는 계속해서 커지고 있다. 사이버 공격을 방어하고 피해를 최소화하기 위해 많은 보안 솔루션들을 도입하여 방어체계를 갖추고 있지만 피해사례와 피해규모는 더욱 커져가고 있다. 계속적으로 증가하는 사이버 공격의 주된 이유 중 약 75%가 소프트웨어 자체적인 취약점에 기인한 것이다[1]. 세계 웹 사이트의 60%이상이 사용하고 있는 오픈소스 OpenSSL의 Heart Bleed 버그가 그 대표적인 예이다. 또한 Microsoft는 보안 운영체제를 개발하는 프로젝트에서 개발 초기 단계부터 보안 요소를 고려하면 보안 요소를 고려하지 않고 개발하여 보안 패치 등과 같은 조치를 취하는 것보다 많은 개발 비용을 절약할 수 있다는 보고를 하였다[2].

취약점을 탐지하기 위한 기법으로는 소스코드를 기반으로 하는 정적 분석 방법과 실행 시스템을 기반으로 하는 동적 분석 방법이 있지만, 이들 기존 방법의 한계로 인해 오탐지, 미탐지, 과탐지 등이 빈번히 발생되고 있다[3].

일반적으로 취약점을 탐지하기 위해서 정적 분석 기법과 동적 분석 기법을 사용하는데 이를 적용한 분석 도구들의 단점, 즉 과탐지, 오탐지, 미탐지가 존재한다. 본 논문에서는 이 단점들을 해결하기 위한 정적분석과 동적분석을 혼용한 취약점 분석 기법을 제안한다. 제안하는 취약점 분석 기법은 기존의 분석 도구들을 이용하여 탐지 정확도를 높이기 위해 분석 도구들 간의 탐지 결과를 상호 분석하도록 설계하였으며, 상호 분석된 결과를 이용하여 재확인함으로 정확도를 높이는 기법이다. 제안하는 기법은 어플리케이션의 취약점을 진단하는 컨설턴트들이 취해야 하는 점검 활동들을 자동으로 처리할 수 있는 장점이 있다.

2. 관련 연구

2.1 취약점 관리 체계

ISO(International Organization for Standardization) 27005는 취약점을 “하나 이상의 위협에 의해 Exploit될 수 있는 자산 또는 자산들의 그룹의 약점”이라고 정의하고 있으며, 국제 인터넷 표준화 기구(IETF : Internet Engineering Task Force)에서는 취약점을 “시스템의 디자인, 구현 또는 작업 그리고 관리에서의 결함이나 약점으로서, 시스템의 보안 정책을 침해하기 위해 Exploit될 수 있는 것”으로 명시하고 있다[4][5].

CVE는 사이버 보안 취약점에 대한 데이터베이스로 취약점이 발생했을 때 발생년도, 발생순서 등을 명시하여 관리하고 있다. 미국의 국가 기관 및 여러 회사들(Apple, MS 등)이 함께 참여하고 있으며, 지금은 미국 국토안보부(U.S Department of Homeland Security)에서 관리하고 있다. CVE는 1999년에 비영리 기업인 MITRE 사에서 시작되었으며, 미국 NIST(National Institute of Standards and Technology, 국립 표준기술연구소)와 협력하기 시작하면서 체계적으로 관리되기 시작하였다. 취약점이 발견되면 MITRE에 등록 요청을 하고 요청받은 CVE는 등록 후보들에 대해 자체 논의를 거쳐 CVE-ID를 부여한다[6].

CWSS는 소프트웨어가 가지고 있는 보안상의 약점들을 정량적으로 평가하는 체계로 미국 정부와 학계, 산업계에서 함께 공동으로 연구를 진행하고 있다. CWSS는 정량적인 평가를 위해서 3가지 영역의 점수를 각각 계산한다. 3가지 영역은 Base Finding Metric 그룹, Attack Surface Metric 그룹, Environmental Metric 그룹으로 구성되어 있다. 이러한 3가지 영역들은 총 16개의 세부적인 항목들을 가지고 있으며, 각 영역별 중간 점수를 계산하고 중간 점수들의 곱으로 최종적인 CWSS의 점수를 계산한다. 현재의 CWSS는 1.0.1 버전이며, 2014년에 최종적으로 업데이트되었다[7].

