논문 22-47-04-14 The Journal of Korean Institute of Communications and Information Sciences '22-04 Vol.47 No.04
https://doi.org/10.7840/kics.2022.47.4.679

정적 분석 도구 성능 비교 및 분석: 오픈소스의 보안 취약점을 기반으로

정 지 인◆, 이 재 혁*, 이 경 률°

Performance Comparison and Analysis of Static Analysis Tools: Based on Vulnerability Analysis of Open Sources

Jiin Jeong◆, Jaehyuk Lee*, Kyungroul Lee°

요 약

현재 전 세계적으로 많은 오픈소스가 공개되는 상황에서, 개별적으로 작성하는 소스코드에 취약점이 내포되는 가능성이 존재한다. 본 논문에서는 취약점을 탐지하는 정적 분석 도구인 Cppcheck, Yasca, Flawfinder를 활용하여 공개된 오픈소스의 취약점 탐지 결과를 기반으로 성능을 비교하고 분석한다. 이를 위하여, 취약점을 포함하는 샘플 소스코드와 암호화 과정을 포함하는 실제 오픈소스를 대상으로, 각 도구의 취약점 탐지 결과를 비교함으로써 성능을 분석하고 비교하였다. 탐지된 소스코드 개수 및 정확도를 기준으로, 탐지 성능을 분석한 결과, Flawfinder가 가장 성능이 높은 것으로 분석되었으며, Yasca가 그다음, Cppcheck가 가장 성능이 낮은 것으로 분석되었다. 하지만 각 도구가 탐지한 CWE가 중복되지 않고 상이하며, 이는 CWE ID를 기준으로 성능을 정량적으로 평가하기 어려운 한계점이 존재할 것으로 판단된다. 이러한 한계점을 극복하기 위하여, 공개된 CWE의 탐지 정확도에 대한 연구 및 탐지된 CWE 라인에 대한 분석을 가지고 동적 분석을 통하여 탐지된 취약점을 검증하는 연구를 진행할 예정이다.

Key Words : Static analysis tool, Vulnerability analysis, Open source software, Performance analysis, CWE(Common Weakness Enumeration)

ABSTRACT

With many open sources being released worldwide, vulnerabilities can be connoted in individual written source codes. In this paper, we analyzed and compared performances based on the detected vulnerability results of open sources using Cppcheck, Yasca, and Flawfinder, which are static analysis tools to detect vulnerabilities. For this purpose, performances were analyzed and compared by comparing the detected vulnerabilities results by using each tool targeting the sample souce codes including the vulnerability and the real open sources including the encryption functions. As a result of analyzing the detection performances based on the number and accuracy of the detected source codes, Flawfinder has the highest performance, Yasca was next, and Cppcheck was analyzed as having the lowest performance. Nevertheless, CEWs detected by each tool are not duplicated and are different, and we consider that there is a limitation in that it is difficult to quantitatively evaluate the detection accuracy of the published CWEs and to verify the detected vulnerabilities based on dynamic analysis with an analysis of the detected CWE results.

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2021R1F1A1050542).
※ 본 논문은 2021년도 한국정보보호학회 영남지부 학술대회에 발표한 우수논문을 개선 및 확장한 것임
◆ First Author : Daegu Catholic University, School of Computer Software, ddd0444@cu.ac.kr, 학생회원
° Corresponding Author : Mokpo National University, Department of Information Security, carpedm@mnu.ac.kr, 정회원
* Daegu Catholic University, Department of Computer Software, gurtmggg@cu.ac.kr, 학생회원
논문번호 : 202111-320-C-RN, Received November 18, 2021; Revised February 3, 2022; Accepted February 11, 2022

Ⅰ. 서 론

전 세계 많은 개발자들이 자신이 작성한 오픈소스를 공개하며, 공개된 오픈소스를 실제 프로그램 및 시스템에 탑재함으로써 빠른 개발이 가능한 장점이 있다[1]. 그럼에도 불구하고, 개별적으로 작성하는 소스코드는 취약점을 포함할 가능성이 있으며, 이로 인하여 보안 위협이 발생한다. 이러한 문제점을 해결하기 위하여, 소스코드에 내포된 취약점을 자동으로 분석하는 다양한 정적 분석 도구들이 등장하였다.

하지만 방대한 정적 분석 도구들이 공개되어 있고, 각 도구마다 취약점 분석 및 탐지 방법이 상이한 문제점이 존재한다. 그뿐만 아니라, 도구들의 취약점 탐지와 관련된 성능의 비교 및 분석에 대한 연구가 미비한 실정이다[2,3].

따라서 본 논문에서는 취약점 분석을 위하여, 오픈소스 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 취약점 탐지 성능을 비교하고자 한다. 성능 비교를 위하여, 암호화 과정을 포함하는 오픈소스를 선정하여, 각 도구의 성능 평가 결과를 비교 및 분석한다.

Ⅱ. 관련 연구

본 논문에서는 공개된 오픈소스에서의 취약점 분석을 위하여, 기존의 성능 평가가 연구되었고, 접근성 및 편의성이 높은 C/C++ 언어가 대상인 오픈소스 기반의 정적 분석 도구를 선정하여 성능을 평가한다[4].

2.1 Cppcheck

Cppcheck는 C/C++ 코드를 위한 정적 분석 도구로, 코드 분석을 통하여 버그, 정의되지 않은 동작 및 위험한 코딩 구조를 집중적으로 탐지한다[5]. 이 도구는 2007년 5월 8일에 처음 공개되었으며, 최근 버전인 v2.6은 2021년 10월에 업데이트되었다.

