정적 분석 도구를 활용한 오픈소스의 보안 취약점 탐지 성능 비교 및 분석

정지인*, 이경률*

*대구가톨릭대학교 컴퓨터소프트웨어학부

Vulnerability Detection Performance Comparison and Analysis of Open Source Software using Static Analysis Tools

Jiin Jeong*, Kyungroul Lee*

*School of Computer Software, Daegu Catholic University

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 분석하였다. 탐지된 소스코드 개수 및 정확도를 기준으로 탐지 성능을 분석한 결과, Flawfinder가 가장 성능이 높은 것으로 분석되었으며, Cppcheck가 그다음, Yasca가 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 동적 분석을 통하여 탐지된 취약점을 검증하는 연구를 진행할 예정이다.

키워드: 정적 분석 도구, 취약점 분석, 오픈소스 소프트웨어, 성능 분석

I. 서론

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 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 취약점 분석을 위하여, 기존의 성능 평가가 연구되었고, 접근성 및 편의성이 높은 C/C++ 언어가 대상인 오픈소스 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 탐지

한다 [5]. 이 도구는 2007년 5월 8일에 처음 공개되었으며, 최근 버전인 2.3은 2020년 12월 5일에 업데이트되었다.

2.2. Yasca

Yasca는 보안 취약성, 코드 품질, 성능 및 프로그램 소스코드의 모범사례에 대한 적합성을 검사하는 오픈소스 프로그램이다. 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 버전은 2017년에 업데이트되었다.

2.3. Flawfinder 도구

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 같은 잘 알려진 문제가 있는 c/c++ 함수의 내장 데이터베이스를 사용한다 [7]. 이 도구는 2001년에 처음 공개되었으며, 최근 버전인 2.0.15는 2021년에 업데이트되었다.

2.4. 기존 성능 평가 연구

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

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 결과를 확인하기 위하여, 버퍼 오버플로우와 포맷 스트링 취약점이 명백하게 존재하는 소스코드를 대상으로 정적 분석 도구인 Cppcheck, Yasca, Flawfinder의 탐지결과를 도출하였으며, 샘플 소스코드 [9] 를 그림 1, 탐지 결과를 표 1에 나타내었다.

```c
//buffer overflow
#include <stdio.h>

int main(int argc, char *argv[]) {
  char buffer[10];
  strcpy(buffer, argv[1]);
  printf("%s\n", &buffer);
}
```

A. 버퍼 오버플로우 소스코드

```c
//format string
#include<stdio.h>

main() {
  char *buffer = "wishfree\n%x\n";
  printf(buffer);
}
```

B. 포맷 스트링 소스코드

그림 1. 버퍼 오버플로우와 포맷 스트링 취약점을 포함하는 샘플 소스코드

표 1. 샘플코드 분석 결과표

도구명	버퍼 오버플로우 예제	포맷 스트링 예제
Cppcheck (v2.3)	X	O
Yasca (v2.0.15)	O	X
Flawfinder (v2.21)	O	O

샘플 소스코드의 취약점 탐지 결과, Cppcheck는 버퍼 오버플로우 취약점을 탐지하지 못하고, 포맷 스트링 취약점만 탐지하였다. Yasca는 버퍼 오버플로우 취약점만 탐지하고, 포맷 스트링 취약점은 탐지하지 못하였다. 마지

막으로 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, 표 2와 같이 암호화 과정을 포함하는 오픈소스 5개를 선정하여 각 도구의 성능을 비교하고 평가하였다.

표 2. 선정한 오픈소스

오픈소스명	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 디렉토리를 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]

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 표 3에 나타내었다.

표 3는 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 포함하지 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.