2.1 취약점 분석 기법

정적 분석 기법은 어플리케이션에 내제되어 있는 취약점을 분석하기 위하여 취약점으로 발전 가능성을 가진 소스 코드들의 형태를 분석한다. 정적 분석 기법은 완성된 어플리케이션에 대해 내제된 취약점을 탐지하기 위해서 사용되기도 하지만 어플리케이션을 개발하는 시점에서 취약점을 발생시킬 수 있는 코드로 판명된 패턴, API 등을 검사할 수 있도록 하며, 해당되는 예외처리 등이 잘 되어 있는지 판단하여 취약점 발생을 미연에 방지할 수 있다. 따라서 완성된 어플리케이션에 대한 유지, 보수비용을 줄일 수 있는 방법으로 많이 사용된다[8]. 정적 분석 기법으로 사용되는 방법들은 패턴 매칭 기법이 대표적인 방법이며, 패턴 매칭 기법은 규칙으로 설정된 모든

패턴을 취약점으로 분류하여 탐지하기 때문에 주석으로 처리된 부분들도 탐지하는 경우가 발생된다. 따라서 과탐지의 가능성이 동적 분석 기법에 비해 상대적으로 많다는 것이 단점이다[9].

동적 분석 기법은 소스 코드가 없어도 취약점을 분석할 수 있는 방법으로 실제로 실행되고 있는 어플리케이션에 값을 입력하여 도출되는 결과를 통하여 취약점을 분석하는 기법이다[10]. 동적 분석 기법은 정적 분석 기법이 할 수 없는 기능적인 결함에 대해서 탐지할 수 있다. 하지만 취약점을 탐지하기 위해서 입력 값을 생성하는데 지식과 노하우가 필요하며, 많은 입력 값을 통해 취약점을 탐지하기 위해서 많은 프로세스가 필요하다는 단점이 있다[11].

3. 크로스 체크기반 취약점 분석 기법

3.1 취약점 분석 전체 프로세스

제안하는 취약점 분석 기법은 크게 5단계로 분류된다. 그림 1은 제안하는 기법의 전체 구조이다. 1단계는 기본 취약점 분석 단계로 기존의 정적 분석 도구와 동적 분석 도구를 활용하여 취약점을 탐지한다.

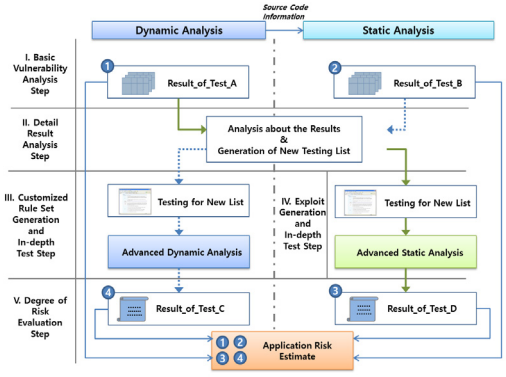


Fig. 1. The whole process of the proposed method

2단계는 기본 취약점 분석 결과를 분석하는 단계로 1단계의 결과를 분석하여 3단계와 4단계에서 활용할 수 있도록 한다. 3단계는 맞춤형 룰 셋 생성 및 심화테스트 단계로 2단계에서 생성된 정보를 이용해서 심화된 정적 분석을 수행한다. 이 때 사용하는 정보는 1단계 기본 취약점 분석 단계의 동적 분석 모듈에서 도출된 결과를 2단계에서 분석한 것이다. 4단계는 1단계의 정적 분석 모듈을 거치고 나온 결과로 2단계에서 이 정보를 이용하여 Exploit을 생성한 후 심화 테스트를 거치게 된다. 이렇게 하여 도출된 결과를 5단계에서 수집하여 해당 어플리케이션의 위험도를 평가한다. 5단계는 어플리케이션에서 검출된 취약점 리스트들을 리포팅하며, 취약점 리스트들의 탐지 정확도를 이용하여 어플리케이션의 위험도를 계산한다. 그림 2는 그림 1을 좀 더 구체적으로 나타낸 그림이며, 각 단계별(Ⅰ,Ⅱ,Ⅲ, Ⅳ, Ⅴ)로 진행되는 프로세스는 다음과 같다.

