2026-04-27

# 【서지사항】

【서류명】 특허출원서

【참조번호】 DPE260025

【출원구분】 특허출원

【출원인】

【명칭】 고려대학교 산학협력단

【특허고객번호】 2-2004-017068-0

【대리인】

【성명】 이대호

【대리인번호】 9-2009-003763-5

【대리인】

【성명】 박건홍

【대리인번호】 9-2012-000137-8

【발명의 국문명칭】 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법 및 장치

【발명의 영문명칭】 METHOD AND APPARATUS FOR DETECTING MISUSE OF APPLICATION PROGRAMMING INTERFACE IN PYTHON SOURCE CODE

【발명자】

【성명】 우승훈

【성명의 영문표기】 WOO, Seunghoon

【국적】 KR

【주민등록번호】

【우편번호】

【주소】

【거주국】

【공보의 주소공개】

【발명자】

【성명】 조서경

【성명의 영문표기】 CHO, Seogyeong

【국적】 KR

【주민등록번호】

【우편번호】

【주소】

【거주국】

【공보의 주소공개】

【출원언어】 국어

【심사청구】 청구

【공지예외적용대상증명서류의 내용】

【공개형태】 논문 공개

【공개일자】 2025.11.16

【공지예외적용대상증명서류의 내용】

【공개형태】 한국금융경제신문을 통해 보도

【공개일자】 2025.12.12

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 2710093880

【과제번호】 RS-2024-00440780

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 정보보호핵심원천기술개발

【연구과제명】 SW공급망 전주기 보안 내재화를 위한 SBOM과 VEX 연계 기반 취약점 통합 관리 플랫폼 기술

【과제수행기관명】 고려대학교산학협력단

【연구기간】 2024.07.01 ~ 2026.12.31

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 2710098455

【과제번호】 RS-2025-00517788

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 한국연구재단

【연구사업명】 (이공)우수신진연구

【연구과제명】 멀티레벨 코드 분석을 통한 지능형 SBOM 생성 및 취약점 분석 자동화 연구

【과제수행기관명】 고려대학교

【연구기간】 2025.03.01 ~ 2029.02.28

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 2370000334

【과제번호】 R2415812

2026-04-27

【부처명】 문화체육관광부

【과제관리(전문)기관명】 한국콘텐츠진흥원

【연구사업명】 지정공모

【연구과제명】 생성형 AI 저작권 관리 및 보호 기술 개발을 위한 국제공동 연구 및 글로벌 인재 양성

【과제수행기관명】 고려대학교

【연구기간】 2024.04.01 ~ 2027.12.31

【취지】 위와 같이 지식재산처장에게 제출합니다.

대리인 이대호 (서명 또는 인)

대리인 박건홍 (서명 또는 인)

【수수료】

【출원료】

【가산출원료】

【우선권주장료】

【심사청구료】

【합계】

【감면사유】

【감면후 수수료】

【첨부서류】

# 【발명의 설명】

## 【발명의 명칭】

파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법 및 장치{METHOD AND APPARATUS FOR DETECTING MISUSE OF APPLICATION PROGRAMMING INTERFACE IN PYTHON SOURCE CODE}

## 【기술분야】

【0001】 본 개시는 소프트웨어 보안 및 프로그램 정적 분석 기술에 관한 것으로, 보다 구체적으로, 파이썬(Python) 소스코드에서 응용 프로그램 인터페이스(Application Programming Interface, API) 호출과 관련된 함수 내부 및 함수 간 정적 의존성을 분석하여 구조화된 분석 컨텍스트를 생성하고, 이를 기반으로 API의 오용을 탐지하기 위한 방법 및 장치에 관한 것이다.

## 【발명의 배경이 되는 기술】

【0002】 현대 소프트웨어는 데이터의 기밀성, 무결성 및 인증을 보장하기 위해 다양한 보안 관련 응용 프로그램 인터페이스를 활용한다. 특히 암호화 API는 키 생성, 난수 생성, 암호 알고리즘 선택 및 운용 모드 설정 등 다수의 보안 구성 요소를 포함하며, 설정 방식이 복잡하여 개발자가 이를 부적절하게 사용할 경우 심각한 보안 취약점이 발생할 수 있다. 파이썬 생태계는 서버 애플리케이션 및 인공지능 시스템에 광범위하게 채택되어 있으며, 이에 따라 파이썬 코드에서의 암호화 API 오용 문제는 실질적인 보안 위협으로 이어지고 있다.

【0003】 이러한 API 오용은 단일 API 호출 지점만으로는 드러나지 않는 경우가 많다. 보안에 중요한 파라미터 값의 기원, 함수 간 전달 과정, 객체의 생성 및 설정 맥락을 종합적으로 고려해야 정확한 판단이 가능하다. 즉, API 오용 판별에 필요한 핵심 문맥 정보가 복수의 함수에 걸쳐 분산되는 컨텍스트 분산(context fragmentation) 문제가 빈번하게 발생하며, 이러한 특성은 파이썬의 동적 타입 결정, 런타임 객체 바인딩 등 언어 고유의 동적 특성에 의해 더욱 심화된다.

【0004】 종래의 API 오용 탐지 기술 중 규칙 기반 분석 방식은 호출 지점 중심의 규칙 적용과 제한적인 데이터 흐름 분석에 의존하여, 암호화 설정 정보가 여러 함수에 걸쳐 전달되는 경우와 같은 깊은 함수 간 문맥을 충분히 포착하지 못하는 한계를 가진다. 예를 들어, 역방향 데이터 흐름 분석을 일부 수행하더라도 함수 경계를 넘어서는 깊은 함수 간 의존성 관계를 복원하지 못하여, 컨텍스트가 분산된 오용 사례를 탐지하지 못하는 경우가 발생한다.

【0005】 정적 타인트 분석(static taint analysis) 기반의 종래 기술은 데이터의 전파 경로를 정밀하게 추적하는 데에는 효과적이나, 동적 객체 생성, 실행 시점에 결정되는 설정 값, 또는 API 사용 맥락에 따라 달라지는 조건부 오용과 같은 복잡한 문맥을 종합적으로 반영하는 데에는 한계를 가진다. 또한, 사전에 정의된 타인트 명세(taint specification)의 범위 내에서만 오용을 탐지할 수 있어, 명세에 포함되지 않은 오용 유형이나 파이썬 고유의 동적 특성에서 비롯되는 맥락 의존적 오용을 효과적으로 처리하지 못하는 한계가 있다.

【0006】 종래의 API 오용 탐지 기술들은 호출 지점 중심 분석 또는 특정 데

이터 흐름 추적에 머무르는 경우가 많아, 오용 판단에 필요한 핵심 문맥 정보가 함수 및 모듈 전반에 분산된 상황을 충분히 복원하지 못한다. 그 결과, 오탐(false positive) 및 미탐(false negative)을 효과적으로 줄이기 어렵거나, 분석 범위를 과도하게 확장하여 실용성이 저하되는 문제가 존재한다.

【0007】 따라서, 파이썬 소스코드에서 함수 내부 및 함수 간의 데이터 흐름과 호출 관계를 체계적으로 분석하여 오용 판별에 필요한 문맥 정보를 정적으로 추출하고 구조화할 수 있는, 통합된 방법 및 장치가 요구된다.

**【선행기술문헌】**

**【특허문헌】**

【0008】 **(특허문헌 0001)** 대한민국 등록특허 제10-2891372호(2025.11.21. 등록)

**【발명의 내용】**

**【해결하고자 하는 과제】**

【0009】 본 개시는 전술한 배경기술에 대응하여 안출된 것으로, 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법 및 장치를 제시하고자 한다.

【0010】 본 개시의 기술적 과제들은 이상에서 언급한 기술적 과제로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 당업자에게

명확하게 이해될 수 있을 것이다.

**【과제의 해결 수단】**

【0011】 전술한 바와 같은 과제를 해결하기 위한 본 개시의 일 실시예에 따라, 컴퓨팅 장치에 의해 수행되는, 파이썬(Python) 소스코드에서 응용 프로그램 인터페이스(Application Programming Interface, API)의 오용을 탐지하기 위한 방법으로서, 입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별하는 단계; 코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 상기 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석하는 단계; 구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 상기 API 호출을 호출하거나 상기 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석하는 단계; 및 상기 함수 내부의 정적 의존성을 분석한 결과 및 상기 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성하는 단계;를 포함할 수 있다.

【0012】 일 실시예에서, 상기 API 호출을 식별하는 단계는, 상기 파이썬 소스코드로부터 메서드 호출(method invocation)을 검출하는 단계; 및 검출된 상기 메서드 호출 중, 사전 정의된 API 목록에 포함된 클래스 또는 함수에 대응되는 호출을 식별함으로써 상기 API 호출을 선별하는 단계;를 포함할 수 있다.

【0013】 일 실시예에서, 상기 함수 내부의 정적 의존성을 분석하는 단계는, 상기 코드 속성 그래프 상에서 데이터 흐름 엣지를 역방향으로 추적하여, 상기 API 호출에 사용되는 객체의 생성 맥락을 파악하기 위한 생성자 호출 및 상기 생성자에

전달된 파라미터를 추적하는 단계; 상기 코드 속성 그래프 상에서 순방향 데이터 흐름을 따라, 상기 API 호출의 반환값이 반환 구문에 사용되거나, 변수에 저장되거나, 또는 다른 함수의 인자로 전달되는 사용 경로를 추적하는 단계; 및 함수 간 정적 의존성 분석의 대상이 되는 함수들을 결정하기 위하여, 상기 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자(caller-callee) 관계를 식별하는 단계;를 포함할 수 있다.

【0014】 일 실시예에서, 상기 함수 간 정적 의존성을 분석하는 단계는, 상기 구문 트리를 역방향으로 추적하여, 상기 API 호출에 사용되는 파라미터의 생성 또는 유입 경로를 추적하는 단계;를 포함할 수 있다.

【0015】 일 실시예에서, 상기 함수 간 정적 의존성을 분석하는 단계는, 상기 복수의 함수들 각각에 대한 구문 트리를 따라 노드를 탐색하여, 상기 API 호출의 인자로 사용되는 리터럴(literal) 값 또는 상수로 정의된 값을 식별하는 단계; 및 식별된 상기 값이 상기 API 호출의 동작 또는 보안 관련 파라미터에 미치는 영향을 분석하는 단계;를 포함할 수 있다.

【0016】 일 실시예에서, 상기 함수 간 정적 의존성을 분석하는 단계는, 상기 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자 관계에 기초하여 함수 호출 그래프를 구성하는 단계; 및 상기 함수 호출 그래프 상에서 상기 API 호출을 종점으로 하는 호출 경로를 따라 상위 함수 방향으로 추적하여 하나 이상의 함수 호출 경로를 식별함으로써, 특정 실행 흐름 하에서 발생되는 API 오용 가능성을 분석하는 단계;를 포함할 수 있다.

【0017】일 실시예에서, 상기 구조화된 분석 컨텍스트를 생성하는 단계는, 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하기 위하여, 상기 함수 내부의 정적 의존성을 분석한 결과로부터 추출된 객체의 생성 맥락 정보, 반환값 사용 정보, 및 호출자-피호출자 관계와, 상기 함수 간 정적 의존성을 분석한 결과로부터 추출된 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 상호 관계에 기초하여 통합함으로써, 상기 구조화된 분석 컨텍스트를 구성하는 단계;를 포함할 수 있다.

【0018】일 실시예에서, 상기 방법은: 상기 구조화된 분석 컨텍스트를 생성하는 단계 이후, 분석 대상 코드, API 오용에 관한 보안 규칙, 및 복수의 함수들에 걸쳐 분산된 API 사용 맥락이 복원된 상기 구조화된 분석 컨텍스트를 포함하는 프롬프트를 구성하는 단계; 및 상기 프롬프트를 사전 학습된 인공지능 모델에 입력하여 API 오용 여부를 판별하는 단계;를 더 포함할 수 있다.

【0019】일 실시예에서, 상기 프롬프트는, 상기 사전 학습된 인공지능 모델이 상기 보안 규칙, 상기 분석 대상 코드, 및 상기 구조화된 분석 컨텍스트를 순차적으로 검토하여 API 오용 여부를 단계적으로 추론하도록 유도하는 연쇄적 사고(Chain-of-Thought, CoT) 방식에 따라 구성될 수 있다.

【0020】전술한 바와 같은 과제를 해결하기 위한 본 개시의 다른 일 실시예에 따라, 프로세서를 포함하는 컴퓨팅 장치가 개시된다. 상기 프로세서는: 입력된 파이썬(Python) 소스코드에서, 사전 정의된 응용 프로그램 인터페이스(Application Programming Interface, API) 목록에 기초하여 API 호출을 식별하는 동작; 코드 속

성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 상기 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석하는 동작; 구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 상기 API 호출을 호출하거나 상기 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석하는 동작; 및 상기 함수 내부의 정적 의존성을 분석한 결과 및 상기 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성하는 동작;을 수행할 수 있다.

**【발명의 효과】**

【0021】 본 개시는 파이썬 소스코드에서 API 오용을 탐지하기 위한 방법 및 장치를 제공할 수 있다. 본 개시는 코드 속성 그래프(Code Property Graph, CPG)에 기초한 함수 내부의 정적 의존성 분석과 구문 트리(Abstract Syntax Tree, AST)에 기초한 함수 간 정적 의존성 분석을 결합함으로써, 단일 API 호출 지점 분석만으로는 포착하기 어려운 컨텍스트 분산 문제를 해소하고, 오용 판별에 필요한 문맥 정보를 정확하게 복원할 수 있다.

【0022】 본 개시는 복수의 함수에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하여 구조화된 분석 컨텍스트를 생성할 수 있다. 본 개시는 객체의 생성 맥락 정보, 반환값 사용 정보, 호출자-피호출자 관계, 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 상호 관계에 기초하여 통합함으로써, API 오용 판별 시스템이 활용 가능한 풍부한 의존성 정보를 제공할 수 있다.

【0023】 본 개시는 추출된 구조화된 분석 컨텍스트를 인공지능 모델(예를 들

어, 대규모 언어 모델(Large Language Model, LLM))과 결합하여 API 오용 여부를 의미론적으로 판별할 수 있다. 본 개시는 분석 대상 코드, 보안 규칙, 및 구조화된 분석 컨텍스트를 포함하는 연쇄적 사고(Chain-of-Thought, CoT) 방식의 프롬프트를 구성함으로써, 인공지능 모델이 문맥을 단계적으로 추론하여 구문적 모호성 및 의미론적 모호성을 모두 처리하도록 할 수 있다.

【0024】 본 개시는 API 인자의 기원, 전달 경로, 사용 맥락을 명시적으로 복원함으로써, 종래 기술에서 발생하는 오탐(False Positive) 및 미탐(False Negative)을 구조적으로 대폭 감소시킬 수 있다.

【0025】 본 개시에서 얻을 수 있는 효과는 이상에서 언급한 효과로 제한되지 않으며, 언급하지 않은 또 다른 효과들은 아래의 기재로부터 본 개시가 속하는 기술분야에서 통상의 지식을 가진 자에게 명확하게 이해될 수 있을 것이다.

**【도면의 간단한 설명】**

【0026】 다양한 양상들이 이제 도면들을 참조로 기재되며, 여기서 유사한 참조 번호들은 총괄적으로 유사한 구성요소들을 지칭하는데 이용된다. 이하의 실시예에서, 설명 목적을 위해, 다수의 특정 세부사항들이 하나 이상의 양상들의 총체적 이해를 제공하기 위해 제시된다. 그러나, 그러한 양상(들)이 이러한 구체적인 세부사항들 없이 실시될 수 있음은 명백할 것이다.

도 1은 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 컴퓨팅 장치를 나타낸 도면이다.

도 2는 본 개시의 일 실시예에 따른 인공지능 모델의 예시적인 구조를 나타낸 도면이다.

도 3은 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법을 나타낸 순서도이다.

도 4는 본 개시의 일 실시예에 따른 응용 프로그램 인터페이스 오용 탐지 시스템의 전체 구성을 나타낸 도면이다.

도 5는 본 개시의 일 실시예에 따른 정적 의존성 분석 단계의 세부 구성을 나타낸 도면이다.

도 6은 본 개시의 일 실시예에 따른 예시적인 컴퓨팅 환경을 나타낸 도면이다.

**【발명을 실시하기 위한 구체적인 내용】**

【0027】 다양한 실시예들이 이제 도면을 참조하여 설명된다. 본 명세서에서, 다양한 설명들이 본 개시의 이해를 제공하기 위해서 제시된다. 그러나, 이러한 실시예들은 이러한 구체적인 설명 없이도 실행될 수 있음이 명백하다.

【0028】 본 명세서에서 사용되는 용어 "컴포넌트", "모듈", "시스템" 등은 컴퓨터-관련 엔티티, 하드웨어, 펌웨어, 소프트웨어, 소프트웨어 및 하드웨어의 조합, 또는 소프트웨어의 실행을 지칭한다. 예를 들어, 컴포넌트는 프로세서상에서 실행되는 처리과정(procedure), 프로세서, 객체, 실행 스레드, 프로그램, 및/또는 컴퓨터일 수 있지만, 이들로 제한되는 것은 아니다. 예를 들어, 컴퓨팅 장치에서

실행되는 애플리케이션 및 컴퓨팅 장치 모두 컴포넌트일 수 있다. 하나 이상의 컴포넌트는 프로세서 및/또는 실행 스레드 내에 상주할 수 있다. 일 컴포넌트는 하나의 컴퓨터 내에 로컬화 될 수 있다. 일 컴포넌트는 2개 이상의 컴퓨터들 사이에 분배될 수 있다. 또한, 이러한 컴포넌트들은 그 내부에 저장된 다양한 데이터 구조들을 갖는 다양한 컴퓨터 판독가능한 매체로부터 실행할 수 있다. 컴포넌트들은 예를 들어 하나 이상의 데이터 패킷들을 갖는 신호(예를 들면, 로컬 시스템, 분산 시스템에서 다른 컴포넌트와 상호작용하는 하나의 컴포넌트로부터의 데이터 및/또는 신호를 통해 다른 시스템과 인터넷과 같은 네트워크를 통해 전송되는 데이터)에 따라 로컬 및/또는 원격 처리들을 통해 통신할 수 있다.

【0029】 더불어, 용어 "또는"은 배타적 "또는"이 아니라 내포적 "또는"을 의미하는 것으로 의도된다. 즉, 달리 특정되지 않거나 문맥상 명확하지 않은 경우에, "X는 A 또는 B를 이용한다"는 자연적인 내포적 치환 중 하나를 의미하는 것으로 의도된다. 즉, X가 A를 이용하거나; X가 B를 이용하거나; 또는 X가 A 및 B 모두를 이용하는 경우, "X는 A 또는 B를 이용한다"가 이들 경우들 어느 것으로도 적용될 수 있다. 또한, 본 명세서에 사용된 "또는", "및/또는"이라는 용어는 열거된 관련 아이템들 중 하나 이상의 아이템의 가능한 모든 조합을 지칭하고 포함하는 것으로 이해되어야 한다.

【0030】 또한, "포함한다" 및/또는 "포함하는"이라는 용어는, 해당 특징 및/또는 구성요소가 존재함을 의미하는 것으로 이해되어야 한다. 다만, "포함한다" 및/또는 "포함하는"이라는 용어는, 하나 이상의 다른 특징, 구성요소 및/또는 이들의

그룹의 존재 또는 추가를 배제하지 않는 것으로 이해되어야 한다. 또한, 달리 특정되지 않거나 단수 형태를 지시하는 것으로 문맥상 명확하지 않은 경우에, 본 명세서와 청구범위에서 단수는 일반적으로 "하나 또는 그 이상"을 의미하는 것으로 해석되어야 한다.

【0031】 그리고, "A 또는 B 중 적어도 하나"이라는 용어는, "A만을 포함하는 경우", "B만을 포함하는 경우", "A와 B의 구성으로 조합된 경우"를 의미하는 것으로 해석되어야 한다.

【0032】 당업자들은 추가적으로 여기서 개시된 실시예들과 관련되어 설명된 다양한 예시적 논리적 블록들, 구성들, 모듈들, 회로들, 수단들, 로직들, 및 알고리즘 단계들이 전자 하드웨어, 컴퓨터 소프트웨어, 또는 양쪽 모두의 조합들로 구현될 수 있음을 인식해야 한다. 하드웨어 및 소프트웨어의 상호교환성을 명백하게 예시하기 위해, 다양한 예시적 컴포넌트들, 블록들, 구성들, 수단들, 로직들, 모듈들, 회로들, 및 단계들은 그들의 기능성 측면에서 일반적으로 위에서 설명되었다. 그러한 기능성이 하드웨어로 또는 소프트웨어로서 구현되는지 여부는 전반적인 시스템에 부과된 특정 어플리케이션(application) 및 설계 제한들에 달려 있다. 숙련된 기술자들은 각각의 특정 어플리케이션들을 위해 다양한 방법들로 설명된 기능성을 구현할 수 있다. 다만, 그러한 구현의 결정들이 본 개시내용의 영역을 벗어나게 하는 것으로 해석되어서는 안 된다.

