정 보 보 호 학 회 지
제 32 권 제5호, 2022. 10

공급망 보안을 위한 오픈소스 소프트웨어 취약점 관리 기술

홍 현 지*, 우 승 훈**, 이 희 조***

요 약

혁신적인 소프트웨어 개발을 위해, 소프트웨어 개발 환경에서 오픈소스 소프트웨어(OSS)를 활용하는 것은 하나의 개발 트렌드로 자리매김했다. 이러한 이점에도 불구하고, 적절한 관리가 이루어지지 않는 OSS 재사용은 취약점의 전파 문제나, 공급망 공격과 같은 위험한 보안 위협을 초래한다. 본 고에서는 OSS 재사용으로 인해 야기되는 다양한 보안 위협을 해결하기 위한 최근 연구들의 동향을 소개하고, 이런 연구들에 기반하여 제작된, OSS 보안성 향상을 위한 기술들을 소개한다.

Ⅰ. 서 론

오픈소스 소프트웨어(OSS)란 소스코드가 대중에게 공개되어 있는 소프트웨어로, 라이선스를 준수하는 범위 내에서 누구나 사용 및 배포가 가능하다. OSS의 재사용은 소프트웨어 개발 환경에서 효율성 증가와 시간 단축 등의 이점을 제공하며, 이로 인해 OSS를 활용하는 것은 하나의 개발 트렌드로 자리매김했다.

이러한 이점에도 불구하고, 적절한 관리가 수반되지 않은 OSS 재사용은 다양한 보안 문제를 일으킬 수 있다: (1) 취약점 전파, (2) 라이선스 위반, (3) 공급망 공격, 그리고 (4) 자원 관리의 비효율성 문제를 야기한다. 취약점 전파의 예로, Apache Log4j에서 발견된 Log4shell 취약점 (CVE-2021-44228)이 있다. Log4j 취약점은 Java의 로깅 유틸리티에서 발생한 취약점으로 35,000개 이상(전체 Maven 패키지의 8%)의 패키지에 영향을 주는 것으로 확인되었고, 해당 취약점을 이용해 시간당 10만 건의 공격 시도가 이루어지면서 수억대의 장치가 위험에 노출됐다 [1,2].

이렇듯, OSS 취약점은 많은 소프트웨어로 퍼져나갈 가능성이 농후하여 그 파급력이 크고, 사용하고 있는 OSS에 대해 정확히 파악하고 있지 않으면 보안 사고 발생 시 즉각적인 대응이 어렵다는 문제가 있다.

관리되지 않은 OSS로 인해 발생하는 보안 위협에 대응하기 위한 해결방안들은 크게 두 가지 방면에서 진행되어오고 있다: (1) 소프트웨어 구성요소 관리 (Software component management), 그리고 (2) 코드 감사(Code auditing)이다.

소프트웨어 구성요소 관리의 경우 소프트웨어가 어떤 OSS를 재사용하는지 파악하고, 이를 상용 및 OSS 구성요소 정보를 포함한 소프트웨어 명세서 (Software Bill of Materials, SBoM)의 형태로 관리하는 것이다. 특히, 2021년 5월 미국 행정부는 '국가 사이버보안 향상에 대한 행정명령'으로 각 소프트웨어 제품의 SBoM을 공개 제공할 것을 규제하고 있다 [3]. SBoM을 이용한 관리는 공급망의 투명성을 보장하고, 라이선스 준수 여부 판단, 취약한 소프트웨어 존재 여부까지 판별함으로써 보안 관리를 용이하게 한다. 코드 감사의 경우 정적, 동적 분석 도구를 이용하여 주기적으로 취약한 코드를 탐지하고 패치하여 보안 위협을 미연에 방지하는데 활용된다.

본 고에서는 OSS 보안 위협을 해결하기 위해 제안된 다양한 연구들을 소개한다 (부문 2 참고). 또한, 소개한 연구 기술에 기반하여 상용화되어 있는 도구들에

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 지역전략산업 융합보안 핵심인재 양성 사업(2022-0-01198), 블록체인 플랫폼 보안취약점 자동분석 기술개발 사업(2019-0-01697), SW 공급망 보안을 위한 SBOM 자동생성 및 무결성 검증기술 개발 사업(2022-0-00277) 결과로 수행되었음.

* 고려대학교 컴퓨터학과 (연구원, hyunji_hong@korea.ac.kr)
** 고려대학교 소프트웨어보안연구소 (연구교수, seunghoonwoo@korea.ac.kr)
*** 고려대학교 컴퓨터학과 (교수, heejo@korea.ac.kr)