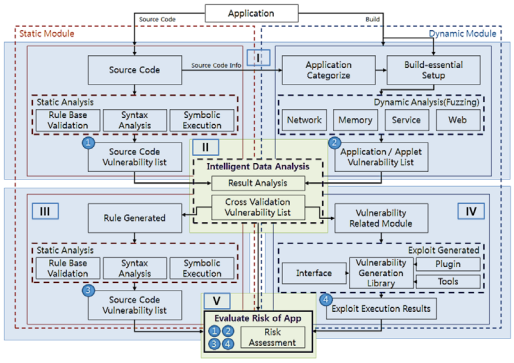


Fig. 2. The detail process of the proposed method

어플리케이션의 취약점을 분석하기 위해 어플리케이션의 코드를 static 모듈로 보낸다. static 모듈에서는 3가지 방법, 미리 셋팅된 룰(취약점 항목)을 이용하는 방법, 문법적인 오류나 보안상 취약점을 내제하고 있는 문법이 있는지 검사하는 방법, 기호실행 엔진을 이용한 방법을 이용하여 취약점을 분석한다. 이렇게 얻은 결과(①)는 2단계인 상세 데이터 분석단계와 위험도 분석단계로 보내진다. 또 다른 1단계인 Dynamic 모듈에서는 빌드되어진 어플리케이션을 실행하여 취약점이 존재하는지 분석한다. 동적 분석을 시작하기 전에 static 모듈에서 소스코드상 어떤 종류의 어플리케이션에 해당하는지 정보를 넘겨받는다. 동적 분석은 어플리케이션이 어떤 형태의 어플리케이션인지에 따라 네트워크, 메모리, 서비스, 웹 퍼징(web fuzzing)의 방법을 선택적으로 실시하여 취약점을 분석한다. 이렇게 도출된 결과(②)는 2단계인 상세 데이터 분석단계와 5단계인 위험도 분석단계로 보내진다. 2단계에서는 1단계의 결과(①,②)를 비교하여 공통적인 취약점으로 탐지된 취약점들을 필터링한다. 필터링된 취약점들은 어플리케이션에 내제되어 있는 것으로 판단하며, 이 리스트들을 5단계인 취약점 분석단계로 보내 취

약점의 정확도를 판단하는데 활용한다. 2단계에서 공통적으로 탐지된 취약점이 아닌 취약점들의 경우, 3단계와 4단계로 보내져 취약점 탐지의 정확도를 높이는 작업을 한다. Dynamic 모듈에서 탐지된 취약점 리스트들 중 공통으로 탐지되지 않은 리스트들은 3단계로 보내져서 한 번 더 취약점 분석을 실시한다. 이 때, 도출된 결과(③)는 5단계로 보내져서 위험도 분석하는데 사용된다. 또한 Static 모듈에서 탐지된 리스트들 중 공통으로 탐지되지 않은 취약점들은 4단계로 보내져서 동적분석을 한 번 더 실시한다. 이 때, 4단계로 보내어진 취약점 리스트들은 이미 취약점으로 의심되는 리스트들이기 때문에 해당 취약점이 맞는지 확인하기 위한 작업을 한다. 어플리케이션의 인터페이스를 통해 어떤 값들이 입력되었을 때, 어떤 기능이 실행되면 취약한 동작이 발현되는지 취약점 라이브러리를 통해 확인하고, 취약한 동작을 발현시키기 위한 툴 및 Plug-in을 이용하여 취약점을 발현시키기 위한 환경을 만들어 실제로 취약점이 맞는지 확인한다. 이 과정을 통해 취약점으로 판단되는 취약점들을 모아 4단계의 결과 리스트(④)를 작성한다. 이 결과 리스트 역시 5단계 위험도 분석 단계로 보내져서 어플리케이션 전체의 위험도를 계산한다. 5단계 위험도 분석 단계에서는 1,2,3,4 단계에서 보내온 결과 리스트를 이용하여, 검사 대상이 되었던 어플리케이션이 얼마나 위험한지를 최종적으로 판단한다.

3.2 취약점 분석 결과에 따른 상세 분석 프로세스

2단계인 상세 분석 단계에서는 1단계의 결과를 분석하여 3단계와 4단계의 취약점 분석을 위한 정보를 생성한다. 그림 3은 2단계의 프로세스를 보여준다. 기본 취약점 분석 단계를 거쳐 도출된 결과가 2단계에 전달되면 소스코드의 같은 위치에서 같은 취약점으로 탐지되었는지 분석한다.

취약점의 특성상 정적 분석 도구와 동적 분석 도구가 탐지할 수 없는 취약점과 탐지할 수 있는 취약점이 존재한다.

따라서 Result Categorization부에서는 두 모듈이 SANS 25의 상위 10개 항목을 공통으로 탐지할 수 있는 리스트와 어느 한쪽만 탐지할 수 있는 취약점 리스트를 작성한다. 정적 모듈과 동적 모듈. 즉 두 모듈에서 공통으로 탐지될 수 있는 리스트는 Mapping Available 모듈로 넘겨지고 함께 탐지할 수 없는 리스트는 Mapping Unavailable 모듈로 보내진다. 그림 4은 2단계의 Result Categorization부에서는 취약점들을 분류하는 프로세스이다.

