The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.7, pp.299~310
eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.7.299

A Study on Open Source Version and License Detection Tool

Ki-Hwan Kim† · Seong-Cheol Yoon† · Su-Hyun Kim†† · Im-Yeong Lee†††

ABSTRACT

Software is expensive, labor-intensive, and time-consuming to develop. To solve this problem, many organizations turn to publicly available open source, but they often do so without knowing exactly what they're getting into. Older versions of open source have various security vulnerabilities, and even when newer versions are released, many users are still using them, exposing themselves to security threats. Additionally, compliance with licenses is essential when using open source, but many users overlook this, leading to copyright issues. To solve this problem, you need a tool that analyzes open source versions, vulnerabilities, and license information. Traditional Blackduck provide a wealth of open source information when you request the source code, but it's a heavy lift to build the environment. In addition, Fossology extracts the licenses of open source, but does not provide detailed information such as versions because it does not have its own database. To solve these problems, this paper proposes a version and license detection tool that identifies the open source of a user's source code by measuring the source code similarity, and then detects the version and license. The proposed method improves the accuracy of similarity over existing source code similarity measurement programs such as MOSS, and provides users with information about licenses, versions, and vulnerabilities by analyzing each file in the corresponding open source in a web-based lightweight platform environment. This solves capacity issues such as BlackDuck and the lack of open source details such as Fossology.

Keywords : Open Source, Version, License, Source Code Similarity

오픈소스 버전 및 라이선스 탐지 도구에 관한 연구

김 기 환† · 윤 성 철† · 김 수 현†† · 이 임 영†††

요 약

소프트웨어는 개발하는 과정에서 많은 비용과 시간이 소모된다. 이를 해결을 위해, 많은 기업들이 공개된 오픈소스를 사용하지만 대부분 오픈소스에 대해 정확히 파악하지 않은 채 사용하고 있다. 특히, 구버전 오픈소스 사용으로 인한 보안 취약점 문제와 라이선스 사항을 간과한 저작권 문제가 발생하고 있다. 따라서, 오픈소스의 버전, 취약점 및 라이선스 정보를 분석하는 도구가 필요하다. 기존의 블랙덕은 오픈소스의 상세한 구성 요소를 제공하지만 환경 구축에 큰 부담을 따르게 된다. 또한, Fossology는 라이선스를 탐지할 수 있지만, 자체 데이터베이스가 존재하지 않아 그 외의 다른 정보들을 제공하지 않는다. 본 논문에서는 소스 코드 유사도 측정을 통해 오픈소스를 파악하여 해당 버전 및 라이선스를 탐지 하는 도구를 제안한다. 본 제안 방식은 TF-IDF 및 코사인 유사도를 기반하여 MOSS라는 기존 소스 코드 유사도 측정 도구인 프로그램보다 정확도를 향상시킨다. 또한, 웹 기반의 경량 플랫폼 환경에서 구축함에도 오픈소스를 분석하여 라이선스, 버전 및 취약점을 제공할 수 있다. 이를 통해, 앞선 블랙덕의 환경 구축에 요구되는 부담 및 Fossology의 오픈소스의 상세 정보의 미제공 문제를 해결할 수 있다.

키워드 : 오픈소스, 버전 판별, 라이선스 탐지, 소스 코드 유사도

1. 서 론

4차 산업혁명 시대에서 소프트웨어 기술은 주요한 역할을 수행하며, 향후에도 그 중요성은 더욱 강조될 것으로 전망된다[1]. 소프트웨어는 기본적으로 사용자 요구사항 분석 기반의 시스템 아키텍쳐 설계 후 소스 코드 작성을 통해 개발된다. 이러한 과정에서 많은 인력이 요구되고 많은 개발 시간과 비용이 소요된다는 단점이 존재한다[2].

위와 같은 문제를 해결하기 위해 많은 기업에서는 오픈소스(Open Source)를 도입하고 있다. 오픈소스는 누구나 무료

※ 이 논문은 한국콘텐츠진흥원의 2024년도 SW저작권 생태계 조성 기술개발 사업(과제명 : 클라우드 서비스 활용 구축 형태별 대규모 소프트웨어 라이선스 검증 기술개발, 과제번호: RS-2023-00224818, 기여율: 50%)과 과학기술정보통신부의 재원으로 한국연구재단(No. 2022R1A2B5B01002490)의 지원을 받아 수행됨.
※ 이 논문은 2023년 ACK 2023의 우수논문으로 "OSS 유사도 및 라이선스 분석 플랫폼에 관한 연구"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 순천향대학교 소프트웨어융합학과 석사과정
†† 정 회 원 : 순천향대학교 컴퓨터소프트웨어공학과 교수
††† 종신회원 : 순천향대학교 컴퓨터소프트웨어공학과 교수
Manuscript Received : December 27, 2023
First Revision : April 24, 2024
Accepted : May 28, 2024
* Corresponding Author : Su-Hyun Kim(kimsh@sch.ac.kr)

로 커스터마이징할 수 있는 공개된 소스 코드이므로, 소프트웨어 개발 과정에서 요구되는 여러 문제들을 해결하여 간편하게 맞춤형 소프트웨어를 개발할 수 있다.

하지만 오픈소스를 정확하게 파악하지 않고 소프트웨어를 개발하는 것은 많은 문제를 야기한다. 아래와 같은 문제들은 빈번하게 발생하며 심각한 문제들을 초래하고 있다.

1.1 오픈소스 버전 취약점 문제

오픈소스는 수정될 때 마다 여러 버전이 공개되고 있다. 특히, 많은 구버전 오픈소스에는 주로 취약한 코드들이 포함되어 있다[3]. 또한, 기업 및 개발자들은 현재도 최신 오픈소스가 아닌 구버전의 오픈소스를 사용하고 있다. 이때, 해커가 구버전의 취약점을 공격하면 데이터나 시스템의 기밀성, 무결성 및 가용성을 손상시킬 수 있다. 취약한 버전의 오픈소스 사용으로 인한 보안 결함의 대표적으로 2021년 11월에 알리바바 클라우드 보안팀에 의해 발견된 Log4J 사태가 있다[4].

1.2 오픈소스 라이선스 미 준수 문제

오픈소스는 무료로 사용할 수 있지만, 무분별한 사용은 지양해야 한다. 오픈소스는 지식재산권으로부터 보호를 받으며, 독점 소프트웨어 같은 법적 저작물이다. 따라서, 이용 규정 사항 및 권한이 명시된 라이선스 사항을 반드시 준수해야 한다. 라이선스는 대표적으로 Table 1과 같이 General Public License(GPL), Affero General Public License(AGPL), Lesser General Public License(LGPL), Berkeley Software Distribution(BSD), Mozilla Public License(MPL), Apache 및 MIT 등이 있다.

하지만 많은 기업에서는 사용 중인 오픈소스를 정확히 인지하지 못한 채 사용하고 있다. 이때, 라이선스를 간과할 시, 저작권 침해로 인한 법적 분쟁으로 이어진다[5]. 대표적으로 2022년 10월에는 마이크로소프트(Microsoft)가 코파일럿(Copilot)의 라이선스를 위반한 저작권 문제가 존재했다.

또한, 소프트웨어 개발에선 여러 개의 오픈소스들을 혼용하는 경우가 많다. 이때, 오픈소스 마다 적용되어 있는 라이선스 사항이 호환되지 않을 시 충돌이 발생할 수 있다[6]. 예를 들어, Table 1과 같이 MIT는 소스 코드 공개 의무가 없지만, GPL은 공개가 필수적이다. 이때, MIT와 GPL이 적용된 두 개의 오픈소스를 사용하여 소프트웨어 산출물을 배포하면 라이선스 충돌로 인하여 저작권 문제가 발생한다.