【0033】 제시된 실시예들에 대한 설명은 본 개시의 기술 분야에서 통상의 지식을 가진 자가 본 개시를 이용하거나 또는 실시할 수 있도록 제공된다. 이러한 실

시예들에 대한 다양한 변형들은 본 개시의 기술 분야에서 통상의 지식을 가진 자에게 명백할 것이다. 여기에 정의된 일반적인 원리들은 본 개시의 범위를 벗어남이 없이 다른 실시예들에 적용될 수 있다. 그리하여, 본 개시는 여기에 제시된 실시예들로 한정되는 것이 아니다. 본 개시는 여기에 제시된 원리들 및 신규한 특징들과 일관되는 최광의의 범위에서 해석되어야 할 것이다.

【0034】 본 개시내용에서의 제1, 제2, 또는 제3과 같이 "제N"으로 표현되는 용어들은 적어도 하나의 엔티티들을 구분하기 위해 사용된다. 예를 들어, 제1과 제2로 표현된 엔티티들은 서로 동일하거나 또는 상이할 수 있다.

【0035】 그리고, "A, B 등"과 같이 "~ 등"이라는 용어는, "A만을 포함하는 경우", "B만을 포함하는 경우", "A와 B의 구성으로 조합된 경우"를 의미하는 것으로 해석되어야 한다.

【0036】 도 1은 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 컴퓨팅 장치를 나타낸 도면이다.

【0037】 도 1을 참조하면, 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 컴퓨팅 장치(100)는 프로세서(110), 메모리(130), 네트워크부(150)를 포함할 수 있다. 상술한 컴퓨팅 장치(100)의 구성은 간략화 하여 나타낸 예시이며, 추가적인 구성요소가 컴퓨팅 장치(100)에 포함되거나 또는 컴퓨팅 장치(100)의 구성요소들 중 일부가 제외 또는 대체될 수도 있다.

【0038】 컴퓨팅 장치(100)는 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법을 수행할 수 있다.

【0039】본 개시에서, 컴퓨팅 장치(100)는 임의의 형태의 서버, 임의의 형태의 단말을 포괄하는 의미로 사용될 수 있다.

【0040】일 실시예에서, 서버는 예를 들어, 마이크로프로세서, 메인프레임 컴퓨터, 디지털 프로세서, 휴대용 디바이스 및 디바이스 제어기 등과 같은 임의의 타입의 컴퓨팅 시스템 또는 컴퓨팅 디바이스를 포함할 수 있다.

【0041】일 실시예에서, 서버는 본 개시에서 사용되는 데이터 및/또는 정보를 저장하기 위한 저장부를 포함할 수 있다. 이러한 저장부는 서버 내에 포함되거나 혹은 서버의 관리 하에 존재할 수 있다. 다른 예시로, 저장부는 서버 외부에 존재하여 서버와 통신가능한 형태로 구현될 수도 있다. 이 경우 서버와는 상이한 다른 외부 서버에 의해 저장부가 관리 및 제어될 수 있다. 전술한 저장부는 메모리(130)와 상호 교환 가능하게 사용될 수 있다.

【0042】일 실시예에서, 단말은 서버 또는 다른 컴퓨팅 디바이스와 상호작용 가능한 임의의 형태의 단말을 포함할 수 있다. 예를 들어, 단말은 휴대폰, 스마트폰(smart phone), 노트북 컴퓨터(laptop computer), PDA(personal digital assistants), 슬레이트 PC(slate PC), 태블릿 PC(tablet PC), 울트라북(ultrabook) 등을 포함할 수 있다.

【0043】일 실시예에서, 프로세서(110)는 하나 이상의 코어로 구성될 수 있으며, 컴퓨팅 장치(100)의 중앙 처리 장치(CPU: central processing unit), 범용 그래픽 처리 장치(GPGPU: general purpose graphics processing unit), 텐서 처리 장치(TPU: tensor processing unit)등의 데이터의 처리에 관련된 동작을 수행하기

위한 프로세서를 포함할 수 있다.

【0044】일 실시예에서, 프로세서(110)는 신경망의 학습을 위한 연산을 수행할 수도 있다. 예를 들어, 프로세서(110)는 딥러닝(DL: deep learning)에서 학습을 위한 입력 데이터의 처리, 입력 데이터에서의 피쳐 추출, 오차 계산, 역전파(backpropagation)를 이용한 신경망의 가중치 업데이트 등의 신경망의 학습을 위한 계산을 수행할 수 있다. 프로세서(110)의 CPU, GPGPU, 및 TPU 중 적어도 하나가 네트워크 함수의 학습을 처리할 수 있다. 예를 들어, CPU 와 GPGPU가 함께 네트워크 함수의 학습, 네트워크 함수를 이용한 데이터 처리를 수행할 수 있다. 또한, 본 개시의 일 실시예에서 복수의 컴퓨팅 장치들의 프로세서들을 함께 사용하여 네트워크 함수의 학습, 네트워크 함수를 이용한 데이터 처리를 수행할 수도 있다.

【0045】프로세서(110)는 통상적으로 컴퓨팅 장치(100)의 전반적인 동작을 제어할 수 있다. 프로세서(110)는 컴퓨팅 장치(100)에 포함된 구성요소들을 통해 입력 또는 출력되는 신호, 데이터, 정보 등을 처리하거나 메모리(130)에 저장된 응용 프로그램을 구동함으로써, 사용자에게 적절한 정보 또는 기능을 제공 또는 처리할 수 있다.

【0046】프로세서(110)는 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법을 수행할 수 있다.

【0047】일 실시예에서, 메모리(130)는 프로세서(110)가 생성하거나 결정한 임의의 형태의 정보 및/또는 네트워크부(150)가 수신한 임의의 형태의 정보를 저장할 수 있다. 프로세서(110)는 메모리(130)에 저장된 데이터를 이용하여 본 개시의

일 실시예에 따른 인공지능 모델을 이용한 정보처리를 위한 동작을 수행할 수 있다. 예를 들어, 프로세서(110)는 입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별하는 동작을 수행할 수 있다. 파이썬 소스코드 및/또는 API 목록은 메모리(130)에 사전 저장되어 있을 수 있다. 프로세서(110)는 코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석할 수 있다. 프로세서(110)는 구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 API 호출을 호출하거나 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석할 수 있다. 프로세서(110)는 함수 내부의 정적 의존성을 분석한 결과 및 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성할 수 있다.

【0048】일 실시예에서, 인공지능 모델은 메모리(130)에 사전 저장되어 있을 수 있다. 컴퓨팅 장치(100)는 인공지능 모델을 외부(예를 들어, 외부 단말, 외부 서버, 외부 저장소 등)로부터 실시간으로 수신할 수도 있다.

【0049】일 실시예에서, 메모리(130)는 플래시 메모리 타입(flash memory type), 하드디스크 타입(hard disk type), 멀티미디어 카드 마이크로 타입(multimedia card micro type), 카드 타입의 메모리(예를 들어 SD 또는 XD 메모리 등), 램(Random Access Memory, RAM), SRAM(Static Random Access Memory), 롬(Read-Only Memory, ROM), EEPROM(Electrically Erasable Programmable Read-Only Memory), PROM(Programmable Read-Only Memory), 자기 메모리, 자기 디스크 및/또

는 광디스크 중 적어도 하나의 타입의 저장매체를 포함할 수 있다. 컴퓨팅 장치(100)는 인터넷(internet) 상에서 메모리(130)의 저장 기능을 수행하는 웹 스토리지(web storage)와 관련되어 동작할 수도 있다. 전술한 메모리에 대한 기재는 예시일 뿐, 본 개시는 이에 제한되지 않는다. 메모리(130)는 프로세서(110)에 의하여 동작될 수 있다.

【0050】 일 실시예에서, 네트워크부(150)는 임의의 형태의 데이터 및 신호 등을 송신 및 수신할 수 있는 임의의 유무선 통신 네트워크를 포함할 수 있다. 본 명세서에서 설명된 기술들은 위에서 언급된 네트워크들뿐만 아니라, 다른 네트워크들에서도 사용될 수 있다.

【0051】 프로세서(110)는 네트워크부(150)와 상호작용함으로써, 본 개시의 일 실시예에 따른 단계들 및 동작들을 수행할 수 있다. 예를 들어, 프로세서(110)는 네트워크부(150)를 통해, 인공지능 모델을 통해 출력된 결과물, 출력 데이터 등을 외부(예를 들어, 외부 단말, 외부 서버, 외부 저장소 등)로 전달할 수 있다. 다른 예를 들어, 프로세서(110)는 생성된 분석 컨텍스트를 외부(예를 들어, 외부 단말, 외부 서버, 외부 저장소 등)로 전달할 수 있다.

【0052】 도 2는 본 개시의 일 실시예에 따른 인공지능 모델의 예시적인 구조를 나타낸 도면이다.

【0053】 본 명세서에 걸쳐, 인공지능 모델, 인공지능 기반 모델, 연산 모델, 신경망, 네트워크 함수, 뉴럴 네트워크(neural network)는 동일한 의미로 사용될 수 있다.

【0054】신경망은 일반적으로 노드라 지칭될 수 있는 상호 연결된 계산 단위들의 집합으로 구성될 수 있다. 이러한 노드들은 뉴런(neuron)들로 지칭될 수도 있다. 신경망은 적어도 하나 이상의 노드들을 포함하여 구성된다. 신경망들을 구성하는 노드(또는 뉴런)들은 하나 이상의 링크에 의해 상호 연결될 수 있다.

【0055】신경망 내에서, 링크를 통해 연결된 하나 이상의 노드들은 상대적으로 입력 노드 및 출력 노드의 관계를 형성할 수 있다. 입력 노드 및 출력 노드의 개념은 상대적인 것으로서, 하나의 노드에 대하여 출력 노드 관계에 있는 임의의 노드는 다른 노드와의 관계에서 입력 노드 관계에 있을 수 있으며, 그 역도 성립할 수 있다. 상술한 바와 같이, 입력 노드 대 출력 노드 관계는 링크를 중심으로 생성될 수 있다. 하나의 입력 노드에 하나 이상의 출력 노드가 링크를 통해 연결될 수 있으며, 그 역도 성립할 수 있다.

【0056】하나의 링크를 통해 연결된 입력 노드 및 출력 노드 관계에서, 출력 노드의 데이터는 입력 노드에 입력된 데이터에 기초하여 그 값이 결정될 수 있다. 여기서 입력 노드와 출력 노드를 상호 연결하는 링크는 가중치(weight)를 가질 수 있다. 가중치는 가변적일 수 있으며, 신경망이 원하는 기능을 수행하기 위해, 사용자 또는 알고리즘에 의해 가변 될 수 있다. 예를 들어, 하나의 출력 노드에 하나 이상의 입력 노드가 각각의 링크에 의해 상호 연결된 경우, 출력 노드는 상기 출력 노드와 연결된 입력 노드들에 입력된 값들 및 각각의 입력 노드들에 대응하는 링크에 설정된 가중치에 기초하여 출력 노드 값을 결정할 수 있다.

【0057】상술한 바와 같이, 신경망은 하나 이상의 노드들이 하나 이상의 링

크를 통해 상호 연결되어 신경망 내에서 입력 노드 및 출력 노드 관계를 형성한다. 신경망 내에서 노드들과 링크들의 개수 및 노드들과 링크들 사이의 연관관계, 링크들 각각에 부여된 가중치의 값에 따라, 신경망의 특성이 결정될 수 있다. 예를 들어, 동일한 개수의 노드 및 링크들이 존재하고, 링크들의 가중치 값이 상이한 두 신경망이 존재하는 경우, 두 개의 신경망들은 서로 상이한 것으로 인식될 수 있다.

【0058】 신경망은 하나 이상의 노드들의 집합으로 구성될 수 있다. 신경망을 구성하는 노드들의 부분 집합은 레이어(layer)를 구성할 수 있다. 신경망을 구성하는 노드들 중 일부는, 최초 입력 노드로부터의 거리들에 기초하여, 하나의 레이어(layer)를 구성할 수 있다. 예를 들어, 최초 입력 노드로부터 거리가 n인 노드들의 집합은, n 레이어를 구성할 수 있다. 최초 입력 노드로부터 거리는, 최초 입력 노드로부터 해당 노드까지 도달하기 위해 거쳐야 하는 링크들의 최소 개수에 의해 정의될 수 있다. 그러나, 이러한 레이어의 정의는 설명을 위한 임의적인 것으로서, 신경망 내에서 레이어의 차수는 상술한 것과 상이한 방법으로 정의될 수 있다. 예를 들어, 노드들의 레이어는 최종 출력 노드로부터 거리에 의해 정의될 수도 있다.

【0059】 본 개시내용의 일 실시예에서, 뉴런들 또는 노드들의 집합은 레이어라는 표현으로 정의될 수 있다.

【0060】 최초 입력 노드는 신경망 내의 노드들 중 다른 노드들과의 관계에서 링크를 거치지 않고 데이터가 직접 입력되는 하나 이상의 노드들을 의미할 수 있다. 또는, 신경망 네트워크 내에서, 링크를 기준으로 한 노드 간의 관계에 있어서, 링크로 연결된 다른 입력 노드들을 가지지 않는 노드들을 의미할 수 있다. 이

와 유사하게, 최종 출력 노드는 신경망 내의 노드들 중 다른 노드들과의 관계에서, 출력 노드를 가지지 않는 하나 이상의 노드들을 의미할 수 있다. 또한, 히든 노드는 최초 입력 노드 및 최후 출력 노드가 아닌 신경망을 구성하는 노드들을 의미할 수 있다.

【0061】본 개시의 일 실시예에 따른 신경망은 입력 레이어의 노드의 개수가 출력 레이어의 노드의 개수와 동일할 수 있으며, 입력 레이어에서 히든 레이어로 진행됨에 따라 노드의 수가 감소하다가 다시 증가하는 형태의 신경망일 수 있다. 또한, 본 개시의 다른 일 실시예에 따른 신경망은 입력 레이어의 노드의 개수가 출력 레이어의 노드의 개수 보다 적을 수 있으며, 입력 레이어에서 히든 레이어로 진행됨에 따라 노드의 수가 감소하는 형태의 신경망일 수 있다. 또한, 본 개시의 또 다른 일 실시예에 따른 신경망은 입력 레이어의 노드의 개수가 출력 레이어의 노드의 개수보다 많을 수 있으며, 입력 레이어에서 히든 레이어로 진행됨에 따라 노드의 수가 증가하는 형태의 신경망일 수 있다. 본 개시의 또 다른 일 실시예에 따른 신경망은 상술한 신경망들의 조합된 형태의 신경망일 수 있다.

【0062】본 개시내용의 일 실시예에 따른 인공지능 기반 모델은 딥 뉴럴 네트워크(DNN: deep neural network, 심층신경망)를 포함할 수 있다. 딥 뉴럴 네트워크는 입력 레이어와 출력 레이어 외에 복수의 히든 레이어를 포함하는 신경망을 의미할 수 있다. 딥 뉴럴 네트워크를 이용하면 데이터의 잠재적인 구조(latent structures)를 파악할 수 있다. 즉, 사진, 글, 비디오, 음성, 단백질 시퀀스 구조, 유전자 시퀀스 구조, 펩타이드 서열의 구조, 음악의 잠재적인 구조(예를 들어, 어

떤 물체가 사진에 있는지, 글의 내용과 감정이 무엇인지, 음성의 내용과 감정이 무엇인지 등), 및/또는 펩타이드와 MHC 간의 결합 친화도를 파악할 수 있다. 딥 뉴럴 네트워크는 컨볼루션 뉴럴 네트워크(CNN: convolutional neural network), 리커런트 뉴럴 네트워크(RNN: recurrent neural network), 오토 인코더(auto encoder), 제한 볼츠만 머신(RBM: restricted boltzmann machine), 심층 신뢰 네트워크(DBN: deep belief network), Q 네트워크, U 네트워크, 샴 네트워크, 적대적 생성 네트워크(GAN: Generative Adversarial Network), 트랜스포머 등을 포함할 수 있다. 전술한 딥 뉴럴 네트워크의 기재는 예시일 뿐이며 본 개시는 이에 제한되지 않는다.

【0063】 본 개시내용의 인공지능 기반 모델은 입력 레이어, 히든 레이어 및 출력 레이어를 포함하는 전술한 임의의 구조의 네트워크 구조에 의해 표현될 수 있다.

【0064】 본 개시내용의 인공지능 기반 모델에서 사용될 수 있는 뉴럴 네트워크는 지도 학습(supervised learning), 비지도 학습(unsupervised learning), 반지도학습(semi supervised learning), 전이 학습(transfer learning), 능동 학습(active learning) 또는 강화학습(reinforcement learning) 중 적어도 하나의 방식으로 학습될 수 있다. 뉴럴 네트워크의 학습은 뉴럴 네트워크가 특정한 동작을 수행하기 위한 지식을 뉴럴 네트워크에 적용하는 과정일 수 있다.

【0065】 뉴럴 네트워크는 출력의 오류를 최소화하는 방향으로 학습될 수 있다. 뉴럴 네트워크의 학습에서 반복적으로 학습 데이터를 뉴럴 네트워크에 입력시키고 학습 데이터에 대한 뉴럴 네트워크의 출력과 타겟의 에러를 계산하고, 에러를

줄이기 위한 방향으로 뉴럴 네트워크의 에러를 뉴럴 네트워크의 출력 레이어에서부터 입력 레이어 방향으로 역전파(backpropagation)하여 뉴럴 네트워크의 각 노드의 가중치를 업데이트 하는 과정이다. 지도 학습의 경우 각각의 학습 데이터에 정답이 라벨링되어있는 학습 데이터를 사용하며(즉, 라벨링된 학습 데이터), 비지도 학습의 경우는 각각의 학습 데이터에 정답이 라벨링되어 있지 않을 수 있다. 즉, 예를 들어 데이터 분류에 관한 지도 학습의 경우의 학습 데이터는 학습 데이터 각각에 카테고리가 라벨링 된 데이터 일 수 있다. 라벨링된 학습 데이터가 뉴럴 네트워크에 입력되고, 뉴럴 네트워크의 출력(카테고리)과 학습 데이터의 라벨을 비교함으로써 오류(error)가 계산될 수 있다. 다른 예로, 데이터 분류에 관한 비지도 학습의 경우 입력인 학습 데이터가 뉴럴 네트워크 출력과 비교됨으로써 오류가 계산될 수 있다. 계산된 오류는 뉴럴 네트워크에서 역방향(즉, 출력 레이어에서 입력 레이어 방향)으로 역전파 되며, 역전파에 따라 뉴럴 네트워크의 각 레이어의 각 노드들의 연결 가중치가 업데이트 될 수 있다. 업데이트 되는 각 노드의 연결 가중치는 학습률(learning rate)에 따라 변화량이 결정될 수 있다. 입력 데이터에 대한 뉴럴 네트워크의 계산과 에러의 역전파는 학습 사이클(epoch)을 구성할 수 있다. 학습률은 뉴럴 네트워크의 학습 사이클의 반복 횟수에 따라 상이하게 적용될 수 있다. 예를 들어, 뉴럴 네트워크의 학습 초기에는 높은 학습률을 사용하여 뉴럴 네트워크가 빠르게 일정 수준의 성능을 확보하도록 하여 효율성을 높이고, 학습 후기에는 낮은 학습률을 사용하여 정확도를 높일 수 있다.

【0066】뉴럴 네트워크의 학습에서 일반적으로 학습 데이터는 실제 데이터

(즉, 학습된 뉴럴 네트워크를 이용하여 처리하고자 하는 데이터)의 부분집합일 수 있으며, 따라서, 학습 데이터에 대한 오류는 감소하나 실제 데이터에 대해서는 오류가 증가하는 학습 사이클이 존재할 수 있다. 과적합(overfitting)은 이와 같이 학습 데이터에 과하게 학습하여 실제 데이터에 대한 오류가 증가하는 현상이다. 예를 들어, 노란색 고양이를 보여 고양이를 학습한 뉴럴 네트워크가 노란색 이외의 고양이를 보고는 고양이임을 인식하지 못하는 현상이 과적합의 일종일 수 있다. 과적합은 머신러닝 알고리즘의 오류를 증가시키는 원인으로 작용할 수 있다. 이러한 과적합을 막기 위하여 다양한 최적화 방법이 사용될 수 있다. 과적합을 막기 위해서는 학습 데이터를 증가시키거나, 레귤라리제이션(regularization), 학습의 과정에서 네트워크의 노드 일부를 비활성화하는 드롭아웃(dropout), 배치 정규화 레이어(batch normalization layer)의 활용 등의 방법이 적용될 수 있다.

【0067】 본 개시의 일 실시예에 따른 데이터 구조를 저장한 컴퓨터 판독가능 매체가 개시된다. 전술한 데이터 구조는 본 개시내용에서의 저장부(예를 들어, 메모리(130))에 저장될 수 있으며, 프로세서(110)에 의해 실행될 수 있으며 그리고 통신부(예를 들어, 네트워크부(150))에 의해 송수신될 수 있다.