〔표 1〕 OSS 보안 연구 기술 현황 및 동향 요약

구분	구성요소 탐지 기술	취약 코드 탐지 기술
기술 현황 및 문제	• 구성요소 탐지 분석 도구(SCA)로 재사용된 OSS 구성요소 탐지 • 대규모의 소프트웨어 구성요소를 확장성있고 정확하게 식별하는 것이 어려움 • 통일된 패키지 매니저로 관리되고 있지 않은 라이브러리를 식별하는 것이 어려움	• 정적, 동적 분석 도구를 활용해 취약코드를 탐지 • 대규모의 소프트웨어에 전파된 취약점을 확장성있고 정확하게 탐지하기 어려움 • 공개 취약점 데이터베이스에서만 바로 가져올 수 있는 패치만 고려한 충분하지 않은 데이터의 문제
연구 동향	• 코드레벨 탐지: 수정된 OSS 구성요소 탐지 기술 (부문 2.1.2-1) • 의존성 탐지: 시스템 레벨 및 애플리케이션 레벨의 라이브러리 탐지 기술 (부문 2.1.2-2)	• 취약 코드 탐지: 알려진 취약코드 및 수정된 취약코드 탐지 기술 (부문 2.2.2-1) • 보안 패치 수집: 취약 데이터베이스 구축 기술 (부문2.2.2-2)

대해서 기술한다 (부문 3 참고).

Ⅱ. 소프트웨어 취약점 탐지 연구 동향

본 고에서는 OSS 보안을 위한 연구를 [표 1]과 같이 두 가지로 분류한다: (1) 구성요소 및 (2) 취약코드 탐지를 위한 기술. 각 기술에 대한 현황 및 문제점을 제시하고 연구 동향에 대해 살펴본다.

2.1. 구성요소 탐지를 위한 기술

본 부문에서는 SBoM 제공에 기반이 되는 오픈소스 구성요소 탐지 기술에 대해 기술한다.

2.1.1. 현황 및 문제

SBoM을 생성하기 위한 가장 첫 걸음은, 재사용된 OSS 구성요소를 명확하게 식별하는 것이다. 비록 소스코드가 공개되어 있더라도, 다음의 이유들로 인해 OSS 구성요소를 식별하는 것은 도전적인 일이 되고 있다.

(1) 확장성 문제 : 수천만 코드라인으로 구성된 대규모의 소프트웨어의 구성요소를 식별하는 문제
(2) 정확도 문제 : OSS 코드의 부분 재사용 및 수정된 코드 재사용으로 인한 구성요소 식별의 어려움
(3) 의존성 (dependency) 식별 문제: 통일된 패키지 매니저로 관리되지 않는 Third-party 라이브러리 탐지의 어려움 (예, C/C++ 언어의 라이브러리)

이러한 요인으로 인해, 확장성있고 정확하게 OSS 구성요소를 식별하는 것은 어렵다.

2.1.2. 연구 동향

앞서 언급한 한계들을 극복하며 OSS 구성요소를 식별하기 위한 기술들이 제안되었다. 이 기술들은 크게 (1) 코드 레벨 탐지 기술 및 (2) 의존성 탐지 기술로 분류된다. 해당 기술들을 활용해 OSS 구성요소를 정확하게 식별한다면, 정확한 SBoM을 생성할 수 있게 되며, 이는 곧 OSS 보안성을 높이는 결과로 이어진다.

2.1.2.1. 코드 레벨 탐지 기술

코드 레벨에서, 수정된 OSS 구성요소까지도 정확하고 확장성 있게 탐지하는 기술로 CENTRIS [4]를 소개한다. CENTRIS는 수정된 구성요소를 정확히 탐지해내기 위해 코드 세분화(Code segmentation)기술을 활용한다. 여기서 코드 세분화 기술이란 OSS의 고유한 코드 영역만을 활용하여 구성요소를 탐지하는 것이다. 구체적으로, OSS 코드베이스에서 고유하지 않은 코드 부분, 즉 빌려온 코드 부분(borrowed code)을 제거하여 고유한 코드 부분만을 남긴다. 이후, 구성요소를 식별할 소프트웨어의 코드베이스와 어떤 OSS의 고유한 코드 영역들을 비교했을 때, 코드 유사도가 임계값 이상이면 해당 OSS를 재사용중인 것으로 판단한다.

OSS의 95%가 수정과 함께 재사용됨에도 불구하고, CENTRIS는 실험에서 91%의 정밀도(precision) 및

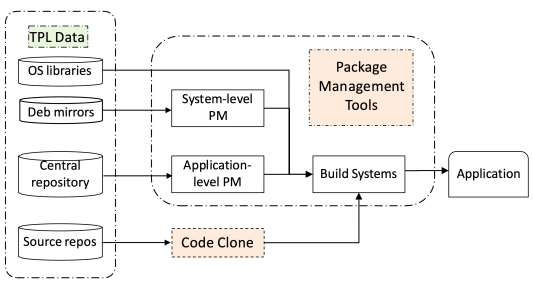
94%의 재현률(recall)로 구성요소를 식별하는데 성공했다. 또한 CENTRIS는 하나의 소프트웨어 당 평균 1분 이내의 시간으로 구성요소를 탐지했으며, 수천만 라인의 타겟 소프트웨어를 대상으로도 구성요소 식별에 성공하여, 높은 정확성과 확장성을 입증해냈다.