2.2 Yasca

Yasca는 보안 취약성, 코드 품질, 성능 및 프로그램 소스코드의 모범사례에 대한 적합성을 검사하는 오픈소스이다. Findbugs, PMD, JLint, JavaScirpt Lint, PHPLint, Cppcheck, ClamAV, Pixy 및 RATS와 같은 외부 오픈소스 프로그램을 활용하여 특정 파일 형식을 스캔하며, 지정 스캐너도 포함할 수 있다[6]. 이 도구는 2007년에 처음 공개되었으며, 최근 버전인 v3.0.5는 2015년 5월에 업데이트되었다.

2.3 Flawfinder

Flawfinder는 C/C++ 소스코드를 검사하며, 발생 가능한 보안 취약점을 위험 수준별로 분류하여 보고하는 도구이다. 취약점 탐지를 위하여, 버퍼 오버플로우 위험 함수, 형식 문자열 문제 함수, 경쟁 조건 문제 함수, 잠재적인 쉘 메타 문자 위험 및 잘못된 난수 획득 함수와 같이 잘 알려진 문제가 있는 C/C++ 함수의 내장 데이터베이스를 사용한다[7]. 이 도구는 2001년에 처음 공개되었으며, 최근 버전인 v2.0.19는 2021년 8월에 업데이트되었다.

2.4 기존 성능 평가 연구

기존 성능 평가 연구에서는 NIST의 취약점 분석 테스트 코드인 Juliet Test Suite for C/C++를 기반으로 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 성능을 평가하였다[4,8].

이 연구에서는 특정 CWE (Common Weakness Enumeration) 인 78, 79, 89, 99, 121, 122, 134, 170, 244, 251, 259, 362, 367, 391, 401, 411, 412, 415, 416, 457, 468, 476, 489를 대상으로, 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 탐지 정확도를 None, Low, Medium, High로 분류하였다. 탐지 방법으로는, 동일한 취약점을 가지는 테스트 코드를 다수 분석하여 정확도를 도출하였으며, 70% 이상 탐지할 경우 High, 40~69%를 탐지한 경우 Medium, 40% 미만인 경우 Low, 0%인 경우 None으로 정의하였다.

이 연구는 공개된 테스트 코드를 대상으로 정적 분석 도구들의 정확도에 대한 성능 평가를 연구하였지만, 오픈소스와 같이 실제 사용되는 소스코드를 대상으로 성능을 평가하지는 않았다. 따라서 본 논문에서는 소스코드에서 특히 중요한 부분인 암호화를 사용하는 공개된 오픈소스를 대상으로 정적 분석 도구들의 성능을 비교하고 분석하고자 한다.

Ⅲ. 정적 분석 도구의 취약점 탐지 성능 비교 및 분석

정적 분석 도구의 취약점 탐지 성능을 비교하고 분석하기 전에, 각 도구의 탐지 결과의 정확도를 확인하기 위하여, 버퍼 오버플로우와 포맷 스트링 취약점이 명백하게 존재하는 소스코드를 대상으로 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 탐지결과를 도출하였으며, 샘플 소스코드[9] 를 Fig. 1., 탐지 결과를 Table 1에 나타내었다.

샘플 소스코드의 취약점 탐지 결과, Cppcheck는

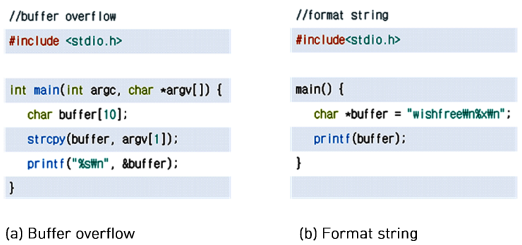

그림 1. 버퍼 오버플로우(a)와 포맷 스트링 취약점(b)을 포함하는 샘플 소스코드
Fig. 1. Sample source codes including buffer overflow (a) and format string (b) vulnerabilities

표 1. 샘플 소스코드 분석 결과표
Table 1. Analysis results of sample source codes

도구명	Buffer overflow	Format string
Cppcheck (v2.6)	X	O
Yasca (v3.0.5)	O	X
Flawfinder (v2.0.19)	O	O

버퍼 오버플로우 취약점을 탐지하지 못하고, 포맷 스트링 취약점만 탐지하였다. Yasca는 버퍼 오버플로우 취약점만 탐지하고, 포맷 스트링 취약점은 탐지하지 못하였다. 마지막으로 Flawfinder는 유일하게 버퍼 오버플로우와 포맷 스트링 취약점 모두를 탐지하였다.

이와 같이, 간단한 샘플 소스코드임에도 불구하고, 각 도구의 결과가 상이한 문제점이 있으며, 이러한 문제점은 성능 평가에서 오탐 및 미탐과 같은 탐지율과 밀접한 관계를 가진다. 따라서 본 논문에서는 실제 사용되는 소스코드 중, Table 2와 같이 암호화 과정을 포함하는 오픈소스 5개를 선정하여 각 도구의 성능을 비교하고 평가하였다.

선정한 오픈소스는 많은 소스코드를 포함하여 모든 결과를 도출하기에는 한계가 있어, 각 오픈소스의 암호화와 관련된 모듈인 Gnupg의 g10, Linux-pam의 modules, Cryptsetup의 crypto_backend, OpenSSH의 cipher, OpenSSL의 aes의 소스코드를 선정하였으며, 탐지된 전체 CWE ID와 개수를 Table 3에 나타내었다.