【0068】 데이터 구조는 데이터에 효율적인 접근 및 수정을 가능하게 하는 데이터의 조직, 관리, 저장을 의미할 수 있다. 데이터 구조는 특정 문제(예를 들어, 데이터 분석, 데이터 검색, 데이터 저장, 데이터 수정) 해결을 위한 데이터의 조직을 의미할 수 있다. 데이터 구조는 특정한 데이터 처리 기능을 지원하도록 설계된, 데이터 요소들 간의 물리적이거나 논리적인 관계로 정의될 수도 있다. 데이터 요소

들 간의 논리적인 관계는 사용자 정의 데이터 요소들 간의 연결관계를 포함할 수 있다. 데이터 요소들 간의 물리적인 관계는 컴퓨터 판독가능 저장매체(예를 들어, 영구 저장 장치)에 물리적으로 저장되어 있는 데이터 요소들 간의 실제 관계를 포함할 수 있다. 데이터 구조는 구체적으로 데이터의 집합, 데이터 간의 관계, 데이터에 적용할 수 있는 함수 또는 명령어를 포함할 수 있다. 효과적으로 설계된 데이터 구조를 통해 컴퓨팅 장치는 컴퓨팅 장치의 자원을 최소한으로 사용하면서 연산을 수행할 수 있다. 구체적으로 컴퓨팅 장치는 효과적으로 설계된 데이터 구조를 통해 연산, 읽기, 삽입, 삭제, 비교, 교환 및 검색의 효율성을 높일 수 있다.

【0069】 데이터 구조는 데이터 구조의 형태에 따라 선형 데이터 구조와 비선형 데이터 구조로 구분될 수 있다. 선형 데이터 구조는 하나의 데이터 뒤에 하나의 데이터만이 연결되는 구조일 수 있다. 선형 데이터 구조는 리스트(List), 스택(Stack), 큐(Queue), 데크(Deque)를 포함할 수 있다. 리스트는 내부적으로 순서가 존재하는 일련의 데이터 집합을 의미할 수 있다. 리스트는 연결 리스트(Linked List)를 포함할 수 있다. 연결 리스트는 각각의 데이터가 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터가 연결된 데이터 구조일 수 있다. 연결 리스트에서 포인터는 다음이나 이전 데이터와의 연결 정보를 포함할 수 있다. 연결 리스트는 형태에 따라 단일 연결 리스트, 이중 연결 리스트, 원형 연결 리스트로 표현될 수 있다. 스택은 제한적으로 데이터에 접근할 수 있는 데이터 나열 구조일 수 있다. 스택은 데이터 구조의 한 쪽 끝에서만 데이터를 처리(예를 들어, 삽입 또는 삭제)할 수 있는 선형 데이터 구조일 수 있다. 스택에 저장된 데이터는 늦게 들어갈수록

빨리 나오는 데이터 구조(LIFO-Last in First Out)일 수 있다. 큐는 제한적으로 데이터에 접근할 수 있는 데이터 나열 구조로서, 스택과 달리 늦게 저장된 데이터일수록 늦게 나오는 데이터 구조(FIFO-First in First Out)일 수 있다. 데크는 데이터 구조의 양 쪽 끝에서 데이터를 처리할 수 있는 데이터 구조일 수 있다.

【0070】비선형 데이터 구조는 하나의 데이터 뒤에 복수개의 데이터가 연결되는 구조일 수 있다. 비선형 데이터 구조는 그래프(Graph) 데이터 구조를 포함할 수 있다. 그래프 데이터 구조는 정점(Vertex)과 간선(Edge)으로 정의될 수 있으며 간선은 서로 다른 두개의 정점을 연결하는 선을 포함할 수 있다. 그래프 데이터 구조는 트리(Tree) 데이터 구조를 포함할 수 있다. 트리 데이터 구조는 트리에 포함된 복수개의 정점 중에서 서로 다른 두개의 정점을 연결시키는 경로가 하나인 데이터 구조일 수 있다. 즉 그래프 데이터 구조에서 루프(loop)를 형성하지 않는 데이터 구조일 수 있다.

【0071】본 명세서에 걸쳐, 인공지능 기반 모델, 연산 모델, 신경망, 네트워크 함수, 뉴럴 네트워크(neural network)는 상호 교환 가능한 의미로 사용될 수 있다. 이하에서는 신경망으로 통일하여 기술한다. 데이터 구조는 신경망을 포함할 수 있다. 그리고 신경망을 포함한 데이터 구조는 컴퓨터 판독가능 매체에 저장될 수 있다. 신경망을 포함한 데이터 구조는 또한 신경망에 의한 처리를 위하여 전처리된 데이터, 신경망에 입력되는 데이터, 신경망의 가중치, 신경망의 하이퍼 파라미터, 신경망으로부터 획득한 데이터, 신경망의 각 노드 또는 레이어와 연관된 활성함수, 신경망의 학습을 위한 손실 함수 등을 포함할 수 있다. 신경망을 포함한 데

이터 구조는 상기 개시된 구성들 중 임의의 구성 요소들을 포함할 수 있다. 즉 신경망을 포함한 데이터 구조는 신경망에 의한 처리를 위하여 전처리된 데이터, 신경망에 입력되는 데이터, 신경망의 가중치, 신경망의 하이퍼 파라미터, 신경망으로부터 획득한 데이터, 신경망의 각 노드 또는 레이어와 연관된 활성 함수, 신경망의 학습을 위한 손실 함수 등 전부 또는 이들의 임의의 조합을 포함하여 구성될 수 있다. 전술한 구성들 이외에도, 신경망을 포함한 데이터 구조는 신경망의 특성을 결정하는 임의의 다른 정보를 포함할 수 있다. 또한, 데이터 구조는 신경망의 연산 과정에 사용되거나 발생되는 모든 형태의 데이터를 포함할 수 있으며 전술한 사항에 제한되는 것은 아니다. 컴퓨터 판독가능 매체는 컴퓨터 판독가능 기록 매체 및/또는 컴퓨터 판독가능 전송 매체를 포함할 수 있다. 신경망은 일반적으로 노드라 지칭될 수 있는 상호 연결된 계산 단위들의 집합으로 구성될 수 있다. 이러한 노드들은 뉴런(neuron)들로 지칭될 수도 있다. 신경망은 적어도 하나 이상의 노드들을 포함하여 구성된다.

【0072】 데이터 구조는 신경망에 입력되는 데이터를 포함할 수 있다. 신경망에 입력되는 데이터를 포함하는 데이터 구조는 컴퓨터 판독가능 매체에 저장될 수 있다. 신경망에 입력되는 데이터는 신경망 학습 과정에서 입력되는 학습 데이터 및/또는 학습이 완료된 신경망에 입력되는 입력 데이터를 포함할 수 있다. 신경망에 입력되는 데이터는 전처리(pre-processing)를 거친 데이터 및/또는 전처리 대상이 되는 데이터를 포함할 수 있다. 전처리는 데이터를 신경망에 입력시키기 위한 데이터 처리 과정을 포함할 수 있다. 따라서 데이터 구조는 전처리 대상이 되는 데이터

및 전처리로 발생되는 데이터를 포함할 수 있다. 전술한 데이터 구조는 예시일 뿐 본 개시는 이에 제한되지 않는다.

【0073】 데이터 구조는 신경망의 가중치를 포함할 수 있다. (본 명세서에서 가중치, 파라미터는 상호 교환가능한 의미로 사용될 수 있다.) 그리고 신경망의 가중치를 포함한 데이터 구조는 컴퓨터 판독가능 매체에 저장될 수 있다. 신경망은 복수개의 가중치를 포함할 수 있다. 가중치는 가변적일 수 있으며, 신경망이 원하는 기능을 수행하기 위해, 사용자 또는 알고리즘에 의해 가변 될 수 있다. 예를 들어, 하나의 출력 노드에 하나 이상의 입력 노드가 각각의 링크에 의해 상호 연결된 경우, 출력 노드는 상기 출력 노드와 연결된 입력 노드들에 입력된 값들 및 각각의 입력 노드들에 대응하는 링크에 설정된 가중치에 기초하여 출력 노드에서 출력되는 데이터 값을 결정할 수 있다. 전술한 데이터 구조는 예시일 뿐 본 개시는 이에 제한되지 않는다.

【0074】 제한이 아닌 예로서, 가중치는 신경망 학습 과정에서 가변되는 가중치 및/또는 신경망 학습이 완료된 가중치를 포함할 수 있다. 신경망 학습 과정에서 가변되는 가중치는 학습 사이클이 시작되는 시점의 가중치 및/또는 학습 사이클 동안 가변되는 가중치를 포함할 수 있다. 신경망 학습이 완료된 가중치는 학습 사이클이 완료된 가중치를 포함할 수 있다. 따라서 신경망의 가중치를 포함한 데이터 구조는 신경망 학습 과정에서 가변되는 가중치 및/또는 신경망 학습이 완료된 가중치를 포함한 데이터 구조를 포함할 수 있다. 그러므로 상술한 가중치 및/또는 각 가중치의 조합은 신경망의 가중치를 포함한 데이터 구조에 포함되는 것으로 한다.

전술한 데이터 구조는 예시일 뿐 본 개시는 이에 제한되지 않는다.

【0075】신경망의 가중치를 포함한 데이터 구조는 직렬화(serialization) 과정을 거친 후 컴퓨터 판독가능 저장 매체(예를 들어, 메모리, 하드 디스크)에 저장될 수 있다. 직렬화는 데이터 구조를 동일하거나 다른 컴퓨팅 장치에 저장하고 나중에 다시 재구성하여 사용할 수 있는 형태로 변환하는 과정일 수 있다. 컴퓨팅 장치는 데이터 구조를 직렬화하여 네트워크를 통해 데이터를 송수신할 수 있다. 직렬화된 신경망의 가중치를 포함한 데이터 구조는 역직렬화(deserialization)를 통해 동일한 컴퓨팅 장치 또는 다른 컴퓨팅 장치에서 재구성될 수 있다. 신경망의 가중치를 포함한 데이터 구조는 직렬화에 한정되는 것은 아니다. 나아가 신경망의 가중치를 포함한 데이터 구조는 컴퓨팅 장치의 자원을 최소한으로 사용하면서 연산의 효율을 높이기 위한 데이터 구조(예를 들어, 비선형 데이터 구조에서 B-Tree, R-Tree, Trie, m-way search tree, AVL tree, Red-Black Tree)를 포함할 수 있다. 전술한 사항은 예시일 뿐 본 개시는 이에 제한되지 않는다.

【0076】데이터 구조는 신경망의 하이퍼 파라미터(Hyper-parameter)를 포함할 수 있다. 그리고 신경망의 하이퍼 파라미터를 포함한 데이터 구조는 컴퓨터 판독가능 매체에 저장될 수 있다. 하이퍼 파라미터는 사용자에 의해 가변되는 변수일 수 있다. 하이퍼 파라미터는 예를 들어, 학습률(learning rate), 비용 함수(cost function), 학습 사이클 반복 횟수, 가중치 초기화(Weight initialization)(예를 들어, 가중치 초기화 대상이 되는 가중치 값의 범위 설정), Hidden Unit 개수(예를 들어, 히든 레이어의 개수, 히든 레이어의 노드 수)를 포함할 수 있다. 전술한 데

이터 구조는 예시일 뿐 본 개시는 이에 제한되지 않는다.

【0077】 본 개시내용의 일 실시예에 따른 인공지능 기반 모델은 대규모 언어모델(Large Language Model, LLM)을 포함할 수 있다. 본 개시내용에서의 대규모 언어모델이란, 자연어 처리(Natural Language Processing)를 수행하도록 방대한 양의 학습 데이터를 이용하여 학습된 인공지능 기반의 모델을 의미할 수 있다. 대규모 언어모델은 트랜스포머(transformer), 트랜스포머의 인코더 계열의 모델 및/또는 트랜스포머의 디코더 계열의 모델을 포함할 수 있다. 트랜스포머의 인코더 계열의 모델은 트랜스포머의 인코더 구조를 사용하는 인공지능 모델에 대응될 수 있다. 트랜스포머의 디코더 계열의 모델은 트랜스포머의 디코더 구조를 사용하는 인공지능 모델에 대응될 수 있다.

【0078】 일 실시예에서, 트랜스포머는 입력 데이터를 인코딩하는 인코더 및 인코딩된 데이터들을 디코딩하는 디코더로 구성될 수 있다. 트랜스포머는 일련의 입력 데이터(a series of input data)를 입력으로 하여, 인코딩 및 디코딩 단계를 거쳐 일련의 출력 데이터를 출력하는 구조를 지닐 수 있다. 일 실시예에서, 일련의 입력 데이터는 트랜스포머가 연산가능한 형태로 가공될 수 있다. 일련의 입력 데이터를 트랜스포머가 연산가능한 형태로 가공하는 과정은 토크나이징 과정 및 임베딩 과정을 포함할 수 있다. 토크나이징 과정이란 일련의 입력 데이터를 일정 단위의 토큰으로 분할하는 과정을 의미할 수 있다. 예를 들어, 일정 단위는 단어 단위를 포함할 수 있다. 임베딩 과정이란 일련의 입력 데이터로부터 토크나이징된 적어도 하나의 토큰을 임베딩 벡터로 변환하는 과정을 의미할 수 있다.

【0079】일 실시예에서, 트랜스포머는 일련의 입력 데이터에 대응되는 적어도 하나의 토큰을 임베딩한 토큰 임베딩 벡터, 토큰 별로 토큰이 포함된 문장을 구분하는 세그멘트(segment) 임베딩 벡터 및 토큰의 위치(position)를 반영한 위치 임베딩 벡터를 합하여 인코더에 입력될 임베딩 벡터를 획득할 수 있다. 트랜스포머의 인코더 계열의 모델 및 디코더 계열의 모델 또한 동일한 방식을 수행하여 임베딩 벡터를 획득할 수 있다.

【0080】일 실시예에서, 트랜스포머가 일련의 입력 데이터를 인코딩 및 디코딩하기 위하여, 트랜스포머 내의 인코더 및 디코더는 어텐션(attention) 알고리즘을 활용할 수 있다. 어텐션 알고리즘이란 주어진 쿼리(Query)에 대해, 쿼리를 키(Key)와 행렬곱한 어텐션 스코어에 softmax 함수를 적용하여 유사도를 산출하고, 산출한 유사도를 밸류(Value)에 행렬곱하여 쿼리에 대한 어텐션(attention) 값을 산출하는 알고리즘을 의미할 수 있다.

【0081】일 실시예에서, 셀프 어텐션 알고리즘은 동일한 임베딩 벡터에 쿼리 가중치, 키 가중치 및 밸류 가중치를 각각 곱하여 생성된 쿼리, 키 및 밸류를 이용하는 어텐션 알고리즘을 의미할 수 있다. 크로스 어텐션 알고리즘은 제1 임베딩 벡터에 쿼리 가중치를 곱하여 생성된 쿼리와, 제2 임베딩 벡터에 키 가중치 및 밸류 가중치를 각각 곱하여 생성된 키 및 밸류를 이용하는 어텐션 알고리즘을 의미할 수 있다. 쿼리 가중치, 키 가중치 및 밸류 가중치는 대규모 언어모델의 학습 과정을 거쳐 업데이트되는 학습 가능한 파라미터(trainable parameter)일 수 있다.

【0082】일 실시예에서, 트랜스포머의 인코더는 임베딩 레이어, 임베딩 벡터

에 셀프 어텐션 알고리즘을 적용하는 셀프 어텐션 레이어, 정규화 레이어 및 피드 포워드 신경망(Feed Forward Neural Network, FFN)을 포함할 수 있다. 또한, 인코더는 셀프 어텐션 레이어, 정규화 레이어 및 피드 포워드 신경망을 포함하는 단위 구조를 N개 연결한 형태를 가질 수 있다. 트랜스포머의 디코더는 임베딩 레이어, 마스킹된(masked) 셀프 어텐션 레이어, 정규화 레이어, 크로스 어텐션 알고리즘을 적용하는 크로스 어텐션 레이어, 피드 포워드 신경망을 포함할 수 있다. 또한, 디코더는 마스킹된 셀프 어텐션 레이어, 정규화 레이어, 크로스 어텐션 레이어 및 피드 포워드 신경망을 포함하는 단위 구조를 N개 연결한 형태를 가질 수 있다. 마스킹된 셀프 어텐션 레이어는 일련의 입력 데이터에 포함된 복수의 단어들에 있어서, 단어들이 순차적으로 포함된 시퀀스들 각각에 대한 어텐션 값을 구하는 레이어에 대응될 수 있다.

【0083】트랜스포머는 인코더 및 디코더 뿐만 아니라, 선형(linear) 레이어, 소프트맥스(softmax) 레이어 등 부가적인 구성요소들 또한 포함할 수 있다. 트랜스포머의 인코더 계열의 모델 및 트랜스포머의 디코더 계열의 모델 또한 각각 인코더 및 디코더 뿐만 아니라 상기 부가적인 구성요소들을 포함할 수 있다. 어텐션 알고리즘을 이용하여 트랜스포머를 구성하는 방법은 Vaswani et al., Attention Is All You Need, 2017 NIPS에 개시된 방법을 포함할 수 있으며, 이는 여기에 참조로서 통합된다.

【0084】일 실시예에서, 셀프 어텐션 레이어, 마스킹된 셀프 어텐션 레이어, 크로스 어텐션 레이어 등의 어텐션 레이어는 복수의 어텐션 레이어들을 병렬적으로

포함하는 멀티-헤드(multi-head) 어텐션 레이어에 대응될 수 있다. 멀티-헤드 어텐션 레이어는 복수의 어텐션 레이어들 각각에서 출력된 어텐션 값을 행렬 연결하고(concatenate), 연결된 행렬에 출력 가중치를 행렬곱하여 출력 어텐션 값을 출력할 수 있다. 멀티-헤드 어텐션 레이어에서 출력된 출력 어텐션 값은 하나의 어텐션 레이어에서 출력된 어텐션 값과 동일한 크기를 가질 수 있다.

【0085】일 실시예에서, 트랜스포머는 Masked Language Model(MLM) 과정, Next Sentence Prediction(NSP) 과정 등을 통해 학습될 수 있다. MLM 과정은 일부 단어가 마스킹된 일련의 학습 데이터를 통해 마스킹된 단어를 예측하는 학습 과정을 의미할 수 있다. NSP 과정은 임의의 두 문장을 포함하는 일련의 학습 데이터에서 두 문장이 연결된 문장인지 여부를 판별하는 학습 과정을 의미할 수 있다.

【0086】일 실시예에서, 대규모 언어모델은 자연어 텍스트 뿐만 아니라 이미지 데이터, 오디오 데이터, 비디오 데이터 등 다양한 데이터 형식을 처리할 수 있다. 대규모 언어모델은 다양한 데이터 형식을 가진 데이터들을 연산가능한 일련의 데이터들로 변환하기 위해, 데이터들을 임베딩할 수 있다. 대규모 언어모델은 일련의 입력 데이터 사이의 상대적 위치관계 또는 위상관계를 표현하는 추가적인 데이터를 처리할 수 있다. 또는 일련의 입력 데이터에 입력 데이터들 사이의 상대적인 위치관계 또는 위상관계를 표현하는 벡터들이 추가적으로 반영되어 일련의 입력 데이터가 임베딩될 수 있다. 일 예에서, 일련의 입력 데이터 사이의 상대적 위치관계는, 자연어 문장 내에서의 어순, 각각의 분할된 이미지의 상대적 위치 관계, 분할된 오디오 파형의 시간 순서 등을 포함할 수 있으나, 이에 제한되지 않는다. 일련

의 입력 데이터들 사이의 상대적인 위치관계 또는 위상관계를 표현하는 정보를 추가하는 과정은 위치 인코딩(positional encoding)으로 지칭될 수 있다.

【0087】 이미지 데이터를 처리하는 대규모 언어모델의 일 예(Vision Transformer, ViT)는 Dosovitskiy, et al., AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE에 개시되어 있으며, 해당 문서는 여기에 참조로서 통합된다.

【0088】 본 개시내용의 일 실시예에 따른 인공지능 모델은 멀티모달(multi-modal) 대규모 언어모델을 포함할 수 있다. 멀티모달 대규모 언어모델은 자연어 텍스트 데이터, 이미지 데이터, 오디오 데이터, 비디오 데이터 등 서로 상이한 데이터 형식 간 관계성을 이해하고 처리할 수 있는 대규모 언어모델을 의미할 수 있다. 멀티모달 언어모델은 각 데이터 형식에 대응되는 입력 데이터를 인코딩하는 복수의 인코더들을 포함할 수 있다. 멀티모달 언어모델은 서로 상이한 데이터 형식의 데이터를 포함하는 학습 데이터를 통해, 각 데이터 형식의 인코더로부터 인코딩된 임베딩 벡터들 간의 유사도를 산출하고, 서로 동일한 짝(pair)에 대한 유사도는 더욱 높게 산출되고, 서로 다른 짝에 대한 유사도는 더욱 낮게 산출되도록 학습될 수 있다.

【0089】 이미지 데이터와 자연어 텍스트 데이터 간 관계성을 이해하고 처리하는 멀티모달 대규모 언어모델의 일 예(Contrastive Language-Image Pre-training, CLIP)는 Alec Radford, et al., LEARNING TRANSFERABLE VISUAL MODELS FROM NATURAL LANGUAGE SUPERVISION에 개시되어 있으며, 해당 문서는 여기에 참조

로서 통합된다.

【0090】 도 3은 본 개시의 일 실시예에 따른 파이썬 소스코드에서 응용 프로그램 인터페이스의 오용을 탐지하기 위한 방법을 나타낸 순서도이다. 도 3에서 도시되는 단계들은 컴퓨팅 장치(100)의 프로세서(110)에 의해 수행될 수 있다. 구현 양태에 따라 도 3에서 도시되는 단계들 중 일부가 생략되거나 또는 추가 단계가 포함될 수도 있다.

【0091】 이하에서는 컴퓨팅 장치(100)에서 파이썬 소스코드에서 응용 프로그램 인터페이스(Application Programming Interface, API)의 오용을 탐지하기 위한 예시적인 방법론을 제시한다.