Table 1. Open Source License Examples

	Free to use	Modifiable	Disclosure
GPL	O	O	O
AGPL	O	O	O
LGPL	O	O	O
BSD	O	O	X
MPL	O	O	O
Apache	O	O	X
MIT	O	O	X

1.3 연구 목표

본 논문에서는 오픈소스 버전 및 라이선스를 정확하게 파악하지 않아 발생하는 문제들을 예방하기 위한 탐지 도구에 관한 연구를 수행한다.

먼저, 소프트웨어 개발에 사용하는 소스 코드가 어떤 오픈소스인지 인식할 수 있어야 한다. 오픈소스는 다양한 기술로 분석이 가능하지만, 본 논문에서는 소스 코드 유사도 측정을 통해 오픈소스 분석을 수행한다. 이후에는 버전 정보를 탐지하여 취약점 및 권고 사항을 확인할 수 있어야 한다. 또한, 해당 오픈소스의 라이선스 사항을 확인할 수 있어야 한다. 마지막으로, 여러 소스 코드를 업로드하고 두 개 이상의 서로 다른 라이선스가 적용된 오픈소스로 분석될 시, 라이선스 충돌 여부를 파악하여 상호 호환이 되는지에 대해서도 탐지할 수 있어야 한다.

이를 통해, 기업 및 개발자들은 소프트웨어 개발에 사용하려는 오픈소스 정보를 명확하게 확인할 수 있다. 또한, 버전을 탐지하여 보안 취약점에 대응할 수 있으며, 라이선스 사항 위반 및 충돌 여부를 탐지하여 저작권 침해로 인한 분쟁 또한 감소시킬 수 있다.

2. 관련 연구

2.1 오픈소스 라이선스 점검 도구

최근 소프트웨어 개발에는 오픈소스 사용의 증가로 버전 보안 취약점 문제와 라이선스 미 준수에 대한 관리가 점차 중요해지고 있다. 이로 인해, 다양한 오픈소스 점검 도구들이 개발되었으며, 개발자 및 기업들이 안전하게 활용할 수 있도록 솔루션을 제공하고 있다. 따라서 본 절에서는 무분별한 오픈소스 사용으로 인해 발생했던 문제들을 해결하기 위한 기존의 점검 도구들에 대한 여러 가지 특징들에 대하여 기술한다. 대표적으로는 블랙덕 소프트웨어와 Fossology가 있다.

1) 블랙덕 소프트웨어

블랙덕 소프트웨어(Blackduck Software)는 시놉시스 사에서 운영하고 있는 널리 알려진 오픈소스 점검 도구 중 하나이다. 블랙덕은 Fig. 1과 같이 동작하며, 소스 코드에 사용된 오픈소스를 탐지 후 각 오픈소스에 대한 라이선스와 보안 취약점을 식별한 결과를 제공한다[7].

블랙덕은 오픈소스 패키지를 수집한 후 해시화하여 데이터베이스에 저장하여, 사용자가 소스 코드 점검을 요청하면 데이터베이스 내 오픈소스와 비교 분석한다. 오픈소스 탐지 후, 라이선스 및 충돌 탐지와 버전 취약점, 취약점 상세 경로 및 해결되어 배포된 버전 등 세부 정보를 가시화한다. 또한, 블랙덕은 소프트웨어 구성 요소를 포함한 자재 명세서를 의미하는

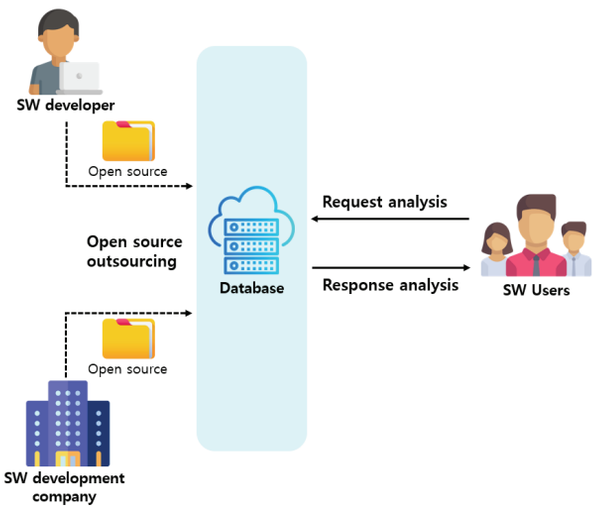

Fig. 1. Structure of Blackduck Software

Software Bill of Material(SBoM)을 제공한다[8]. 이를 통해, 실제로 사용자가 어떤 오픈소스를 사용하고 있는지 명확하게 파악할 수 있어 오픈소스를 사용한 소프트웨어 공급망에서의 보안을 강화할 수 있다.

블랙덕은 오픈소스 소프트웨어 라이선스 및 보안 측면에서 취약점을 효과적으로 분석하기 위해 Table 2와 같은 기법들을 사용한다. 상기와 같은 방법으로 사용자의 소스 코드가 어떤 오픈소스이며, 어떤 버전과 라이선스가 적용되었는 지의 결과를 제공할 수 있다.

블랙덕은 오픈소스 통합 관리 솔루션으로써, 특히 방대한 정보를 제공함의 이점으로 기업 단위에서 주로 사용하고 있다. 블랙덕을 사용하기 위해서는 Jenkins와 같은 Continuous Integration(CI) 도구가 필요하다. 또한, 사용자의 로컬 PC에 구축하는 데에 있어서 80GB의 저장 공간이 요구되게 된다. 만약, 개발자가 단순히 하나의 소프트웨어를 상용화하기 위해, 사용하고 있는 소스 코드의 오픈소스 라이선스 및 버전 정보를 파악하기 위해 블랙덕을 사용해야 할 경우가 생기면 상기와 같은 구축 과정을 수행해야 한다.

Table 2. Blackduck Software Analytical Methods

Analytical methods	Detail contents
Dependence analysis	Track interdependencies between declared components and built libraries, modules, packages, etc
Code printing analysis	Analyzing file or directory metadata signatures to detect modified open sources
Match code pieces	Identify open source code fragments containing potential copyright and licensing obligations
Binary analysis	Analyze compiled software, firmware, or installation programs without source code access
Custom components detection	Identify non-open source internal or third-party commercial components with string search and code printing

Table 3. Fossology Analytical Methods

Analytical methods	Detail contents
String-based analysis	Detection of license-related keywords, patterns, etc. based on strings
Dependence analysis	Track interdependencies between declared components and built libraries, modules, packages, etc
Keyword matching	Identify licenses by detecting specific keywords you specify
File hash analysis	Create a hash value and identify a license by comparing it to a stored hash value

2) Fossology

Fossology는 리눅스 기반으로 개발되었으며, 오픈소스 라이선스를 스캔하여 결과를 제공하는 도구이다[9]. Fossology는 Docker를 기반으로 솔루션을 사용할 수 있으며, Vagrant와 VirtualBox를 사용해야 한다.

사용자가 오픈소스로 작성된 소스 코드를 업로드하면 라이선스를 스캔 후 추출하여 제공한다. Fossology의 라이선스 스캔은 Table 3의 기법들로 수행된다. 이때, 사용자는 소유한 소스 코드가 오픈소스임을 알고 있는 상황이다.

하지만 Fossology는 오픈소스 라이선스 스캔 기능만 제공하며, 오픈소스 버전 취약점 등 세부적인 정보를 제공하지 않는다. Fossology는 자체적인 오픈소스 데이터베이스가 구축되어 있지 않기 때문에, 사용자가 소유한 소스 코드가 어떤 오픈소스인지를 모르는 경우 이를 파악하고 싶어도 저장된 오픈소스들과 비교를 수행할 수 없기에, 이에 대한 결과를 제공할 수 없다.

