(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2023-0062204
(43) 공개일자 2023년05월09일

(51) 국제특허분류(Int. Cl.)
G06F 21/56 (2013.01)
(52) CPC특허분류
G06F 21/56 (2013.01)
(21) 출원번호 10-2021-0147222
(22) 출원일자 2021년10월29일
심사청구일자 없음

(71) 출원인
삼성에스디에스 주식회사
서울특별시 송파구 올림픽로35길 125 (신천동)
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
신호철
서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)
최진우
서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)
(뒷면에 계속)
(74) 대리인
특허법인가산

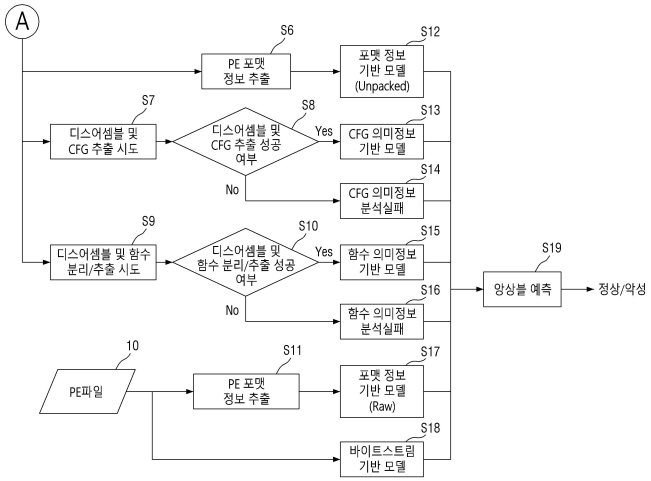
전체 청구항 수 : 총 22 항

(54) 발명의 명칭 **악성 코드 탐지 방법 및 그 장치**

(57) 요 약

본 개시는 정적 분석 방식의 악성 코드 탐지 방법을 제시한다. 본 개시에 따른 악성 코드 탐지 방법은 실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 특징 벡터를 얻는 단계와, 상기 CFG의 각 베이직 블록 간 연결 관계와, 각 베이직 블록의 상기 특징 벡터를 이용하여 상기 CFG의 표현(representation)을 생성하는 단계와, 상기 CFG의 표현에서 입력 특징 벡터를 생성하는 단계와, 상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계와, 상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함할 수 있다.

대 표 도



(72) 발명자

지윤찬

서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)

이상근

경기도 하남시 덕풍서로 65, 503동 103호(덕풍동, 하남풍산아이파크5단지)

김휘강

서울특별시 서초구 서운로 104, 201동 804호(서초동, 래미안서초에스티지에스)

유정도

서울특별시 은평구 증산로11길 31-1, 202호(증산동)

이정현

서울특별시 성북구 고려대로22길 31, 301호(안암동5가)

박상민

경기도 남양주시 와부읍 덕소로 270, 106동 1001호(한강우성아파트)

김동영

경기도 용인시 기흥구 구성로39번길 13, 103동 101호(마북동, 우림필유아파트)

한성민

서울특별시 동대문구 무학로45길 10, 401호(용두동)

조규환

서울특별시 강남구 영동대로 22, 813동 303호(일원동, 디에이치 자이 개포)

명 세 서

청구범위

청구항 1

컴퓨팅 장치에 의하여 수행되는 방법에 있어서,

실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 특징 벡터를 얻는 단계;

상기 CFG의 각 베이직 블록 간 연결 관계와, 각 베이직 블록의 상기 특징 벡터를 이용하여 상기 CFG의 표현(representation)을 생성하는 단계;

상기 CFG의 표현에서 입력 특징 벡터를 생성하는 단계;

상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계; 및

상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함하되,

상기 실행 가능 파일은 복수의 인스트럭션(instruction)으로 구성된 것이고, 각각의 인스트럭션은 명령어(opcode) 및 피연산자(operand)를 포함하는 토큰에 대응되는 것인,

악성 코드 탐지 방법.

청구항 2

제1 항에 있어서,

상기 토큰 임베딩 기반 특징 벡터를 얻는 단계 이전에, 복수의 실행 가능 파일을 포함하는 학습 데이터 셋을 이용하여 토큰 및 그 토큰 임베딩으로 구성된 토큰 사전을 구성하는 단계를 더 포함하고,

상기 토큰 임베딩 기반 특징 벡터를 얻는 단계는,

상기 베이직 블록에 대응되는 각각의 토큰의 토큰 임베딩을 상기 토큰 사전에서 조회하여 얻는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 3

제2 항에 있어서,

상기 토큰 사전을 구성하는 단계는,

상기 학습 데이터 셋에 포함된 제1 실행 가능 파일을 얻는 단계;

상기 제1 실행 가능 파일의 CFG를 얻는 단계;

상기 제1 실행 가능 파일을 디스어셈블(disassemble)하여, 상기 제1 실행 가능 파일의 어셈블리 코드(assembly code)를 얻는 단계;

상기 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과와, 상기 어셈블리 코드를 이용하여 상기 제1 실행 가능 파일에 대한 아티클(article)을 얻는 단계;

상기 아티클에서 복수의 토큰을 추출하는 단계;

상기 추출된 각각의 토큰에 대하여 토큰 임베딩을 연산하는 단계;