【0092】 일 실시예에서, 파이썬 소스코드는 파이썬(Python) 프로그래밍 언어로 작성된 소스코드를 포괄적으로 의미할 수 있다. 파이썬은 동적 타입 결정(dynamic typing), 런타임 객체 바인딩(runtime object binding) 등의 언어 고유의 동적 특성을 가지며, 서버 애플리케이션, 인공지능 시스템 등 다양한 분야에서 광범위하게 활용될 수 있다. 이러한 동적 특성으로 인해, 파이썬 소스코드에서 응용 프로그램 인터페이스의 사용 맥락은 단일 함수 내에 국한되지 않고 복수의 함수에 걸쳐 분산되는 경우가 빈번하게 나타날 수 있다.

【0093】 일 실시예에서, 동적 타입 결정은 변수의 자료형이 컴파일 시점이 아닌 실행 시점에 결정되는 파이썬의 언어적 특성을 의미할 수 있다. 예를 들어, 동일한 변수에 정수, 문자열, 객체 등 서로 다른 자료형의 값이 순차적으로 할당될 수 있으며, 함수에 전달되는 인자의 자료형 또한 호출 시점에 따라 달라질 수

있다. 이러한 특성으로 인해, API 호출에 전달되는 파라미터의 자료형 및 값을 소스코드 분석만으로 정적으로 결정하기 어려운 경우가 발생할 수 있다.

【0094】일 실시예에서, 런타임 객체 바인딩은 객체와 메서드 간의 연결 관계가 실행 시점에 결정되는 파이썬의 언어적 특성을 의미할 수 있다. 예를 들어, update()와 같이 복수의 객체 타입에서 공통적으로 사용되는 메서드 이름은 호출 대상 객체의 타입에 따라 암호화 연산을 수행하는 cipher 객체의 메서드일 수도 있고, 해시 연산을 수행하는 hash 객체의 메서드일 수도 있다. 정적 분석만으로는 호출 대상 객체의 타입과 생성 경로를 완전히 파악하기 어렵기 때문에, 객체의 생성 맥락을 역방향으로 추적하는 분석이 요구된다.

【0095】일 실시예에서, 응용 프로그램 인터페이스는 소프트웨어 구성 요소 간의 상호작용을 정의하는 인터페이스를 포괄적으로 의미할 수 있다. 예를 들어, 응용 프로그램 인터페이스는 암호화 라이브러리에 포함된 클래스 및 함수를 포함할 수 있다. 구체적인 일례로, 응용 프로그램 인터페이스는 대칭 암호화(예를 들어, AES(Advanced Encryption Standard)), 키 유도(예를 들어, PBKDF2HMAC(Password-Based Key Derivation Function 2 with HMAC)), 해시 함수(예를 들어, SHA-256(Secure Hash Algorithm 256)) 등의 암호화 프리미티브(cryptographic primitive)를 제공하는 pycryptodome, PyNaCl, cryptography 등의 라이브러리에 포함된 클래스 및/또는 함수를 포함할 수 있다. 전술한 응용 프로그램 인터페이스의 기재는 예시일 뿐이며, 본 개시는 이에 제한되지 않는다.

【0096】일 실시예에서, 응용 프로그램 인터페이스의 오용(API misuse)은 응

용 프로그램 인터페이스가 보안 요구사항 또는 올바른 사용 규칙에 부합하지 않는 방식으로 사용되는 것을 포괄적으로 의미할 수 있다. 예를 들어, 응용 프로그램 인터페이스의 오용은 취약한 암호화 알고리즘의 사용, 하드코딩된 암호화 키의 사용, 안전하지 않은 블록 암호 운용 모드의 사용, 고정된 초기화 벡터(Initialization Vector, IV)의 사용, 안전하지 않은 난수 생성기의 사용, 패스워드 기반 암호화(Password-Based Encryption, PBE)에서의 불충분한 반복 횟수 설정 등을 포함할 수 있다. 이러한 오용은 단일 API 호출 지점만으로는 드러나지 않는 경우가 많으며, API 호출에 전달되는 인자 값의 기원과 함수 간 전달 과정, 객체의 생성 및 설정 맥락을 종합적으로 고려해야 정확한 판단이 가능하다. 전술한 응용 프로그램 인터페이스의 오용의 기재는 예시일 뿐이며, 본 개시는 이에 제한되지 않는다.

【0097】 도 3을 참조하면, 컴퓨팅 장치(100)는 입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별할 수 있다(310).

【0098】 예를 들어, 컴퓨팅 장치(100)는 파이썬 소스코드로부터 메서드 호출(method invocation)을 검출할 수 있다. 구체적인 일례로, 컴퓨팅 장치(100)는 구문 트리(Abstract Syntax Tree, AST) 및 함수 호출 구조를 이용하여 파이썬 소스코드 내에 존재하는 메서드 호출을 탐지할 수 있다. 컴퓨팅 장치(100)는 단순 문자열 매칭이 아닌, 메서드 호출의 완전한 식별자(method full name)를 기준으로 메서드 호출을 탐지함으로써, 실제 실행 가능한 메서드 호출만을 대상으로 할 수 있다. 임포트(import) 문이나 미사용 코드 영역은 분석 대상에서 제외될 수 있다.

【0099】 일 실시예에서, 구문 트리는 소스코드의 문법 구조를 트리 형태로

표현한 자료구조를 의미할 수 있다. 구문 트리는 소스코드를 파싱(parsing)하여 생성되며, 변수 선언, 함수 정의, 메서드 호출, 인자 전달 등의 구문적 요소들을 노드(node)와 엣지(edge)로 표현할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 파이썬의 내장 구문 트리 모듈을 활용하여 파이썬 소스코드로부터 구문 트리를 추출하고, 추출된 구문 트리의 노드를 탐색함으로써 메서드 호출, 인자 값, 변수 할당 등의 구문적 정보를 분석할 수 있다.

【0100】 일 실시예에서, 함수 호출 구조는 소스코드 내에서 함수 또는 메서드가 호출되는 방식과 호출 관계를 의미할 수 있다. 예를 들어, 함수 호출 구조는 호출자(caller) 함수와 피호출자(callee) 함수 사이의 관계, 호출 시 전달되는 인자의 구조, 및 반환값의 사용 방식 등을 포함할 수 있다. 컴퓨팅 장치(100)는 구문 트리에서 함수 호출 구조를 분석함으로써, 메서드 호출의 완전한 식별자를 결정하고 API 호출 여부를 판별할 수 있다.

【0101】 일 실시예에서, 메서드 호출의 완전한 식별자는 메서드 호출을 고유하게 식별하기 위한 정보로서, 호출 대상 객체 또는 클래스의 이름과 메서드 이름을 포함하는 식별자를 의미할 수 있다. 예를 들어, AES.new()의 완전한 식별자는 클래스 이름 AES와 메서드 이름 new를 결합한 형태로 표현될 수 있으며, hashlib.pbkdf2_hmac()의 완전한 식별자는 모듈 이름 hashlib와 함수 이름 pbkdf2_hmac를 결합한 형태로 표현될 수 있다. 컴퓨팅 장치(100)는 완전한 식별자를 기준으로 API 호출을 탐지함으로써, 동일한 메서드 이름을 가진 서로 다른 클래스의 메서드를 정확하게 구별할 수 있다.

【0102】일 실시예에서, 임포트 문은 외부 모듈 또는 라이브러리를 현재 소스코드에서 사용할 수 있도록 불러오는 파이썬의 구문을 의미할 수 있다. 예를 들어, from Crypto.Cipher import AES 또는 import hashlib와 같은 구문이 임포트 문에 해당할 수 있다. 임포트 문은 실제 API 호출이 아니라 라이브러리를 참조하는 선언에 해당하므로, 컴퓨팅 장치(100)는 임포트 문을 API 호출 식별의 분석 대상에서 제외할 수 있다.

【0103】일 실시예에서, 미사용 코드 영역은 소스코드 내에 존재하나 실제 실행 흐름에서 도달하거나 실행되지 않는 코드 영역을 의미할 수 있다. 예를 들어, 조건이 항상 거짓인 분기문 내부의 코드, 반환문 이후에 위치한 코드, 또는 정의되었으나 어디에서도 호출되지 않는 함수의 코드가 미사용 코드 영역에 해당할 수 있다. 컴퓨팅 장치(100)는 미사용 코드 영역을 API 호출 식별의 분석 대상에서 제외함으로써, 실제 실행 가능한 API 호출만을 대상으로 분석을 수행할 수 있다.

【0104】일 실시예에서, 컴퓨팅 장치(100)는 검출된 메서드 호출 중, 사전 정의된 API 목록에 포함된 클래스 또는 함수에 대응되는 호출을 식별함으로써 API 호출을 선별할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 검출된 메서드 호출 각각의 완전한 식별자를 사전 정의된 API 목록과 대조하여, API 목록에 포함된 클래스 또는 함수에 대응되는 메서드 호출만을 API 호출로 선별할 수 있다.

【0105】일 실시예에서, 컴퓨팅 장치(100)는 파이썬 소스코드를 분석 대상 프로젝트의 파일 단위로 입력받을 수 있다. 구체적인 일례로, 컴퓨팅 장치(100)는 분석 대상 프로젝트 내의 파이썬 소스코드 파일을 순차적으로 입력받아 분석을 수

행할 수 있다.

【0106】 다른 일 실시예에서, 컴퓨팅 장치(100)는 파이썬 소스코드를 외부 저장소 또는 네트워크부(150)를 통해 수신하는 방식으로 입력받을 수도 있다. 구체적인 일례로, 컴퓨팅 장치(100)는 깃허브(GitHub)와 같은 공개 소스코드 저장소로부터 파이썬 소스코드를 수신하여 분석을 수행할 수 있다.

【0107】 일 실시예에서, 사전 정의된 API 목록은 분석 대상 응용 프로그램 인터페이스의 식별에 활용하기 위해 사전에 정의된 클래스 및 함수의 집합을 의미할 수 있다. 사전 정의된 API 목록은 pycryptodome, PyNaCl, cryptography 등의 암호화 라이브러리에서 수집된 복수의 암호화 프리미티브에 해당하는 클래스 및 함수를 포함할 수 있다. 구체적인 일례로, 사전 정의된 API 목록은 각 라이브러리의 공식 문서, 사용 예시 및 개발자 가이드의 분석을 통해 수집된 예를 들어 204개와 같은 소정 개수의 암호화 프리미티브를 포함할 수 있다.

【0108】 일 실시예에서, 메서드 호출은 객체 또는 클래스에 정의된 메서드를 실행하는 코드 구문을 의미할 수 있다. 예를 들어, cipher.encrypt(plain_text)는 cipher 객체에 정의된 encrypt 메서드를 호출하는 메서드 호출에 해당할 수 있다.

【0109】 일 실시예에서, 클래스는 객체의 속성과 메서드를 정의하는 코드 구조를 의미할 수 있다. 예를 들어, 암호화 라이브러리에서 AES는 대칭 암호화 연산을 수행하는 객체를 생성하기 위한 클래스에 해당할 수 있으며, AES.new(key, AES.MODE_GCM)와 같이 클래스의 생성자를 호출하여 암호화 객체를 초기화할 수 있다.

【0110】일 실시예에서, 함수는 특정 동작을 수행하도록 정의된 코드 블록을 의미할 수 있다. 예를 들어, 암호화 라이브러리에서 hashlib.pbkdf2_hmac()는 패스워드 기반 키 유도 연산을 수행하는 함수에 해당할 수 있다.

【0111】일 실시예에서, API 호출은 사전 정의된 API 목록에 포함된 클래스 또는 함수에 대응되는 메서드 호출로서, 파이썬 소스코드 내에서 실제 실행 가능한 형태로 존재하는 호출을 의미할 수 있다. 예를 들어, AES.new(key, AES.MODE_ECB)는 AES 클래스의 생성자를 호출하는 API 호출에 해당하며, hashlib.sha256()은 SHA-256 해시 함수를 호출하는 API 호출에 해당할 수 있다.

【0112】컴퓨팅 장치(100)는 코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석할 수 있다(320). 예를 들어, 컴퓨팅 장치(100)는 파이썬 소스코드를 파싱(parsing)하여 코드 속성 그래프를 생성하고, 생성된 코드 속성 그래프를 이용하여 API 호출이 포함된 함수 내부의 데이터 흐름 및 호출 관계를 분석할 수 있다. 구체적인 일례로, 컴퓨팅 장치(100)는 Joern 파서(Joern parser)를 활용하여 파이썬 소스코드로부터 코드 속성 그래프를 생성할 수 있다.

【0113】일 실시예에서, 코드 속성 그래프는 소스코드의 구문적 및 의미론적 관계를 통합적으로 표현하는 코드 표현 구조를 의미할 수 있다. 코드 속성 그래프는 구문 트리, 제어 흐름 그래프(Control Flow Graph, CFG) 및 데이터 흐름 그래프(Data Flow Graph, DFG)를 하나의 통합된 그래프로 병합한 자료구조일 수 있다. 예를 들어, 코드 속성 그래프는 변수 할당, 함수 호출, 데이터 전달 등의 관계를 노

드(node)와 엣지(edge)로 표현함으로써, 소스코드 내의 데이터 흐름 및 제어 흐름을 정밀하게 추적하는 데 활용될 수 있다.

【0114】일 실시예에서, 제어 흐름 그래프는 프로그램의 실행 순서와 분기 구조를 그래프 형태로 표현한 자료구조를 의미할 수 있다. 제어 흐름 그래프에서 각 노드는 순차적으로 실행되는 코드 블록을 나타내고, 각 엣지는 코드 블록 간의 실행 흐름 전이를 나타낼 수 있다. 예를 들어, 조건문(if-else), 반복문(for, while), 함수 호출 등에 의해 발생하는 실행 경로의 분기 및 합류가 제어 흐름 그래프의 엣지로 표현될 수 있다. 컴퓨팅 장치(100)는 제어 흐름 그래프를 활용하여 특정 실행 흐름 하에서만 발생하는 API 오용 가능성을 분석할 수 있다.

【0115】일 실시예에서, 데이터 흐름 그래프는 프로그램 내에서 데이터 값이 생성되고 전달되며 사용되는 경로를 그래프 형태로 표현한 자료구조를 의미할 수 있다. 데이터 흐름 그래프에서 각 노드는 데이터를 생성하거나 사용하는 코드 요소를 나타내고, 각 엣지는 데이터 값이 한 코드 요소에서 다른 코드 요소로 전달되는 관계를 나타낼 수 있다. 예를 들어, 변수에 할당된 값이 이후 함수 호출의 인자로 전달되는 관계가 데이터 흐름 그래프의 엣지로 표현될 수 있다. 컴퓨팅 장치(100)는 데이터 흐름 그래프를 활용하여 API 호출에 전달되는 파라미터의 기원을 역방향으로 추적하거나, API 호출의 반환값이 이후 코드에서 어떻게 사용되는지를 순방향으로 추적할 수 있다.

【0116】일 실시예에서, 노드는 코드 속성 그래프에서 소스코드의 구성 요소를 표현하는 기본 단위를 의미할 수 있다. 예를 들어, 노드는 변수 선언, 함수 정

의, 메서드 호출, 인자 값, 반환문 등 소스코드의 구문적 요소 각각에 대응될 수 있다. 컴퓨팅 장치(100)는 코드 속성 그래프의 노드를 탐색함으로써 API 호출 지점, 객체 생성 지점, 파라미터 할당 지점 등을 식별할 수 있다.

【0117】일 실시예에서, 엣지는 코드 속성 그래프에서 노드 간의 관계를 표현하는 연결선을 의미할 수 있다. 엣지는 표현하는 관계의 유형에 따라 구문적 관계를 나타내는 구문 트리 엣지, 실행 흐름 관계를 나타내는 제어 흐름 엣지, 및 데이터 전달 관계를 나타내는 데이터 흐름 엣지로 구분될 수 있다. 예를 들어, 데이터 흐름 엣지는 변수에 할당된 값이 이후 API 호출의 인자로 전달되는 관계를 표현할 수 있으며, 컴퓨팅 장치(100)는 데이터 흐름 엣지를 역방향으로 추적하여 API 호출에 사용되는 객체의 생성 맥락을 파악할 수 있다.

【0118】일 실시예에서, 함수 내부의 정적 의존성은 API 호출이 포함된 단일 함수의 경계 내에서 데이터 흐름 및 호출 관계를 분석하여 추출되는 의존성 정보를 의미할 수 있다. 함수 내부의 정적 의존성은 API 호출에 사용되는 객체(object)의 생성 맥락 정보, API 호출의 반환값 사용 정보, 및/또는 함수 내부에서 식별된 호출자-피호출자(caller-callee) 관계 중 적어도 하나를 포함할 수 있다.

【0119】일 실시예에서, 객체는 클래스의 생성자 호출을 통해 생성된 인스턴스(instance)를 의미할 수 있다. 객체는 클래스에 정의된 속성과 메서드를 보유하며, 메서드 호출을 통해 특정 동작을 수행할 수 있다. 예를 들어, cipher = AES.new(key, AES.MODE_GCM)와 같이 AES 클래스의 생성자를 호출하여 생성된 cipher가 객체에 해당하며, cipher.encrypt(plain_text)와 같이 객체에 정의된 메

서드를 호출함으로써 암호화 연산을 수행할 수 있다. 파이썬의 동적 특성으로 인해 동일한 변수명의 객체가 실행 시점에 따라 서로 다른 클래스의 인스턴스일 수 있으므로, 객체의 타입과 생성 경로를 정적으로 파악하기 위해서는 역방향 데이터 흐름 추적이 요구될 수 있다.

【0120】일 실시예에서, 생성 맥락 정보는 API 호출에 사용되는 객체가 어떤 생성자 호출을 통해 초기화되었으며, 초기화 시 어떤 파라미터가 전달되었는지에 관한 정보를 의미할 수 있다. 예를 들어, cipher 객체의 생성 맥락 정보는 AES.new()라는 생성자 호출, 및 해당 생성자에 전달된 키 값과 운용 모드(예를 들어, AES.MODE_ECB)를 포함할 수 있다. 컴퓨팅 장치(100)는 생성 맥락 정보를 추출함으로써, 암호화 객체의 초기화 방식과 보안 설정 정보를 파악하고 API 오용 여부를 판별하는 데 활용할 수 있다.

【0121】일 실시예에서, 반환값 사용 정보는 API 호출의 결과로 반환된 값이 이후 코드에서 어떻게 사용되는지에 관한 정보를 의미할 수 있다. 컴퓨팅 장치(100)는 코드 속성 그래프 상에서 순방향 데이터 흐름을 따라, API 호출의 반환값이 반환 구문에 사용되거나, 변수에 저장되거나, 또는 다른 함수의 인자로 전달되는 직접적인 사용 경로를 추적함으로써 반환값 사용 정보를 추출할 수 있다. 예를 들어, cipher.encrypt(plain_text)의 반환값이 추가적인 보호 조치 없이 즉시 반환 구문에 사용되는 경우, 컴퓨팅 장치(100)는 이를 반환값 사용 정보로 기록할 수 있다.

【0122】일 실시예에서, 호출자-피호출자 관계는 함수 호출 관계에서 호출을

수행하는 함수와 호출되는 함수 사이의 관계를 의미할 수 있다. 호출자(caller)는 다른 함수를 호출하는 함수를 의미하고, 피호출자(callee)는 호출되는 함수를 의미할 수 있다. 예를 들어, handle_request() 함수가 encrypt_data() 함수를 호출하는 경우, handle_request()가 호출자, encrypt_data()가 피호출자에 해당할 수 있다. 컴퓨팅 장치(100)는 API 호출이 포함된 함수와 해당 함수를 호출하는 상위 함수 사이의 호출자-피호출자 관계를 식별함으로써, 이후 함수 간 정적 의존성 분석의 대상이 되는 함수들을 결정할 수 있다.

【0123】일 실시예에서, 컴퓨팅 장치(100)는 코드 속성 그래프 상에서 데이터 흐름 엣지를 역방향으로 추적하여, API 호출에 사용되는 객체의 생성 맥락을 파악하기 위한 생성자 호출 및 생성자에 전달된 파라미터를 추적할 수 있다. 예를 들어, cipher.encrypt(plain_text)와 같은 API 호출이 존재하는 경우, 컴퓨팅 장치(100)는 코드 속성 그래프에서 cipher 변수에 해당하는 노드를 식별하고, 데이터 흐름 엣지를 역방향으로 따라가며 cipher 객체가 AES.new(key, AES.MODE_ECB)와 같은 생성자 호출을 통해 초기화되었음을 파악할 수 있다. 컴퓨팅 장치(100)는 해당 생성자 호출에 전달된 key 및 AES.MODE_ECB와 같은 파라미터를 추출함으로써, 암호화 객체의 초기화 방식과 보안 설정 정보를 식별할 수 있다.

【0124】일 실시예에서, 데이터 흐름 엣지는 코드 속성 그래프에서 데이터 값이 한 노드에서 다른 노드로 전달되는 관계를 나타내는 엣지를 의미할 수 있다. 예를 들어, 변수 할당 구문에서 우변의 값이 좌변의 변수로 전달되는 관계, 함수 호출 시 인자 값이 파라미터로 전달되는 관계, 또는 함수의 반환값이 호출 지점의

변수로 전달되는 관계가 데이터 흐름 엣지로 표현될 수 있다. 컴퓨팅 장치(100)는 데이터 흐름 엣지를 역방향으로 추적함으로써 특정 변수 또는 값의 기원을 소급하여 파악할 수 있다.