Table 3은 선정한 모듈에서의 암호화 기능을 제공하는 일부 소스코드를 대상으로 취약점 탐지 결과를 탐지된 전체 CWE ID 및 개수, 기존 연구에 포함되지 않은 CWE ID 및 개수, 기존 연구에 포함된 CWE ID 및 개수로 정리하였다. 이와 같이 분류한 목적은 탐지된 전체 CWE ID 중 기존 연구에서 탐지 정확도를 포

표 2. 선정한 오픈소스
Table 2. Selected open sources

오픈소스	설명
Cryptsetup	• DM 커널 모듈을 기반으로 디스크 암호화를 편리하게 설정하는 도구[10]
Gnupg	• RFC 4880 (OpenPGP Message Format) 에 정의된 OpenPGP 표준 구현 • 데이터 및 통신 암호화 제공 • 키 관리 시스템 및 공개키 디렉터리를 위한 접근 모듈 제공[11]
Linux-pam	• 시스템에서 애플리케이션이나 서비스의 인증을 처리하는 라이브러리[12]
OpenSSH	• SSH 프로토콜을 사용한 원격 로그인을 위한 도구 • 도청, 하이재킹 및 다른 공격을 제거하기 위하여 모든 트래픽 암호화[13]
OpenSSL	• TLS (Transport Layer Security) 및 SSL (Secure Sockets Layer) 프로토콜들을 위한 모든 기능을 제공하는 범용 암호화 라이브러리[14]

함하지 않은 ID와 정적 분석 도구의 성능을 평가하기 위하여 기존 연구에 포함된 ID를 비교하기 위함이다.

비교 결과, Cryptsetup은 소스코드마다 약 5~10개의 취약점이 탐지되었고, 전반적으로 CWE-119와 120이 가장 많이 탐지되었다. 기존 연구에 포함된 CWE-476는 crypto_cipher_kernel.c에서 탐지되었다. Gnupg는 다른 오픈소스보다 많은 개수의 취약점이 탐지되었으며, 그중 CWE-119, 120, 126, 398이 가장 많이 탐지되었다. 기존 연구에 포함된 CWE-476는 keyid.c에서 탐지되었고, CWE-362는 keyring.c에서 탐지되었다. Linux-pam은 다양한 취약점의 종류와 개수가 탐지되었으며, 그중, CWE-120와 362가 가장 많이 탐지되었다. 특히, 이 오픈소스는 탐지된 대부분의 취약점이 기존 연구에 포함되었다. OpenSSH는 다른 오픈소스보다 취약점이 탐지된 소스코드의 개수가 적으며, CWE-120과 398이 가장 많이 탐지되었다. 이 CWE는 cipher.c에서 탐지되었다. OpenSSL 역시 OpenSSH와 비슷하게 다른 오픈소스보다 취약점이 탐지된 소스코드의 개수가 적으며, CWE-398과 457이 탐지되었다. 기존 연구에 포함된 CWE-457은 aes_ige.c에서 탐지되었다.

Table 3의 비교 결과를 토대로 탐지된 모든 CWE의 성능을 비교하고 평가하기 위한 기준이 연구되지 않은 한계점이 존재한다. 따라서 탐지 정확도의 기준을 포함하는 CWE를 대상으로 Table 4와 같이 정적 분석 도구의 성능을 비교하였다.

표 3. 탐지된 전체 CWE ID와 개수 및 기존 연구 포함 및 미포함 CWE ID와 개수 비교
Table 3. Comparing the total number of detected CWE IDs and the number of CWE IDs with and without existing research

오픈소스	소스코드	탐지된 전체 CWE		기존 연구 미포함 CWE		기존 연구 포함 CWE	
		ID	개수	ID	개수	ID	개수
Crypt setup (v2.3.4)	crypto_kernel.c	20, 119, 120, 398	5	20, 119, 120, 398	5	-	-
	crypto_cipher_kernel.c	20, 119, 120, 476	10	20, 119, 120	9	476	1
	crypto_gcrypt.c	119, 120, 563, 571	7	119, 120, 563, 571	7	-	-
	crypto_nettle.c	120, 398, 563	6	120, 398, 563	6	-	-
	crypto_nss.c	119, 120, 398	6	119, 120, 398	6	-	-
	crypto_openssl.c	119, 120, 398	6	119, 120, 398	6	-	-
	crypto_storage.c	119, 120, 563	5	119, 120, 563	5	-	-
Gnupg (v2.3)	keydb.c	119, 120, 126, 398, 477, 571, 732	29	119, 120, 126, 398, 477, 571, 732	29	-	-
	keyid.c	119, 120, 126, 398, 476	30	119, 120, 126, 398	29	476	1
	encrypt.c	119, 120, 126, 563	10	119, 120, 126, 563	10	-	-
	keylist.c	119, 120, 126, 190, 398, 563, 686	37	119, 120, 126, 190, 398, 563, 686	37	-	-
	keyring.c	119, 120, 126, 362, 398, 477, 732	11	119, 120, 126, 398, 477, 732	10	362	1
	gpgcompose.c	119, 120, 126, 398, 786	47	119, 120, 126, 398, 786	47	-	-
	keyedit.c	119, 120, 126, 190, 398, 563, 571, 686, 758	71	119, 120, 126, 190, 398, 563, 571, 686, 758	71	-	-
Linux -pam (v1.5.1)	pam_access.c	119, 120, 126, 362, 398, 563, 571, 758	37	119, 120, 126, 398, 563, 571, 758	35	362	2
	pam_filter.c	78, 120, 126, 362, 398, 476, 590	29	120, 126, 398, 590	15	78, 362, 476	14
	pam_keyinit.c	134, 398	5	398	3	134	2
	pam_securetty.c	119, 120, 126, 362	12	119, 120, 126	9	362	3
	pam_xauth.c	20, 78, 119, 120, 126, 362, 367, 377, 398, 477, 807	40	20, 119, 120, 126, 377, 398, 477, 807	36	78, 362, 367	4
Open SSH (v8.4)	cipher.c	120, 126, 398, 401, 476, 477	10	120, 126, 398, 477	8	401, 476	2
	cipher-aes.c	120, 398	8	120, 398	8	-	-
Open SSL (v1.1.1)	aes_ige.c	119, 120, 398, 457, 563	29	119, 120, 398, 563	26	457	3
	aes_core.c	398, 563	2	398, 563	2	-	-