상기 연산 된 토큰 임베딩을 CBOW(Continuous Bag of Words) 방식을 적용하여 최적화하는 단계; 및

상기 추출된 토큰과 상기 최적화된 토큰 임베딩을 상기 토큰 사전에 추가하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 4

제3 항에 있어서,

상기 CBOW 방식을 적용하여 최적화하는 단계는,

악성 코드 탐지 정확도가 제1 레벨로 세팅 된 경우, 제1 개수의 주변 단어(context word)를 이용하여 상기 CBOW 방식을 적용하는 단계; 및

상기 악성 코드 탐지 정확도가 상기 제1 레벨보다 낮은 제2 레벨로 세팅 된 경우, 상기 제1 개수보다 작은 제2 개수의 주변 단어를 이용하여 상기 CBOW 방식을 적용하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 5

제2 항에 있어서,

상기 토큰 사전을 구성하는 단계는,

상기 학습 데이터 셋에 포함된 실행 가능 파일에 대하여, 상기 토큰 사전으로의 토큰 추가 프로세스를 수행하는 단계; 및

상기 토큰 추가 프로세스를 상기 학습 데이터 셋에 포함된 각각의 실행 가능 파일에 대하여 반복하되, 상기 반복에 따른 추가 토큰 개수가 기준치에 미달하는 경우, 잔여 실행 가능 파일이 존재하더라도 상기 토큰 추가 프로세스를 종료하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 6

제2 항에 있어서,

상기 베이직 블록에 대응되는 각각의 토큰의 토큰 임베딩을 상기 토큰 사전에서 조회하여 얻는 단계는,

상기 토큰 사전에서 조회되지 않는 신규 토큰의 토큰 임베딩을 영(zero) 임베딩으로 할당하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 7

제1 항에 있어서,

상기 입력 특징 벡터를 GNN 기반의 그래프 분류 모델에 입력하는 단계는,

상기 입력 특징 벡터를 GIN 레이어(Graph Isomorphism Network layer)을 포함하는 상기 그래프 분류 모델에 입력하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 8

제7 항에 있어서,

상기 그래프 분류 모델은,

5개 이하의 GIN 레이어를 포함하는 것인,

악성 코드 탐지 방법.

청구항 9

제7 항에 있어서,

상기 입력 특징 벡터를 GIN 레이어(Graph Isomorphism Network layer)을 포함하는 상기 그래프 분류 모델에 입력하는 단계는,

악성 코드 탐지 정확도가 제1 레벨로 세팅 된 경우, 상기 입력 특징 벡터를 제1 개수의 GIN 레이어를 포함하는 제1 그래프 분류 모델에 입력하는 단계; 및

상기 악성 코드 탐지 정확도가 상기 제1 레벨보다 낮은 제2 레벨로 세팅 된 경우, 상기 입력 특징 벡터를 상기 제1 개수보다 작은 제2 개수의 GIN 레이어를 포함하는 제2 그래프 분류 모델에 입력하는 단계를 포함하되,

악성 코드 탐지 방법.

청구항 10

컴퓨팅 장치에 의하여 수행되는 방법에 있어서,

실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계;

상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 특징 벡터를 이용하여 상기 함수 호출 그래프의 표현(representation)을 생성하는 단계;

상기 함수 호출 그래프의 표현에서 입력 특징 벡터를 생성하는 단계;

상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계; 및

상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 11

제10 항에 있어서,

상기 함수 각각에 대한 특징 벡터를 얻는 단계 이전에, 복수의 실행 가능 파일을 포함하는 학습 데이터 셋을 이용하여 토큰 및 그 토큰 임베딩으로 구성된 토큰 사전을 구성하는 단계와, 함수의 특징 벡터를 최적화 하기 위한 최적화 모델을 학습시키는 단계를 더 포함하고,

상기 최적화 모델을 학습시키는 단계는,

상기 학습 데이터셋에 포함된 제1 실행 가능 파일의 제1 함수의 식별정보에 대한 함수 임베딩을 생성하는 단계;

상기 제1 함수에 대하여 제1 함수 CFG(control flow graph)를 얻는 단계;

상기 제1 함수의 어셈블리 코드(assembly code)를 얻는 단계;

상기 제1 함수의 어셈블리 코드, 상기 제1 함수 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과, 상기 제1 함수 CFG에서 샘플링 된 엣지에 연결된 인스트럭션들을 이용하여 상기 제1 함수에 대한 아티클(article)을 얻는 단계; 및

상기 아티클을 구성하는 토큰에 대한 토큰 임베딩을 상기 토큰 사전에서 조회하는 단계; 및

상기 조회된 토큰 임베딩과 상기 함수 임베딩을 PV-DM(Paragraph Vector - Distributed Memory) 모델에 입력하여, 상기 PV-DM 모델을 학습시키는 단계를 포함하되,

상기 최적화 모델은, 상기 PV-DM 모델인,

악성 코드 탐지 방법.