2.2 소스 코드 유사도 측정

오픈소스 버전 및 라이선스를 위해서는 소유한 소스 코드가 어떤 오픈소스인지 파악해야 한다. 이를 위해서는, 소유한 소스 코드와 오픈소스의 내부 소스 코드 간의 유사도를 측정함으로써 확인할 수 있어야 한다.

소스 코드 유사도란 두 소스 코드 간 문맥 상 얼마나 유사한 지에 대해 나타낸 수치 값이다[10]. 오픈소스 분석에 소스 코드 유사도 측정을 활용하면 오픈소스가 무엇인지에 대한 정보를 간편하고 명확하게 파악할 수 있다.

소스 코드 유사도 측정 기법은 다양하게 존재하며, 아래에서는 관련 유사도 측정 기법의 연구에 대하여 기술한다.

1) TF-IDF

Term Frequency-Inverse Document Frequency(TF-IDF)는 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 표현하는 수치를 의미한다[11]. 문서의 핵심 단어를 추출하거나 문서들 사이에서 유사한 정도를 빈도 분석하여 판별할 수 있으며, 소스 코드 유사도 측정에서도 TF-IDF를 사용할 수 있다. 유사도 측정의 중요도가 낮은 요소들에 대한 전처리 작업

후, 각 소스 코드 문서 d에서 각 키워드 t의 빈도를 계산한다. 이는 각 키워드가 문서 내에서 얼마나 자주 등장하는지를 나타내며 Equation (1)과 같이 TF로 표현된다.

TF 계산 후, 다음으로 소스 코드의 각 키워드 t의 역 문서 빈도 IDF를 Equation (2)와 같이 계산한다. IDF는 자주 나오는 키워드에 대해서는 낮은 가중치를 부여하고 그렇지 않은 키워드에는 높은 가중치를 부여한다.

앞에서 구한 TF와 IDF 값을 구하면 $TF-IDF$ 값을 Equation (3)과 같이 구할 수 있고, 비교하는 소스 코드들은 $TF-IDF$ 가중치 기반의 방향성을 가진 소스 코드 벡터로 표현된다. 소스 코드 벡터는 자체적으로 내적을 활용하여 유사도를 측정할 수 있다.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d} \quad (1)$$

$$IDF(t) = \log\frac{N}{1+df} \quad (2)$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t) \quad (3)$$

2) 유클리드 거리

유클리드 거리(Euclidean Distance)는 두 점 간의 거리를 구하는 기법이다. 소스 코드 유사도 분석에서는 벡터화 한 소스 코드의 벡터 간 거리를 계산해서 측정할 수 있다[12].

두 개의 소스 코드 벡터 간의 유클리드 거리는 각 차원의 차이를 제곱하여 더한 후, 제곱근 값을 취한다.

이를 통해, 유클리드 거리는 0에서 1 사이의 값으로 반환하여 Equation (4)과 같이 계산하여 유사도를 측정한다.

이 방법은 벡터 간의 거리를 통해 유사도를 측정하며, 소스 코드 간의 구조적 유사도를 고려한다. 하지만 유클리드 거리는 벡터의 크기에 민감하게 반응하기 때문에, 소스 코드의 길이가 차이가 클 경우 정확한 유사도 측정이 어렵다.

$$similarity = \sqrt{\sum_{u=1}^{n}(Ai - Bi)^2} \quad (4)$$

3) 코사인 유사도

코사인 유사도란, 두 벡터간의 방향적 유사도를 측정하는 지표로 소스 코드 유사도 측정에서 빈번하게 사용된다.

앞서 언급한 소스 코드 벡터에 단순 내적을 적용하면, 소스 코드 문서 길이에 의해 영향을 받게 되며 크기에 의해 정확도가 낮아지게 된다. 하지만 코사인 유사도는 방향성을 고려하면서도 벡터의 크기에 영향을 받지 않는다. 따라서 소스 코드 길이의 영향을 받지 않으면서도 정확하고 유사도를 측정할 수 있다.

코사인 유사도는 Equation (5)와 같이 두 소스 코드 벡터 A와 B의 내적에서 노름의 곱을 나누어 계산한다. 세부적으로는 Equation (6)과 같이 표현되며, 이와 같이 측정된 코사인 유사도 값은 Fig. 2와 같이 0-1 사이의 범위를 가진다. 이때,

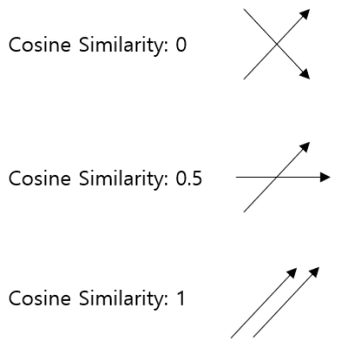

Fig. 2. Cosine Similarity Result

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\ \|B\|} \quad (5)$$

$$\frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (6)$$

1에 가까울수록 두 소스 코드가 높은 유사도로 측정됨을 표현한다. 이와 같이 $TF-IDF$와 코사인 유사도를 이용하면 유사도를 더욱 정확하게 측정할 수 있다[13]. 따라서 소스 코드 벡터화 후 코사인 유사도 기법을 적용하면 정확하게 소스 코드 유사도를 측정할 수 있다.

4) Measure Of Software Similarity(MOSS)

소스 코드 유사도 측정 프로그램으로는 스탠포드 대학교에서 개발한 MOSS가 있다. 핑거프린팅 기법을 사용하는 MOSS는 학생들의 프로그래밍 과제의 소스 코드를 유사도를 측정하는 등 교육 기관에서 주로 사용되고 있다[14].

하지만 MOSS는 널리 사용됨에도 불구하고, 정확도 측면에서 부족한 점이 다수 존재한다. 특히, 변수 및 함수명을 변경하면 유사도가 대폭 하락하게 된다. 또한, 동일한 소스 코드 비교에는 100%가 나와야 하지만, 이 또한 정확히 도출되지 않는다. 따라서 MOSS와 같이 소스 코드 유사도를 측정하는 프로그램보다 정확성을 향상시킨 유사도 기법을 활용하여 오픈소스 유사도를 측정해야 한다.

본 논문에서는 앞 절에서 언급한 TF-IDF와 코사인 유사도를 기반으로 유사도를 측정한다[15]. 또한, 중요하지 않은 키워드는 비교 분석 이전에 사전에 제거하는 전처리 작업을 수행하여 정확도 측면의 문제를 해결한다.

3. 제안 방식

본 장에서는 소스 코드 유사도 측정을 통한 오픈소스 분석 후 버전 및 라이선스 탐지 기법에 대하여 제안한다. 본 제안 방식의 전체 시나리오는 Fig. 3과 같다.

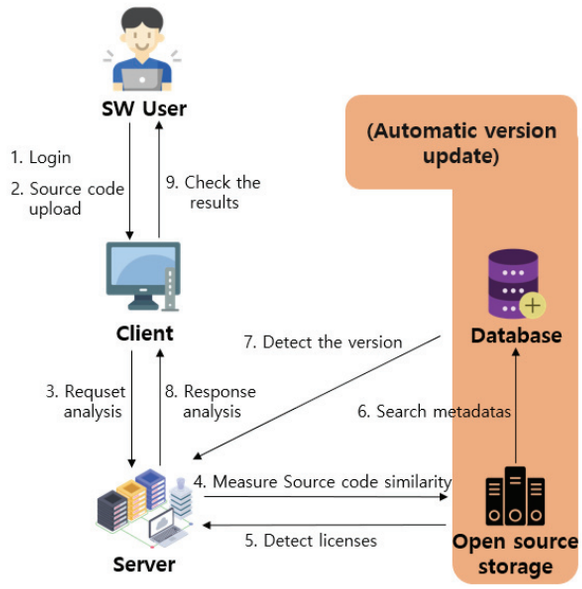

Fig. 3. Overall Scenario

3.1 제안 방식 구축 환경

클라이언트는 Hypertext Markup Language(HTML), Cascading Style Sheets(CSS), JavaScript로 구성하고 서버는 Python 기반의 Django를 사용한다. 클라이언트와 서버는 unified Web Server Gateway Interface(uWSGI)를 활용해 상호 통신이 가능하도록 한다.