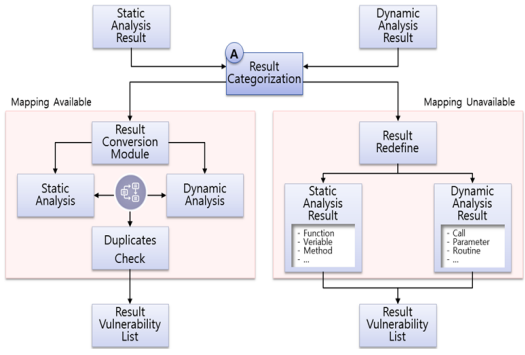


Fig. 3. Setting the rules

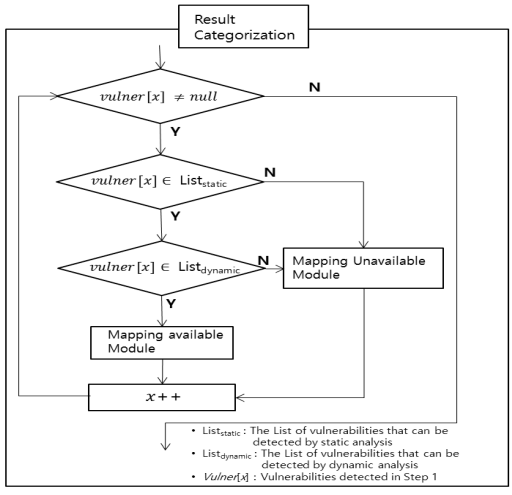


Fig. 4. The Process of Result Categorization

어플리케이션의 분석 결과가 같이 정적 모듈과 동적 모듈에서 함께 탐지된 항목이라면 Mapping Available 모듈로 분류한다.

만일 정적 모듈 또는 동적 모듈에서만 탐지할 수 있는 취약점들이라면 Mapping Unavailable 모듈로 분류한다. Mapping Unavailable 모듈로 분류된 취약점 항목들은 그림 5와 같이 1단계의 어느 모듈의 결과인지를 판단하여, 3단계와 4단계로 전달한다. 또한 5단계의 위험도 분석 단계에도 전달하여 추후 어플리케이션 위험도 평가에 반영한다.

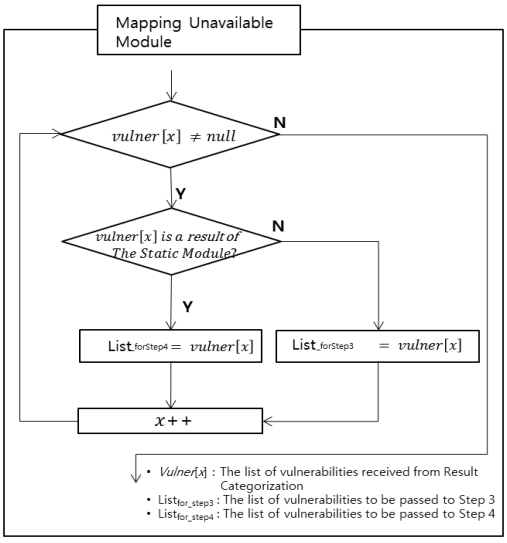


Fig. 5. The Process of Mapping Unavailable Module

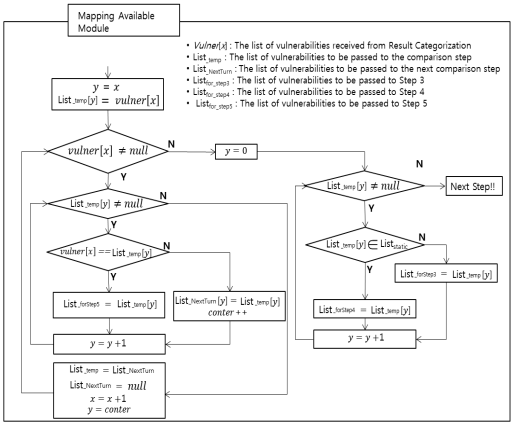


Fig. 6. The Process of Mapping Available Module

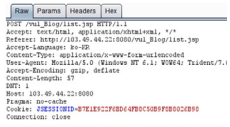
그림 6은 Mapping Available 모듈의 프로세스이다. Mapping Available 모듈로 분류되어진 취약점 리스트들 중에 정적 모듈로 탐지한 취약점과 동적 분석으로 탐지한 취약점이 같은 원인으로 발생한 취약점인지를 식별한다. 같은 원인으로 탐지된 취약점들은 탐지 리스트로 작성하여 5단계인 어플리케이션 위험도 분석 단계로 보내진다.

Mapping Available 모듈로 분류되어진 취약점 리스트들 중에 같은 원인으로 탐지되지 않은 취약점들은 과탐지와 오탐지의 확률이 있다. 이 취약점들은 취약점 탐지의 정확도를 높이기 3단계와 4단계의 과정, 즉 심화 테스팅 과정을 거친다. 따라서 같은 원인으로 탐지되지 않은 취약점들은 1단계 어느 모듈의 결과인지를 분석하여 그림 6과 같이 동적 모듈의 결과이면 3단계로, 정적 모듈의 결과이면 4단계로 전달하기 위한 리스트를 작성한다.