【0125】일 실시예에서, API 호출에 사용되는 객체는 객체 지향 방식으로 호출되는 API에서 메서드 호출의 대상이 되는 수신 객체(receiver object)를 의미할 수 있다. 예를 들어, cipher.encrypt(plain_text)에서 cipher가 API 호출에 사용되는 객체에 해당하며, hashlib.sha256()에서 hashlib 모듈이 API 호출에 사용되는 객체에 해당할 수 있다. API 호출에 사용되는 객체의 타입 및 초기화 방식에 따라, 동일한 메서드 이름이 서로 다른 암호화 연산을 수행할 수 있다.

【0126】일 실시예에서, 생성 맥락은 API 호출에 사용되는 객체가 어떤 생성자 호출을 통해 초기화되었으며, 초기화 시 어떤 파라미터가 전달되었는지에 관한 정보를 포괄적으로 의미할 수 있다. 예를 들어, cipher 객체의 생성 맥락은 AES.new()라는 생성자 호출, 및 해당 생성자에 전달된 키 값과 운용 모드(예를 들어, AES.MODE_ECB)를 포함할 수 있다. 생성 맥락을 파악함으로써, 컴퓨팅 장치(100)는 API 호출의 보안 설정이 적절한지 여부를 판별하는 데 필요한 정보를 확보할 수 있다.

【0127】일 실시예에서, 생성자 호출은 클래스의 인스턴스를 생성하고 초기화하기 위한 호출을 의미할 수 있다. 예를 들어, AES.new(key, AES.MODE_GCM)는 AES 클래스의 생성자를 호출하여 암호화 객체를 생성하는 생성자 호출에 해당할 수 있다. 컴퓨팅 장치(100)는 API 호출에 사용되는 객체의 생성자 호출을 역방향 데이

터 흐름 추적을 통해 식별함으로써, 객체의 초기화 방식과 보안 설정 정보를 파악할 수 있다.

【0128】 일 실시예에서, 생성자에 전달된 파라미터는 생성자 호출 시 인자로 전달된 값을 의미할 수 있다. 예를 들어, AES.new(key, AES.MODE_ECB)에서 key 및 AES.MODE_ECB가 생성자에 전달된 파라미터에 해당하며, 각각 암호화 키와 운용 모드를 나타낼 수 있다. 컴퓨팅 장치(100)는 생성자에 전달된 파라미터를 추출함으로써, 암호화 키의 길이 및 안전성, 운용 모드의 적절성 등 보안 관련 설정 정보를 분석할 수 있다.

【0129】 일 실시예에서, 컴퓨팅 장치(100)는 코드 속성 그래프 상에서 순방향 데이터 흐름을 따라, API 호출의 반환값이 반환 구문(return statement)에 사용되거나, 변수(variable)에 저장되거나, 또는 다른 함수의 인자(argument)로 전달되는 사용 경로를 추적할 수 있다. 예를 들어, cipher.encrypt(plain_text)의 반환값이 encrypted_data 변수에 저장된 후 함수의 반환 구문에 즉시 사용되는 경우, 컴퓨팅 장치(100)는 코드 속성 그래프에서 cipher.encrypt(plain_text) 호출 노드로부터 순방향 데이터 흐름 엣지를 따라 encrypted_data 변수의 할당 노드를 거쳐 반환 구문 노드에 이르는 사용 경로를 추적할 수 있다. 컴퓨팅 장치(100)는 반환값의 전이적(transitive) 전파 경로 전체를 추적하지 않고, 직접적인 사용에 해당하는 단계까지만 추적함으로써 과도한 근사화(over-approximation)를 방지할 수 있다.

【0130】 일 실시예에서, 반환 구문은 함수의 실행을 종료하고 호출자에게 값을 반환하는 코드 구문을 의미할 수 있다. 예를 들어, return encrypted_data와 같

이 API 호출의 반환값이 추가적인 처리 없이 즉시 반환 구문에 사용되는 경우, 암호화 출력값이 무결성 검증 등의 추가적인 보호 조치 없이 외부에 노출될 가능성이 있다. 컴퓨팅 장치(100)는 API 호출의 반환값이 반환 구문에 사용되는 경로를 추적함으로써, 이러한 보안 위험 가능성을 식별할 수 있다.

【0131】일 실시예에서, 변수는 데이터 값을 저장하기 위한 이름이 부여된 저장 공간을 의미할 수 있다. 예를 들어, encrypted_data = cipher.encrypt(plain_text)에서 encrypted_data가 API 호출의 반환값을 저장하는 변수에 해당할 수 있다. 컴퓨팅 장치(100)는 API 호출의 반환값이 저장되는 변수를 식별하고, 해당 변수가 이후 코드에서 어떻게 사용되는지를 순방향으로 추적함으로써 반환값 사용 정보를 추출할 수 있다.

【0132】일 실시예에서, 인자는 함수 또는 메서드를 호출할 때 전달되는 값을 의미할 수 있다. 예를 들어, cipher.encrypt(plain_text)에서 plain_text가 encrypt 메서드에 전달되는 인자에 해당할 수 있으며, AES.new(key, AES.MODE_ECB)에서 key 및 AES.MODE_ECB가 AES 생성자에 전달되는 인자에 해당할 수 있다. 컴퓨팅 장치(100)는 API 호출의 반환값이 다른 함수의 인자로 전달되는 경로를 추적함으로써, 반환값이 후속 처리에 활용되는 방식을 파악할 수 있다.

【0133】일 실시예에서, 컴퓨팅 장치(100)는 함수 간 정적 의존성 분석의 대상이 되는 함수들을 결정하기 위하여, API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자(caller-callee) 관계를 식별할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 API 호출이 포함된 함수의 본문(body)을 탐색하여 해당

함수를 호출하는 상위 함수를 식별하고, 식별된 상위 함수를 호출자로, API 호출이 포함된 함수를 피호출자로 기록할 수 있다. 구체적인 일례로, encrypt_data() 함수 내에 cipher.encrypt()와 같은 API 호출이 포함되어 있고, handle_request() 함수가 encrypt_data() 함수를 호출하는 경우, 컴퓨팅 장치(100)는 handle_request()를 호출자로, encrypt_data()를 피호출자로 식별할 수 있다. 이렇게 식별된 호출자-피호출자 관계는 이후 함수 간 정적 의존성 분석 단계에서 파라미터 전파 분석, 상수 분석, 및 호출 체인 분석의 대상이 되는 함수들을 결정하는 데 활용될 수 있다.

【0134】 컴퓨팅 장치(100)는 구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 API 호출을 호출하거나 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석할 수 있다(330). 예를 들어, 컴퓨팅 장치(100)는 함수 내부의 정적 의존성 분석 단계(320)에서 식별된 호출자-피호출자 관계에 기초하여 분석 대상 함수들을 결정하고, 결정된 복수의 함수들 각각에 대한 구문 트리를 이용하여 파라미터 전파 분석, 상수 분석, 및 호출 체인 분석을 수행할 수 있다.

【0135】 일 실시예에서, 함수 간 정적 의존성은 API 호출과 관련된 복수의 함수들에 걸쳐 파라미터의 전달 경로, 상수 사용 여부, 및 함수 호출 체인을 분석하여 추출되는 의존성 정보를 의미할 수 있다. 함수 간 정적 의존성은 파라미터 전파 분석을 통해 추출된 파라미터 전달 경로, 상수 분석을 통해 추출된 상수 사용 여부, 및 호출 체인 분석을 통해 재구성된 호출 체인을 포함할 수 있다.

【0136】 일 실시예에서, 컴퓨팅 장치(100)는 함수 간 정적 의존성을 분석하

기 위해, 파라미터 전파를 분석할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 구문 트리를 역방향으로 추적하여, API 호출에 사용되는 파라미터의 생성 또는 유입 경로를 추적할 수 있다. 구체적인 일례로, derive_key() 함수에서 hashlib.pbkdf2_hmac()에 전달되는 반복 횟수(iterations) 파라미터가 get_iterations()라는 별도의 함수로부터 반환된 값인 경우, 컴퓨팅 장치(100)는 구문 트리에서 iterations 변수에 해당하는 노드를 식별하고, 해당 노드로부터 구문 트리를 역방향으로 추적하여 iterations 변수의 값이 get_iterations() 함수 호출의 반환값으로부터 유입되었음을 파악할 수 있다. 컴퓨팅 장치(100)는 호출자-피호출자 관계를 이용하여 복수의 함수 호출 계층에 걸친 파라미터 전달 경로를 재구성함으로써, 보안 파라미터가 외부 입력, 전역 변수, 또는 상수 등 어디에서 유래하는지를 판별할 수 있다.

【0137】일 실시예에서, 컴퓨팅 장치(100)는 함수 간 정적 의존성을 분석하기 위해, 상수를 분석할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 복수의 함수들 각각에 대한 구문 트리를 따라 노드를 탐색하여, API 호출의 인자로 사용되는 리터럴(literal) 값 또는 상수로 정의된 값을 식별할 수 있다. 구체적인 일례로, derive_key() 함수에서 hashlib.pbkdf2_hmac()에 전달되는 솔트(salt) 인자가 SALT = b'12345678'과 같이 전역 변수로 정의된 상수로부터 유래하는 경우, 컴퓨팅 장치(100)는 구문 트리의 노드를 탐색하여 해당 인자의 값이 고정된 상수임을 식별할 수 있다.

【0138】그리고, 컴퓨팅 장치(100)는 식별된 값이 API 호출의 동작 또는 보

안 관련 파라미터에 미치는 영향을 분석할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 식별된 리터럴 값 또는 상수로 정의된 값이 암호화 키, 초기화 벡터(IV), 솔트(salt), 반복 횟수 등의 보안 관련 파라미터로 사용되는지 여부를 판별하고, 해당 값이 고정되거나 예측 가능한 경우 API 오용에 해당할 수 있음을 분석 결과로 기록할 수 있다. 구체적인 일례로, AES.new(b'hardcodedkey1234', AES.MODE_ECB)와 같이 암호화 키가 리터럴 값으로 직접 전달되는 경우, 컴퓨팅 장치(100)는 해당 리터럴 값이 키 관리 보안 규칙에 위반될 수 있음을 식별할 수 있다.

【0139】일 실시예에서, 솔트는 패스워드 기반 암호화(Password-Based Encryption, PBE) 또는 키 유도 함수(Key Derivation Function, KDF)에서 동일한 패스워드로부터 매번 다른 암호화 키가 생성되도록 하기 위해 추가되는 임의의 값을 의미할 수 있다. 솔트는 동일한 패스워드에 대해 항상 동일한 키가 생성되는 것을 방지함으로써, 사전 공격(dictionary attack) 및 레인보우 테이블 공격(rainbow table attack)에 대한 보안성을 높이는 역할을 할 수 있다. 예를 들어, hashlib.pbkdf2_hmac()에서 솔트는 키 유도 연산에 입력되는 임의의 바이트 값으로, os.urandom(16)과 같이 매 호출 시마다 새롭게 생성된 예측 불가능한 값이 사용되어야 한다. 반면, SALT = b'12345678'과 같이 고정된 리터럴 값 또는 상수로 정의된 값이 솔트로 사용되는 경우, 동일한 패스워드로부터 항상 동일한 키가 생성되어 보안 취약점이 발생할 수 있으며, 이는 API 오용에 해당할 수 있다.

【0140】일 실시예에서, 리터럴 값은 소스코드 내에 직접 기재된 고정된 값을 의미할 수 있다. 예를 들어, b'hardcodedkey1234'와 같은 바이트 문자열, 1000

과 같은 정수, 또는 'sha256'과 같은 문자열이 리터럴 값에 해당할 수 있다. 리터럴 값은 실행 시점과 무관하게 항상 동일한 값을 가지므로, 암호화 키, 초기화 벡터, 솔트 등의 보안 관련 파라미터로 사용되는 경우 예측 가능성으로 인한 보안 취약점을 유발할 수 있다.

【0141】일 실시예에서, 상수로 정의된 값은 소스코드 내에서 변경되지 않는 값을 가지도록 정의된 변수 또는 식별자를 의미할 수 있다. 예를 들어, SALT = b'12345678'과 같이 모듈 수준에서 정의된 전역 변수, 또는 클래스 내에서 정의된 클래스 변수가 상수로 정의된 값에 해당할 수 있다. 상수로 정의된 값은 리터럴 값과 달리 변수명을 통해 참조되므로, 컴퓨팅 장치(100)는 구문 트리를 역방향으로 추적하여 해당 변수의 정의 지점을 확인함으로써 상수 여부를 판별할 수 있다.

【0142】일 실시예에서, 보안 관련 파라미터는 API 호출의 보안 수준을 결정하는 인자를 의미할 수 있다. 예를 들어, 보안 관련 파라미터는 암호화 키(encryption key), 초기화 벡터(Initialization Vector, IV), 솔트, 패스워드 기반 암호화(Password-Based Encryption, PBE)의 반복 횟수(iteration count), 암호화 알고리즘의 종류, 및/또는 블록 암호 운용 모드 등을 포함할 수 있다. 보안 관련 파라미터가 고정된 리터럴 값 또는 상수로 정의된 값으로 설정되는 경우, 해당 API 호출은 오용에 해당할 수 있다.

【0143】일 실시예에서, 컴퓨팅 장치(100)는 함수 간 정적 의존성을 분석하기 위해, 호출 체인을 분석할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자 관계에 기초하

여 함수 호출 그래프를 구성할 수 있다. 구체적인 일례로, 함수 내부의 정적 의존성 분석 단계(320)에서 추출된 호출자-피호출자 쌍들을 노드와 엣지로 표현하여 함수 호출 그래프를 구성할 수 있다. 예를 들어, handle_request() 함수가 derive_key() 함수를 호출하고, derive_key() 함수가 hashlib.pbkdf2_hmac()와 같은 API 호출을 포함하는 경우, 컴퓨팅 장치(100)는 handle_request()와 derive_key() 각각을 노드로, 두 함수 간의 호출 관계를 엣지로 표현하는 함수 호출 그래프를 구성할 수 있다.

【0144】그리고, 컴퓨팅 장치(100)는 함수 호출 그래프 상에서 API 호출을 종점으로 하는 호출 경로를 따라 상위 함수 방향으로 추적하여 하나 이상의 함수 호출 경로를 식별함으로써, 특정 실행 흐름 하에서 발생되는 API 오용 가능성을 분석할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 함수 호출 그래프에서 API 호출이 포함된 함수를 종점으로 설정하고, 함수 호출 그래프를 재귀적으로 탐색하여 API 호출에 도달하는 하나 이상의 함수 호출 경로를 열거할 수 있다. 구체적인 일례로, handle_request() -> derive_key() -> hashlib.pbkdf2_hmac()와 같은 호출 체인이 재구성되는 경우, 컴퓨팅 장치(100)는 handle_request() 함수가 특정 조건 하에서만 derive_key()를 호출하는 구조임을 파악하고, 해당 실행 흐름에서만 발생할 수 있는 API 오용 가능성을 분석할 수 있다. 이를 통해, 컴퓨팅 장치(100)는 단일 함수 분석만으로는 포착하기 어려운 조건부 오용 패턴을 식별할 수 있다.

【0145】컴퓨팅 장치(100)는 함수 내부의 정적 의존성을 분석한 결과 및 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조

화된 분석 컨텍스트를 생성할 수 있다(340). 예를 들어, 컴퓨팅 장치(100)는 함수 내부의 정적 의존성 분석 단계(320) 및 함수 간 정적 의존성 분석 단계(330)에서 추출된 의존성 정보들을 API 오용 판별에 활용 가능한 형태로 통합하여, 구조화된 분석 컨텍스트를 생성할 수 있다.

【0146】일 실시예에서, 구조화된 분석 컨텍스트는 복수의 함수에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하여 API 오용 판별에 활용 가능한 형태로 정형화한 의존성 정보의 집합을 의미할 수 있다. 구조화된 분석 컨텍스트는 함수 내부의 정적 의존성을 분석한 결과로부터 추출된 객체의 생성 맥락 정보, 반환값 사용 정보, 및 호출자-피호출자 관계와, 함수 간 정적 의존성을 분석한 결과로부터 추출된 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 포함할 수 있다. 예를 들어, hashlib.pbkdf2_hmac() API 호출에 대한 구조화된 분석 컨텍스트는 pbkdf2_hmac() 함수의 호출 정보, 해당 API 호출에 전달된 솔트 파라미터가 SALT = b'12345678'과 같은 전역 상수로부터 유래한다는 파라미터 전달 경로, 반복 횟수(iterations) 파라미터가 1000과 같은 리터럴 값으로 직접 전달된다는 상수 사용 여부, 및 handle_request() -> derive_key()->hashlib.pbkdf2_hmac()와 같은 호출 체인을 포함할 수 있다. 구조화된 분석 컨텍스트는 규칙 기반 분석, 자동화된 판별 시스템, 및/또는 인공지능 모델(예를 들어, 대규모 언어 모델(Large Language Model, LLM)) 등 외부 분석 도구에서 API 오용 여부를 판단하기 위한 입력으로 활용될 수 있다. 구조화된 분석 컨텍스트는 암호화 API를 대표적인 적용 예로 하되 인자 값의 흐름 및 함수 간 의존성 분석이 중요한 다른 유형의 API 오용 탐지에도

동일하게 적용될 수 있다.

【0147】일 실시예에서, 컴퓨팅 장치(100)는 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하기 위하여, 함수 내부의 정적 의존성을 분석한 결과로부터 추출된 객체의 생성 맥락 정보, 반환값 사용 정보, 및 호출자-피호출자 관계와, 함수 간 정적 의존성을 분석한 결과로부터 추출된 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 상호 관계에 기초하여 통합함으로써, 구조화된 분석 컨텍스트를 구성할 수 있다. 즉, 컴퓨팅 장치(100)는 생성 맥락 정보, 반환값 사용 정보, 호출자-피호출자 관계, 파라미터 전달 경로, 상수 사용 여부 및/또는 호출 체인을 상호 관계에 기초하여 통합함으로써, 구조화된 분석 컨텍스트를 구성할 수 있다.

【0148】예를 들어, handle_request() -> derive_key() -> hashlib.pbkdf2_hmac()와 같은 호출 체인이 존재하는 경우, 컴퓨팅 장치(100)는 pbkdf2_hmac() API 호출에 대하여 파라미터 전달 경로(password 파라미터가 handle_request()의 지역 변수로부터 유입됨), 상수 사용 여부(SALT가 전역 상수로 정의된 고정값임), 및 호출 체인(handle_request() -> derive_key() -> pbkdf2_hmac())을 통합하고, 함수 내부 분석을 통해 추출된 생성 맥락 정보 및 반환값 사용 정보와 결합하여, 하나의 구조화된 분석 컨텍스트로 구성할 수 있다. 이를 통해, 단일 함수 분석만으로는 복원하기 어려운, 복수의 함수들에 걸쳐 분산된 API 사용 맥락이 단일 분석 단위로 복원될 수 있다.

【0149】일 실시예에서, 파라미터 전달 경로는 API 호출에 사용되는 파라미

터가 생성된 지점으로부터 API 호출 지점에 이르기까지 함수 간에 전달되는 경로를 의미할 수 있다. 파라미터 전달 경로는 파라미터의 기원(예를 들어, 외부 입력, 전역 변수, 또는 상수)과 함수 호출 계층을 거치며 전달되는 과정을 포함할 수 있다. 예를 들어, handle_request() 함수의 지역 변수로 정의된 password 값이 derive_key() 함수의 인자로 전달되고, derive_key() 함수 내에서 hashlib.pbkdf2_hmac()의 인자로 전달되는 경로가 파라미터 전달 경로에 해당할 수 있다.

【0150】일 실시예에서, 호출 체인(call chain)은 API 호출에 이르기까지 함수들이 순차적으로 호출되는 전체 실행 경로를 의미할 수 있다. 호출 체인은 최상위 진입 함수로부터 API 호출이 포함된 함수에 이르기까지의 호출자-피호출자 관계의 연속으로 표현될 수 있다. 예를 들어, handle_request() -> derive_key() -> hashlib.pbkdf2_hmac()와 같이 handle_request() 함수가 derive_key() 함수를 호출하고, derive_key() 함수가 hashlib.pbkdf2_hmac() API를 호출하는 일련의 호출 순서가 호출 체인에 해당할 수 있다. 컴퓨팅 장치(100)는 호출 체인을 재구성함으로써, API 호출이 어떤 실행 경로를 통해 호출되는지를 파악하고 특정 실행 흐름 하에서만 발생하는 API 오용 가능성을 분석할 수 있다.

【0151】일 실시예에서, 컴퓨팅 장치(100)는 구조화된 분석 컨텍스트를 생성하는 단계(340) 이후, 분석 대상 코드, API 오용에 관한 보안 규칙, 및 복수의 함수들에 걸쳐 분산된 API 사용 맥락이 복원된 구조화된 분석 컨텍스트를 포함하는 프롬프트를 구성할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 분석 대상 코드, API

오용에 관한 보안 규칙, 및 구조화된 분석 컨텍스트를 하나의 프롬프트로 결합하되, 사전 학습된 인공지능 모델(예를 들어, 대규모 언어 모델 등)이 보안 규칙을 먼저 검토한 후 분석 대상 코드와 구조화된 분석 컨텍스트를 순차적으로 검토하여 API 오용 여부를 단계적으로 추론할 수 있도록 프롬프트를 구성할 수 있다.