오픈소스 버전 및 라이선스 탐지를 하기 위해서는 사용자의 소스 코드와 비교할 오픈소스가 필요하다. 본 제안 방식에서는 GitHub Application Programming Interface(API) 연동이 아닌, 서버에 오픈소스 저장 공간을 구축한다. 이때, 오픈소스 등록 및 업데이트를 위한 등록 기능을 구현한다.

또한, 데이터베이스에는 오픈소스 버전, 등록 날짜, 등록자의 식별 ID와 같은 구성 요소를 저장한다.

아래 절에서는 본 제안 방식의 기능들에 대해 기술한다.

3.2 오픈소스 등록 및 데이터베이스 구축

오픈소스 등록은 오픈소스 개발자가 개발한 오픈소스를 플랫폼에 등록하여 배포하는 기능이다. 입력한 내용을 기반으로 서버에 오픈소스 프로젝트를 저장하며, 데이터베이스에는 등록될 때의 오픈소스 메타데이터를 저장한다.

1) 오픈소스 등록 및 버전 업데이트

오픈소스는 개발자는 Fig. 4와 같은 폼에 프로젝트명, 라이선스, 소스 코드, 언어 및 README 내용을 입력한다. 이때, 프로젝트명+버전의 이름의 디렉토리를 생성한 후, 라이선스 파일(License)과 README 마크다운 파일(REAMDE.md)을 추가하여 서버 저장 공간에 저장한다.

본 논문에서 제안하는 버전 관리는 최초 등록 시 1.0으로 자동 생성하며, 이후 같은 개발자가 같은 오픈소스를 플랫폼에

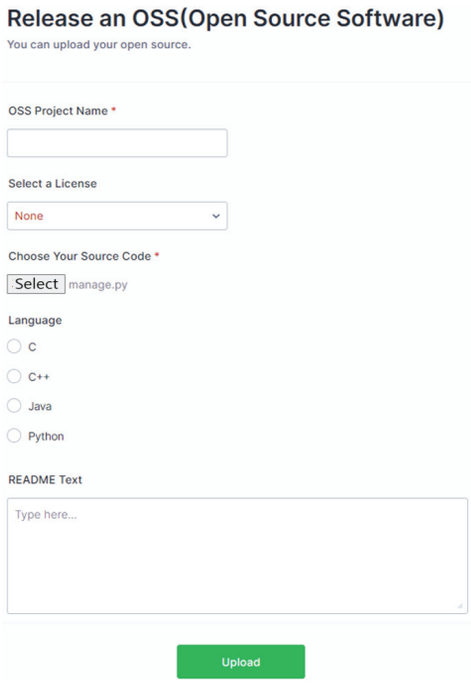

Fig. 4. Open Source Deployment Form

다시 등록할 때, 버전 업데이트 버튼을 표기한다. 이때, 대규모 패치를 수행한 경우 '주 번호 업데이트'를 클릭하고, 버그 패치나 편의성 개선 등의 작은 패치가 수행되었다면 '부 번호 업데이트'를 클릭한다. '주 번호 업데이트' 클릭 시, 버전의 주 번호를 자동 +1 증가시키며, '부 번호 업데이트'를 클릭 시, 버전의 부 번호를 자동으로 +1씩 자동 증가시켜 업데이트를 수행한다. 이와 같은 버전 업데이트를 통해, 효율적으로 버전 관리가 가능하다.

2) 오픈소스 데이터베이스 구축

오픈소스 데이터베이스는 SQLite로 구축하며, 오픈소스가 등록될 때, Fig. 5와 같이 날짜, 등록자 식별 name 그리고 버전 정보를 추가하여 메타데이터의 속성으로 설정한다. 속성의 id는 등록되는 오픈소스의 순서를 의미한다. 새로 업데이트된 버전의 경우의 id 역시 플랫폼에 추가된 오픈소스의 순서대로 할당된다.

```
migrations.CreateModel(
    name='Metadata',
    fields=[
        ('id', models.BigAutoField(auto_created=True, pr
        ('name', models.CharField(max_length=50)),
        ('version', models.CharField(max_length=50)),
        ('disributor', models.CharField(max_length=50)),
        ('language', models.CharField(max_length=50)),
        ('licen', models.CharField(max_length=50)),
        ('content', models.TextField()),
        ('create_date', models.DateTimeField()),
    ],
),
```

Fig. 5. Open Source Database Columns

3.3 소스 코드에 대한 오픈소스 분석 과정

Fig. 6과 같은 소스 코드 업로드 폼에 사용자가 로컬 PC에 소유하고 있는 소스 코드 파일(C, C++, JAVA, Python) 및 소스 코드 압축 파일을 업로드한다. 이때, 올바른 소스 코드의 확장자가 아닌 파일이 업로드가 된다면 알림 메시지를 설정하여 에러 발생을 사전에 방지한다.

소스 코드 업로드 후 분석하기 버튼을 클릭하면 서버로 소스 코드가 전송되며 분석 과정이 진행된다. 분석이 수행되는 과정은 Fig. 7과 같다. 소스 코드 분석이 요청된 후, 서버에 있는 오픈소스를 구성하는 소스 코드들과 유사도 측정을 수행하며 가장 유사한 오픈소스에 대한 결과를 제공한다.

1) 소스 코드 전처리 단계

사용자가 요청한 소스 코드에 적용되는 오픈소스 버전 및 라이선스를 분석하기 위해서는 어떤 오픈소스가 적용되었는지부터 파악해야 한다. 이를 위해, 소스 코드 유사도 측정을 통해 가장 높은 유사도를 보이는 오픈소스를 분석해야 한다.

Fig. 6. Open Source analysis form

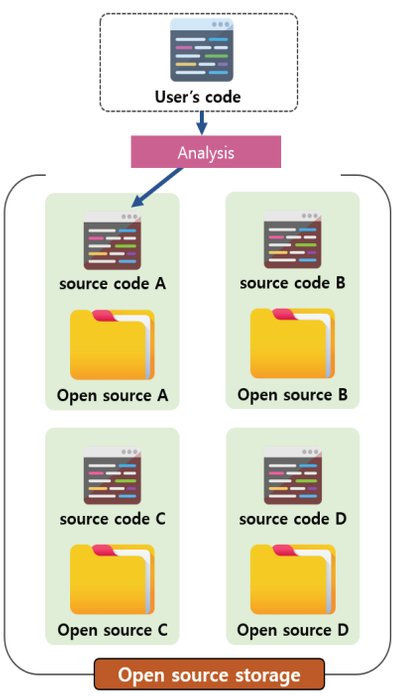

Fig. 7. Process of Finding the Similar Open Source

Output String Cancellation	Source Code Lowercase

for(int i=0; i<5; i++) //comments
{
if(MyNum==0) //comments
{
printf();
} canceled
}
	for(int i=0; i<5; i++) //comments
	{
	if(mynum==0) //comments
	{
	printf();
	}
	}

Stopwords Cancellation	**Punctuations Cancellation**

for(i=0; i<5; i++)
{
if(i%2==0)
{
printf();
}
}
	for i=0 i<5 i++ if mynum==0 printf

Fig. 8. Source Code Pre-processing

하지만 대부분의 소스 코드는 핵심 구문을 제외하고 중요도가 낮은 여러 단어들이 포함되어 구성되어 있다.