표 4. 도구별 오픈소스 취약점 탐지 정확도 (C: CWE ID, F: Flaw, D: 탐지 정확도)
Table 4. Vulnerability detection accuracy of open source by tool (C: CWE ID, F: Flaw, D: Detection accuracy)

오픈 소스	모듈 (경로)	소스 코드	정적 분석 도구명								
			Cppcheck			Flawfinder			Yasca		
			C	F	D	C	F	D	C	F	D
Crypt setup	cryptsetup-v2.3.4 (\lib\crypto_backend)	crypto_ cipher_ kernel.c	476	Null Pointer Dereference	Low	-			-		
Gnupg	gnupg-gnupg-2.3.0 (\g10)	keyid.c	-			-			476	Null Pointer Dereference	Low
		keyring.c	-			362	Race Condition	Medium	-		
Linux -pam	linux-pam-1.5.1 (\modules)	pam_ access.c	-			362	Race Condition	Medium	-		
		pam_ filter.c	476	Null Pointer Dereference	Low	78	OS Command Injection	High	-		
						362	Race Condition	Medium			
		pam_ keyinit.c	-			134	Uncontrolled Format String	High	-		
		pam_ securetty.c	-			362	Race Condition	Medium	-		
		pam_ xauth.c	-			78	OS Command Injection	High	-		
						362	Race Condition	Medium			
						367	TOCTOU Race Condition	High			
Open SSH	openssh -portable-V_8_4 (\)	cipher.c	401	Memory Leak	Low	-			-		
			476	Null Pointer Dereference	Low						
Open SSL	openssl -OpenSSL_1_1_1 (\crypto\aes)	aes_ige.c	457	Use of Uninitialized Variable	None	-			-		

비교 결과를 살펴보면, 각 도구에서 탐지된 CWE ID가 중복되는 항목이 하나도 없으며, Cppcheck와 Yasca는 탐지 정확도가 Low이거나 None인 CWE-401, 457, 476만 탐지되었다.

탐지된 CWE ID의 소스코드 개수를 기준으로 각 도구의 성능을 비교하면, Yasca는 오직 keyid.c로 가장 성능이 낮으며, 그다음으로 Cppcheck가 crypto_cipher_kernel.c, pam_filter.c cipher.c, aes_ige.c로 4개, Flawfinder는 keyring.c, pam_access.c, pam_filter.c, pam_keyinit.c, pam_securetty.c, pam_xauth.c는 6개로 가장 성능이 높은 것으로 나타났다.

탐지된 CWE ID의 탐지 정확도를 기준으로 각 도구의 성능을 비교하면, Cppcheck는 CWE-401, 457, 476이 탐지되었고, CWE-401과 476의 탐지 정확도는 Low, CWE-457의 탐지 정확도는 None이다. Flawfinder는 CWE-78, 134, 362, 367을 탐지하였으며, 각각의 탐지 정확도는 High, High, Medium, High

로 높은 정확도를 가지는 CWE ID를 가장 많이 탐지하였다. Yasca는 다른 도구들에서 탐지한 대부분의 CWE를 탐지하지 못하고, keyid.c에서 CWE-476만 탐지하였다. 따라서 탐지 정확도가 높고, 가장 많은 CWE를 탐지한 Flawfinder가 가장 성능이 높은 것으로 나타났다.

마지막으로 탐지된 CWE ID를 기준으로 각 도구의 성능을 비교하면, Cppcheck와 Yasca는 CWE-476인 Null Pointer Dereference를 가장 많이 탐지하였으며, Flawfinder는 CWE-362인 Race Condition을 가장 많이 탐지하였다. 따라서 각 도구가 탐지한 CWE가 중복되지 않고 상이하며, 이는 CWE ID를 기준으로 성능을 평가하기에는 한계가 있을 것으로 판단된다.

상기 각 도구의 성능 비교 결과를 토대로 성능 평가 결과를 분석하면, 탐지된 CWE ID의 소스코드 개수 및 탐지 정확도를 기준으로 Flawfinder가 가장 성능이 높은 것으로 판단된다.

Ⅳ. 탐지된 취약점 상세 분석 및 결과 해석

상기 정적 분석 도구의 성능 평가 결과를 살펴보면, 각 도구가 탐지한 CWE ID들이 중복되지 않고 상이하므로, 도구 결과만으로는 성능을 평가하기에 한계가 있을 것으로 판단되며, 탐지 결과의 정확성을 기대하기는 어려울 것으로 사료된다. 이러한 이유로 본 논문에서는 탐지된 코드를 상세하게 분석함으로써 취약점이 발생하는 이유와 취약점으로 인하여 발생하는 위협을 해석하고자 한다. 탐지된 전체 코드 중, 많이 탐지된 코드를 대상으로 상세하게 분석하였고, 해당 소스코드를 Table 5에 나타내었다.

표 5. 탐지된 취약점 요약
Table 5. Summary of detected vulnerabilities

취약점 종류	소스코드	Cppcheck (v2.6)	Flawfinder (v2.0.19)	Yasca (v3.0.5)
Null Pointer Dereference	crypto_cipher_kernel.c	O	X	X
Race Condition	keyring.c	X	O	X
	pam_access.c			
	pam_filter.c			
	pam_securetty.c			
	pam_xauth.c			
OS Command Injection	pam_xauth.c	X	O	X
	pam_filter.c			
Use of Uninitialized Variable	aes_ige.c	O	X	X

4.1 Null Pointer Dereference

Null Pointer Dereference는 포인터를 유효한 주소가 아닌 Null 값으로 역참조하는 경우에 발생하는 취약점이다[15].