【0152】 그리고, 컴퓨팅 장치(100)는 프롬프트를 사전 학습된 인공지능 모델(예를 들어, 대규모 언어 모델 등)에 입력하여 API 오용 여부를 판별할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 구성된 프롬프트를 사전 학습된 인공지능 모델에 입력하고, 사전 학습된 인공지능 모델로부터 출력된 결과에 기초하여 분석 대상 코드에 API 오용이 존재하는지 여부를 판별할 수 있다. 컴퓨팅 장치(100)는 판별의 신뢰성을 높이기 위해 동일한 프롬프트를 복수 회 입력하고, 다수결 방식에 따라 최종 판별 결과를 결정할 수 있다.

【0153】 일 실시예에서, 분석 대상 코드는 API 오용 여부를 판별하고자 하는 파이썬 소스코드의 특정 코드 영역을 의미할 수 있다. 분석 대상 코드는 API 호출이 포함된 함수 및 함수 간 정적 의존성 분석의 대상이 된 복수의 함수들을 포함할 수 있다. 예를 들어, hashlib.pbkdf2_hmac() API 호출이 포함된 derive_key() 함수 및 해당 함수를 호출하는 handle_request() 함수의 소스코드가 분석 대상 코드에 해당할 수 있다.

【0154】 일 실시예에서, API 오용에 관한 보안 규칙은 API의 올바른 사용 방법과 오용 조건을 정의한 규칙의 집합을 의미할 수 있다. API 오용에 관한 보안 규칙은 OWASP(Open Web Application Security Project), NIST(National Institute of

Standards and Technology), PKCS(Public-Key Cryptography Standards) 등의 보안 표준에 기반하여 구성될 수 있다. 예를 들어, API 오용에 관한 보안 규칙은 안전한 대칭 암호화 알고리즘의 사용, 충분한 비대칭 암호화 키 크기의 사용, 취약한 해시 함수의 사용 금지, 안전하지 않은 블록 암호 운용 모드의 사용 금지, 하드코딩된 키의 사용 금지, 암호학적으로 안전한 난수 생성기의 사용, 예측 가능한 시드(seed) 사용 금지, 초기화 벡터의 무작위화, 솔트의 무작위화, 패스워드 기반 암호화에서의 충분한 반복 횟수 설정(예를 들어, 소정 횟수 이상 등), 및 인증된 암호화 모드의 사용 등을 포함할 수 있다.

【0155】일 실시예에서, 인공지능 모델은 대규모 데이터 학습을 통해 패턴을 인식하고 추론을 수행하는 컴퓨팅 알고리즘 구조를 의미할 수 있다. 예를 들어, 인공지능 모델은 대규모 언어 모델(Large Language Model, LLM)을 포함할 수 있다. 대규모 언어 모델은 방대한 양의 텍스트 데이터를 이용하여 사전 학습된 인공지능 기반의 자연어 처리 모델을 의미할 수 있다. 대규모 언어 모델은 입력된 프롬프트에 포함된 코드, 보안 규칙, 및 의존성 정보를 종합적으로 검토하여 API 오용 여부에 관한 추론 결과를 출력할 수 있다. 예를 들어, 대규모 언어 모델은 구조화된 분석 컨텍스트를 통해 복원된 API 사용 맥락을 바탕으로, 단순한 구문 분석만으로는 판별하기 어려운 의미론적 오용 패턴을 식별할 수 있다. 인공지능 모델의 구체적인 구조 및 연산 과정에 대해서는 앞서 도 2를 참조하여 설명된 인공지능 모델에 관한 설명이 참조될 수 있다.

【0156】일 실시예에서, 프롬프트는 사전 학습된 인공지능 모델(예를 들어,

대규모 언어 모델 등)에 입력되는 텍스트 형식의 질의 또는 지시문을 의미할 수 있다. 프롬프트는 사전 학습된 인공지능 모델(예를 들어, 대규모 언어 모델 등)이 보안 규칙, 분석 대상 코드, 및 구조화된 분석 컨텍스트를 순차적으로 검토하여 API 오용 여부를 단계적으로 추론하도록 유도하는 연쇄적 사고(Chain-of-Thought, CoT) 방식에 따라 구성될 수 있다. 예를 들어, 프롬프트는 보안 분석가로서의 역할 지시, API 오용에 관한 보안 규칙의 열거, 분석 대상 코드의 제시, 구조화된 분석 컨텍스트의 제시, 및 API 오용 여부를 사전 결정된 형식(예를 들어, JSON(JavaScript Object Notation) 형식 등)으로 출력하도록 하는 지시를 포함할 수 있다.

【0157】 일 실시예에서, 연쇄적 사고 방식은 인공지능 모델(예를 들어, 대규모 언어 모델 등)이 복잡한 추론 문제를 해결할 때 중간 추론 단계를 순차적으로 거쳐 최종 결론에 도달하도록 유도하는 프롬프트 설계 방식을 의미할 수 있다. 예를 들어, 연쇄적 사고 방식에 따라 구성된 프롬프트는 사전 학습된 인공지능 모델이 API 오용에 관한 보안 규칙을 먼저 검토하고, 분석 대상 코드의 구문적 구조를 파악한 후, 구조화된 분석 컨텍스트에 포함된 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 순차적으로 검토하여 API 오용 여부를 단계적으로 추론하도록 유도할 수 있다. 이를 통해, 컴퓨팅 장치(100)는 단순히 코드만을 제공하는 방식 대비 더 높은 정확도로 API 오용 여부를 판별할 수 있다.

【0158】 도 4는 본 개시의 일 실시예에 따른 응용 프로그램 인터페이스 오용 탐지 시스템의 전체 구성을 나타낸 도면이다.

【0159】도 4를 참조하면, 컴퓨팅 장치(100)는 입력(410)으로 수신된 파이썬 소스코드 프로젝트에서 API 호출을 식별하고(420), 정적 의존성 분석(430) 및 사전 학습된 인공지능 모델 기반 오용 탐지(460)를 순차적으로 수행하여 암호화 API 오용 여부를 출력(470)할 수 있다.

【0160】일 실시예에서, 파이썬 소스코드 프로젝트는 하나 이상의 파이썬 소스코드 파일로 구성된 소프트웨어 프로젝트를 의미할 수 있다. 파이썬 소스코드 프로젝트는 단일 파이썬 소스코드 파일로 구성될 수도 있고, 복수의 파이썬 소스코드 파일 및 디렉토리로 구성된 프로젝트 단위일 수도 있다. 예를 들어, 파이썬 소스코드 프로젝트는 공개 소스코드 저장소에 공개된 오픈소스 프로젝트를 포함할 수 있다.

【0161】일 실시예에서, 입력(410)은 컴퓨팅 장치(100)가 수신하는 분석 대상 파이썬 소스코드 프로젝트를 의미할 수 있다. 컴퓨팅 장치(100)는 분석 대상 프로젝트에 포함된 파이썬 소스코드 파일을 입력으로 수신하여 이후 단계들을 수행할 수 있다.

【0162】일 실시예에서, 컴퓨팅 장치(100)는 입력(410)으로 수신된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별할 수 있다(420). 컴퓨팅 장치(100)는 파이썬 소스코드로부터 메서드 호출을 검출하고, 검출된 메서드 호출 중 사전 정의된 API 목록에 포함된 클래스 또는 함수에 대응되는 호출을 API 호출로 선별할 수 있다. 식별된 API 호출은 이후 정적 의존성 분석(430)의 입력으로 활용될 수 있다.

【0163】일 실시예에서, 컴퓨팅 장치(100)는 식별된 API 호출을 중심으로 함수 내부의 정적 의존성 분석(Intra-procedural analysis) 및 함수 간 정적 의존성 분석(Inter-procedural analysis)을 수행할 수 있다(430). 컴퓨팅 장치(100)는 코드 속성 그래프에 기초하여 API 호출이 포함된 함수 내부의 객체 생성 맥락, 반환값 사용 정보, 및 호출자-피호출자 관계를 추출하고, 구문 트리에 기초하여 복수의 함수들에 걸친 파라미터 전파, 상수 사용 여부, 및 호출 체인을 분석할 수 있다. 도 4에 도시된 바와 같이, 컴퓨팅 장치(100)는 함수 내부의 정적 의존성 분석의 결과를 함수 간 정적 의존성 분석의 대상 함수를 결정하는 데 활용하며, 두 분석을 상호 연계하여 수행할 수 있다. 정적 의존성 분석(430)에 대한 보다 구체적인 설명은 앞서 도 3의 단계(320) 및 단계(330)를 참조하여 설명된 내용이 참조될 수 있다.

【0164】일 실시예에서, 컴퓨팅 장치(100)는 정적 의존성 분석(430)을 통해 의존성 정보(440)를 추출할 수 있다. 의존성 정보(440)는 컴퓨팅 장치(100)가 정적 의존성 분석(430)을 통해 생성한 구조화된 분석 컨텍스트를 의미할 수 있다. 의존성 정보(440)는 객체 생성 맥락 정보, 반환값 사용 정보, 호출자-피호출자 관계, 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인(예를 들어, call chain 등)을 포함할 수 있다. 컴퓨팅 장치(100)는 의존성 정보(440)를 이후 사전 학습된 인공지능 모델 기반 오용 탐지(460)에서 프롬프트의 구성 요소로 활용할 수 있다.

【0165】일 실시예에서, 사전 정의 정보(450)는 컴퓨팅 장치(100)가 API 오용 탐지에 활용하는 암호화 API 오용 규칙(Cryptographic API misuse rule)을 의미

할 수 있다. 컴퓨팅 장치(100)는 OWASP, NIST, PKCS 등의 보안 표준에 기반하여 사전에 정의된 보안 규칙의 집합을 사전 정의 정보(450)로 활용할 수 있다.

【0166】 일 실시예에서, 사전 정의 정보(450)는 메모리(130)에 사전 저장되어 있을 수 있다. 예를 들어, 컴퓨팅 장치(100)는 보안 표준에 기반하여 수동으로 구성된 보안 규칙의 집합을 메모리(130)에 사전 저장하고, API 오용 탐지 시 메모리(130)에 저장된 사전 정의 정보(450)를 불러와 활용할 수 있다.

【0167】 다른 일 실시예에서, 사전 정의 정보(450)는 네트워크부(150)를 통해 외부로부터 수신될 수도 있다. 예를 들어, 컴퓨팅 장치(100)는 외부 서버 또는 외부 저장소로부터 최신 보안 표준에 기반하여 갱신된 보안 규칙의 집합을 수신하여 사전 정의 정보(450)로 활용할 수 있다.

【0168】 일 실시예에서, 보안 표준은 소프트웨어 및 암호화 시스템의 보안 요구사항과 올바른 구현 방법을 정의한 공식적인 지침 또는 규격을 의미할 수 있다. 예를 들어, 보안 표준은 웹 애플리케이션 보안 취약점 및 대응 방안을 정의하는 OWASP, 암호화 키 생성 및 관리에 관한 권고사항을 제공하는 NIST, 및 공개키 암호화 표준을 정의하는 PKCS 등을 포함할 수 있다.

【0169】 일 실시예에서, 암호화 API 오용 규칙은 앞서 설명된 API 오용에 관한 보안 규칙과 동일한 의미로 사용될 수 있다. 구체적인 일례로, 암호화 API 오용 규칙은 암호화 API의 올바른 사용 방법과 오용 조건을 정의한 규칙의 집합으로서, 보안 표준에 기반하여 구성된 보안 규칙을 암호화 API의 맥락에서 특정한 것을 의미할 수 있다.

【0170】일 실시예에서, 수동으로 구성된 보안 규칙의 집합은 자동화된 방식이 아닌, 보안 전문가 또는 개발자가 보안 표준 문서, 공식 API 문서, 사용 예시 및 개발자 가이드를 직접 분석하여 정의한 규칙의 집합을 의미할 수 있다. 예를 들어, 각 암호화 라이브러리의 공식 문서 및 OWASP, NIST 등의 보안 표준을 참조하여 안전한 알고리즘의 종류, 최소 키 길이, 권장 운용 모드 등의 보안 요구사항을 규칙의 형태로 직접 정의하고 구성한 집합이 수동으로 구성된 보안 규칙의 집합에 해당할 수 있다.

【0171】일 실시예에서, 컴퓨팅 장치(100)는 사전 정의 정보(450)를 정적 의존성 분석(430) 및 사전 학습된 인공지능 모델 기반 오용 탐지(460) 모두에 활용할 수 있다. 예를 들어, 컴퓨팅 장치(100)는 사전 정의 정보(450)에 포함된 보안 규칙을 정적 의존성 분석(430)에서 분석의 기준으로 활용하고, 사전 학습된 인공지능 모델 기반 오용 탐지(460)에서는 프롬프트의 구성 요소로 활용할 수 있다.

【0172】일 실시예에서, 컴퓨팅 장치(100)는 의존성 정보(440) 및 사전 정의 정보(450)를 활용하여 프롬프트를 구성하고, 구성된 프롬프트를 사전 학습된 인공지능 모델에 입력하여 API 오용 여부를 판별할 수 있다(460). 컴퓨팅 장치(100)는 분석 대상 코드(Target code), 보안 규칙(Security rule), 및 의존성 정보(Dependency)를 포함하는 프롬프트를 구성할 수 있다. 컴퓨팅 장치(100)는 연쇄적 사고(Chain-of-Thought, CoT) 방식에 따라 프롬프트를 구성함으로써, 사전 학습된 인공지능 모델이 보안 규칙, 분석 대상 코드, 및 의존성 정보를 순차적으로 검토하여 API 오용 여부를 단계적으로 추론하도록 유도할 수 있다. 사전 학습된 인공지능

모델 기반 오용 탐지(460)에 대한 보다 구체적인 설명은 앞서 도 3의 단계(340) 이후를 참조하여 설명된 내용이 참조될 수 있다.

【0173】 일 실시예에서, 출력(470)은 컴퓨팅 장치(100)가 사전 학습된 인공지능 모델 기반 오용 탐지(460)를 통해 판별한 암호화 API 오용(Cryptographic API misuse) 결과를 의미할 수 있다. 컴퓨팅 장치(100)는 사전 학습된 인공지능 모델로부터 출력된 결과에 기초하여 분석 대상 코드에 API 오용이 존재하는지 여부를 최종적으로 판별하고, 판별된 결과를 출력(470)으로 제공할 수 있다.

【0174】 도 5는 본 개시의 일 실시예에 따른 정적 의존성 분석 단계의 세부 구성을 나타낸 도면이다.

【0175】 도 5를 참조하면, 컴퓨팅 장치(100)는 파이썬 소스코드(510)를 입력으로 수신하고, 정적 의존성 분석(520)을 통해 6가지 의존성 사실(dependency facts)을 포함하는 컨텍스트 주입(530)을 수행하여 구조화된 분석 컨텍스트(540)를 생성하고, 생성된 구조화된 분석 컨텍스트(540)를 프롬프트(550)의 구성 요소로 주입할 수 있다.

【0176】 일 실시예에서, 의존성 사실은 API 호출의 사용 맥락을 구성하는 개별 의존성 정보의 단위를 의미할 수 있다. 컴퓨팅 장치(100)는 함수 내부의 정적 의존성 분석(521) 및 함수 간 정적 의존성 분석(522)을 통해 수신 객체(Receiver object), 반환값(Return value), 호출 계층(Call hierarchy), 파라미터 전파(Parameter propagation), 상수 사용(Constant usage), 및 호출 체인(Call chain)의 6가지 의존성 사실을 추출할 수 있다. 이 중 수신 객체, 반환값, 및 호출 계층

은 함수 내부의 정적 의존성 분석(521)을 통해 추출되고, 파라미터 전파, 상수 사용, 및 호출 체인은 함수 간 정적 의존성 분석(522)을 통해 추출될 수 있다. 컴퓨팅 장치(100)는 6가지 의존성 사실을 통합하여 구조화된 분석 컨텍스트(540)를 구성함으로써, 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원할 수 있다.

【0177】일 실시예에서, 파이썬 소스코드(510)는 API 오용 여부를 분석하고자 하는 분석 대상 파이썬 소스코드를 의미할 수 있다. 파이썬 소스코드(510)에 대한 보다 구체적인 설명은 앞서 도 3 및 도 4를 참조하여 설명된 내용이 참조될 수 있다.

【0178】일 실시예에서, 정적 의존성 분석(520)은 파이썬 소스코드(510)를 대상으로 수행되는 함수 내부의 정적 의존성 분석(521) 및 함수 간 정적 의존성 분석(522)을 포함할 수 있다. 컴퓨팅 장치(100)는 코드 속성 그래프에 기초하여 함수 내부의 정적 의존성 분석(521)을 수행하고, 구문 트리에 기초하여 함수 간 정적 의존성 분석(522)을 수행할 수 있다. 정적 의존성 분석(520)에 대한 보다 구체적인 설명은 앞서 도 3의 단계(320) 및 단계(330)를 참조하여 설명된 내용이 참조될 수 있다.

【0179】일 실시예에서, 컨텍스트 주입(530)은 함수 내부의 정적 의존성 분석(521) 및 함수 간 정적 의존성 분석(522)을 통해 추출된 6가지 의존성 사실을 구조화된 분석 컨텍스트(540)로 통합하고, 이를 프롬프트(550)의 의존성 정보(553)로 주입하는 과정을 의미할 수 있다. 컴퓨팅 장치(100)는 6가지 의존성 사실을 통합함

으로써, 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하여 프롬프트(550)에 제공할 수 있다.

【0180】일 실시예에서, 구조화된 분석 컨텍스트(540)는 함수 내부의 정적 의존성 분석(521)으로부터 추출된 함수 내부의 의존성 정보(541) 및 함수 간 정적 의존성 분석(522)으로부터 추출된 함수 간 의존성 정보(542)를 포함할 수 있다.

【0181】일 실시예에서, 함수 내부의 의존성 정보(541)는 함수 내부의 정적 의존성 분석(521)으로부터 추출된 수신 객체(Receiver object), 반환값(Return value), 및 호출 계층(Call hierarchy)을 포함할 수 있다.

【0182】일 실시예에서, 함수 간 의존성 정보(542)는 함수 간 정적 의존성 분석(522)으로부터 추출된 파라미터 전파(Parameter propagation), 상수 사용(Constant usage), 및 호출 체인(Call chain)을 포함할 수 있다.

【0183】일 실시예에서, 컴퓨팅 장치(100)는 6가지 의존성 사실을 상호 관계에 기초하여 통합함으로써 구조화된 분석 컨텍스트(540)를 구성할 수 있다.

【0184】일 실시예에서, 수신 객체는 API 호출에 사용되는 객체의 생성 맥락 정보를 의미할 수 있다. 컴퓨팅 장치(100)는 코드 속성 그래프 상에서 데이터 흐름 엣지를 역방향으로 추적하여 수신 객체의 생성자 호출 및 생성자에 전달된 파라미터를 추출할 수 있다.

【0185】일 실시예에서, 반환값은 API 호출의 반환값이 반환 구문에 사용되거나, 변수에 저장되거나, 또는 다른 함수의 인자로 전달되는 사용 경로 정보를 의미할 수 있다. 컴퓨팅 장치(100)는 코드 속성 그래프 상에서 순방향 데이터 흐름을

따라 반환값의 직접적인 사용 경로를 추적할 수 있다.

【0186】일 실시예에서, 호출 계층은 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자 관계를 의미할 수 있다. 컴퓨팅 장치(100)는 API 호출이 포함된 함수의 본문을 탐색하여 호출자-피호출자 관계를 식별하고, 이를 함수 간 정적 의존성 분석의 대상 함수를 결정하는 데 활용할 수 있다.

【0187】일 실시예에서, 파라미터 전파는 API 호출에 사용되는 파라미터의 생성 또는 유입 경로를 의미할 수 있다. 컴퓨팅 장치(100)는 구문 트리를 역방향으로 추적하여 API 호출에 사용되는 파라미터의 기원을 판별할 수 있다.

【0188】일 실시예에서, 상수 사용은 API 호출의 인자로 사용되는 리터럴 값 또는 상수로 정의된 값, 및 해당 값이 API 호출의 동작 또는 보안 관련 파라미터에 미치는 영향에 관한 정보를 의미할 수 있다. 컴퓨팅 장치(100)는 복수의 함수들 각각에 대한 구문 트리를 따라 노드를 탐색하여 상수 사용 여부를 식별할 수 있다.

【0189】일 실시예에서, 호출 체인은 API 호출에 이르기까지 함수들이 순차적으로 호출되는 전체 실행 경로를 의미할 수 있다. 컴퓨팅 장치(100)는 호출자-피호출자 관계로부터 함수 호출 그래프를 구성하고, 함수 호출 그래프를 재귀적으로 탐색하여 API 호출에 이르는 하나 이상의 함수 호출 경로를 식별할 수 있다.

【0190】일 실시예에서, 프롬프트(550)는 사전 학습된 인공지능 모델에 입력되는 텍스트 형식의 질의 또는 지시문을 의미할 수 있다. 예를 들어, 프롬프트(550)는 분석 대상 코드(Target code)(551), 보안 규칙(Security rule)(552), 및

의존성 정보(Dependency)(553)를 포함할 수 있다. 컴퓨팅 장치(100)는 구조화된 분석 컨텍스트(540)를 의존성 정보(553)로서 프롬프트(550)에 주입함으로써, 사전 학습된 인공지능 모델이 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 반영하여 API 오용 여부를 추론할 수 있도록 할 수 있다. 프롬프트(550)에 대한 보다 구체적인 설명은 앞서 도 3 및 도 4를 참조하여 설명된 내용이 참조될 수 있다.