본 제안 방식에서는 이와 같은 중요도가 낮은 구문들을 제거한 후, 핵심 구문들로 유사도를 측정할 수 있도록 전처리 과정을 수행한다. 전처리 과정은 소스 코드 출력 문자열 제거, 소스 코드 소문자화, 불용어 제거 및 구두점 제거로써, Fig. 8과 같이 총 4단계로 구성된다.

a) 소스 코드 출력 문자열 제거

첫 번째로, 소스 코드 출력 문자열은 프로그램이 실행될 때 생성되는 문자열로, 주로 화면에 출력되거나 파일에 기록된다. 사용자들에게 정보를 전달하기 위해 활용되며 콘솔 또는 Graphical User Interface(GUI) 화면에서 출력된다.

이러한 출력 문자열은 유사도를 하락시키는 가장 큰 원인이 된다. 구문 구조가 동일한 소스 코드여도 출력 문자열의 수정이 이루어지면 유사도 수치가 크게 하락하게 될 수 있다. python의 예로, print()에서 이름값을 입력받을 때 출력문이 “name: ”인 경우와, “Input name: ”인 경우, 실행 결과에는 아무런 영향을 끼치지 않지만 유사도가 하락하게 된다.

따라서 본 제안 방식에서는 정규 표현식을 사용하여 따옴표로 둘러싸인 출력 문자열을 Fig. 9와 같이 정적으로 제거하는 과정을 수행한다. 변수의 값을 출력하는 코드가 있을 시, 이를 제거하면 컴파일에 문제가 생기지만, 본 제안 방식은 유사도 측정이 우선이기 때문에, 이를 고려하지 않아도 별도의 문제점이 되지 않는다.

b) 소스 코드 소문자화

두 번째로, 비교하는 소스 코드의 구문을 소문자로 변경한다. 소스 코드의 대소문자를 통일하면 의미에 영향을 주지 않고, 유사도 측정을 더욱 일관되고 정확하게 수행한다.

```
def remove_quoted_strings(code):
    # Remove the quoted output string
    without_str = re.sub(r'["\'](.*?)["\']', '', code)
    return without_str
```

Fig. 9. Output String Cancellation Code

본 제안 방식에서는 파이썬에는 lower() 내장 함수를 사용하여 소스 코드 소문자화를 진행한다.

c) 소스 코드 불용어 제거

세 번째로, 소스 코드 불용어(Stopword)를 제거한다. 불용어는 텍스트 분석이나 정보 검색에서 일반적이고 빈번하게 등장하는 단어들을 의미한다. 소스 코드에서의 불용어는 프로그래밍에서 필수적으로 사용되지만, 소스 코드의 구문 의미를 파악하는 데에는 크게 기여하지 않는다. 따라서 본 제안 방식에서는 Table 4와 같은 불용어들을 제거하여 유사도를 향상시키는 과정을 수행한다.

d) 소스 코드 구두점 제거 및 토큰화

구두점(Punctuation)은 소스 코드를 이루는 부호나 기호를 나타내는 용어이다. 소스 코드 유사도 측정의 전처리 단계에서 구두점 제거는 가장 필수적인 과정이다. 이는 코드의 구성을 해치지 않는 무의미한 기호들을 제거함으로써, 보다 정확한 유사도 측정을 위해서 반드시 필요하다.

본 제안 방식에서는 Table 5의 소스 코드에서 빈번히 사용되는 구두점들을 제거하는 과정을 수행한다.

소스 코드의 높은 유사도 측정을 위한 구두점을 제거하는 전처리 작업을 수행하는 코드는 Fig. 10과 같다.

Table 4. Type of Stopwords

Stopwords	Features in source code
Comments	Text documenting a description of a code that does not affect the code
indentation	For readability of the structure of the code block, ignored for similarity measurements
Data types	Data types of variables and lists. If all float variables are double, the similarity measurement will decrease even though the function is similar

Table 5. Type of Punctuations

Punctuations	Features in source code
Comma(,)	Used primarily to list variables or arrangements on a single line
Period(.)	Mainly used to access member elements of a structure or class
Semicolon(;)	Used primarily to indicate the end of a source code line or to isolate a line
Colon(:)	Used primarily for conditional statements, repeat statements, function definitions, and the beginning of the division of code blocks
Quotes(‘ “)	Used primarily to define the strings and messages that are output to the console and GUI
Parenthesis ({} () [])	Used primarily to represent code blocks, function calls, array and list components

```
# Remove punctuation and divide and save
pattern = r'[.,;"(){}\[\]\'\"]'
replacement = ''
sourceWordList = re.sub(pattern, replacement).split()
```

Fig. 10. Punctuations Cancellation Code

위와 같은 4단계의 전처리 작업을 통해, 핵심 기능을 제외하고 소스 코드 구문에 영향을 끼치지 않는 요소들을 제거함으로써, 유사도의 정확성을 향상시킬 수 있다.

2) 소스 코드 유사도 측정 단계

본 절에서는 전처리 과정을 수행 후, 사용자가 요청한 소스 코드가 서버에 있는 어떤 오픈소스와 가장 유사한 지에 대해 분석하기 위해 유사도 측정 알고리즘을 적용한다.

비교하는 두 소스 코드의 전처리 과정 후, 유사도 측정은 2장에서 기술한 TF-IDF 벡터화 및 코사인 유사도 측정을 기반으로 수행된다.

a) TF-IDF 벡터화

전처리 작업 후. 키워드 단위로 토큰을 벡터화하는 과정이며, 두 소스 코드가 각각 서로 특정 키워드가 얼마나 자주 나타나는 지에 대한 TF 빈도 값을 계산한다.

다음으로, 키워드의 중요도를 나타내는 IDF값을 계산한 후 TF와 IDF값을 곱하여 문서를 벡터로 표현한다.

$TF-IDF$ 벡터는 각 키워드의 중요도를 반영하여 두 소스 코드의 벡터 간 유사도 측정을 위해 생성한다.

$TF-IDF$ 벡터화 과정은 Fig. 11과 같이 수행된다.

(b) 코사인 유사도 측정

Fig. 12와 같이 두 소스 코드 벡터 방향에 대한 코사인 유사도를 측정한다. 이때, 두 벡터가 유사할수록 1에 가까워지고 서로 다를수록 0에 가까운 실수 값이 나오게 된다.

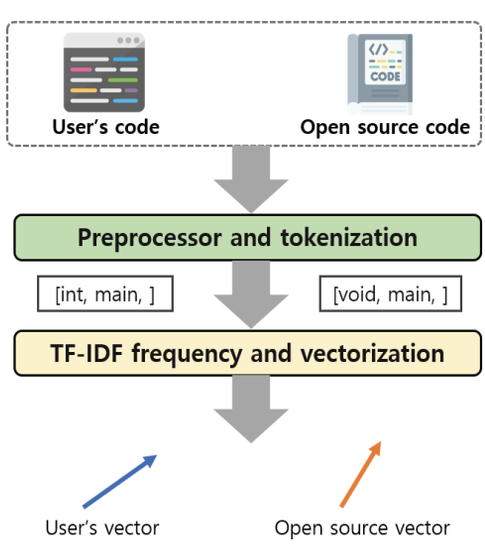

Fig. 11. Source Code Vectorization

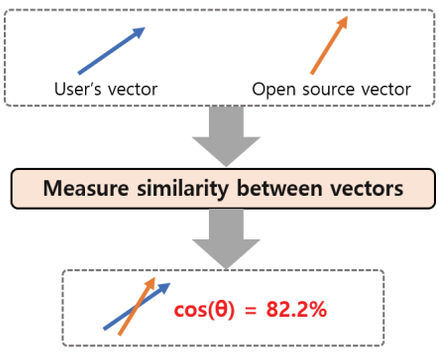

Fig. 12. Source Code Similarity Measurement

유사도 측정은 Equation (7)과 같이 토큰화된 소스 코드 벡터를 각 차원별로 곱하여 A와 B의 내적을 계산한다.