2.1.2.2. 의존성 탐지 기술

의존성 탐지 기술로는 CCScanner [5]가 있다. CCScanner은 의존성 레벨에서 범용적이고 정확하게 탐지할 수 있는 기술로써, 이들은 라이브러리를 다음의 두 가지로 분류한다: (1) 시스템 레벨의 라이브러리, (2) 애플리케이션 레벨의 라이브러리이다 ([그림 1] 참고).

먼저, 시스템에 레벨의 라이브러리는 시스템 패키지 매니저에 따른 OS 라이브러리를 지칭한다. 예를 들어, 리눅스의 경우 APT를, 맥(MacOS)의 경우 Homebrew를 고려한다. 애플리케이션 레벨의 라이브러리는 프로그래밍 언어별로 특화된 패키지 매니저에 기반한 라이브러리이다. 예를 들어, Java의 경우 Maven을, JavaScript의 경우 NPM을 고려한다. 통일된 패키지 매니저로 관리가 되고있는 Java, JavaScript 등의 언어는 패키지 정보를 담고 있는 파일을 간단하게 파싱함으로서 비교적 쉽게 의존성을 식별할 수 있다. 하지만, C/C++ 언어의 경우에는 통일된 패키지 매니저로 관리가 되어있지 않다. 따라서, 모든 패키지 정보를 저장하고 있는 파일이 명확하지 않은데, CCScanner는 빌드 시스템과 패키지 매니저에 따른 라이브러리를 추출할 수 있는 프로그램 내에 구성 파일(configuration file)을 스캔하여 의존성을 식별한다.

CCScanner를 구현하여 실험했을 때, 86% 정밀도, 80.1% 재현율의 높은 정확도로 의존성을 탐지했다.



〔그림 1〕 CCScanner에서 정의한 의존성 라이프사이클 〔5〕 (PM: 패키지 매니저, TPL: Third-party 라이브러리 지칭)

2.2. 취약코드 탐지 기술

본 부문에서는 소스코드레벨의 취약코드 탐지에 관련한 기술에 대해 소개한다.

2.2.1. 현황 및 문제

전파된 취약 코드를 탐지해내는 것은 소프트웨어 보안의 관점에서 아주 중요한 부분이다. 기존에도 다양한 취약코드 탐지 기술들이 제안되었지만, 이들은 다음과 같은 세 가지 한계점이 보였다:

(1) 확장성 문제: 수천만 코드라인으로 구성된 대규모 소프트웨어 취약점을 분석하는 문제
(2) 정확도 문제: 수정된 OSS 코드 재사용으로 인한 탐지의 어려움
(3) 충분하지 않은 취약점 데이터베이스: 공개 취약점 데이터베이스에서 바로 가져올 수 있는 패치만 고려

이러한 요인으로 확장성있고 정확하게 취약 코드를 탐지하는 것은 도전적인 일이 되었다.

2.2.2. 연구 동향

위의 한계점을 극복하며 정확하게 취약 코드를 탐지해내기 위해, 다양한 기술들이 제안되었다: (1) 취약코드 탐지 기술, 그리고 (2) 보안 패치 수집 기술.

2.2.2.1. 취약코드 탐지 기술

취약코드 탐지 기술로 VUDDY [6]와 MOVERY [7] 가 있다. OSS 재사용 특징 중 하나는, 개발자들이 이미 구현되어 있는 코드의 일부 또는 전체를 복제하여 사용하는 코드 클론 (Code clone) 이 자주 발생된다는 것이다. 코드 클론에 취약점이 포함되어있는 경우 이를 취약한 코드 클론(Vulnerable code clone)이라 지칭하며, VUDDY와 MOVERY는 취약한 코드 클론을 탐지하는 것을 목표로 한다.

VUDDY는 확장성있게 취약코드를 탐지하기 위해, 함수기반 코드클론 탐지 방식을 제안했다. 소프트웨어 프로그램에서 모든 함수를 추출하고, 각 함수에 대해 추상화(abstraction) 및 정규화(normalization)를 적용

하여 지문(fingerprint)을 생성한다. 추상화는 함수 내의 데이터 타입, 파라미터, 변수명 등을 동일한 문자로 대체하는 기술이며, 정규화는 함수 내의 공백, 주석 등 화이트스페이스 문자를 제거하는 기술이다. 이를 통해 취약 코드 전파 과정에서 발생한 함수의 작은 형상 변화에 유연하게 대처할 수 있다.

타겟 프로그램의 함수 집합과, 취약 코드의 함수 집합을 비교하여, 동일 함수가 존재하는 경우, VUDDY는 이 취약 함수가 타겟 프로그램에 존재하는 것으로 판단한다.

실험에서, VUDDY는 오탐율 0%의 높은 정확도로 취약 코드 클론을 탐지하는데 성공했다, VUDDY는 기존 기술 대비 2배 이상 빠른 속도로 취약 코드 클론 탐지에 성공했고, 수 천만 라인으로 구성된 타겟 프로그램내의 취약 코드도 탐지해 냄으로써 높은 확장성을 입증하였다.