청구항 12

제11 항에 있어서,

상기 PV-DM 모델을 학습시키는 단계는,

상기 아티클에 포함된 3개의 연속적인 토큰으로 구성되는 윈도우를 세팅하는 단계;

상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록 상기 함수 임베딩 및 상기 윈도우의 각 토큰의 토큰 임베딩을 업데이트 하는 단계; 및

상기 아티클의 마지막에 도달할 때까지 상기 윈도우를 쉬프트 하면서 상기 세팅하는 단계 및 상기 업데이트 하는 단계를 반복하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 13

제10 항에 있어서,

상기 특징 벡터를 얻는 단계는,

상기 실행 가능 파일의 제1 함수의 식별정보에 대한 함수 임베딩을 생성하는 단계;

상기 제1 함수에 대하여 제1 함수 CFG(control flow graph)를 얻는 단계;

상기 제1 함수의 어셈블리 코드(assembly code)를 얻는 단계;

상기 제1 함수의 어셈블리 코드, 상기 제1 함수 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과, 상기 제1 함수 CFG에서 샘플링 된 엣지에 연결된 인스트럭션들을 이용하여 상기 제1 함수에 대한 아티클(article)을 얻는 단계; 및

상기 아티클을 구성하는 토큰에 대한 토큰 임베딩을 상기 토큰 사전에서 조회하는 단계;

상기 조회된 토큰 임베딩과 상기 함수 임베딩을 기 학습된 PV-DM 모델에 입력하여, 상기 함수 임베딩을 최적화하는 단계; 및

상기 최적화된 함수 임베딩을 상기 제1 함수에 대한 특징 벡터로 결정하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 14

제13 항에 있어서,

상기 함수 임베딩을 최적화하는 단계는,

상기 아티클에 포함된 3개의 연속적인 토큰으로 구성되는 윈도우를 세팅하는 단계;

상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록, 상기 함수 임베딩을 업데이트 하는 단계; 및

상기 아티클의 마지막에 도달할 때까지 상기 윈도우를 쉬프트 하면서 상기 세팅하는 단계 및 상기 업데이트 하는 단계를 반복하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 15

제14 항에 있어서,

상기 함수 임베딩을 업데이트 하는 단계는,

상기 PV-DM 모델의 파라미터 및 상기 윈도우의 토큰이 고정된 상태에서 상기 함수 임베딩을 업데이트 하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 16

컴퓨팅 장치에 의하여 수행되는 방법에 있어서,

실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계;

각 함수의 특징 벡터들을 포함하는 입력 특징 벡터를 구성하는 단계;

상기 입력 특징 벡터를 CNN(Convolutional Neural Network) 기반의 분류 모델에 입력하는 단계; 및

상기 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 17

제16 항에 있어서,

상기 각 함수의 특징 벡터들을 포함하는 입력 특징 벡터를 구성하는 단계는,

각 함수의 주소에 따라 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써 상기 입력 특징 벡터를 구성하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 18

제17 항에 있어서,

상기 각 함수의 주소에 따라 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써 상기 입력 특징 벡터를 구성하는 단계는,

각 함수의 주소에 따라 오름 차순으로 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써, 상기 입력 특징 벡터를 구성하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 19

제16 항에 있어서,

상기 CNN 기반의 분류 모델은,

1D 컨볼루션 연산에 의하여 학습된 1D CNN 기반의 분류 모델인,

악성 코드 탐지 방법.

청구항 20

컴퓨팅 장치에 의하여 수행되는 방법에 있어서,

실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계;

상기 실행 가능 파일의 함수 호출 그래프의 추출을 시도하는 단계;

상기 함수 호출 그래프의 추출이 성공한 경우, 제1 프로세스를 수행하고, 상기 함수 호출 그래프의 추출이 실패한 경우, 제2 프로세스를 수행하는 단계를 포함하되,

상기 제1 프로세스는,

상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 특징 벡터를 이용하여 상기 함수 호출 그래프의 표현(representation)을 생성하는 단계;

상기 함수 호출 그래프의 표현에서 입력 특징 벡터를 생성하는 단계;

상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계; 및

상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함하고,

상기 제2 프로세스는,

실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계;

각 함수의 특징 벡터들을 포함하는 입력 데이터를 구성하는 단계;

상기 입력 데이터를 CNN(Convolutional Neural Network) 기반의 분류 모델에 입력하는 단계; 및

상기 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함하는,

악성 코드 탐지 방법.

청구항 21

인스트럭션의 명령어(opcode) 및 그 피연산자(operand)를 가리키는 토큰과, 상기 토큰의 임베딩(embedding)을 포함하는 토큰 사전 데이터를 저장하는 스토리지;

메모리; 및

상기 메모리에 로드(load)된 악성 코드 탐지 프로그램을 실행하는 프로세서를 포함하되,

상기 악성 코드 탐지 프로그램은,

악성 코드 탐지 대상인 실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 제1 특징 벡터를 얻고, 상기 CFG의 각 베이직 블록 간 연결 관계와 각 베이직 블록의 상기 제1 특징 벡터를 이용하여 상기 CFG를 표현하는 제1 입력 특징 벡터를 생성하는 제1 인스트럭션(instruction);

상기 실행 가능 파일에 포함된 함수 각각에 대한 제2 특징 벡터를 얻는 제2 인스트럭션;

상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 제2 특징 벡터를 이용하여 상기 함수 호출 그래프를 표현하는 제2 입력 특징 벡터를 생성하는 제3 인스트럭션;

상기 실행 가능 파일에 포함된 함수 각각에 대한 상기 제2 특징 벡터들을 포함하는 제3 입력 특징 벡터를 생성하는 제4 인스트럭션;

상기 제1 입력 특징 벡터를 입력 받은 제1 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제5 인스트럭션;

상기 제2 입력 특징 벡터를 입력 받은 제2 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제6 인스트럭션; 및

상기 제3 입력 특징 벡터를 입력 받은 CNN(Convolutional Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제7 인스트럭션을 포함하는,

악성 코드 탐지 장치.