다음으로, 두 벡터의 크기 $\|A\|$와 $\|B\|$를 Equation (8)과 (9)를 통해 구한 후 곱셈을 수행한다. 최종적으로는 벡터의 내적 $A \cdot B$를 $\|A\|$와 $\|B\|$의 곱으로 나눔으로써, 코사인 유사도를 계산한다.

$$A \cdot B = A_1 \times B_1 + \ldots + A_n \times B_n \tag{7}$$

$$\|A\| = \sqrt{A_1^2 + A_1^2 + \ldots + A_n^2} \tag{8}$$

$$\|B\| = \sqrt{B_1^2 + B_1^2 + \ldots + B_n^2} \tag{9}$$

3) 소스 코드 라이선스 탐지

앞선 절 2)에서 사용자가 업로드한 소스 코드와 가장 높은 유사도를 갖는 오픈소스를 분석할 수 있다. 이를 통해, 소프트웨어 개발하고자 사용하려는 소스 코드가 어떤 오픈소스가 적용되어 있는지를 파악할 수 있다.

다음 단계로, 분석한 오픈소스가 어떤 라이선스가 적용되어 있는지 정확하게 탐지해야 한다. 라이선스는 Fig. 13과

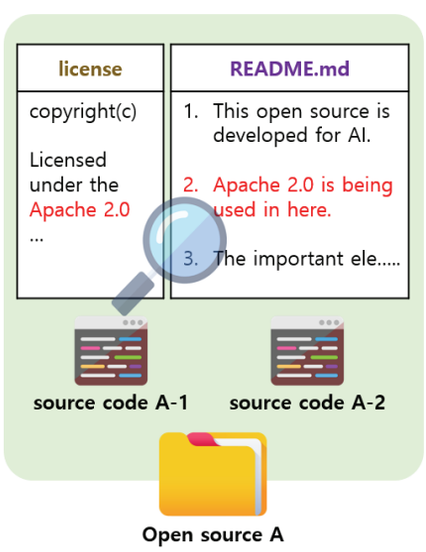

Fig. 13. Open Source License Detection

같이 대부분 라이선스 파일, README 마크다운 파일에 존재하며, 오픈소스의 소스 코드 주석 상단에 포함되는 경우도 있다. 본 제안 방식에서는 이러한 라이선스를 탐지하여 해당 소스 코드의 규정 사항을 제공할 수 있어야 한다.

a) 라이선스 파일 탐색

라이선스 탐지를 위해 가장 유사도가 높은 오픈소스의 루트 디렉토리에서 라이선스 관련 파일 탐색을 수행한다. 가장 먼저, 라이선스(License 또는 License.txt) 파일 여부를 확인한다. 해당 파일의 탐색이 성공하면 파일에서 라이선스 정보를 추출하는 작업을 수행한다.

먼저, 파일을 텍스트 모드로 열어 행 단위로 읽으며 라이선스 탐색을 시작한다. 본 제안 방식에서 추출할 라이선스 문자열로는 GPL, AGPL, MPL, EPL, LGPL, BSD, Apache 그리고 MIT로 설정한다.

다음으로, 행 단위로 토큰화하여 라이선스 문자열을 식별한 후 파싱하는 작업을 수행한다. 토큰이 상기에 설정한 라이선스 문자열이고, 다음 토큰이 버전 정보(Ex. GPL 3.0)라면 해당 라이선스 문자열을 현재의 오픈소스 라이선스로 해석한다. 만약 다음 토큰이 License 문자열이라면, 그 다음 토큰을 다시 비교하고 이때 버전 정보가 나오면(Ex. AGPL License 2.1) 해당 라이선스 문자열과 마지막의 버전 정보를 사용하여 오픈소스에 적용된 라이선스로 해석한다.

파싱된 결과는 (라이선스 문자열, 버전)의 구조를 가지며, 이를 JSON형식을 활용하여 키-값 쌍으로 구성된 구조를 사용한다. 본 제안 방식에는 파이썬의 딕셔너리를 활용하여 라이선스 정보를 파싱 후 추출하여 탐지한다.

b) README 마크다운 파일 탐색

하지만 오픈소스 개발자가 등록할 때 라이선스를 선택하지 않고 README 파일에 라이선스를 입력하는 경우도 많다. 이때, 오픈소스 설명 문서인 README 마크다운 파일을 앞선 방법과 동일하게 라이선스 탐지를 수행한다.

c) 소스 코드 주석 탐색

만약 README에서도 라이선스에 대한 내용을 탐지하지 못했을 경우, 전처리 작업을 이전의 오픈소스를 구성하는 소스 코드의 주석을 탐색한다. 행 단위로 탐색 하면서 같은 방법으로 라이선스를 추출하는 작업을 수행한다.

이와 같이, License 파일, README 파일 및 소스 코드 주석을 탐색을 수행할 수 있지만, 등록자가 관련 파일을 변경 및 삭제할 수도 있다. 이때, COPYING 파일을 비롯한 루트 디렉토리의 읽기 가능한 파일을 모두 탐색하여 라이선스를 분석한다. 상기 과정에도 정보를 획득하지 못했다면 서브 디렉토리의 파일을 확인하고, 최종적으로 라이선스를 탐지하지 못했을 경우, 라이선스가 적용되지 않은 유연하게 사용 가능한 소스 코드인 결과물을 사용자에게 제공한다.

4) 오픈소스 버전 및 라이선스 탐지 확인

앞선 과정을 통해, Fig. 14와 같이 서버는 사용자가 업로드한 소스 코드와 가장 유사한 오픈소스와의 유사도 퍼센테이지의 결과를 제공한다. 그리고 데이터베이스에서 Table 6과 같이 오픈소스의 버전을 포함한 배포 날짜, 등록자의 정보와 같은 메타데이터또한 제공할 수 있다. 또한, Fig. 15와 분석한 오픈소스에 적용되는 라이선스가 어떤 것인지에 대하여 파악할 수 있다.

위 과정을 통해, 오픈소스 버전 및 라이선스와 다른 구성요소들을 탐지할 수 있다. 분석한 오픈소스의 버전 정보, Apache 2.0 라이선스 규정 사항 및 세부 메타데이터를 Table 6과 Fig. 14, 15에서 확인할 수 있다.

또한, 해당 오픈소스의 버전 별 유의 및 권고 사항에 대해 확인할 수 있다. Fig. 16은 앞서 분석한 1.2 버전의 오픈소스에 대한 버전 권고 사항을 확인한다. 분석한 오픈소스는 1.0, 1.1 그리고 1.2 버전으로 플랫폼에 등록되어 있음을 확인할 수 있고, 각 버전의 권고 사항은 각 오픈소스 별 URL에서 확인할 수 있다. Fig. 16은 기존에 등록되었던 1.0 버전과 1.1 버전의 코드 취약점이 발견된 점을 README에서 확인할 수 있다. 이를 통해, 버전 보안 취약점을 사전에 파악함으로써, 공격자로부터의 공격을 사전 대응할 수 있다.