Fig. 2.는 Cppcheck가 cryptsetup의 crypto_cipher_kernel.c에서 Null Pointer Dereference를 탐지한 코드이다. 코드에서 setsockopt 함수는 소켓의 세부사항들을 설정하는 함수이며, 이를 위하여 4번째 매개변수는 옵션 값이 저장된 버퍼의 포인터이다. 이때, 전달되는 버퍼의 포인터가 Null인 경우, 유효한 주소가 아닌 경우, 프로그램이 사용할 수 있는 메모리 공간이 아닌 경우에는 역참조할 때, 메모리 접근 오류로 인하여 서비스 거부가 발생한다. Fig. 2.에서는 포인터의 값이 Null일 경우, 오류가 발생할 가능성이 존재하며, 따라서 함수를 호출하기 전, 해당 버퍼의 포인터에 저장된 주소의 유효성을 검증하는 코드가 요구된다. 이를 검증하기 위하여, 버퍼의 주소에 임의의 주소를 넣은 결과, Cppcheck가 취약점을 탐지하지 않음을 확인하였다.

```
//crypto_cipher_kernel.c
if (tag_length && setsockopt(ctx->tfmfd, SOL_ALG,
ALG_SET_AEAD_AUTHSIZE, NULL, tag_length) < 0)
```

그림 2. crypto_cipher_kernel.c에서 탐지된 취약한 코드
Fig. 2. Detected vulnerable codes in crypto_cipher_kernel.c

4.2 Race Condition

Race Condition은 동일한 환경에서 둘 이상의 명령어가 동시에 같은 공유된 자원에 접근하려고 하는 경우에 발생한다[15].

Fig. 3.은 Flawfinder가 Gnupg의 keyring.c에서 Race Condition을 탐지한 코드이며, Flawfinder는 문제가 있는 함수인 chmod 함수를 탐지하였다. chmod 함수는 파일의 접근권한을 변경하는 함수이며, 코드에서는 fname 변수가 가리키는 파일의 소유자 접근권한을 읽기와 쓰기로 변경한다. 따라서 fname의 경로를

```
//keyring.c
if (!gnupg_stat (bakfname, &statbuf)
&& !chmod (fname, statbuf.st_mode))
```

그림 3. keyring.c에서 탐지된 취약한 코드
Fig. 3. Detected vulnerable codes in keyring.c

매개변수로 전달받아 권한을 변경하기 때문에, 공격자가 대상 파일을 삭제하고 같은 이름의 심볼릭 링크 파일을 생성함으로써 다른 파일의 접근권한을 읽기와 쓰기로 변경하는 것이 가능하다. 이러한 취약점을 해결하기 위하여, Flawfinder는 동일한 기능을 수행하지만, 파일 경로가 아닌 파일 디스크립터를 매개변수로 전달받는 fchmod 함수의 사용을 권유한다.

Fig. 4.는 Flawfinder가 Linux-pam의 pam_access.c, pam_filter.c, pam_securetty.c, pam_xauth.c에서 Race Condition을 탐지한 코드이며, open 함수와 fopen 함수를 탐지하였다. open 함수와 fopen 함수는 파일을 열기 위하여 사용하는 함수이며, 열고자 하는 파일의 경로를 매개변수로 전달받는다. 따라서 상기 chmod 함수에서의 취약점과 마찬가지로, Fig. 5.와 같이 공격자가 열기를 원하는 대상 파일을 삭제하고 같은 이름의 심볼릭 링크 파일을 생성함으로써 다른 파일에 접근이 가능한 취약점이 발생한다[9]. 이러한 문제점을 해결하기 위하여, 파일과 관련된 함수를 호출하기 전, 해당 파일의 심볼릭 링크 파일의 여부를 확인하는 코드가 요구된다.

Race Condition 공격의 일례를 Fig. 5.에 나타내었다. 프로그램이 정상적으로 실행되는 경우에는 Fig. 4.에 나타낸 str, DEV_PTMX와 같은 프로그램에서 요구하는 파일을 열지만, 공격자가 원래 파일을 삭제한 후, 같은 이름의 심볼릭 링크 파일을 생성할 경우에는 공격자가 원하는 파일을 열어 처리하는 등의 조작이

```
//pam_access.c
FILE *fp = fopen(str, "r");
if ((fd = open(DEV_PTMX, O_RDWR)) >=0)
```

```
//pam_filter.c
if ((fd = open(DEV_PTMX, O_RDWR)) >=0)
int t = open("/dev/tty", O_RDWRIO_NOCTTY);
int t = open("/dev/tty", O_RDWR);
fd[1] = open(terminal, O_RDWR);
```

```
//pam_securetty.c
ttyfile = fopen(securettyfile, "r");
cmdlinefile = fopen(CMDLINE_FILE, "r");
consoleactivefile = fopen(CONSOLEACTIVE_FILE, "r");
```

```
//pam_xauth.c
fd = open(path, O_RDONLY | O_NOCTTY);
```

그림 4. pam_access.c, pam_filter.c, pam_securetty.c, pam_xauth.c에서 탐지된 취약한 코드
Fig. 4. Detected vulnerable codes in pam_access.c, pam_filter.c, pam_securetty.c and pam_xauth.c

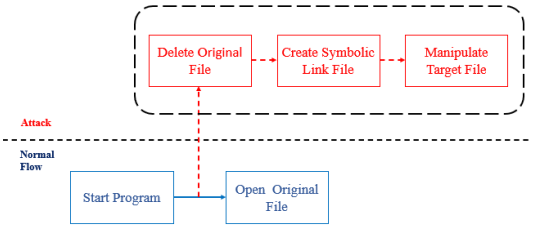

그림 5. Race Condition 공격 일례
Fig. 5. Race Condition attack example

가능하다.

4.3 OS Command Injection

OS Command Injection은 운영체제 명령에 대한 이름, 경로, 매개변수에 신뢰할 수 없는 데이터가 포함되더라도 공격자가 직접 시스템 명령을 사용할 수 있는 취약점이다[15].