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

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

표 3. 탐지된 전체 CWE ID와 개수 및 기존 연구 포함 및 미포함 CWE ID와 개수 비교

오픈소스명	소스코드명	탐지된 전체 CWE		기존 연구 미포함 CWE		기존 연구 포함 CWE	
		ID	개수	ID	개수	ID	개수
Cryptsetup (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_qcrypt.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
openssh (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	-	-
openssl (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: Flow, D: Detection accuracy)

오픈 소스명	소스 코드명	정적 분석 도구명								
		Cppcheck			Flawfinder			Yasca		
		C	F	D	C	F	D	C	F	D
crypt setup	crypto_ cipher_ kernel.c	476	Null Pointer Dereference	Low	-			-		
gnupg	keyid.c	-			-			476	Null Pointer Dereference	Low
	keyring.c	-			362	Race Condition	Medium	-		
Linux -pam	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	TOUTOU Race Condition	High			
Open ssh	cipher.c	401	Memory Leak	Low	-			-		
		476	Null Pointer Dereference	Low						
Open ssl	aes_ige.c	457	Use of Uninitialized Variable	None	-			-		

탐지된 CWE ID의 소스코드 개수를 기준으로 각 도구의 성능을 비교하면, Yasca는 오직 keyid.c로 가장 성능이 낮으며, 그다음으로 Cppcheck가 crypto_cipher_kernel.c, cipher.c, aes_ige.c로 3개, 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, cppchek는 CWE-401, 457, 476가 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, High로 높은 정확도를 가지는 CWE ID를 가장 많이 탐지하였다. Yasca는 gnupg의 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. 결론

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 모두 다른 결과가 도출되었으며, Flawfinder만 존재하는 취약점을 모두 탐지하였다. 이를 통하여 각 도구의 결과가 상이한 문제점을 도출하였으며, 오픈소스를 대상으로 각 도구의 성능을 비교하고 분석하였다.

암호화 과정을 포함하는 오픈소스 5개를 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, Yasca가 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, 표 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 예정이다. 그뿐만 아니라, 정적 분석 도구의 한계점인 오탐 및 미탐을 실증하기 위하여, 본 논문의 결과를 기반으로 동적 분석을 통하여 탐지된 취약점을 검증하는 연구를 진행할 예정이다.

감사의 글

"본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학지원사업의 연구결과로 수행되었음"(2019-0-01056)

[참고문헌]

[1] 류원옥, 강신각, “주요 오픈소스 라이선스 현황,”, 한국통신학회 학회지 (정보와 통신), 제36권, 제11호, pp. 32-41, 2019년 10월.

[2] 이용준, 안준선, 최진영, “보안약점 종류에 따른 정적분석 도구의 탐지 성능 연구,” 한국정보과학회 학술발표논문집, pp. 272-274, 2019년 12월.

[3] 서현지, 박영관, 김태환, 한격숙, 표창우, “Juliet과 STONESOUP 테스트 모음을 사용한 C/C++ 프로그램의 보안약점 탐지를 위한 정적 분석기 평가,” 한국컴퓨터정보학회 논문지, 제22권 제3호, pp. 17-25, 2017년 3월.

[4] C. L. Blackmon, D. F. Sang, and C. S. Peng, “Performance Evaluation of Automated Static Analysis Tools,” GSTF Journal on Computing (JoC), vol. 2, no. 1, pp. 214-219, Apr. 2012.

[5] Cppcheck, “A tool for static C/C++ code analysis”, http://cppcheck.sourceforge.net,

2021년 2월 5일 접속.

[6] Yasca, “Yet Another Source Code Analyzer,” http://scovetta.github.io/yasca, 2021년 2월 5일 접속.

[7] Flawfinder, https://dwheeler.com/flawfinder, 2021년 2월 5일 접속.

[8] NIST, “Test Suites,” https://samate.nist.gov/SRD/testsuite.php, 2021년 2월 5일 접속.

[9] 양대일, “시스템 해킹과 보안: 정보 보안 개론과 실습 (3판),” 한빛아카데미, 2018년 11월 12일 출판.

[10] GitLab, “cryptsetup,” https://gitlab.com/cryptsetup/cryptsetup, 2021년 2월 5일 접속.

[11] GnuPG, “The Gnu Privacy Guard,” https://gnupg.org, 2021년 2월 5일 접속.

[12] Linux Documentation, “pam.d(8) - Linux man page,” https://linux.die.net/man/8/pam.d, 2021년 2월 5일 접속.

[13] OpenSSH, https://www.openssh.com, 2021년 2월 5일 접속.

[14] OpenSSL, “Cryptography and SSL/TLS Toolkit,” https://www.openssl.org, 2021년 2월 5일 접속.