Table 6. Detected Open Source Metadatas

Metadatas	Element
Open source	Perceptron program
Version	1.2
Released date	September 18, 2023
Developer	Cse1234
Uesd language	Python
Recommendations	1.0 and 1.1 vulnerabilities found

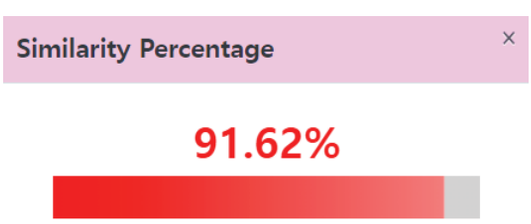

Fig. 14. Open Source Similarity Percentage

Fig. 15. Detected Open Source License

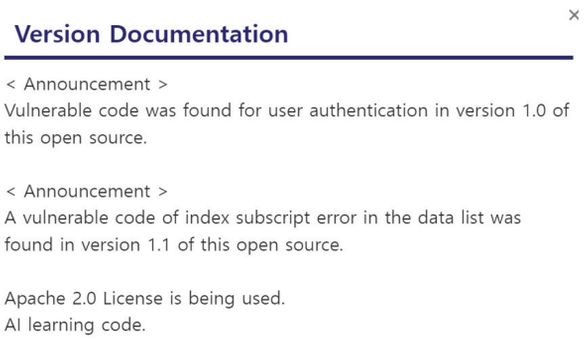

Fig. 16. Vulnerable Versions of Open Source

5) 오픈소스 라이선스 충돌 확인

앞선 과정에서는 한 개의 소스 코드 적용된 오픈소스 정보를 탐지했지만, 실제로는 소프트웨어 개발에 여러 오픈소스를 혼용하는 경우가 많다. 이를 탐지하는 형태로는 여러 개의 소스 코드를 한 번에 분석을 요청하는 경우이다. 본 제안 방식에서는 압축한 여러 소스 코드를 ZIP 파일로 구성하여 분석 요청 후 여러 오픈소스 탐지가 가능하다.

하지만 이때 규정이 호환되지 않는 라이선스 충돌이 발생할 확률이 높다. 따라서 라이선스 충돌 여부를 분석하는 기능을 구현한다.

a) 라이선스 규정 사항 테이블 구축

앞서 언급한 8 종류의 대표적인 라이선스 종류와 라이선스 사항을 활용하여 SQLite 데이터베이스를 구성한다.

테이블의 열은 라이선스 종류들로 구성하며, 테이블의 행은 라이선스 사항으로 구성한다. 라이선스 사항으로는 Fig. 17과 같이 무료 이용가능, 배포 허용가능, 소스 코드 취득 가능, 소스 코드 수정 가능, 2차적 저작물 공개 의무 그리고 독점 소프트웨어 결합 가능 여부로 구성한다.

상기 테이블을 활용하면 두 개 이상의 오픈소스에 적용된 라이선스들의 충돌 여부를 탐지할 수 있다. 각 오픈소스 별 라이선스 탐지 후 상기 테이블로 분석한 라이선스 사항들을 비교함으로써, 규정이 충돌 여부를 탐지한다.

	GPL	LGPL	BSD
Free use	Y	Y	Y
Allow Deployment	Y	Y	Y
Code acquisition	Y	Y	Y
Code Modifying	Y	Y	Y
Code Disclosure	**Y**	Y	**X**
SW Combination	**X**	Y	**Y**

Fig. 17. Process of License Conflict Analysis

Fig. 18. Result of License Conflict

동일한 라이선스가 적용된 두 개 이상의 소스 코드 압축 파일을 업로드하면 하나만 결과만 출력된다. 하지만 두 개 이상의 라이선스가 적용된 오픈소스로 분석되면, 라이선스 테이블에서 비교한 후, 호환되지 않을 시 충돌임을 알린다.

Fig. 18은 사용자가 두 개의 소스 코드 압축 파일을 업로드한 후, 분석된 오픈소스가 각각 GPL과 BSD 라이선스가 적용된 경우이다. 이때, Fig. 18의 충돌 검사 버튼에서 라이선스 충돌 여부를 확인할 수 있다. 버튼 클릭 시, 라이선스 사항 테이블에서 각 열 마다 적용되는 사항을 비교하게 되고, GPL과 BSD가 상기 테이블의 가장 하단 두 개의 사항이 서로 상호 호환되지 않음을 탐지하게 된다. 이를 통해, 두 라이선스가 충돌되는 결과를 제공한다.

4. 제안 방식 분석

본 제안 방식은 소스 코드 유사도 측정을 활용하여 어떤 오픈소스인지 분석하고, 각 버전의 취약점과 라이선스를 파악할 수 있다. 본 장에서는 제안 방식과 기존 도구들과의 장단점에 대해 비교 분석한다.

Table 7은 MOSS와 본 제안 방식의 소스 코드 유사도 측정 결과 분석표이며, Table 8은 블랙덕, Fossology와 본 제안 방식의 장단점에 대해 기술한 분석표를 나타낸다.

4.1 오픈소스 유사도 측정 비교

MOSS는 소스 코드 유사도 측정을 수행해주는 프로그램이지만, 정확도 면에서 문제점이 존재한다. 특히, Fig. 19와 같이 변수 및 함수명 변경 및 선언 위치를 변경하면 유사도가 대폭 하락하게 된다. 심지어는 동일한 소스 코드 간 유사도를 비교에도 100%가 나오지 않는 문제도 존재한다.

Table 7. Comparison Proposed Method with MOSS

	MOSS	Proposed method
Analytical method	Fingerprinting	TF-IDF and cosine similarity
Same code similarity	97.8%	100%
Variables renamed code similarity	93.5%	97.4%
Function renamed code similarity	93.0%	98.8%
Change function declaration location	87.4%	100%

Table 8. Comparison Proposed Method with Existing Tools

	Blackduck Software	Fossology	Proposed method
Analytical method	Code printing and binary analysis, etc	String-based and file hash analysis, etc	TF-IDF and cosine similarity
Storage Capacity Required	80GB	4GB	3GB
Version	Detectable	Undetectable	Detectable
License	Detectable	Detectable	Detectable

Before Change

```
int main(void)
{
   int a=10, b=20;
   printf("%d", getSum(a,b));
}

int getSum(int x, int y)
{
   return x + y;
}

int getAvg(int x, int y)
{
   return (x + y) / 2;
}
```

After change

```
int main(void)
{
   int m=10, n=20;
   printf("%d", calSum(m,n));
}

int calAvg(int x, int y)
{
   return (x + y) / 2;
}

int calSum(int x, int y)
{
   return x + y;
}
```

Fig. 19. Example of Source Code Change

이를 해결하기 위해, 본 제안방식에서는 4단계의 전처리 작업을 수행하고 TF-IDF 기반의 코사인 유사도를 적용했다. Table 7은 MOSS와 본 제안 방식 간 동일한 소스 코드, 변수명 변경, 함수명 변경을 수행한 경우의 유사도 비교 결과를 나타낸다. 실험군은 CodeUp 사이트 정보올림피아드 영역의 80-150라인 C언어 소스 코드 20개를 사용했다. 유사도 측정 결과 동일한 소스 코드 비교 시, 본 제안 방식은 100% 정확도를 도출하며, 변수 및 함수명의 변경에도 유사도가 근소 감소하지만, MOSS와 비교하여 더욱 높은 정확도의 결과를 도출할 수 있었다.

4.2 오픈소스 점검 도구 비교

본 제안 방식에서는 오픈소스 저장 공간 및 데이터베이스의 실험군으로는 CodeUp의 80-150 라인의 정보올림피아드 C언어, python 소스 코드 각 20개와, BeakJoon 알고리즘 카테고리의 C++, Java 소스 코드 각 30개를 사용했다.