Fig. 6.은 Flawfinder가 Linux-pam의 pam_xauth.c와 pam_filter.c에서 OS Command Injection을 탐지한 코드이다. 두 함수는 시스템 호출 함수이며, execv 함수는 경로를 포함한 실행파일의 이름인 명령을 매개변수로 전달받아 실행한다. execle 함수도 동일하게 filtername에 경로를 포함한 실행파일의 이름인 명령을 매개변수로 전달받아 실행한다. 따라서 두 함수 모두 실행파일의 경로를 전달함으로써 공격자가 운영체제의 명령을 임의로 실행이 가능하며, 이러한 취약점으로 인하여 서버정보 확인, 권한 상승, 다른 취약점과 결합하여 민감한 파일에 접근하거나 서버를 제어할 수 있다. 이러한 문제점을 해결하기 위하여, 명령어 실행 전, 실행 가능한 명령어 목록을 기반으로 전달되는 파일경로를 검증하는 코드가 요구된다.

```
//pam_xauth.c
execv(command, (char *const *) args);
```

```
//pam_filter.c
execle(filtername, "<pam_filter>", NULL, evp);
```

그림 6. pam_xauth.c, pam_filter.c에서 탐지된 취약한 코드
Fig. 6. Detected vulnerable codes in pam_xauth.c and pam_filter.c

4.4 Use of Uninitialized Variable

Use of Uninitialized Variable은 초기화되지 않은 변수를 사용함으로써 예측할 수 없거나 의도하지 않은 결과가 발생하는 취약점이다[15].

Fig. 7.은 Cppcheck가 OpenSSL의 aes_ige.c에서

```
//aes_ige.c
tmp2.data[n] = tmp.data[n] ^ iv.data[n];
iv2 = tmp;
memcpy(ivec + AES_BLOCK_SIZE, iv2.data, AES_BLOCK_SIZE);
```

그림 7. aes_ige.c에서 탐지된 취약한 코드
Fig. 7. Detected vulnerable codes in aes_ige.c

Use of Uninitialized Variable을 탐지한 코드이며, 코드에서는 data, tmp, data의 순서로 각 변수에서 Use of Uninitialized Variable이 발생하였다. 이를 자세하게 살펴보면, tmp와 iv2는 구조체로 선언되어 있으며, 해당 구조체는 data 변수를 포함한다. 탐지된 코드는 tmp 변수가 초기화되지 않았기 때문에 iv2도 영향을 받았으며, 총 3곳에서 취약점이 탐지되었다. 또한 소스코드에서 load_block 함수가 memcpy 함수를 호출함으로써 구조체를 초기화한다. 하지만, memcpy는 복사되는 데이터의 크기를 검사하지 않는 취약한 함수이므로 버퍼 오버플로우 공격이 발생할 가능성이 존재한다.

이와 같이 상기 상세 분석 결과를 살펴볼 때, 정적 분석 도구에서 탐지한 일부 취약점들은 실제 공격이 가능한 수준일 것으로 판단되며, 동적 분석을 활용한다면 실제 공격 가능성을 검증할 수 있을 것으로 판단된다.

추가적으로 다양한 오픈소스를 기반으로 결과를 도출하고 성능을 평가하기 위하여, 4차 산업혁명의 주요 기술인 3D, IoT(Internet of Things), Deep Learning과 관련된 오픈소스를 선정하였다[16]. 선정한 오픈소스를 자세하게 설명하면, 3D를 위한 FreeCAD, OpenSCAD, Blender, IoT를 위한 Esphome, RIOT, Tasmota, Deep Learning을 위한 Tensorflow, Darknet, PyTorch, Caffe이다. 선정한 오픈소스에서의 취약점 탐지를 위하여, 본 논문에서는 각 오픈소스에서 제공하는 특정 모듈과 관련된 소스코드를 선정하였으며, 자세하게는, 3D에서 모델링된 객체를 이미지나 영상으로 표현하는 렌더링과 관련된 소스코드, IoT에서 사물 인터넷 통신을 위한 Esp32와 관련된 소스코드, Deep Learning에서는 딥러닝과 관련된 소스코드를 대상으로 정적 분석 도구를 통한 취약점 탐지 결과를 도출하였다.

취약점 탐지 성능을 비교하기 위하여, 기존 성능 평가에서 연구한 탐지 정확도를 포함하는 CWE를 대상으로, 성능 평가 결과를 Table 6에 나타내었으며, 기존 연구에 포함된 CWE가 탐지되지 않은 OpenSCAD, Tensorflow, PyTorch는 성능 평가에서 제외하였다. 비교 결과를 살펴보면, Blender와 Darknet에서는 Cppcheck와 Yasca가 탐지한 CWE-401, 457, 476이 중복으로 탐지되었으며, 이를 제외하고는 각 도구에서 탐지한 CWE ID가 중복으로 탐지되지 않았다.

탐지된 CWE ID의 소스코드 개수를 기준으로 각 도구의 성능을 평가하면, Cppcheck는 Blender의 engines에서 1개, Esphome의 esp32에서 2개, Darknet의 src에서 6개가 탐지되었으며, 총 9개를 탐지하여 가장 성능이 낮은 것으로 나타났다. 그다음으로 Yasca는 Blender의 engines에서 3개, RIOT의 esp32에서 3개, Tasmota의 libesp32에서 1개, Darknet의 src에서 7개, Caffe의 src에서 1개가 탐지되었으며, 총 15개를 탐지하여 Cppcheck보다 성능이 높은 것으로 나타났다. 마지막으로 Flawfinder는 FreeCAD의 Raytacing에서 2개, Esphome의 esp32에서 2개, RIOT의 esp32에서 5개, Tasmota의 libesp32에서 10개, Darknet의 src에서 19개, Caffe의 src에서 2개가 탐지되었으며, 총 40개를 탐지하여 가장 성능이 높은 것으로 나타났다.