3.3 동적 모듈 결과와 정적 모듈 결과의 매칭

같은 종류의 취약점이 같은 원인으로 발생한 취약점인지를 판단하기 위해서 동적 모듈에서 취약점이 발생된 위치, 사용된 파라미터 값, 호출된 메서드 등이 사용된다. 정적 모듈에서는 소스코드를 기반으로 매칭 작업을 하게 된다. 표 1은 1단계 분석 결과에서 같은 원인으로 발생한 취약점을 찾는 과정을 XSS 취약점을 예로 들어 보여준다.

Table 1. The process of matching vulnerabilities of the same cause after dynamic and static analysis

Item	Dynamic Module	Static Module
Location of vulnerability	 http://localhost:8080/vul_Blog/list.jsp	C:\testcode\vul_Blog\src\main\WebContent\list.jsp 93 94 <%=src_value %> 95 <%
Parameter	 srcvalue	src_value -> srcvalue 22 23 String src_value = request.getParameter("srcvalue"); 24

동적 모듈에서 취약점의 발생을 'list.jsp'의 위치로 알려준다. 정적 모듈에서 발생한 취약점 중 맵핑이 되는 취약점점에서 'list.jsp' 파일에서 발생한 취약점이 있는지 파악하고 'list.jsp' 파일이 WebContent 디렉터리 밑에 있는지 파악한다. 그 후 동적 모듈에서의 결과는 파라미터 'srcvalue'를 지목하여 문제가 있다고 알려주고 있다. 따라서 정적 모듈에서 list.jsp 파일의 'srcvalue'라는 파라미터가 사용되는지 파악한 후 발생원인이 동적 분석에서와 같은 값을 갖도록 코딩되어 있는지 확인한다. 마지막으로 사용되는 파라미터가 같은 함수에서 사용되는 파라미터라면 같은 원인으로 탐지된 취약점이라고 확신할 수 있다.

이렇게 매칭이 완료된 취약점은 정확도가 아주 높은 취약점 탐지 리스트로 분류되어 5단계 어플리케이션 위험도 분석 단계로 보내진다.

3.4 맞춤형 룰 생성

2단계에서 보내오는 정적 모듈의 결과는 해당 취약점이 탐지된 위치정보(URL 등), 호출되는 메소드, 파라미터에 들어가는 값 등이다. 이 정보들은 소스코드가 어느 위치에 존재하며, 해당 함수에 어떤 데이터가 들어갔을 때 문제를 일으키는지에 대한 정보이다. 이러한 정보는 기본 분석 단계에서 탐지하지 못한 취약점을 소스코드로 확인할 수 있는 좋은 단서가 된다. 따라서 이 정보를 활용하여 정적 분석 도구로 소스코드를 확인할 수 있는 맞춤형 룰을 생성할 수 있다.

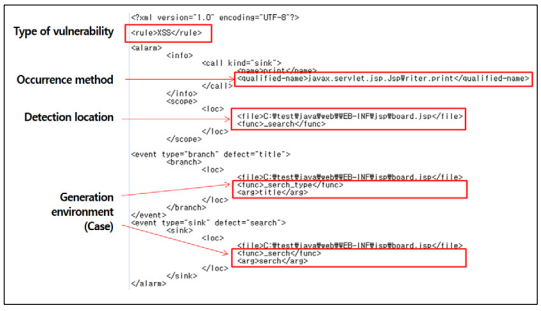


Fig. 7. Custom Rule Creation Example(XSS)

그림 7은 맞춤형 룰이 생성된 예시이다. 기본 분석 단계에서 XSS 취약점을 탐지하였고 발생경로에 있는 board.jsp로 보이는 페이지에서 '_search'함수에 'title'과 'search'라는 값이 입력되어 실행되었을 때 문제가 되는 메서드가 호출된다는 탐지를 주었다. 이 정보를 이용해서 정적 분석 모듈이 해당 파일의 해당 함수에서 XSS 취약점이 발생할 수 있는 소스코드 패턴이 있는지 확인하는 룰을 만들어 낼 수 있다. 이러한 맞춤형 룰은 특정한 패턴을 만들어 내는 것이 아니라 취약점이 발생한 곳을 재확인할 수 있도록 하는 것이다. 이렇게 생성된 룰을 이용하여 해당 부분만 정적 분석 모듈이 재확인하여 XSS를 탐지해 낼 수 있으면 3단계의 취약점 리스트로 작성하게 된다.