MOVERY는 VUDDY에서 한 걸음 더 나아가, 취약한 코드가 많은 코드 변화와 함께 전파된 경우에도 정확하게 이를 탐지하기 위한 기술을 제안하였다. MOVERY의 핵심 기술은 취약 패치에서 핵심적인 코드라인만 추출하여 이를 활용하는 것이다. 구체적으로, MOVERY는 다음의 세 가지 핵심 코드라인을 고려한다: 보안 패치에서 삭제된/추가된 코드라인을 대상으로 (1) 취약점의 공개 시점 함수와 가장 오래된 버전의 취약 함수에 모두 존재하는 필수 코드 라인 (essential code lines), (2) 필수 코드 라인과 제어 및 데이터 의존성이 있는 종속적 코드 라인 (dependent code lines), (3) 필수 코드 라인이 속한 함수의 시작부터 필수 코드 라인까지 도달하는 경로 내에 존재하는 모든 제어 흐름 코드 라인(control flow code lines)을 고려한다.

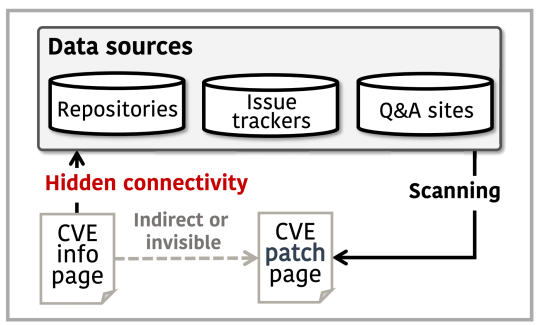
MOVERY는 취약점별로 핵심 코드라인들만 수집하여 취약/패치 시그니처를 생성한다. 이후, 타겟 소프트웨어의 특정 함수가 다음 조건을 만족할 때 이를 취약코드클론으로 판단한다: (1) 취약 시그니처의 모든 코드라인을 포함하고, (2) 패치 시그니처의 어떠한 코드라인도 포함하지 않으며, (3) 함수의 구문이 취약점의 공개 시점 함수나 가장 오래된 버전의 취약 함수와 유사할 경우이다. MOVERY는 GitHub의 인기 소프트웨어에서 96% 정밀도 및 96% 재현율로 수정된 취약 코드 클론을 탐지하는데 성공했으며, 이는 기존 기술 대비 6배 이상 더 많은 탐지 결과이다.

2.2.2.2. 보안 패치 수집 기술

취약점 패치 수집 기술로 xVDB [8]와 DICOS [9]가 있다. 보안패치는 취약점을 탐지하고 패치하는데 사용됨으로, 취약점 탐지 기술의 효용성을 향상시키기 위해서 충분히 많은 보안 패치를 수집하여 데이터베이스를 구축해야 한다. 하지만, 기존에 취약점을 수집하는 기술들은 한정된 데이터소스 (예, GitHub)만 고려하거나, 취약 공개데이터베이스(예, NVD)의 참고 URL에서 직접 가져올 수 있는 패치들만 수집하여, 커버리지에 한계점이 있었다.

이를 해결하고자, xVDB와 DICOS라는 연구들이 제안되었다. xVDB는 NVD와 같은 취약 공개데이터베이스와 보안패치 사이에 숨겨진 연결성을 활용하여 다양한 보안패치를 수집하는 기술이다 ([그림 2] 참고). 먼저, 한정된 데이터소스 문제를 해결하기 위해 다음의 3가지 데이터 소스를 모두 고려한다: 저장소 (repositories), 이슈 트래커(issue trackers), Q&A 사이트 (Q&A sites). 각 데이터소스에서 찾을 수 있는 보안 패치와 취약 공개 데이터베이스 사이의 연결성을 세 가지로 분류하여 수집 방법을 제안한다: (1) 직접 (direct), (2) 간접(indirect), 그리고 (3) 보이지 않는 (invisible) 링크. 직접링크의 경우 취약 공개 데이터베이스의 참고 URL에 보안패치가 직접적으로 제공되는 경우이며, 간접링크는 참고 URL에 접속 시 보안 패치로 연결되는 링크가 제공되는 경우이다. 보이지 않는 링크의 경우는 공개 데이터베이스에서 패치 URL이 앞선 두 가지 방식으로 제공되지 않는 경우이다.

직접 링크의 경우, 해당 보안패치 URL에 접근함으로써 쉽게 패치 수집이 가능하다. 간접 링크로 제공되는 패치의 경우, 참고 URL에 접속하여 보안패치로 연결되는 정보를 추출하여 보안패치를 수집한다. 취약



〔그림 2〕 xVDB 보안패치 수집 모델

공개 데이터베이스에 직, 간접적인 링크가 없을 때 보이지 않는 링크 수집 방식을 적용할 수 있는데, 이는 DICOS 연구를 활용한다.