청구항 22

제21 항에 있어서,

상기 악성 코드 탐지 프로그램은,

상기 실행 가능 파일에서 상기 함수 호출 그래프를 추출하는 것이 성공하면 상기 제3 인스트럭션과 상기 제6 인스트럭션으로 구성된 제1 그룹 인스트럭션을 실행하고, 상기 함수 호출 그래프를 추출하는 것이 실패하면 상기 제4 인스트럭션과 상기 제7 인스트럭션으로 구성된 제2 그룹 인스트럭션을 실행하는,

악성 코드 탐지 장치.

발명의 설명

기 술 분 야

[0001] 본 개시는 악성 코드를 탐지하는 방법, 그 장치 또는 그 시스템에 관한 것이다. 보다 자세하게는, 정적 분석(static analysis) 기반의 악성 코드를 탐지하는 방법, 그 장치 또는 그 시스템에 관한 것이다.

배 경 기 술

[0002] 실행 가능 파일(Executable File)에서 악성 코드를 탐지하는 기술이 제공된다. 상기 실행 가능 파일은, 예를 들어 컴파일 된 바이너리 파일일 수 있으며, 윈도우 운영체제에서 사용되는 실행 파일, DLL(Dynamic Linked Library) 파일 등은 PE 포맷 기반으로 구성되어 있어 PE 파일로도 지칭된다. 상기 PE 파일은 호출되는 서브루틴에 대한 정보 등 악성 코드의 탐지에 필요한 다양한 정보를 포함하며, 상기 정보는 상기 PE 파일의 실행 없이도 확인될 수 있다.

[0003] 실행 가능 파일에 대한 실행이 요구되는 동적 분석(dynamic analysis) 대비, 정적 분석은 몇몇 장점을 가진다. 예를 들어, 정적 분석은 실행 가능 파일의 실행을 위한 격리 실행 환경 구축, 실행에 따른 연산 자원 소비, 다양한 실행 환경 구축 등을 필요로 하지 않는다. 반면, 상기 실행 가능 파일로부터 특징 값을 추출하여 사전에 알려진 악성 코드와 비교하는 방식을 취하는 형태의 정적 분석은 변종 악성 코드에 취약한 문제가 있다.

[0004] 알려지지 않은 악성 코드 또는 변종 악성 코드까지 탐지하기 위하여 기계 학습된 악성 코드 탐지 모델을 이용하여 악성 코드를 탐지하는 정적 분석 기술도 제공되고 있으나, 형식 정보에 강하게 의존하여 정상 파일의 형식과 유사한 악성 코드에는 취약하고, 크기가 큰 악성 코드의 경우 미분석 영역이 발생될 수 있거나, 패킹 등 정적 분석 방지 기술이 적용된 실행 가능 파일에는 적용되기 어렵거나, 악성 코드 탐지 모델의 학습을 위하여 다수의 학습 데이터가 요구되는 등의 한계가 존재한다. 학습 데이터가 부족한 현실을 극복하기 위하여 딥러닝 등의 비지도 학습(unsupervised-learning) 기반 악성 코드 탐지 모델을 이용하는 악성 코드 탐지 기술도 제공되고 있으나, 그 정확도 측면에서 상용화에 한계가 존재한다.

선행기술문헌

특허문헌

[0005] (특허문헌 0001) 미국공개특허 제2021-0029145호 (2021.01.28 공개)

(특허문헌 0002) 미국공개특허 제2021-0029157호 (2021.01.28 공개)

(특허문헌 0003) 미국공개특허 제2019-0362074호 (2019.11.28 공개)

(특허문헌 0004) 한국등록특허 제1880686호 (2018.07.20 공개)

(특허문헌 0005) 미국공개특허 제2019-0034632호 (2019.01.31 공개)

비특허문헌

[0006] (비특허문헌 0001) Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin, "DEEPBINDIFF: Learning Program-Wide Code Representations for Binary Difng" in Network and Distributed Systems Security (NDSS) Symposium 2020, 23-26 February 2020 (https://dx.doi.org/10.14722/ndss.2020.24311)

(비특허문헌 0002) Aravind Nair, Avigit Roy, Karl Meinke, "funcGNN: A Graph Neural Network Approach to Program Similarity", 2020 (https://doi.org/10.1145/3382494.3410675)

(비특허문헌 0003) Lan Zhang, Peng Liu, Yoon-Ho Choi, "Semantics-preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection", 2020 (https://arxiv.org/abs/2009.05602)

(비특허문헌 0004) Samuel Kim,"PE Header Analysis for Malware Detection PE Header Analysis for Malware Detection", 2018(https://doi.org/10.31979/etd.q3dd-gp9u)