3.4 어플리케이션의 위험도 계산

어플리케이션의 최종 위험도 평가는 중간 점수에 탐지되는 취약점들의 발견 개수 및 빈도와 탐지된 취약점들이 어플리케이션에 미치는 위험도가 함께 반영되어 결정된다. 이러한 요소들을 반영하기 위해 취약점 자체 위험도 점수의 영역을 표 2과 같이 3가지로 분류하였다. 각 영역별 중간 점수 구간은 65점미만, 66점~85점, 85점 이상이며, 각 영역별로 가장 작은 점수 영역은 1~9사이의 숫자로 최종 위험도를 표현하며, 두 번째 영역은 10 ~ 90사이의 숫자로, 세 번째 영역은 100~200사이의 숫자로 취약점의 위험도를 표현하였다. 경험적으로 CWSS의 높은 점수를 가진 취약점이 한 개 존재하는 것이 낮은 점수의 취약점이 여러 개 있을 존재하는 것보다 더 위험하다는 판단에 의해 위험도 점수를 부여하였다. 즉, 위험도 점수는 중간 점수 구간의 값을 기반으로 하여 어플리케이션의 위험도를 직관적으로 더욱 쉽게 파악할 수 있도록 보정한 값이다.

Table 2. The ways to describe final vulnerability scores by sections of mid scores

Section of Mid Scores	Degree of Risk
Below 65	1st digit : 1 ~ 9
65 ~ 84	10th digit : 10 ~ 90
Above 85	100th digit: 100 ~ 200

1 ~ 9사이의 숫자로 표현하는 중간 점수 65점미만 구간을 위한 식은 중간 점수가 일 때, 식 1과 같다.

$$f_1(x) = \sum_{x<65}\left\{\frac{8*(x-50)}{15}+1\right\} \quad \text{(식 1)}$$

10 ~ 90사이의 숫자로 표현하는 중간 점수 65점 이상, 85점미만인 구간을 위한 식은 중간 점수가 x일 때, 식 2와 같다.

$$f_2(x) = \sum_{x<85}\left\{80*\left(\frac{(x-65)}{20}\right)+10\right\} \quad \text{(식 2)}$$

100 ~ 200사이의 숫자로 표현하는 중간 점수 85점 이상, 100점 이하인 구간을 위한 식은 중간 점수가 x일 때, 식 3과 같다.

$$f_3(x) = \sum_{x\le 100}\left\{100*\left(\frac{(x-85}{15}\right)+100\right\} \quad \text{(식 3)}$$

도출된 각 구간별 취약점의 위험도를 모두 합하여 해당 어플리케이션의 최종 위험도(R_{final})를 식 4와 같이 계산한다.

$$R_{final} = f_1(x) + f_2(x) + f_3(x) \quad \text{(식 4)}$$

상기한 각 구간별 취약점 위험도 계산 수식은 CWSS의 정량적 평가항목인 총 16개의 세부적인 항목들의 점수 계산 방식을 참고하여 취약점들의 발견 개수와 빈도와 탐지된 취약점들이 어플리케이션에 미치는 위험도가 함께 반영되어 본 제안 시스템에 적합하도록 수정 및 보완한 수식이다.

4. 구현 및 평가

4.1 시스템 구현

제안하는 기법을 테스트하기 위하여 프로토타입으로 취약점 분석 시스템을 구현하였다. 시스템은 동적 모듈과 정적 모듈로 구성되어 있으며, 동적 모듈의 도구들과 정적 모듈의 도구들은 Open Source를 이용하였다. 표 3은 정적 모듈과 동적 모듈에서 사용한 분석 도구들이다. 시스템에 업로드된 특정 오픈소스의 취약점 분석을 수행한 후 상세 결과를 보여준다.

Table 3. Vulnerability Analysis Tools used for Test bed

Static Module	Dynamic Module
PMD	Zap
	Wfuzz
Find Bugs	Wapiti
	SPIKE
	Scratch
	ROTOS
	peach puzzer

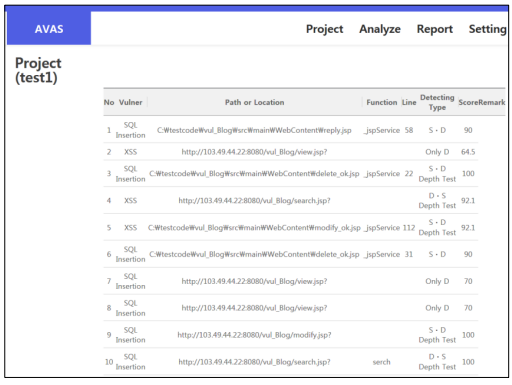


Fig. 8. Detail results for Application Vulnerability Analysis

그림 8은 구현된 취약점 분석 시스템의 취약점 분석 화면이다. 상세 결과에서 취약점이 발생되는 코드 또는 실행 중 취약점이 발생되는 기능의 상세 정보는 해당 취약점을 선택하면 상세히 볼 수 있도록 구현하였다.