DICOS는 스택오버플로우(StackOverflow)[1]와 같은 개발자 Q&A 사이트에서 안전하지 않은 코드스니펫을 탐지하는 기술이다. 구체적으로, 개발자 사이트의 각 포스트의 패치들을 분석하여 보안 패치 특징이 나타나는지 확인함으로서 안전하지 않은 포스트를 찾는 방식을 사용한다. DICOS는 보안 패치의 특징으로 다음 세 가지 특징을 고려한다: (1) 보안에 민감한 API의 변화, (2) 보안 관련 키워드의 변화, 그리고 (3) 코드의 제어 흐름의 변화이다. 개발자 사이트에서 DICOS 방식을 그대로 적용하여 수집하며, 저장소 및 이슈트래커에서는 보안 키워드로 “CVE-20”를 매칭하여 보안패치를 수집한다.

xVDB와 DICOS를 구현하여 내부 실험을 진행한 결과, 기존 기술 대비 약 4배 더 많은 CVE 패치를 수집했다. 뿐만 아니라, 개발자 사이트에서 CVE로 보고되지 않은 잠재적인 보안패치까지 수집하여 확장된 데이터베이스를 구축했다.

Ⅲ. 응 용

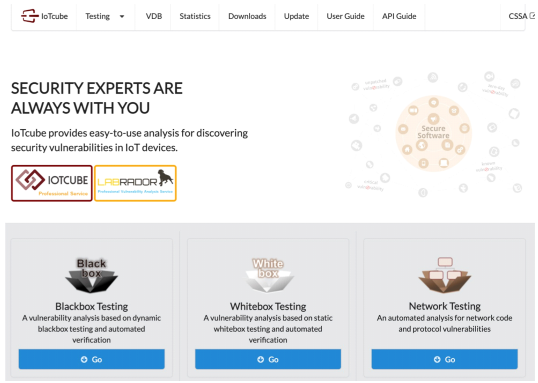
본 단원에서는 실제 상용화되어 있는 탐지 도구들을 소개한다. 소프트웨어의 취약점 탐지 서비스를 제공하는 오픈 플랫폼 IoTcube (부문 3.1), 그리고 상용 및 오픈소스 소프트웨어의 구성요소를 탐지하여 SBoM을 생성할 수 있는 Labrador (부문 3.2)를 소개한다.

3.1. IoTcube

IoTcube [10] 는 2016년부터 취약점을 탐지하기 위한 다양한 보안도구들을 제공해 온 오픈플랫폼이다. 상용 소프트에어 개발자를 비롯하여 오픈소스 개발자, IoT 기기 제조업자 등 많은 사용자들이 해당 플랫폼을 이용하고 있다. [그림 3]은 IoTcube 메인 화면이다.

IoTcube는 현재 블랙박스, 화이트박스, 네트워크 테스트를 지원하는 여러 분석 도구를 제공하고 있다. 본 단원에서는 본 고에서 소개한 xVDB 수집 기술을 탑재한 VDB와, VUDDY 기술을 구현한 Hmark에 대

1) https://stackoverflow.com/

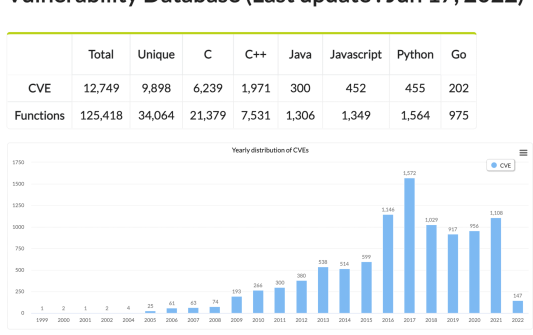


〔그림 3〕 IoTcube 메인 페이지

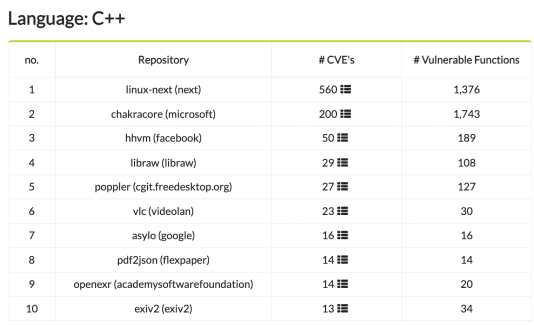
해 기술한다.

3.1.1. VDB (취약점 데이터베이스)

IoTcube는 xVDB 기술이 적용된 취약 데이터베이스, VDB를 지원한다. VDB는 가장 많은 CVE 취약점을 보고한 상위 6가지 언어(C, C++, Java, JavaScript, Go, Python) 지원한다. IoTcube VDB 메뉴에서 [그림



〔그림 4〕 IoTcube VDB 통계 (언어별, 연도별 통계)



〔그림 5〕 IoTcube VDB 페이지 통계 (저장소별 통계)

4], [그림 5]와 같이 언어별, 연도별로 저장소별로 통계치를 제공한다.