(비특허문헌 0005) Edward Raff, Jared Sylvester, Charles Nicholas, "Learning the PE Header, Malware Detection with Minimal Domain Knowledge", 2017, (https://doi.org/10.1145/3128572.3140442)

(비특허문헌 0006) Angelo Oliveira, "MALWARE ANALYSIS DATASETS: RAW PE AS IMAGE", 2019, (https://dx.doi.org/10.21227/8brp-j220)

(비특허문헌 0007) Damin Moon, JaeKoo Lee, MyungKeun Yoon, "Compact feature hashing for machine learning based malware detection", 2021.8, (https://doi.org/10.1016/j.icte.2021.08.005)

(비특허문헌 0008) Jifeng Xuan, He Jiang, Zhilei Ren, Yan Hu, Zhongxuan Luo, "A Random Walk Based Algorithm for Structural Test Case Generation", 2017(https://arxiv.org/pdf/1704.04772)

발명의 내용

해결하려는 과제

[0007] 본 개시의 몇몇 실시예들을 통하여 달성하고자 하는 기술적 과제는, 실행 가능 파일에서 추출된 각 서브 루틴(예를 들어, 함수) 별 의미 정보(semantic information)를 기반으로, 우회가 곤란한 정적 분석 기반의 악성 코드 탐지 방법 및 그 장치를 제공하는 것이다.

[0008] 본 개시의 몇몇 실시예들을 통하여 달성하고자 하는 다른 기술적 과제는, 실행 가능 파일에서 추출된 베이직 블록(basic block)의 의미 정보(semantic information)를 기반으로, 우회가 곤란한 정적 분석 기반의 악성 코드 탐지 방법 및 그 장치를 제공하는 것이다.

[0009] 본 개시의 몇몇 실시예들을 통하여 달성하고자 하는 또 다른 기술적 과제는, 실행 가능 파일에 정적 분석 방지 기술이 적용된 경우에도 작동하는 정적 분석 기반의 악성 코드 탐지 방법 및 그 장치를 제공하는 것이다.

[0010] 본 개시의 몇몇 실시예들을 통하여 달성하고자 하는 또 다른 기술적 과제는, 다양한 방식의 정적 분석의 결과를 종합적으로 고려하여 악성 코드 여부를 최종 결정함으로써, 높은 정확도의 정적 분석 기반 악성 코드 탐지 방법 및 그 장치를 제공하는 것이다.

[0011] 본 개시의 기술적 과제들은 이상에서 언급한 기술적 과제들로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 본 개시의 기술분야에서의 통상의 기술자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0012] 본 개시의 일 실시예에 따른 악성 코드 탐지 방법은, 실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 특징 벡터를 얻는 단계와, 상기 CFG의 각 베이직 블록 간 연결 관계와, 각 베이직 블록의 상기 특징 벡터를 이용하여 상기 CFG의 표현(representation)을 생성하는 단계와, 상기 CFG의 표현에서 입력 특징 벡터를 생성하는 단계와, 상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계와, 상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함한다. 이 때, 상기 실행 가능 파일은 복수의 인스트럭션(instruction)으로 구성된 것이고, 각각의 인스트럭션은 명령어(opcode) 및 피연산자(operand)를 포함하는 토큰에 대응되는 것이다.

[0013] 일 실시예에서, 상기 악성 코드 탐지 방법은 상기 토큰 임베딩 기반 특징 벡터를 얻는 단계 이전에, 복수의 실행 가능 파일을 포함하는 학습 데이터 셋을 이용하여 토큰 및 그 토큰 임베딩으로 구성된 토큰 사전을 구성하는 단계를 더 포함할 수 있다. 또한, 상기 토큰 임베딩 기반 특징 벡터를 얻는 단계는 상기 베이직 블록에 대응되는 각각의 토큰의 토큰 임베딩을 상기 토큰 사전에서 조회하여 얻는 단계를 포함할 수 있다. 상기 베이직 블록에 대응되는 각각의 토큰의 토큰 임베딩을 상기 토큰 사전에서 조회하여 얻는 단계는, 상기 토큰 사전에서 조회되지 않는 신규 토큰의 토큰 임베딩을 영(zero) 임베딩으로 할당하는 단계를 포함할 수 있다.

[0014] 상기 토큰 사전을 구성하는 단계는, 상기 학습 데이터 셋에 포함된 제1 실행 가능 파일을 얻는 단계와, 상기 제1 실행 가능 파일의 CFG를 얻는 단계와, 상기 제1 실행 가능 파일을 디스어셈블(disassemble)하여, 상기 제1 실행 가능 파일의 어셈블리 코드(assembly code)를 얻는 단계와, 상기 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과와, 상기 어셈블리 코드를 이용하여 상기 제1 실행 가능 파일에 대한 아티클(article)을 얻는 단계와, 상기 아티클에서 복수의 토큰을 추출하는 단계와, 상기 추출된 각각의 토큰에 대하여 토큰 임베딩을 연산하는 단계와, 상기 연산 된 토큰 임베딩을 CBOW(Continuous Bag of Words) 방식을 적용하여 최적화하는 단계와, 상기 추출된 토큰과 상기 최적화된 토큰 임베딩을 상기 토큰 사전에 추가하는 단계를 포함할 수 있다. 이 때, 상기 CBOW 방식의 적용에 있어서 사용되는 주변 단어(context word)의 개수는 요구되는 악성 코드 탐지 정확도에 따라 조정될 수 있다. 예를 들어, 상기 CBOW 방식을 적용하여 최적화하는 단계는 악성 코드 탐지 정확도가 제1 레

벨로 세팅 된 경우, 제1 개수의 주변 단어(context word)를 이용하여 상기 CBOW 방식을 적용하는 단계와, 상기 악성 코드 탐지 정확도가 상기 제1 레벨보다 낮은 제2 레벨로 세팅 된 경우, 상기 제1 개수보다 작은 제2 개수의 주변 단어를 이용하여 상기 CBOW 방식을 적용하는 단계를 포함할 수 있다.