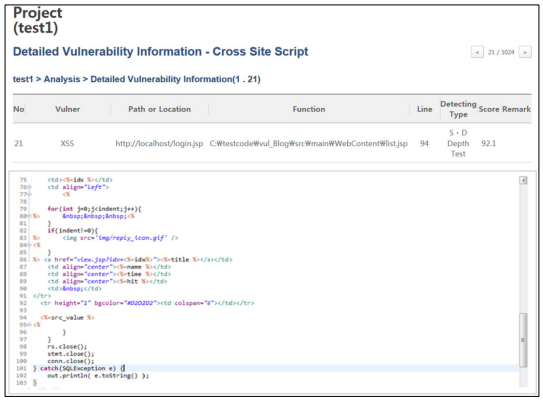


Fig. 9. Details of Detection Vulnerability based on Source Code

그림 9는 한 개의 취약점을 선택했을 때 나타나는 취약점의 상세 정보이다. 정적 분석으로 탐지된 코드를 자세히 볼 수 있도록 구현하였다. 또한 탐지 유형이 정적 모듈, 동적 모듈이 동시에 탐지한 경우이거나 정적 또는 동적 모듈의 재검증을 통해 취약점으로 탐지된 경우라면 해당 소스 코드를 빌드하여 소스코드가 구현한 기능의 실제 어플리케이션 분석 화면을 나타내어 취약점이 발생한 위치를 정확히 찾을 수 있도록 하였다.

4.2 실험 평가

실험환경은 CPU 쿼드코어 2.66GHz, RAM 4GB, Windows 7 운영체제를 설치하였고 하드 디스크 500G인 컴퓨터를 이용하여 분석 도구들을 설치하여 실험하였다.

표 5는 SANS 25의 상위 10개 항목에 대한 기존 정적 분석도구와 제안 기법의 전체 탐지 및 정탐지, 오탐지에 대한 결과를 나타낸 표이다. 버퍼 오버플로우의 취약점과 적절한 인증 없는 중요기능 허용에 대한 취약점은 해당 어플리케이션들에 취약점으로 내제하지 않았기 때문으로 판단된다. 제안하는 기법은 한 개의 항목을 제외하고 모든 항목에서 정탐의 개수가 많이 도출되어 기존의 정적 분석도구보다 취약점을 탐지하는 능력이 우수한 것으로 판단된다.

Table 5. Comparison of number of vulnerability detection when compared to existing static tool

No.	Top 10 of SANS 25	Existing Static Module			Proposed Method		
		Total	Proper Detect	False Detect	Total	Proper Detect	False Detect
1	SQL input	482	299	183	563	384	179
2	OS Command input	3	3	0	11	3	8
3	Buffer overflow	0	0	0	0	0	0
4	Cross site script	61	45	16	457	410	47
5	Allowance of critical functions without proper authorization	0	0	0	0	0	0
6	Inadequate authorization	14	14	0	18	18	0
7	Hard coded critical information	112	107	5	50	48	2
8	Clear text transmission and save of critical information	2	2	0	54	48	6
9	File uploads	2	2	0	9	8	1
10	Inappropriate input value used in security function decisions	16	11	5	19	14	5

Table 6. Comparison of vulnerable detection numbers when compared with existing dynamic module

No.	Top 10 of SANS 25	Existing Dynamic Module			Proposed Method		
		Total	Proper Detect	False Detect	Total	Proper Detect	False Detect
1	SQL input	19	19	0	563	384	179
2	OS Command input	11	3	8	3	3	0
3	Buffer overflow	0	0	0	0	0	0
4	Cross site script	98	68	30	457	410	47
5	Allowance of critical functions without proper authorization	0	0	0	0	0	0
6	Inadequate authorization	8	8	0	18	18	0
7	Hard coded critical information	0	0	0	50	48	2
8	Clear text transmission and save of critical information	23	17	6	54	48	6
9	File uploads	8	7	1	9	8	1
10	Inappropriate input value used in security function decisions	1	1	0	19	14	5

표 6은 SANS 25의 상위 10개 항목에 대한 기존 동적 분석도구와 제안 기법 취약점 탐지 개수를 나타낸다. 탐지되지 않은 두 개의 항목을 제외하고 모든 취약점 항목에서 정탐지한 취약점의 개수가 많은 것으로 나타났다. 따라서 기존 동적 분석도구의 취약점 탐지 능력보다 제안하는 기법의 취약점 탐지 능력이 더 우수한 것으로 판단된다.