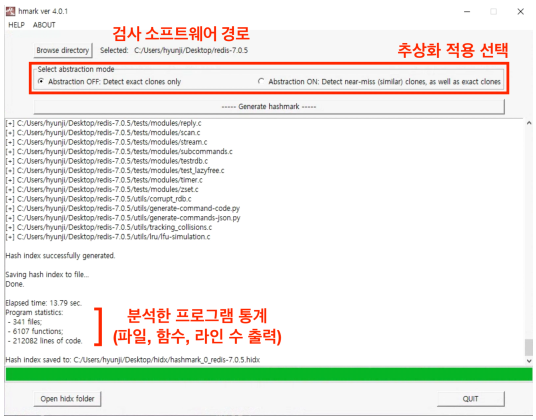
3.1.2. Hmark (취약한 코드클론 탐지기)

IoTcube는 VUDDY 기술에 기반한 취약 코드 클론 탐지 도구인 Hmark를 지원한다. 사용자는 Hmark를 이용해서 검사 소프트웨어의 함수 해시 지문을 생성하고, 이를 IoTcube 플랫폼에 업로드하여 취약 코드 클론 탐지 결과를 확인할 수 있다.

Hmark를 이용해 분석결과를 확인하는 방법은 다음과 같다. 먼저 Hmark 도구를 이용해 검사 소프트웨어를 분석해 “hidx” 파일을 생성한다. [그림 6]와 같이 사용자는 추상화 적용 여부를 설정하여 분석을 진행한다. 분석이 완료되면 총 분석한 파일, 함수, 코드 라인 개수를 출력하며, 분석한 함수의 해시값의 모음인 hidx 파일이 생성된다.

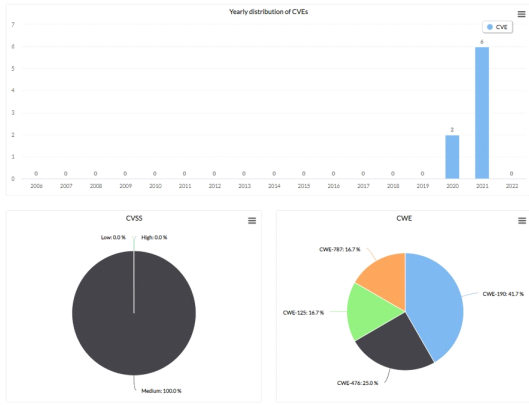
생성한 hidx 파일을 IoTcube 플랫폼에 업로드하여 결과를 확인할 수 있다. [그림 7]는 널리 활용되는 데이터베이스인 Redis[2]의 옛 버전 (v5.0.5)를 분석한 결과이다. 탐지된 취약 함수 개수, CVE 개수, 취약 함수의 원본 OSS 정보를 출력한다. 또한 [그림 8]과 같이 탐지된 CVE를 기준으로 연도별, CVSS, CWE 등의 통계치를 보여주어 다양한 관점에서 우선적으로 패치해야 할 취약점 정보를 제공한다.

또한, 검사 소프트웨어의 탐지된 CVE 취약점 정보만 출력해주는 것이 아닌 취약한 함수 위치 정보, 함수

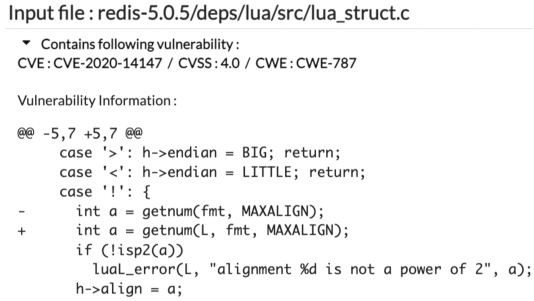


〔그림 6〕 Hmark 도구를 이용한 검사 소프트웨어 분석

〔그림 7〕 IoTcube의 Redis 분석 결과 페이지



〔그림 8〕 IoTcube의 Redis 분석 결과 통계 페이지



〔그림 9〕 IoTcube의 Redis 분석 결과 중 패치 화면
(Redis 최신버전에서는 패치가 완료된 취약 코드)

내용, 패치 정보까지 제공한다. 따라서, 개발자들은 제공된 정보를 통해 쉽게 취약한 코드를 찾고 패치할 수 있는 이점이 있다 ([그림 9] 참고).

2) https://github.com/redis/redis

3.2. Labrador

소프트웨어 개발 과정에서는 세 가지를 통합적으로 관리해나가는 것이 중요하다: (1) 구성요소 식별, (2) 취약코드 탐지, (3) 라이선스 탐지이다. Labrador [11]는 CENTRIS와 VUDDY 기술에 기반하여 통합 솔루션을 제공하고, SBoM 생성을 지원하는 도구이다.

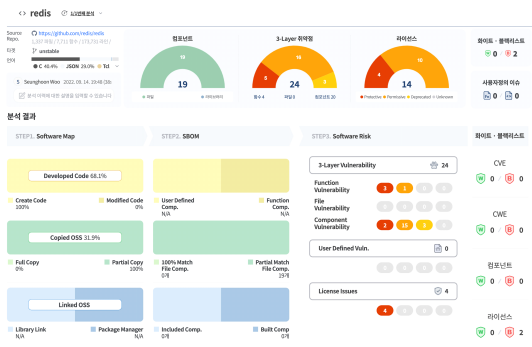
먼저 [그림 10]에서 볼 수 있듯이, Labrador는 그룹별로 다수의 소프트웨어를 관리할 수 있는 기능을 제공하며, 분석한 소프트웨어의 이력을 보유하고 있어 지속적으로 소프트웨어를 관리해 나가기에 용이하다.