[0015] 또한, 상기 토큰 사전을 구성하는 단계는, 상기 학습 데이터 셋에 포함된 실행 가능 파일에 대하여, 상기 토큰 사전으로의 토큰 추가 프로세스를 수행하는 단계와, 상기 토큰 추가 프로세스를 상기 학습 데이터 셋에 포함된 각각의 실행 가능 파일에 대하여 반복하되, 상기 반복에 따른 추가 토큰 개수가 기준치에 미달하는 경우, 잔여 실행 가능 파일이 존재하더라도 상기 토큰 추가 프로세스를 종료하는 단계를 포함할 수도 있다.

[0016] 일 실시예에서, 상기 입력 특징 벡터를 GNN 기반의 그래프 분류 모델에 입력하는 단계는, 상기 입력 특징 벡터를 GIN 레이어(Graph Isomorphism Network layer)을 포함하는 상기 그래프 분류 모델에 입력하는 단계를 포함할 수 있다. 상기 그래프 분류 모델은, 5개 이하의 GIN 레이어를 포함하는 것일 수 있다. 상기 그래프 분류 모델은, 예를 들어 3개의 GIN 레이어를 포함하는 것일 수 있다.

[0017] 상기 입력 특징 벡터를 GIN 레이어(Graph Isomorphism Network layer)을 포함하는 상기 그래프 분류 모델에 입력하는 단계는, 악성 코드 탐지 정확도가 제1 레벨로 세팅 된 경우, 상기 입력 특징 벡터를 제1 개수의 GIN 레이어를 포함하는 제1 그래프 분류 모델에 입력하는 단계와, 상기 악성 코드 탐지 정확도가 상기 제1 레벨보다 낮은 제2 레벨로 세팅 된 경우, 상기 입력 특징 벡터를 상기 제1 개수보다 작은 제2 개수의 GIN 레이어를 포함하는 제2 그래프 분류 모델에 입력하는 단계를 포함할 수 있다.

[0018] 본 개시의 다른 실시예에 따른 악성 코드 탐지 방법은, 실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계와, 상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 특징 벡터를 이용하여 상기 함수 호출 그래프의 표현(representation)을 생성하는 단계와, 상기 함수 호출 그래프의 표현에서 입력 특징 벡터를 생성하는 단계와, 상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계와, 상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함할 수 있다.

[0019] 상기 악성 코드 탐지 방법은 상기 함수 각각에 대한 특징 벡터를 얻는 단계 이전에, 복수의 실행 가능 파일을 포함하는 학습 데이터 셋을 이용하여 토큰 및 그 토큰 임베딩으로 구성된 토큰 사전을 구성하는 단계와, 함수의 특징 벡터를 최적화 하기 위한 최적화 모델을 학습시키는 단계를 더 포함할 수 있다. 상기 최적화 모델을 학습시키는 단계는 상기 학습 데이터셋에 포함된 제1 실행 가능 파일의 제1 함수의 식별정보에 대한 함수 임베딩을 생성하는 단계와, 상기 제1 함수에 대하여 제1 함수 CFG(control flow graph)를 얻는 단계와, 상기 제1 함수의 어셈블리 코드(assembly code)를 얻는 단계와, 상기 제1 함수의 어셈블리 코드, 상기 제1 함수 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과, 상기 제1 함수 CFG에서 샘플링 된 엣지에 연결된 인스트럭션들을 이용하여 상기 제1 함수에 대한 아티클(article)을 얻는 단계와, 상기 아티클을 구성하는 토큰에 대한 토큰 임베딩을 상기 토큰 사전에서 조회하는 단계와, 상기 조회된 토큰 임베딩과 상기 함수 임베딩을 PV-DM(Paragraph Vector - Distributed Memory) 모델에 입력하여, 상기 PV-DM 모델을 학습시키는 단계를 포함할 수 있다. 이 때, 상기 최적화 모델은, 상기 PV-DM 모델을 가리키는 것으로 이해될 수 있을 것이다.

[0020] 상기 PV-DM 모델을 학습시키는 단계는, 상기 아티클에 포함된 3개의 연속적인 토큰으로 구성되는 윈도우를 세팅하는 단계와, 상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록 상기 함수 임베딩 및 상기 윈도우의 각 토큰의 토큰 임베딩을 업데이트 하는 단계와, 상기 아티클의 마지막에 도달할 때까지 상기 윈도우를 쉬프트 하면서 상기 세팅하는 단계 및 상기 업데이트 하는 단계를 반복하는 단계를 포함할 수 있다.