탐지된 CWE ID의 탐지 정확도를 기준으로 각 도구의 성능을 평가하면, Cppcheck는 CWE-401와 457, 476을 탐지하였고, CWE-401과 476의 탐지 정확도는 Low, CWE-457의 탐지 정확도는 None으로, 상대적으로 낮은 정확도를 가지는 CWE ID를 탐지하였다. Flawfinder는 CWE-78과 134, 362, 367을 탐지하였고, CWE-78과 134, 367의 탐지 정확도는 High, CWE-362의 탐지 정확도는 Medium이다. Yasca는 CWE-259와 401, 415, 457, 476을 탐지하였고, CWE-401의 탐지 정확도는 Medium, CWE-259와 415, 457, 476의 탐지 정확도는 Low이다. 따라서 가장 많은 CWE를 탐지한 도구는 Yasca이며, 정확도가 높은 CWE를 가장 많이 탐지한 도구는 Flawfinder으로 나타났다.

마지막으로 탐지된 CWE ID를 기준으로 각 도구의 성능을 평가하면, Cppcheck와 Yasca는 CWE-457인 Use of Uninitialized Variable과 CWE-476인 Null Pointer Dereference를 가장 많이 탐지하였으며, Flawfinder는 CWE-362인 Race Condition을 가장 많이 탐지하였다.

본 논문에서 선정한 암호화 과정을 포함하는 오픈소스와 3D, IoT, Deep Learning과 관련된 오픈소스를 대상으로, 각 정적 분석 도구에 따른 취약점 탐지의 성능 평가 결과를 종합하여 Table 7에 나타내었다.

결과를 종합하면, 가장 많은 CWE를 탐지한 도구는 Yasca이며, 정확도가 높은 CWE를 가장 많이 탐지한 도구는 Flawfinder로 나타났다. 탐지된 CWE ID의

표 6. 오픈소스 카테고리별 취약점 탐지 결과(C: CWE ID, F: Flaw, D: 탐지 정확도, N: 탐지된 소스코드 개수)
Table 6. Vulnerability detection accuracy by open source category(C: CWE ID, F: Flaw, D: Detection accuracy, N: Number of detected Source Codes)

분류	오픈 소스	모듈 (경로)	정적 분석 도구명											
			Cppcheck				Flawfinder				Yasca			
			C	F	D	N	C	F	D	N	C	F	D	N
3D	FreeCAD (v0.19.3)	FreeCAD-0.19.3 (\src\Mod \Raytracing)	-	-	-	-	362	Race Condition	Medium	2	-	-	-	-
	Blender (v3.0.0)	blender-3.0.0 (\source\blender \draw\engines)	476	Null Pointer Dereference	Low	1	-	-	-	-	476	Null Pointer Dereference	Low	3
IoT	Esphome (v2022.1.1)	esphome-2022.1.1 (\esphome \components \esp32*)	457	Use of Uninitialized Variable	None	2	134	Uncontrolled Format String	High	2	-	-	-	-
							362	Race Condition	Medium					
	RIOT (v2021.10)	RIOT-2021.10 (\cpu\esp32)	-	-	-	-	134	Uncontrolled Format String	High	5	259	Hard-Coded Password	Low	3
	Tasmota (v10.1.0)	Tasmota-10.1.0 (\lib\libesp32)	-	-	-	-	78	OS Command Injection	High	10	457	Use of Uninitialized Variable	Low	1
							134	Uncontrolled Format String	High					
							362	Race Condition	Medium					
Deep Learn -ing	Darknet (YOLOv4)	darknet-yolov4 (\src)	401	Memory Leak	Low	6	78	OS Command Injection	High	19	401	Memory Leak	Medium	7
			457	Use of Uninitialized Variable	None		134	Uncontrolled Format String	High		415	Double Free	Low	
			476	Null Pointer Dereference	Low		362	Race Condition	Medium		457	Use of Uninitialized Variable	Low	
							367	TOCTOU Race Condition	High		476	Null Pointer Dereference	Low	
	Caffe (v1.0)	caffe-1.0 (\src)	-	-	-	-	362	Race Condition	Medium	2	457	Use of Uninitialized Variable	Low	1
											476	Null Pointer Dereference	Low	

표 7. 성능 평가 결과표
Table 7. Performance evaluation results

정적 분석 도구명	CWE ID	탐지 정확도	소스코드 개수
Cppcheck	401	Low	13
	457	None	
	476	Low	
Flawfinder	78	High	46
	134	High	
	362	Medium	
	367	High	
Yasca	259	Low	16
	401	Medium	
	415	Low	
	457	Low	
	476	Low	

소스코드 개수, 탐지된 CWE ID의 탐지 정확도, 탐지된 CWE ID를 기준으로 성능을 종합하여 평가한 결과, 암호화 과정을 포함하는 오픈소스에서의 성능 평가와 동일하게 Flawfinder가 가장 성능이 높은 것으로 판단된다.

V. 결 론

본 논문에서는 오픈소스 기반의 취약점 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 탐지 성능을 비교하고 분석하였다. 성능 비교를 위하여, 버퍼 오버플로우와 포맷 스트링 취약점이 명백하게 존재하는 샘플 소스코드를 대상으로 각 도구의 탐지 성능을 비교한 결과, 각 도구가 다른 취약점을 탐지하는 것을 확인하였으며, Flawfinder만 모든 취약점을 탐지하였다. 이를 통하여 각 도구의 결과가 상이한 문제점을 도출하였으며, 오픈소스를 대상으로 각 도구의 성능을 비교하고 분석하였다.