Labrador를 이용해 소프트웨어를 분석하는 방법은 다음과 같다. 사용자는 검사 소프트웨어의 저장소 주소를 입력하거나, Labrador 스캐너를 이용한 분석 파일을 플랫폼에 업로드하여 분석결과를 확인할 수 있다. [그림 11]는 오픈소스 기반 데이터베이스 관리 소프트웨어인 Redis를 분석한 결과이다. 분석한 프로젝트의 메인 페이지에서는 탐지한 구성요소, 취약점, 라이선스 이슈 등을 요약하여 보여준다.

Labrador 탐지 결과에서 주목해 볼 것은, 3-Layer 취약점 분석 결과이다: (1) 함수 기반 취약점 탐지, (2) 파일 기반 취약점 탐지, 그리고 (3) 컴포넌트 (구성요소) 기반 취약점 탐지이다.

함수 기반 탐지방식은 함수 단위로 분석한 결과로서, VUDDY 기술에 기반한 탐지 결과를 제공한다. 부문 3.1 도구와 마찬가지로 취약함수에 대해 정보를 제공하며, 패치를 제안한다. 다음으로, 파일 기반 탐지방식은 파일 단위로 분석한 방식이다. 취약한 파일 내에 함수가 존재하지 않는 경우가 있으므로, 이러한 취약 파일을 탐지하고 패치를 제공한다. 또한, 파일 기반 탐

〔그림 10〕 Labrador 그룹 메인 페이지



〔그림 11〕 Labrador의 Redis 분석 결과 페이지

지방식은 프로그래밍 언어의 제약을 덜 받는다는 장점이 있다. 마지막으로, 컴포넌트 기반 탐지방식은 CENTRIS 기술을 활용해 구성요소를 식별한 후, 해당 OSS 구성요소 및 버전에 알려진 취약점이 존재하는지를 판단함으로써, 전파된 취약점을 탐지하는 방식이다.

[그림 12]은 Labrador가 출력하는 Redis에서 탐지된 컴포넌트의 목록을 나타낸다. 여기에는 컴포넌트의 이름, 버전, 라이선스 정보뿐만 아니라, 취약한 컴포넌트의 개수 등 컴포넌트의 보안성을 향상시킬 수 있는 다양한 정보를 제공한다. 또한, 취약한 컴포넌트의 경우 취약점이 존재하지 않는 안전한 버전 정보까지도 명시함으로써, 개발자가 취약점을 해결할 때 수행해야 할 추가 작업을 최소화하여 구성요소 관리의 편의성을 제공한다.

Labrador의 세 가지 분석 방식은 서로 다른 접근법으로 취약점을 분석하기 때문에, 상호보완적인 분석결과를 제공한다.

이외에도, Labrador는 탐지한 구성요소 결과를 기반으로 SBoM 표준문서인 SPDX, CyclonDX로 변환하는 SBoM 생성 기술을 제공한다. 현존하는 오픈소스 취약점 분석 도구는 많지만, 구성요소를 정확하게 탐지하여 SBoM 생성까지 지원하는 도구는 거의 없다 [12]. [그림 13]은 Labrador에서 생성한 Redis SBoM

〔그림 12〕 Labrador의 컴포넌트 분석결과 페이지

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
DocumentNamespace: http://labrador.labradorlabs.ai/sbom/2022/9/29/9e30617b-aa36-48e0-90e2-f335b079f35b/sbom-spdx.spdx
DocumentName: redis
SPDXID: SPDXRef-DOCUMENT

## Creation Information
Creator: Organization : LABRADOR LABS. (contact@labradorlabs.ai)
Creator: Tool : Labrador (https://labrdor.labradorlabs.ai)
Created: 2022-09-29T02:43:55Z