[0021] 상기 특징 벡터를 얻는 단계는, 상기 실행 가능 파일의 제1 함수의 식별정보에 대한 함수 임베딩을 생성하는 단계와, 상기 제1 함수에 대하여 제1 함수 CFG(control flow graph)를 얻는 단계와, 상기 제1 함수의 어셈블리 코드(assembly code)를 얻는 단계와, 상기 제1 함수의 어셈블리 코드, 상기 제1 함수 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과, 상기 제1 함수 CFG에서 샘플링 된 엣지에 연결된 인스트럭션들을 이용하여 상기 제1 함수에 대한 아티클(article)을 얻는 단계와, 상기 아티클을 구성하는 토큰에 대한 토큰 임베딩을 상기 토큰 사전에서 조회하는 단계와, 상기 조회된 토큰 임베딩과 상기 함수 임베딩을 기 학습된 PV-DM 모델에 입력하여, 상기 함수 임베딩을 최적화하는 단계와, 상기 최적화된 함수 임베딩을 상기 제1 함수에 대한 특징 벡터로 결정하는 단계를 포함할 수 있다.

[0022] 상기 함수 임베딩을 최적화하는 단계는, 상기 아티클에 포함된 3개의 연속적인 토큰으로 구성되는 윈도우를 세

팅하는 단계와, 상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록, 상기 함수 임베딩을 업데이트 하는 단계와, 상기 아티클의 마지막에 도달할 때까지 상기 윈도우를 쉬프트 하면서 상기 세팅하는 단계 및 상기 업데이트 하는 단계를 반복하는 단계를 포함할 수 있다. 상기 함수 임베딩을 업데이트 하는 단계는, 상기 PV-DM 모델의 파라미터 및 상기 윈도우의 토큰이 고정된 상태에서 상기 함수 임베딩을 업데이트 하는 단계를 포함할 수 있다.

[0023] 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법은, 실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계와, 각 함수의 특징 벡터들을 포함하는 입력 특징 벡터를 구성하는 단계와, 상기 입력 특징 벡터를 CNN(Convolutional Neural Network) 기반의 분류 모델에 입력하는 단계와, 상기 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함할 수 있다.

[0024] 상기 각 함수의 특징 벡터들을 포함하는 입력 특징 벡터를 구성하는 단계는, 각 함수의 주소에 따라 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써 상기 입력 특징 벡터를 구성하는 단계를 포함할 수 있다. 또한, 상기 각 함수의 주소에 따라 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써 상기 입력 특징 벡터를 구성하는 단계는, 각 함수의 주소에 따라 오름 차순으로 정렬된 순서로 상기 각 함수의 특징 벡터들을 배열함으로써, 상기 입력 특징 벡터를 구성하는 단계를 포함할 수 있다.

[0025] 상기 CNN 기반의 분류 모델은, 1D 컨볼루션 연산에 의하여 학습된 1D CNN 기반의 분류 모델일 수 있다.

[0026] 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법은, 실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계와, 상기 실행 가능 파일의 함수 호출 그래프의 추출을 시도하는 단계와, 상기 함수 호출 그래프의 추출이 성공한 경우, 제1 프로세스를 수행하고, 상기 함수 호출 그래프의 추출이 실패한 경우, 제2 프로세스를 수행하는 단계를 포함할 수 있다. 상기 제1 프로세스는, 상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 특징 벡터를 이용하여 상기 함수 호출 그래프의 표현(representation)을 생성하는 단계와, 상기 함수 호출 그래프의 표현에서 입력 특징 벡터를 생성하는 단계와, 상기 입력 특징 벡터를 GNN(Graph Neural Network) 기반의 그래프 분류 모델에 입력하는 단계와, 상기 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함할 수 있다. 또한, 상기 제2 프로세스는, 실행 가능 파일에 포함된 함수 각각에 대한 특징 벡터를 얻는 단계와, 각 함수의 특징 벡터들을 포함하는 입력 데이터를 구성하는 단계와, 상기 입력 데이터를 CNN(Convolutional Neural Network) 기반의 분류 모델에 입력하는 단계와, 상기 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일이 악성코드인지 여부를 결정하는 단계를 포함할 수 있다.

[0027] 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 장치는, 인스트럭션의 명령어(opcode) 및 그 피연산자(operand)를 가리키는 토큰과, 상기 토큰의 임베딩(embedding)을 포함하는 토큰 사전 데이터를 저장하는 스토리지와, 메모리와, 상기 메모리에 로드(load)된 악성 코드 탐지 프로그램을 실행하는 프로세서를 포함할 수 있다.

[0028] 상기 악성 코드 탐지 프로그램은, 악성 코드 탐지 대상인 실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 제1 특징 벡터를 얻고, 상기 CFG의 각 베이직 블록 간 연결 관계와 각 베이직 블록의 상기 제1 특징 벡터를 이용하여 상기 CFG를 표현하는 제1 입력 특징 벡터를 생성하는 제1 인스트럭션(instruction)과, 상기 실행 가능 파일에 포함된 함수 각각에 대한 제2 특징 벡터를 얻는 제2 인스트럭션과, 상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 제2 특징 벡터를 이용하여 상기 함수 호출 그래프를 표현하는 제2 입력 특징 벡터를 생성하는 제3 인스트럭션과, 상기 실행 가능 파일에 포함된 함수 각각에 대한 상기 제2 특징 벡터들을 포함하는 제3 입력 특징 벡터를 생성하는 제4 인스트럭션과, 상기 제1 입력 특징 벡터를 입력 받은 제1 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제5 인스트럭션과, 상기 제2 입력 특징 벡터를 입력 받은 제2 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제6 인스트럭션과, 상기 제3 입력 특징 벡터를 입력 받은 CNN(Convolutional Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제7 인스트럭션을 포함할 수 있다.