【0191】도 1 내지 도 5를 참조하여 상술한 바와 같이, 컴퓨팅 장치(100)는 파이썬 소스코드에서 사전 정의된 API 목록에 기초하여 API 호출을 식별하고, 코드 속성 그래프에 기초한 함수 내부의 정적 의존성 분석 및 구문 트리에 기초한 함수 간 정적 의존성 분석을 수행하여 6가지 의존성 사실을 추출하고, 추출된 의존성 사실을 통합한 구조화된 분석 컨텍스트를 사전 학습된 인공지능 모델에 제공함으로써, 복수의 함수들에 걸쳐 분산된 API 사용 맥락을 복원하여 API 오용 여부를 판별할 수 있다.

【0192】본 개시의 일 실시예에 따른 효과를 검증하기 위한 특정 실험 예에서, 컴퓨팅 장치(100)는 공개 벤치마크 데이터셋(PyCryptoBench)을 대상으로 암호화 API 오용 탐지 정확도를 측정하였다. 그 결과, 컴퓨팅 장치(100)는 기존 정적 분석 기반 기법 대비 예를 들어 약 95% 수준과 같이 높은 F1 점수(F1-score)를 달성하는 것을 확인하였다. 또한, 함수 간 문맥 의존성이 중요한 파라미터 및 컴포넌트 관리 오용 유형에서, 컴퓨팅 장치(100)는 예를 들어 약 91% 수준과 같이 높은 재현율(recall)을 달성하는 것을 확인하였다. 나아가, 실제 오픈소스 저장소 기반 데이터셋을 대상으로 한 실험에서도, 컴퓨팅 장치(100)는 예를 들어 약 84% 수준과

같이 높은 F1 점수를 달성하여, 복잡한 실환경 코드에 대해서도 안정적인 오용 탐지가 가능함을 확인하였다.

【0193】 다만, 이러한 수치적 결과는 특정한 모델 아키텍처 및 하드웨어 환경 하에서 도출된 일례일 뿐이며, 본 개시의 권리범위가 기재된 구체적인 수치에 의해 한정되는 것은 아니다.

【0194】 도 6은 본 개시의 실시예들이 구현될 수 있는 예시적인 컴퓨팅 환경(예를 들어, 컴퓨팅 장치(100))에 대한 간략하고 일반적인 개략도를 도시한다.

【0195】 본 개시가 일반적으로 컴퓨팅 장치에 의해 구현될 수 있는 것으로 전술되었지만, 당업자라면 본 개시가 하나 이상의 컴퓨터 상에서 실행될 수 있는 컴퓨터 실행가능 명령어 및/또는 기타 프로그램 모듈들과 결합되어 및/또는 하드웨어와 소프트웨어의 조합으로써 구현될 수 있다는 것을 잘 알 것이다.

【0196】 일반적으로, 프로그램 모듈은 특정의 태스크를 수행하거나 특정의 추상 데이터 유형을 구현하는 루틴, 프로그램, 컴포넌트, 데이터 구조, 기타 등등을 포함한다. 또한, 당업자라면 본 개시의 방법이 단일-프로세서 또는 멀티프로세서 컴퓨터 시스템, 미니컴퓨터, 메인프레임 컴퓨터는 물론 퍼스널 컴퓨터, 핸드헬드(handheld) 컴퓨팅 장치, 마이크로프로세서-기반 또는 프로그램가능 가전 제품, 기타 등등(이들 각각은 하나 이상의 연관된 장치와 연결되어 동작할 수 있음)을 비롯한 다른 컴퓨터 시스템 구성으로 실시될 수 있다는 것을 잘 알 것이다.

【0197】 본 개시의 설명된 실시예들은 또한 어떤 태스크들이 통신 네트워크를 통해 연결되어 있는 원격 처리 장치들에 의해 수행되는 분산 컴퓨팅 환경에서

실시될 수 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈은 로컬 및 원격 메모리 저장 장치 둘 다에 위치할 수 있다.

【0198】 컴퓨터는 통상적으로 다양한 컴퓨터 판독가능 매체를 포함한다. 컴퓨터에 의해 액세스 가능한 매체는 그 어떤 것이든지 컴퓨터 판독가능 매체가 될 수 있고, 이러한 컴퓨터 판독가능 매체는 휘발성 및 비휘발성 매체, 일시적(transitory) 및 비일시적(non-transitory) 매체, 이동식 및 비-이동식 매체를 포함한다. 제한이 아닌 예로서, 컴퓨터 판독가능 매체는 컴퓨터 판독가능 저장 매체 및 컴퓨터 판독가능 전송 매체를 포함할 수 있다.

【0199】 컴퓨터 판독가능 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보를 저장하는 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성 매체, 일시적 및 비-일시적 매체, 이동식 및 비이동식 매체를 포함한다. 컴퓨터 판독가능 저장 매체는 ROM, EEPROM, 플래시 메모리 또는 기타 메모리 기술, CD-ROM, DVD(digital video disk) 또는 기타 광 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 기타 자기 저장 장치, 또는 컴퓨터에 의해 액세스될 수 있고 원하는 정보를 저장하는 데 사용될 수 있는 임의의 기타 매체를 포함하지만, 이에 한정되지 않는다.

【0200】 컴퓨터 판독가능 전송 매체는 통상적으로 반송파(carrier wave) 또는 기타 전송 메커니즘(transport mechanism)과 같은 피변조 데이터 신호(modulated data signal)에 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터 등을 구현하고 모든 정보 전달 매체를 포함한다. 피변조 데이터

신호라는 용어는 신호 내에 정보를 인코딩하도록 그 신호의 특성들 중 하나 이상을 설정 또는 변경시킨 신호를 의미한다. 제한이 아닌 예로서, 컴퓨터 판독가능 전송 매체는 유선 네트워크 또는 직접 배선 접속(direct-wired connection)과 같은 유선 매체, 그리고 음향, RF, 적외선, 기타 무선 매체와 같은 무선 매체를 포함한다. 상술된 매체들 중 임의의 것의 조합도 역시 컴퓨터 판독가능 전송 매체의 범위 안에 포함되는 것으로 한다.

【0201】 컴퓨터(1102)를 포함하는 본 개시의 여러가지 측면들을 구현하는 예시적인 환경이 나타내어져 있으며, 컴퓨터(1102)는 처리 장치(1104), 시스템 메모리(1106) 및 시스템 버스(1108)를 포함한다. 시스템 버스(1108)는 시스템 메모리(1106)(이에 한정되지 않음)를 비롯한 시스템 컴포넌트들을 처리 장치(1104)에 연결시킨다. 처리 장치(1104)는 다양한 상용 프로세서들 중 임의의 프로세서일 수 있다. 듀얼 프로세서 및 기타 멀티프로세서 아키텍처도 역시 처리 장치(1104)로서 이용될 수 있다.

【0202】 시스템 버스(1108)는 메모리 버스, 주변장치 버스, 및 다양한 상용 버스 아키텍처 중 임의의 것을 사용하는 로컬 버스에 추가적으로 상호 연결될 수 있는 몇 가지 유형의 버스 구조 중 임의의 것일 수 있다. 시스템 메모리(1106)는 판독 전용 메모리(ROM)(1110) 및 랜덤 액세스 메모리(RAM)(1112)를 포함한다. 기본 입/출력 시스템(BIOS)은 ROM, EPROM, EEPROM 등의 비휘발성 메모리(1110)에 저장되며, 이 BIOS는 시동 중과 같은 때에 컴퓨터(1102) 내의 구성요소들 간에 정보를 전송하는 일을 돕는 기본적인 루틴을 포함한다. RAM(1112)은 또한 데이터를 캐싱하기

위한 정적 RAM 등의 고속 RAM을 포함할 수 있다.

【0203】 컴퓨터(1102)는 또한 내장형 하드 디스크 드라이브(HDD)(1114)(예를 들어, EIDE, SATA) -이 내장형 하드 디스크 드라이브(1114)는 또한 적당한 섀시(도시 생략) 내에서 외장형 용도로 구성될 수 있음-, 자기 플로피 디스크 드라이브(FDD)(1116)(예를 들어, 이동식 디스켓(1118)으로부터 판독을 하거나 그에 기록을 하기 위한 것임), 및 광 디스크 드라이브(1120)(예를 들어, CD-ROM 디스크(1122)를 판독하거나 DVD 등의 기타 고용량 광 매체로부터 판독을 하거나 그에 기록을 하기 위한 것임)를 포함한다. 하드 디스크 드라이브(1114), 자기 디스크 드라이브(1116) 및 광 디스크 드라이브(1120)는 각각 하드 디스크 드라이브 인터페이스(1124), 자기 디스크 드라이브 인터페이스(1126) 및 광 드라이브 인터페이스(1128)에 의해 시스템 버스(1108)에 연결될 수 있다. 외장형 드라이브 구현을 위한 인터페이스(1124)는 USB(Universal Serial Bus) 및 IEEE 1394 인터페이스 기술 중 적어도 하나 또는 그 둘 다를 포함한다.

【0204】 이들 드라이브 및 그와 연관된 컴퓨터 판독가능 매체는 데이터, 데이터 구조, 컴퓨터 실행가능 명령어, 기타 등등의 비휘발성 저장을 제공한다. 컴퓨터(1102)의 경우, 드라이브 및 매체는 임의의 데이터를 적당한 디지털 형식으로 저장하는 것에 대응한다. 상기에서의 컴퓨터 판독가능 매체에 대한 설명이 HDD, 이동식 자기 디스크, 및 CD 또는 DVD 등의 이동식 광 매체를 언급하고 있지만, 당업자라면 집 드라이브(zip drive), 자기 카세트, 플래시 메모리 카드, 카트리지, 기타 등등의 컴퓨터에 의해 판독가능한 다른 유형의 매체도 역시 예시적인 운영 환경에

서 사용될 수 있으며 또 임의의 이러한 매체가 본 개시의 방법들을 수행하기 위한 컴퓨터 실행가능 명령어를 포함할 수 있다는 것을 잘 알 것이다.

【0205】운영 체제(1130), 하나 이상의 애플리케이션 프로그램(1132), 기타 프로그램 모듈(1134) 및 프로그램 데이터(1136)를 비롯한 다수의 프로그램 모듈이 드라이브 및 RAM(1112)에 저장될 수 있다. 운영 체제, 애플리케이션, 모듈 및/또는 데이터의 전부 또는 그 일부분이 또한 RAM(1112)에 캐싱될 수 있다. 본 개시가 여러가지 상업적으로 이용가능한 운영 체제 또는 운영 체제들의 조합에서 구현될 수 있다는 것을 잘 알 것이다.

【0206】사용자는 하나 이상의 유선/무선 입력 장치, 예를 들어, 키보드(1138) 및 마우스(1140) 등의 포인팅 장치를 통해 컴퓨터(1102)에 명령 및 정보를 입력할 수 있다. 기타 입력 장치(도시 생략)로는 마이크, IR 리모콘, 조이스틱, 게임 패드, 스타일러스 펜, 터치 스크린, 기타 등등이 있을 수 있다. 이들 및 기타 입력 장치가 종종 시스템 버스(1108)에 연결되어 있는 입력 장치 인터페이스(1142)를 통해 처리 장치(1104)에 연결되지만, 병렬 포트, IEEE 1394 직렬 포트, 게임 포트, USB 포트, IR 인터페이스, 기타 등등의 기타 인터페이스에 의해 연결될 수 있다.

【0207】모니터(1144) 또는 다른 유형의 디스플레이 장치도 역시 비디오 어댑터(1146) 등의 인터페이스를 통해 시스템 버스(1108)에 연결된다. 모니터(1144)에 부가하여, 컴퓨터는 일반적으로 스피커, 프린터, 기타 등등의 기타 주변 출력 장치(도시 생략)를 포함한다.

【0208】컴퓨터(1102)는 유선 및/또는 무선 통신을 통한 원격 컴퓨터(들)(1148) 등의 하나 이상의 원격 컴퓨터로의 논리적 연결을 사용하여 네트워크화된 환경에서 동작할 수 있다. 원격 컴퓨터(들)(1148)는 워크스테이션, 컴퓨팅 디바이스 컴퓨터, 라우터, 퍼스널 컴퓨터, 휴대용 컴퓨터, 마이크로프로세서-기반 오락 기기, 피어 장치 또는 기타 통상의 네트워크 노드일 수 있으며, 일반적으로 컴퓨터(1102)에 대해 기술된 구성요소들 중 다수 또는 그 전부를 포함하지만, 간략함을 위해, 메모리 저장 장치(1150)만이 도시되어 있다. 도시되어 있는 논리적 연결은 근거리 통신망(LAN)(1152) 및/또는 더 큰 네트워크, 예를 들어, 원거리 통신망(WAN)(1154)에의 유선/무선 연결을 포함한다. 이러한 LAN 및 WAN 네트워킹 환경은 사무실 및 회사에서 일반적인 것이며, 인트라넷 등의 전사적 컴퓨터 네트워크(enterprise-wide computer network)를 용이하게 해주며, 이들 모두는 전세계 컴퓨터 네트워크, 예를 들어, 인터넷에 연결될 수 있다.

【0209】LAN 네트워킹 환경에서 사용될 때, 컴퓨터(1102)는 유선 및/또는 무선 통신 네트워크 인터페이스 또는 어댑터(1156)를 통해 로컬 네트워크(1152)에 연결된다. 어댑터(1156)는 LAN(1152)에의 유선 또는 무선 통신을 용이하게 해줄 수 있으며, 이 LAN(1152)은 또한 무선 어댑터(1156)와 통신하기 위해 그에 설치되어 있는 무선 액세스 포인트를 포함하고 있다. WAN 네트워킹 환경에서 사용될 때, 컴퓨터(1102)는 모뎀(1158)을 포함할 수 있거나, WAN(1154) 상의 통신 컴퓨팅 디바이스에 연결되거나, 또는 인터넷을 통하는 등, WAN(1154)을 통해 통신을 설정하는 기타 수단을 갖는다. 내장형 또는 외장형 및 유선 또는 무선 장치일 수 있는 모뎀

(1158)은 직렬 포트 인터페이스(1142)를 통해 시스템 버스(1108)에 연결된다. 네트워크화된 환경에서, 컴퓨터(1102)에 대해 설명된 프로그램 모듈들 또는 그의 일부분이 원격 메모리/저장 장치(1150)에 저장될 수 있다. 도시된 네트워크 연결이 예시적인 것이며 컴퓨터들 사이에 통신 링크를 설정하는 기타 수단이 사용될 수 있다는 것을 잘 알 것이다.

【0210】 컴퓨터(1102)는 무선 통신으로 배치되어 동작하는 임의의 무선 장치 또는 개체, 예를 들어, 프린터, 스캐너, 데스크톱 및/또는 휴대용 컴퓨터, PDA(portable data assistant), 통신 위성, 무선 검출가능 태그와 연관된 임의의 장비 또는 장소, 및 전화와 통신을 하는 동작을 한다. 이것은 적어도 Wi-Fi 및 블루투스 무선 기술을 포함한다. 따라서, 통신은 종래의 네트워크에서와 같이 미리 정의된 구조이거나 단순하게 적어도 2개의 장치 사이의 애드혹 통신(ad hoc communication)일 수 있다.

【0211】 Wi-Fi(Wireless Fidelity)는 유선 없이도 인터넷 등으로의 연결을 가능하게 해준다. Wi-Fi는 이러한 장치, 예를 들어, 컴퓨터가 실내에서 및 실외에서, 즉 기지국의 통화권 내의 아무 곳에서나 데이터를 전송 및 수신할 수 있게 해주는 셀 전화와 같은 무선 기술이다. Wi-Fi 네트워크는 안전하고 신뢰성 있으며 고속인 무선 연결을 제공하기 위해 IEEE 802.11(a, b, g, 기타)이라고 하는 무선 기술을 사용한다. 컴퓨터를 서로에, 인터넷에 및 유선 네트워크(IEEE 802.3 또는 이더넷을 사용함)에 연결시키기 위해 Wi-Fi가 사용될 수 있다. Wi-Fi 네트워크는 비인가 2.4 및 5GHz 무선 대역에서, 예를 들어, 11Mbps(802.11a) 또는 54

Mbps(802.11b) 데이터 레이트로 동작하거나, 양 대역(듀얼 대역)을 포함하는 제품에서 동작할 수 있다.

【0212】본 개시의 기술 분야에서 통상의 지식을 가진 자는 정보 및 신호들이 임의의 다양한 상이한 기술들 및 기법들을 이용하여 표현될 수 있다는 것을 이해할 것이다. 예를 들어, 위의 설명에서 참조될 수 있는 데이터, 지시들, 명령들, 정보, 신호들, 비트들, 심볼들 및 칩들은 전압들, 전류들, 전자기파들, 자기장들 또는 입자들, 광학장들 또는 입자들, 또는 이들의 임의의 결합에 의해 표현될 수 있다.

【0213】본 개시의 기술 분야에서 통상의 지식을 가진 자는 여기에 개시된 실시예들과 관련하여 설명된 다양한 예시적인 논리 블록들, 모듈들, 프로세서들, 수단들, 회로들 및 알고리즘 단계들이 전자 하드웨어, (편의를 위해, 여기에서 소프트웨어로 지칭되는) 다양한 형태들의 프로그램 또는 설계 코드 또는 이들 모두의 결합에 의해 구현될 수 있다는 것을 이해할 것이다. 하드웨어 및 소프트웨어의 이러한 상호 호환성을 명확하게 설명하기 위해, 다양한 예시적인 컴포넌트들, 블록들, 모듈들, 회로들 및 단계들이 이들의 기능과 관련하여 위에서 일반적으로 설명되었다. 이러한 기능이 하드웨어 또는 소프트웨어로서 구현되는지 여부는 특정한 애플리케이션 및 전체 시스템에 대하여 부과되는 설계 제약들에 따라 좌우된다. 본 개시의 기술 분야에서 통상의 지식을 가진 자는 각각의 특정한 애플리케이션에 대하여 다양한 방식들로 설명된 기능을 구현할 수 있으나, 이러한 구현 결정들은 본 개시의 범위를 벗어나는 것으로 해석되어서는 안 될 것이다.

【0214】여기서 제시된 다양한 실시예들은 방법, 장치, 또는 표준 프로그래밍 및/또는 엔지니어링 기술을 사용한 제조 물품(article)으로 구현될 수 있다. 용어 제조 물품은 임의의 컴퓨터-판독가능 저장장치로부터 액세스 가능한 컴퓨터 프로그램, 캐리어, 또는 매체(media)를 포함한다. 예를 들어, 컴퓨터-판독가능 저장 매체는 자기 저장 장치(예를 들면, 하드 디스크, 플로피 디스크, 자기 스트립, 등), 광학 디스크(예를 들면, CD, DVD, 등), 스마트 카드, 및 플래시 메모리 장치(예를 들면, EEPROM, 카드, 스틱, 키 드라이브, 등)를 포함하지만, 이들로 제한되는 것은 아니다. 또한, 여기서 제시되는 다양한 저장 매체는 정보를 저장하기 위한 하나 이상의 장치 및/또는 다른 기계-판독가능한 매체를 포함한다.

【0215】제시된 프로세스들에 있는 단계들의 특정한 순서 또는 계층 구조는 예시적인 접근들의 일례임을 이해하도록 한다. 설계 우선순위들에 기반하여, 본 개시의 범위 내에서 프로세스들에 있는 단계들의 특정한 순서 또는 계층 구조가 재배열될 수 있다는 것을 이해하도록 한다. 첨부된 방법 청구항들은 샘플 순서로 다양한 단계들의 엘리먼트들을 제공하지만 제시된 특정한 순서 또는 계층 구조에 한정되는 것을 의미하지는 않는다.

【0216】제시된 실시예들에 대한 설명은 임의의 본 개시의 기술 분야에서 통상의 지식을 가진 자가 본 개시를 이용하거나 또는 실시할 수 있도록 제공된다. 이러한 실시예들에 대한 다양한 변형들은 본 개시의 기술 분야에서 통상의 지식을 가진 자에게 명백할 것이며, 여기에 정의된 일반적인 원리들은 본 개시의 범위를 벗어남이 없이 다른 실시예들에 적용될 수 있다. 그리하여, 본 개시는 여기에 제시된

실시예들로 한정되는 것이 아니라, 여기에 제시된 원리들 및 신규한 특징들과 일관되는 최광의의 범위에서 해석되어야 할 것이다.

## 【청구범위】

### 【청구항 1】

컴퓨팅 장치에 의해 수행되는, 파이썬(Python) 소스코드에서 응용 프로그램 인터페이스(Application Programming Interface, API)의 오용을 탐지하기 위한 방법으로서,

입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별하는 단계;

코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 상기 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석하는 단계;

구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 상기 API 호출을 호출하거나 상기 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석하는 단계; 및

상기 함수 내부의 정적 의존성을 분석한 결과 및 상기 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성하는 단계;

를 포함하는,

방법.

### 【청구항 2】

제1항에 있어서,

상기 API 호출을 식별하는 단계는,

상기 파이썬 소스코드로부터 메서드 호출(method invocation)을 검출하는 단계; 및

검출된 상기 메서드 호출 중, 사전 정의된 API 목록에 포함된 클래스 또는 함수에 대응되는 호출을 식별함으로써 상기 API 호출을 선별하는 단계;

를 포함하는,

방법.