5. 결론

본 논문에서는 어플리케이션에 내제되어 있는 취약점의 탐지 정확도를 높이기 위해서 기존의 분석도구를 통해 얻은 결과를 재검증하여 정확도를 높이는 정적 및 동적 분석을 이용한 크로스 체크기반의 취약점 분석 기법을 제안하였다. 맞춤형 정적 분석 룰을 생성하고, Exploit 코드를 생성하여 정적 모듈과 동적 모듈로 재검증함으로써 취약점의 탐지 정확도를 높일 수 있는 기법을 제안하였다. 또한 CWSS에서 제공하는 취약점 자체 위험도 점수와 본 논문에서 제안한 취약점 탐지 정확도를 이용하여 어플리케이션 위험도 평가 점수를 제시하였다.

실험 결과 제안하는 기법의 취약점 탐지 능력은 기존 정적, 동적 분석 도구의 취약점 탐지 능력보다 전반적으로 높게 나타났다. 또한 본 논문에서 제안한 취약점 탐지 기법을 이용하여 분석 대상이 되는 어플리케이션의 위험도 계산 방법을 제안하였고, 해당 어플리케이션이 내제된 취약점으로 인해 얼마나 위험한 상태인지를 한눈에 알아볼 수 있어 기존 취약점 분석 도구들에 비해 장점으로 평가할 수 있다.

References

[1] Luca Allodi and Fabio Massacci, “Comparing Vulnerability Severity and Exploits Using Case-Control Studies,” Journal of ACM Transactions on Information and System Security (TISSEC), Vol. 17, Issue 1, pp. 1-12, August, 2014.
DOI: https://dx.doi.org/10.1145/2630069

[2] More Secure Software, https://www.microsoft.com/en-us/sdl/about/benefits.aspx, Date accessed: May, 2016.

[3] Sam Ransbotham and Sabyasachi Mitra, The Impact of Immediate Disclosure on Attack Diffusion and Volume, Economics of Information Security and Privacy III, Springer, New York, pp.1-12, 2013.

[4] Introduction To ISO 27005(ISO27005), http://www.27000.org/iso-27005.htm, Date accessed: May, 2016.

[5] Agawal, Monica, Singh, Abhinav, Metasploit Penetration testing Cookbook(Second Edition), Packt Publishing, UK, pp.23-50, Oct., 2013.

[6] CVE(Common Vulnerabilities and Exposures), http://cve.mitre.org/, Date accessed: May, 2016.

[7] Common Weakness Scoring System(CWSS), http://cwe.mitre.org/cwss/cwss_v1.0.1.html/, May, 2016.

[8] Joon-Ho Kim, A study of utilization of source code security vulnerabilities detection tools and improving tools by using symbolic execution engine, Master's

Thesis, Department of Information Protection Graduate School, Soongsil University, pp.9-15, Jun, 2015.

[9] Seokmo Kim, A Study of Dynamic Analysis Compensation of Over-detection Problem of Static Analysis : In Case of Software Vulnerability Detection, Master's Thesis, Department of Computer Science and Engineering Graduate School, Dankook University, pp.14-15, Dec., 2015.

[10] M. D. Ernst, “Static and dynamic analysis: synergy and duality”, In Proceedings of the ICSE Workshop on Dynamic Analysis(WODA ’03), pp.24 - 27, 2003.

[11] Ari Takanen, Jared D. DeMott and Charles Miller, Fuzzing for Software Security Testing and Quality Assurance, Artech House Publishers, UK, pp.80-84, 2008.

송 준 호(Jun-Ho Song) **[정회원]**

- 2012년 2월 : 숭실대학교 일반대학원 컴퓨터학과 (공학석사)
- 2015년 3월 ~ : 숭실대학교 일반대학원 컴퓨터학과 (박사과정)

<관심분야>
정보통신, 정보보안

김 광 직(Kwang-Jik Kim) **[정회원]**

- 1988년 2월 : 명지대학교 전자공학과(학사)
- 2016년 8월 : 숭실대학교 정보과학대학원 IT경영학과(공학석사)
- 2018년 8월 : 숭실대학교 IT정책경영학과 박사 수료
- 2000년 1월 ~ : ALE Korea 한국지사장

<관심분야>
정보경영, 정보통신

고 용 선(Yong-Sun Ko) **[정회원]**

- 2011년 8월 : 숭실대학교 경영학과 (학사)
- 2016년 8월 : 숭실대학교 정보과학대학원 IT경영학과(공학석사)
- 2017년 3월 ~ : 숭실대학교 IT정책경영학과 박사 과정
- 2006년 3월 ~ : ㈜서브나라 대표이사

<관심분야>
정보통신, 정보보안

박 재 표(Jae-Pyo Park) **[종신회원]**

- 1998년 8월 : 숭실대학교 대학원 컴퓨터학과 (공학석사)
- 2004년 8월 : 숭실대학교 대학원 컴퓨터학과 (공학박사)
- 2010년 3월 ~ 현재 : 숭실대학교 정보과학대학원 교수

<관심분야>
네트워크 보안, 암호학, 디지털포렌식, 핀테크