## Package Information
PackageName: hiredis@v1.0.0-rc1
SPDXID: SPDXRef-DC2C80D1C2058B22573AAA695B64BF97
PackageVersion: v1.0.0-rc1
PackageDownloadLocation: https://github.com/redis/hiredis.git
PackageLicenseConcluded: BSD-3-Clause
PackageLicenseDeclared: BSD-3-Clause
PackageCopyrightText: <text>NOASSERTION</text>
PackageAttributionText: <text>CVE-2021-32765 is patched in v1.0.1</text>
## Relationships
Relationship: SPDXRef-DC2C80D1C2058B22573AAA695B64BF97 CONTAINED_BY SPDXRef-root
FilesAnalyzed: false
```

〔그림 13〕 SPDX 양식으로 생성한 Redis SBoM

의 일부분을 나타낸다.

SBoM을 통해 재사용중인 구성요소를 명확히 알 수 있고, 어떤 버전이 재사용되었는지, 어떤 공급망 경로를 통해 내 소프트웨어에 포함되었는지 등의 정보까지도 파악할 수 있다. Labrador의 이런 기능들은 소프트웨어 생태계에서 안전하고 투명한 공급망 관리가 이루어질 수 있도록 돕는다.

Ⅳ. 결 론

본 고에서는 공급망 보안에 위협이 되는 OSS 취약점에 대한 인식을 제고하고, OSS 취약점을 해결하기 위한 기술 및 도구를 소개했다. OSS 취약점을 해결하기 위해, OSS 구성요소를 정확히 식별하고 취약코드를 탐지, 패치하는 것이 중요하다. 따라서, 본 고에서는 다음 세 가지 방안으로 더욱 안전한 소프트웨어 개발환경을 만들어갈 것을 권고한다.

- 소프트웨어 개발 수명주기(SDLC) 내에 SBoM을 도입하여 OSS를 체계적으로 관리하고 보안성을 점검해야 함
- CENTRIS, CCScanner와 같은 연구를 활용하여 소프트웨어 내에 OSS 구성요소 및 의존성을 정확히 파악하고 SBoM을 생성
- xVDB, DICOS와 같은 연구를 활용하여 풍부한 취약 데이터베이스를 구축하고, 취약코드 탐지 기술인 VUDDY, MOVERY 등을 통해 소프트웨어 내에 전파 취약점이 있는지 정기적으로 확인

참 고 문 헌

[1] The Wall Street Journal, “The Log4j Vulnerability: Millions of Attempts Made Per Hour to Exploit Software Flaw“, https://www.wsj.com/articles/what-is-the-log4j-vulnerability-11639446180

[2] Google, “Understanding the Impact of Apache Log4j Vulnerability”, https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html

[3] The White house, “Executive Order on Improving the Nation’s Cybersecurity”, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[4] Seunghoon Woo, Sunghan Park, Seulbae Kim, Heejo Lee, and Hakjoo Oh, "CENTRIS: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse", 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021

[5] Wei Tang, Zhengzi Xu, Chengwei Liu, Jiahui Wu, Shouguo Yang, Yi Li, Ping Luo, and Yang Liu, “Towards Understanding Third-party Library Dependency in C/C++ Ecosystem”, The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2022

[6] Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh, “VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery”, 2017 IEEE Symposium on Security and Privacy (SP), 2017

[7] Seunghoon Woo, Hyunji Hong, Eunjin Choi, and Heejo Lee, “MOVERY: A Precise Approach for Modified Vulnerable Code Clone Discovery from Modified Open-Source Software Components”, 31st USENIX Security Symposium (Security), 2022

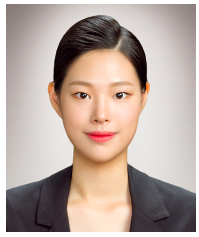
[8] Hyunji Hong, Seunghoon Woo, Eunjin Choi, Jihyun Choi, and Heejo Lee, “xVDB: A High-Coverage Approach for Constructing a Vulnerability Database”, IEEE ACCESS (ACCESS), 2022

[9] Hyunji Hong, Seunghoon Woo, and Heejo Lee, “DICOS: Discovering Insecure Code Snippets from Stack Overflow Posts by Leveraging User Discussions”, Annual Computer Security Applications Conference (ACSAC), 2021

[10] CSSA, “IoTcube”, https://iotcube.net/

[11] Labrador Labs, “Labrador”, https://labradorlabs.ai
[12] KAIST CSRC Weblog, “오픈소스 취약점 분석 누가 누가 잘하나?”, https://csrc.kaist.ac.kr/blog/2022/03/04

〈저자소개〉



홍 현 지 (Hyunji Hong)

2020 년: 한신대학교 컴퓨터공학 학사
2020년~현재 : 고려대학교 컴퓨터공학 석사과정
<관심 분야> 오픈소스 소프트웨어 보안, 취약점 탐지 및 분석

우 승 훈 (Seunghoon Woo)

2016년 : 고려대학교 컴퓨터공학 학사
2022년 : 고려대학교 컴퓨터공학 석, 박사 통합
2022년~현재 : LABRADOR LABS Inc. CSO
2022~현재 : 고려대학교 소프트웨어 보안연구소(CSSA) 연구 교수
<관심 분야> 오픈소스 소프트웨어 보안, 소프트웨어 구성요소 분석, 소프트웨어 취약점 탐지, 코드클론 탐지

이 희 조 (Heejo Lee)

종신회원
1993년 : 포항공과대학교 컴퓨터공학 학사
1995년 : 포항공과대학교 컴퓨터공학 석사
2000년 : 포항공과대학교 컴퓨터공학 박사
2000년~2001년 : Purdue Univ. CS 및 CERIAS 연구원
2001년~200년 : 안철수연구소 CTO
2004년~현재 : 고려대 컴퓨터공학 교수
2015년~현재 : 고려대 소프트웨어보안연구소(CSSA) 연구 소장
2018년~현재 : LABRADOR LABS Inc. Co-CEO
<관심 분야> DDoS 방지, 봇넷 탐지, 악성코드 탐지