**【청구항 3】**

제1항에 있어서,

상기 함수 내부의 정적 의존성을 분석하는 단계는,

상기 코드 속성 그래프 상에서 데이터 흐름 엣지를 역방향으로 추적하여, 상기 API 호출에 사용되는 객체의 생성 맥락을 파악하기 위한 생성자 호출 및 상기 생성자에 전달된 파라미터를 추적하는 단계;

상기 코드 속성 그래프 상에서 순방향 데이터 흐름을 따라, 상기 API 호출의 반환값이 반환 구문에 사용되거나, 변수에 저장되거나, 또는 다른 함수의 인자로 전달되는 사용 경로를 추적하는 단계; 및

함수 간 정적 의존성 분석의 대상이 되는 함수들을 결정하기 위하여, 상기 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자(caller-callee) 관계를 식별하는 단계;

를 포함하는,

방법.

**【청구항 4】**

제1항에 있어서,

상기 함수 간 정적 의존성을 분석하는 단계는,

상기 구문 트리를 역방향으로 추적하여, 상기 API 호출에 사용되는 파라미터의 생성 또는 유입 경로를 추적하는 단계;

를 포함하는,

방법.

**【청구항 5】**

제1항에 있어서,

상기 함수 간 정적 의존성을 분석하는 단계는,

상기 복수의 함수들 각각에 대한 구문 트리를 따라 노드를 탐색하여, 상기 API 호출의 인자로 사용되는 리터럴(literal) 값 또는 상수로 정의된 값을 식별하는 단계; 및

식별된 상기 값이 상기 API 호출의 동작 또는 보안 관련 파라미터에 미치는 영향을 분석하는 단계;

를 포함하는,

방법.

**【청구항 6】**

제1항에 있어서,

상기 함수 간 정적 의존성을 분석하는 단계는,

상기 API 호출을 포함하는 함수와 이를 호출하는 상위 함수 사이의 호출자-피호출자 관계에 기초하여 함수 호출 그래프를 구성하는 단계; 및

상기 함수 호출 그래프 상에서 상기 API 호출을 종점으로 하는 호출 경로를 따라 상위 함수 방향으로 추적하여 하나 이상의 함수 호출 경로를 식별함으로써, 특정 실행 흐름 하에서 발생되는 API 오용 가능성을 분석하는 단계;

를 포함하는,

방법.

**【청구항 7】**

제1항에 있어서,

상기 구조화된 분석 컨텍스트를 생성하는 단계는,

복수의 함수들에 걸쳐 분산된 API 사용 맥락을 단일 분석 단위로 복원하기 위하여, 상기 함수 내부의 정적 의존성을 분석한 결과로부터 추출된 객체의 생성 맥락 정보, 반환값 사용 정보, 및 호출자-피호출자 관계와, 상기 함수 간 정적 의존성을 분석한 결과로부터 추출된 파라미터 전달 경로, 상수 사용 여부, 및 호출 체인을 상호 관계에 기초하여 통합함으로써, 상기 구조화된 분석 컨텍스트를 구성하는 단계;

를 포함하는,

방법.

**【청구항 8】**

제1항에 있어서,

상기 구조화된 분석 컨텍스트를 생성하는 단계 이후,

분석 대상 코드, API 오용에 관한 보안 규칙, 및 복수의 함수들에 걸쳐 분산된 API 사용 맥락이 복원된 상기 구조화된 분석 컨텍스트를 포함하는 프롬프트를 구성하는 단계; 및

상기 프롬프트를 사전 학습된 인공지능 모델에 입력하여 API 오용 여부를 판별하는 단계;

를 더 포함하는,

방법.

**【청구항 9】**

제8항에 있어서,

상기 프롬프트는,

상기 사전 학습된 인공지능 모델이 상기 보안 규칙, 상기 분석 대상 코드, 및 상기 구조화된 분석 컨텍스트를 순차적으로 검토하여 API 오용 여부를 단계적으로 추론하도록 유도하는 연쇄적 사고(Chain-of-Thought, CoT) 방식에 따라 구성되는,

방법.

**【청구항 10】**

프로세서를 포함하는 컴퓨팅 장치에 있어서,

상기 프로세서는:

입력된 파이썬(Python) 소스코드에서, 사전 정의된 응용 프로그램 인터페이스(Application Programming Interface, API) 목록에 기초하여 API 호출을 식별하는 동작;

코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 상기 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석하는 동작;

구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 상기 API 호출을 호출하거나 상기 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석하는 동작; 및

상기 함수 내부의 정적 의존성을 분석한 결과 및 상기 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성하는 동작;

을 수행하는,

컴퓨팅 장치.

# 【요약서】

## 【요약】

컴퓨팅 장치에 의해 수행되는, 파이썬(Python) 소스코드에서 응용 프로그램 인터페이스(Application Programming Interface, API)의 오용을 탐지하기 위한 방법이 개시된다.

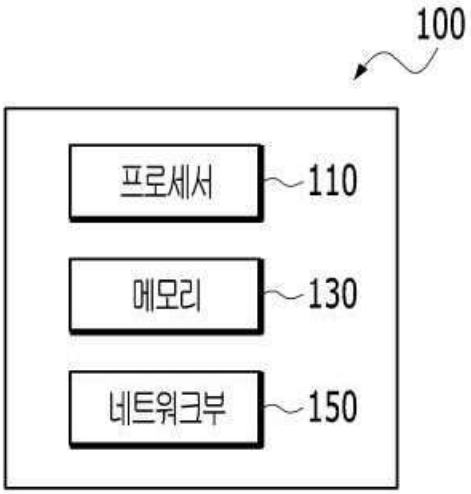
상기 방법은, 입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별하는 단계, 코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 상기 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석하는 단계, 구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 상기 API 호출을 호출하거나 상기 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석하는 단계, 및 상기 함수 내부의 정적 의존성을 분석한 결과 및 상기 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성하는 단계를 포함할 수 있다.

## 【대표도】

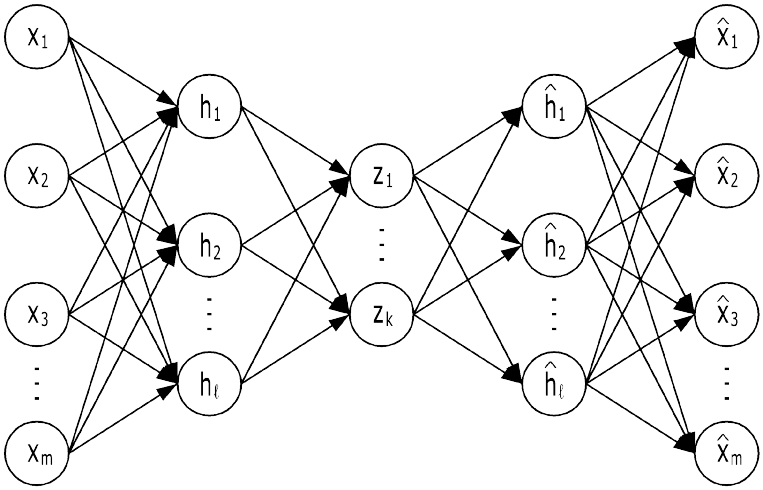
도 3

# 【도면】

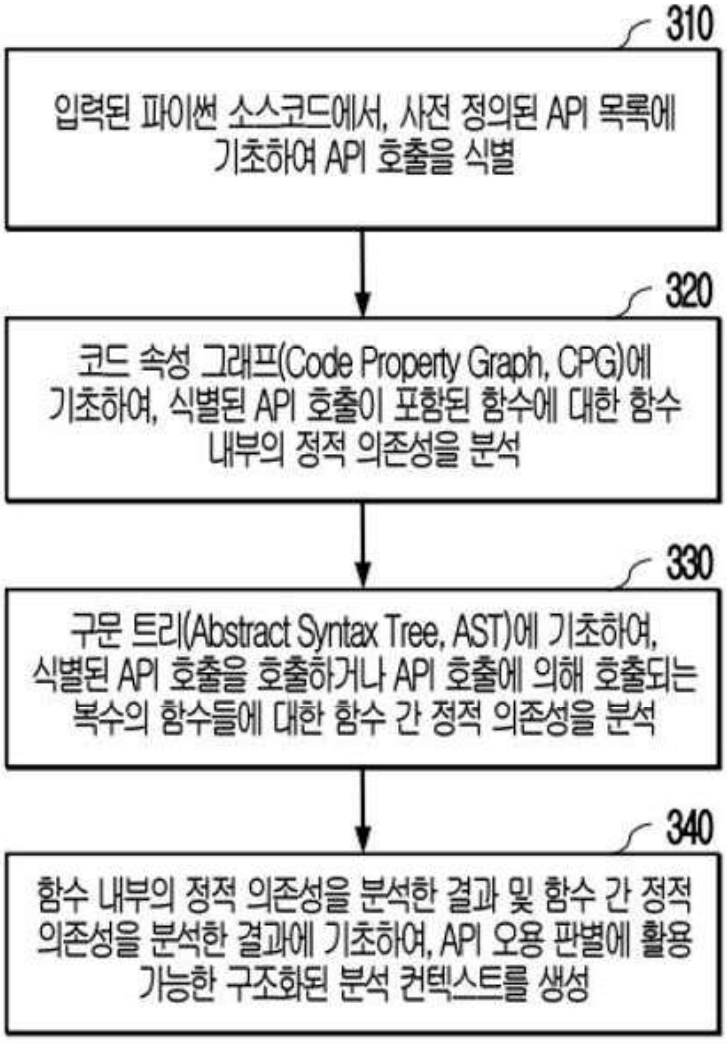
## 【도 1】


100
프로세서
110
메모리
130
네트워크부
150

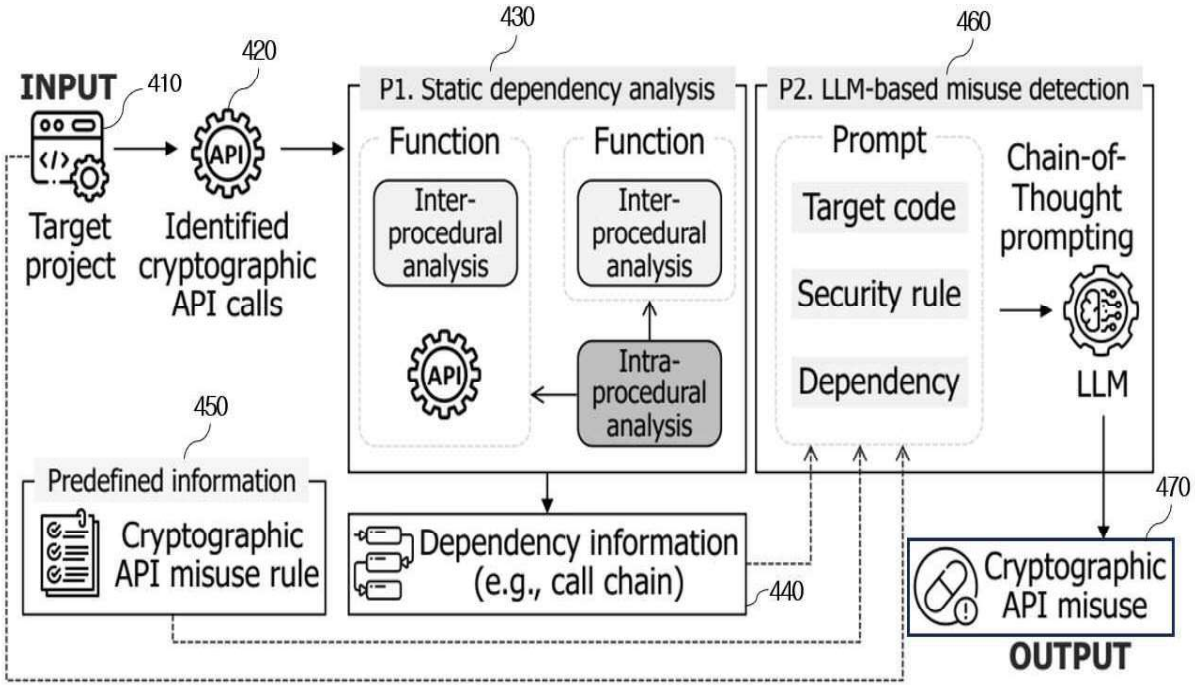
## 【도 2】


x1
x2
x3
xm
h1
h2
hℓ
z1
zk
ĥ1
ĥ2
ĥℓ
x̂1
x̂2
x̂3
x̂m

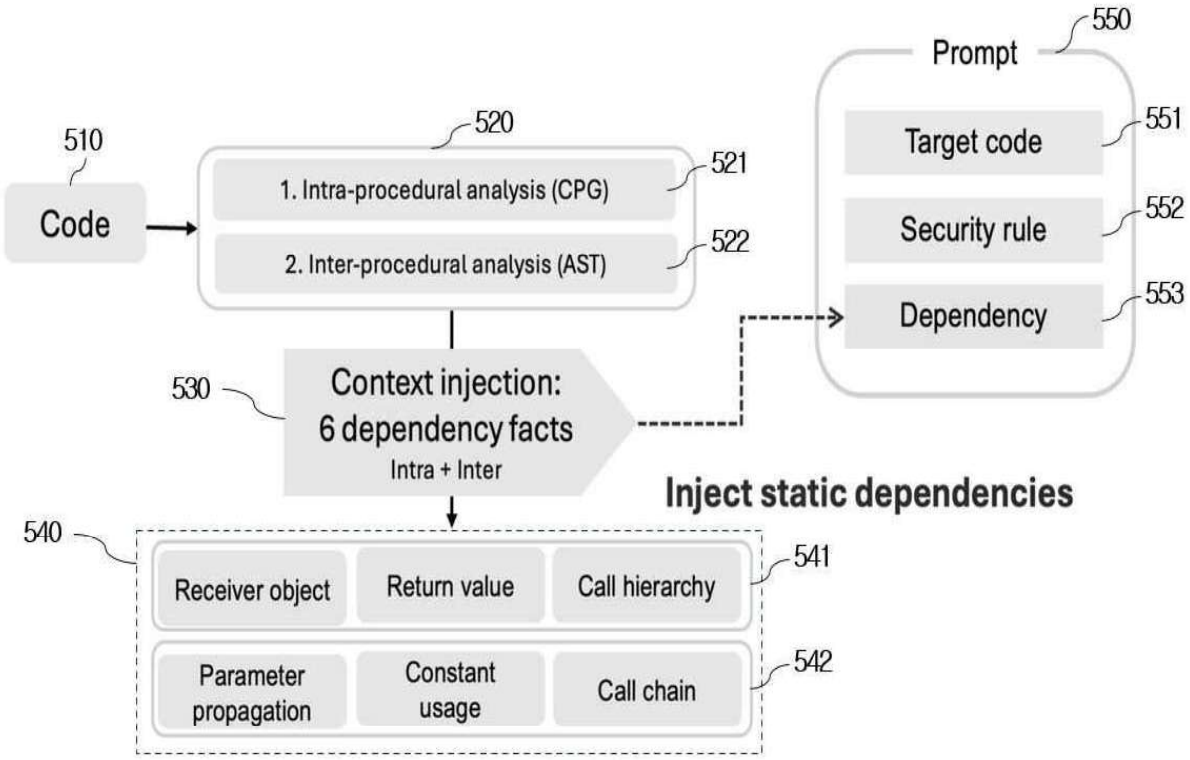
【도 3】


310
입력된 파이썬 소스코드에서, 사전 정의된 API 목록에 기초하여 API 호출을 식별
320
코드 속성 그래프(Code Property Graph, CPG)에 기초하여, 식별된 API 호출이 포함된 함수에 대한 함수 내부의 정적 의존성을 분석
330
구문 트리(Abstract Syntax Tree, AST)에 기초하여, 식별된 API 호출을 호출하거나 API 호출에 의해 호출되는 복수의 함수들에 대한 함수 간 정적 의존성을 분석
340
함수 내부의 정적 의존성을 분석한 결과 및 함수 간 정적 의존성을 분석한 결과에 기초하여, API 오용 판별에 활용 가능한 구조화된 분석 컨텍스트를 생성

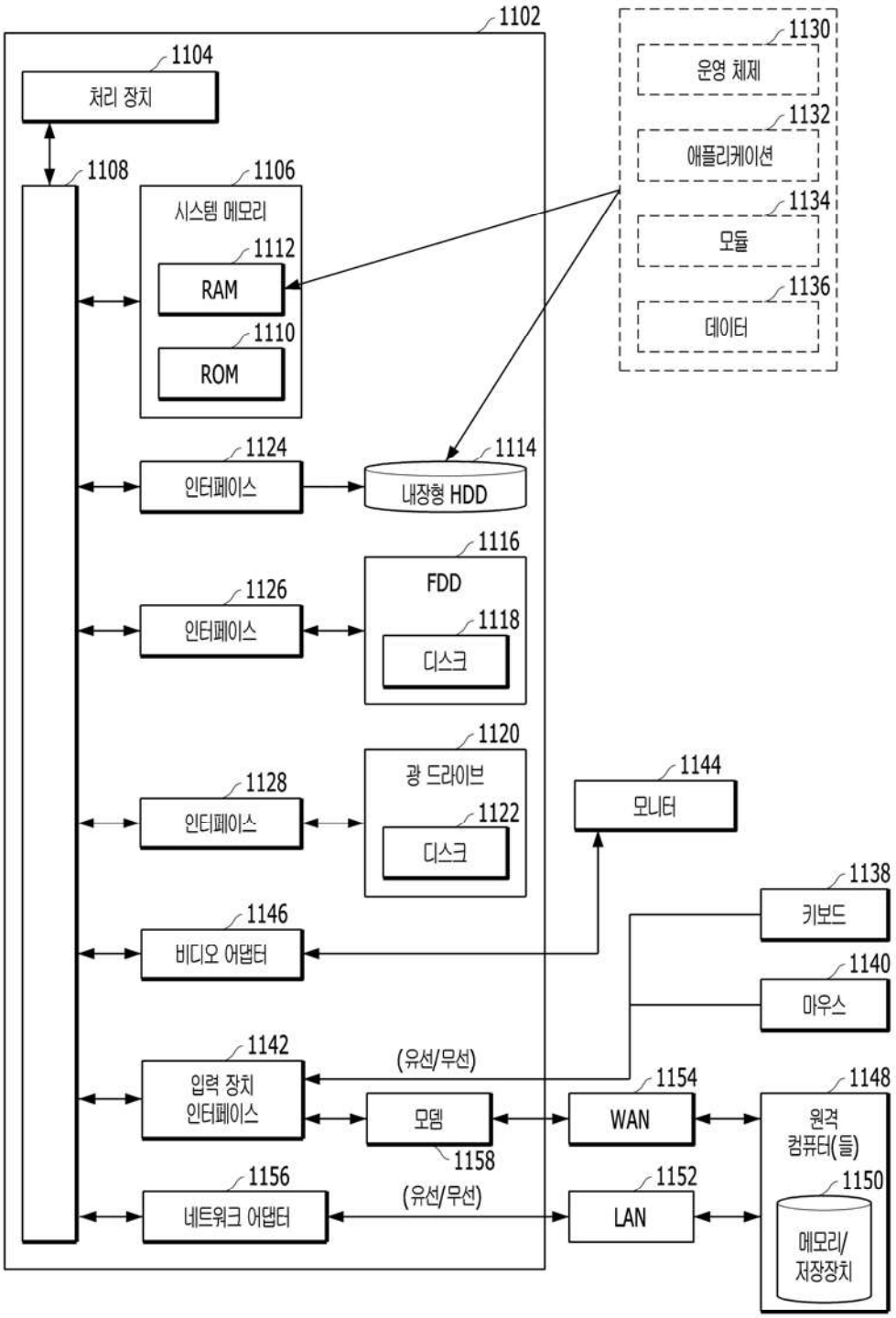
【도 4】


INPUT
410
420
430
460
P1. Static dependency analysis
P2. LLM-based misuse detection
Function
Function
Inter-procedural analysis
Inter-procedural analysis
API
Intra-procedural analysis
Prompt
Target code
Security rule
Dependency
Chain-of-Thought prompting
LLM
Target project
Identified cryptographic API calls
450
Predefined information
Cryptographic API misuse rule
Dependency information (e.g., call chain)
440
470
Cryptographic API misuse
OUTPUT

【도 5】


510
Code
520
1. Intra-procedural analysis (CPG)
2. Inter-procedural analysis (AST)
521
522
530
Context injection:
6 dependency facts
Intra + Inter
Prompt
550
Target code
551
Security rule
552
Dependency
553
Inject static dependencies
540
Receiver object
Return value
Call hierarchy
541
Parameter propagation
Constant usage
Call chain
542

【도 6】


1102
1104
처리 장치
1108
1106
시스템 메모리
1112
RAM
1110
ROM
1130
운영 체제
1132
애플리케이션
1134
모듈
1136
데이터
1124
인터페이스
1114
내장형 HDD
1116
FDD
1126
인터페이스
1118
디스크
1120
광 드라이브
1128
인터페이스
1122
디스크
1144
모니터
1138
키보드
1146
비디오 어댑터
1140
마우스
1142
입력 장치
인터페이스
(유선/무선)
1154
WAN
1148
원격
컴퓨터(들)
모뎀
1158
1156
네트워크 어댑터
(유선/무선)
1152
LAN
1150
메모리/
저장장치