암호화 과정을 포함하는 오픈소스 5개와 3D과 관련 오픈소스 3개, IoT과 관련된 오픈소스 3개, Deep Learning과 관련된 오픈소스 4개를 선정하여 탐지된 CWE ID 및 개수를 비교하고 분석한 결과, 탐지된 모든 CWE의 성능을 비교하고 평가하기 위한 기준이 연구되지 않은 한계점이 존재하였다. 이에 따라, 탐지 정확도의 기준을 포함하는 CWE를 대상으로, 정적 분석 도구의 성능을 탐지된 CWE ID의 소스코드 개수 및 탐지 정확도를 기준으로 탐지 성능을 비교하고 분석하였다. 그 결과, Flawfinder가 가장 성능이 높은 것으로 분석되었으며, Cppcheck가 가장 성능이 낮은 것으로 분석되었다.

이러한 결과가 도출되었음에도 불구하고, 각 도구가 탐지한 CWE가 중복되지 않고 상이하며, 이는 CWE ID를 기준으로 성능을 정량적으로 평가하기 어려운 한계점이 존재할 것으로 판단된다. 그뿐만 아니라, Table 2에 나타낸 것과 같이 기존 연구에서 모든 CWE ID의 탐지 정확도가 연구되지 않은 한계점, 각 도구가 탐지한 CWE가 중복되지 않은 한계점도 포함하는 것으로 나타났다.

이러한 한계점을 극복하기 위하여, 향후 연구로, 공개된 CWE ID의 탐지 정확도에 대한 연구를 진행할 예정이며, 정적 분석 도구의 한계점인 오탐 및 미탐을 실증하기 위하여, 본 논문의 결과와 탐지된 코드의 분석을 기반으로 동적 분석을 통하여 탐지된 취약점을 검증하는 연구를 진행할 예정이다.

References

[1] W. Ryu and S. Gang, "Current trends of major open source licenses," *J. KICS*, vol. 36, no. 11, pp. 32-41, Oct. 2019.

[2] Y. Lee, J. Ahn, and J. Choi, "Performance analysis of static analysis tools by software weakness," in *Proc. 2019 KIISE*, pp. 272-274, Dec. 2019.

[3] H. Seo, Y. Park, T. Kim, K. Han, and C. Pyo, "Evaluation of static analyzers for weakness in C/C++ programs using juliet and STONESOUP test suites," *J. Korea Soc. Comput. and Inf.*, vol. 22, no. 3, pp. 17-25, Mar. 2017.

[4] C. L. Blackmon, D. F. Sang, and C. S. Peng, "Performance evaluation of automated static analysis tools," *GSTF J. Comput.*, vol. 2, no. 1, pp. 214-219, Apr. 2012.

[5] Cppcheck, "*A tool for static C/C++ code analysis*," Retrieved May 10, 2021, from http://cppcheck.sourceforge.io.

[6] D. A. Wheeler, "*Flawfinder*," Retrieved May 10, 2021, from https://dwheeler.com/flawfinder

[7] Yasca, "*Yet Another Source Code Analyzer*," Retrieved May 10, 2021, from http://scovetta.github.io/yasca.

[8] NIST, "*Test Suites*," Retrieved May 10, 2021, from https://samate.nist.gov/SRD/testsuite.php.

[9] D. Yang, "*System hacking and security: Principles and practices*," 3rd Ed., HANBIT Academy, Nov. 2018.
[10] GitLab, "*cryptsetup*," Retrieved May 10, 2021, from https://gitlab.com/cryptsetup/cryptsetup.
[11] GnuPG, "*The Gnu Privacy Guard*," Retrieved May 10, 2021, from https://gnupg.org.
[12] Linux Documentation, "*pam.d(8) - Linux man page*," Retrieved May 10, 2021, from https://linux.die.net/man/8/pam.d.
[13] OpenBSD Foundation, "*OpenSSH*," Retrieved May 10, 2021, from https://www.openssh.com.
[14] OpenSSL, "*Cryptography and SSL/TLS Toolkit*," Retrieved May 10, 2021, from https://www.openssl.org.
[15] MITRE, "*Common Weakness Enumeration*," Retrieved May 10, 2021, from https://cwe.mitre.org.
[16] K. Chun, H. Kim, K. Park, and K. Lee, "A study on 5 platform technology trends for 4th industrial revolution," in *Proc. Korea Technol. Innovation Soc. Conf*, pp. 1375-1389, Jeju island, South Korea, Nov. 2017.

정 지 인 (Jiin Jeong)

2019년 3월~현재 : 대구가톨릭대학교 컴퓨터소프트웨어학부 사이버보안전공 학사과정
<관심분야> 정보보호, 취약점 분석, Offensive security
[ORCID:0000-0002-7079-2862]

이 재 혁 (Jaehyuk Lee)

2021년 2월 : 대구가톨릭대학교 컴퓨터소프트웨어학부 사이버보안전공 (공학사)
2021년 3월~현재 : 대구가톨릭대학교 컴퓨터소프트웨어학과 석사과정
<관심분야> 정보보호, 취약점 분석, 역공학, Offensive security
[ORCID:0000-0003-1492-1241]

이 경 룔 (Kyungroul Lee)

2008년 8월 : 순천향대학교 정보보호학과 (공학사)
2010년 8월 : 순천향대학교 정보보호학과 (공학석사)
2015년 2월 : 순천향대학교 정보보호학과 (공학박사)
2011년 5월~2011년 12월 : (미) 퍼듀대학교 방문연구원
2015년 6월~2016년 2월 : 순천향대학교 박사후연구원
2016년 3월~2020년 8월 : 순천향대학교 연구 조교수
2020년 9월~2022년 2월 : 대구가톨릭대학교 컴퓨터소프트웨어학부 교수
2022년 3월~현재 : 목포대학교 정보보호학과 교수
<관심분야> 정보보호, 취약점 분석, 시스템 보안, 하드웨어 보안, 역공학, Offensive security
[ORCID:0000-0003-3551-6130]