기존 오픈소스 점점 도구인 블랙덕은 사용자가 업로드 한 소스 코드의 오픈소스 정보, 라이선스 위반 및 충돌 여부, 버전 취약점과 같은 정보를 확인할 수 있다. 또한, 블랙덕은 하지만 블랙덕은 사용을 위해, 큰 저장 공간이 요구된다. 로컬 PC 환경에 구축하기 위한 저장 공간은 약 80GB가 요구되며, 이는 용량 측면에서 많은 부담을 제공하게 된다.

Fossology는 사용자가 업로드한 오픈소스의 라이선스를 스캔할 수 있지만, 그 외 정보를 파악할 수 없다는 문제가 존재한다. 솔루션 사용을 위해서는 커널 구축 및 환경 설정에 4GB의 저장 공간이 요구되며, 이는 블랙덕에 비해 상대적으로 적은 용량을 요구한다. 하지만, 상기와 같이 라이선스를 제외한 구성요소를 확인하는 것은 불가능하다.

본 제안 방식은 Vscode, Django 및 SQLite 등 웹 기반의 경량 플랫폼 환경에서 소스 코드 유사도 측정 기반의 오픈소스 버전 및 라이선스를 탐지하는 도구이다. 구축에는 약 3GB의 저장 공간을 요구되고 Fossology와 유사한 크기이며, 블랙덕에 비해 상대적으로 용량을 크게 절감할 수 있었다. 또한, 적은 저장 공간의 경량 플랫폼임에도 사용자가 업로드한 소스 코드에 대한 오픈소스의 라이선스 및 버전 취약점 정보들을 제공할 수 있다.

5. 결 론

오픈소스의 무분별한 사용으로 인한 문제는 끊임없이 발생하며, 사용자에게 다양한 피해를 가져온다. 특히, 취약한 버전의 오픈소스 사용으로 인한 취약점 노출과 라이선스 위반 및 충돌로 인한 저작권 분쟁이 끊임없이 발생하고 있다.

또한, 국내에는 오픈소스 점검 도구들이 부족한 실정이며 해외 도구들에 많이 의존하고 있다. 하지만 이러한 도구들 또한 사용하는 데 큰 저장 공간을 요구하거나, 주요 기능을 제공하지 않는 부분에 있어서 여전히 문제점이 존재한다.

본 논문에서는 이러한 문제들을 극복하기 위한 오픈소스 버전 및 라이선스 탐지 도구를 제안한다. 경량화된 플랫폼 환경에서 사용자는 소스 코드 업로드를 함으로써, 어떤 오픈소스를 사용하고 있는지를 인식할 수 있다. 또한, 인식한오픈소스가 최신 버전 또는 오래된 버전인지에 파악하고 각 취약점에 대한 유의 사항들을 확인할 수 있다. 이를 통해, 취약한 구버전 오픈소스 사용으로 발생했던 다양한 보안 위협으로부터 효과적으로 대응할 수 있다.

또한, 오픈소스 라이선스 및 충돌 여부를 탐지함으로써, 준수해야 할 오픈소스 저작권 규정 사항에 대해 명확하게 인식할 수 있다. 이를 통해, 기존에 라이선스 미 준수로 발생했던 저작권 법적 분쟁의 문제들을 해결할 수 있다.

향후 본 제안 방식에 오픈소스의 세부적인 구성 명세를 나타내는 SBoM을 도입하여 소프트웨어 공급망 보안을 강화하고, 여러 프로그래밍 언어를 분석할 수 있도록 분석 가능 언어를 확장하는 연구가 필요할 것으로 사료된다.

References

[1] S.-R. Son and Y. Kang, “Business models proposal for Korean open source GIS software companies,” *Journal of Cadastre & Land InformatiX*, Vol.48, No.1, pp.187-199, https://koreascience.kr/article/JAKO201821142175225.page, Jun, 2018.

[2] R. Gurikar and G. S. Gururaj, “Use of open source software in indian institutional digital repositories: A study,” *Library Philosophy and Practice* (e-journal), 2021.

[3] K.-H. Lee and J.-P. Park, “A software vulnerability analysis system using learning for source code weakness history,” *Korea Academia-Industrial Cooperation Society*, Vol.18, No.11, pp.46-52, 2017.

[4] Y. Agarwal, “Apache Log4j logging framework and its vulnerability,” in Metropolia University of Applied Sciences Master of Engineering Information Technology Master’s Thesis, Mar. 2022.

[5] D.-G. Lee and Y.-S. Seo, “A study on the identification of open source license compatibility violations” *KIPS Transactions on Software and Data Engineering*, Vol.7, No.12, pp.451-460, 2018.

[6] H. Schoettle, “Open source license compliance-why and how?” *Computer*, Vol.52, No.8, pp.63-67, 2019. DOI: https://doi.org/10.1109/MC.2019.291569

[7] D. Bellamkonda “Software engineering tools for secure application development,” in Cluminating Projects in Information Assurance, May. 2023.

[8] S.-W. Kim and K.-H. Son, “SBOM trends for OSS traceability,” *The Korea Institute of Information Security and Cryptology*, Vol.32, No.5, pp.53-66, 2022.

[9] T. Tuunanen, J. Koskinen, and T. Kärkkäinen, “Automated software license analysis,” *Automated Software Engineering*, Vol.16, pp.455-490, 2009.

[10] C. Ragkhitwetsagul, J. Krinke, and D. Clark “Similarity of source code in the presence of pervasive modifications,” in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2016.

[11] C.-H. Huang, J. Yin, and F. Hou, “A text similarity measurement combining word semantic information with TF-IDF method,” *Chinese Journal of Computers*, No.5, pp.856-864, 2011.

[12] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," in *The 7th International Student Conference on Advanced Science and Technology ICAST 2012*, At: Seoul, South Korea, Oct. 2012.
[13] Z. Pauzi and A. Capiluppi, "Text similarity between concepts extracted from source code and documentation," in Intelligent Data Engineering and Automated Learning – IDEAL 2020, pp.124-135, Oct. 2020.
[14] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 76-85, Jun. 2003.
[15] K.-H. Kim, S. C. Yoon, S.-H. Kim, and I.-Y. Lee, "A study on platform for OSS similarity and license analysis," in *Proceedings of the Annual Conference of Korea Information Processing Society Conference (KIPS) 2023*, Vol.30, No.2, pp.317-318, Nov. 2023.

김 기 환
https://orcid.org/0009-0007-3496-9142
e-mail : 20247089@sch.ac.kr
2024년 순천향대학교
컴퓨터소프트웨어공학과(학사)
2024년 ~ 현 재 순천향대학교
소프트웨어융합학과 석사과정
관심분야 : 정보보호, 암호학

윤 성 철
https://orcid.org/0009-0007-2365-3804
e-mail : ysc1346@sch.ac.kr
2023년 순천향대학교
컴퓨터소프트웨어공학과(학사)
2023년 ~ 현 재 순천향대학교
소프트웨어융합학과 석사과정
관심분야 : 정보보호, 암호학

김 수 현
https://orcid.org/0000-0002-6224-3273
e-mail : kimsh@sch.ac.kr
2010년 순천향대학교 정보기술공학부(학사)
2012년 순천향대학교 컴퓨터학과(석사)
2016년 순천향대학교 컴퓨터학과(박사)
2023년 ~ 현 재 순천향대학교
컴퓨터소프트웨어공학과 교수
관심분야 : 암호 프로토콜, 클라우드 컴퓨팅 보안, IoT 보안

이 임 영
https://orcid.org/0000-0002-8856-0103
e-mail : imylee@sch.ac.kr
1981년 홍익대학교 전자공학과(학사)
1986년 오사카대학 통신공학전공(석사)
1989년 오사카대학 통신공학전공(박사)
1994년 ~ 현 재 순천향대학교
컴퓨터소프트웨어공학과 교수
관심분야 : 암호이론, 암호 프로토콜, 컴퓨터보안, 블록체인