[0029] 상기 악성 코드 탐지 프로그램은, 상기 실행 가능 파일에서 상기 함수 호출 그래프를 추출하는 것이 성공하면 상기 제3 인스트럭션과 상기 제6 인스트럭션으로 구성된 제1 그룹 인스트럭션을 실행하고, 상기 함수 호출 그래프를 추출하는 것이 실패하면 상기 제4 인스트럭션과 상기 제7 인스트럭션으로 구성된 제2 그룹 인스트럭션을 실행하는 것일 수 있다. 즉, 상기 악성 코드 탐지 프로그램은 상기 제1 그룹 인스트럭션과 상기 제2 그룹 인스트럭션 중 어느 하나를 선택적으로 실행할 수 있는 것이다.

도면의 간단한 설명

[0030] 도 1은 본 개시의 일 실시예에 따른 악성 코드 탐지 시스템의 구성도이다.

도 2 내지 도 3은 본 개시의 다른 실시예에 따른 악성 코드 탐지 방법의 순서도이다.

도 4는 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법의 순서도이다.

도 5는 본 개시의 몇몇 실시예에서 참조되는 토큰 사전 구성 방법을 설명하기 위한 도면이다.

도 6은 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법의 순서도이다.

도 7은 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법의 순서도이다.

도 8은 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법의 순서도이다.

도 9는 본 개시의 몇몇 실시예들에서 구성요소로서 사용될 수 있는 컴퓨팅 장치의 하드웨어 구성도이다.

발명을 실시하기 위한 구체적인 내용

[0031] 이하, 첨부된 도면을 참조하여 본 개시의 실시예들을 상세히 설명한다. 본 개시의 실시예들의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명의 기술적 사상은 이하의 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 수 있으며, 단지 이하의 실시예들은 본 발명의 기술적 사상을 완전하도록 하고, 본 개시의 실시예들이 속하는 기술분야에서 통상의 지식을 가진 자에게 본 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명의 기술적 사상은 청구항의 범주에 의해 정의될 뿐이다.

[0032] 각 도면의 구성요소들에 참조부호를 부가함에 있어서, 동일한 구성요소들에 대해서는 비록 다른 도면상에 표시되더라도 가능한 한 동일한 부호를 가지도록 하고 있음에 유의해야 한다. 또한, 본 개시의 실시예들을 설명함에 있어, 관련된 공지 구성 또는 기능에 대한 구체적인 설명이 요지를 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명은 생략한다.

[0033] 다른 정의가 없다면, 본 개시에서 사용되는 모든 용어(기술 및 과학적 용어를 포함)는 본 개시의 실시예들이 속하는 기술분야에서 통상의 지식을 가진 자에게 공통적으로 이해될 수 있는 의미로 사용될 수 있다. 또 일반적으로 사용되는 사전에 정의되어 있는 용어들은 명백하게 특별히 정의되어 있지 않는 한 이상적으로 또는 과도하게 해석되지 않는다. 본 개시에서 사용된 용어는 실시예들을 설명하기 위한 것이며 본 개시의 실시예들을 제한하고자 하는 것은 아니다. 본 개시에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다.

[0034] 또한, 본 명세서의 실시예들의 구성 요소를 설명하는 데 있어서, 제1, 제2 등의 용어를 사용할 수 있다. 이러한 용어는 그 구성 요소를 다른 구성 요소와 구별하기 위한 것일 뿐, 그 용어에 의해 해당 구성 요소의 본질이나 차례 또는 순서 등이 한정되지 않는다. 어떤 구성 요소가 다른 구성요소에 "연결", "결합" 또는 "접속"된다고 기재된 경우, 그 구성 요소는 그 다른 구성요소에 직접적으로 연결되거나 또는 접속될 수 있지만, 각 구성 요소 사이에 또 다른 구성 요소가 "연결", "결합" 또는 "접속"될 수도 있다고 이해되어야 할 것이다.

[0035] 이하, 몇몇 실시예들에 대하여 첨부된 도면을 참조하여 설명한다.

[0036] 도 1을 참조하여, 본 개시의 일 실시예에 따른 악성 코드 탐지 시스템의 구성 및 동작을 설명한다. 본 실시예에 따른 악성 코드 탐지 시스템은 서비스 서버(100), 제1 데이터베이스 서버(101) 및 제2 데이터베이스 서버(102)를 포함할 수 있다.

[0037] 서비스 서버(100)는 실행 가능 파일(executable file)을 얻어서, 상기 실행 가능 파일이 악성 코드인지 여부를 판정하며, 상기 판정 결과를 출력한다. 상기 실행 가능 파일은 서비스 서버(100)의 스토리지 장치 또는 메모리에 저장된 것이거나, 서비스 서버(100)가 네트워크를 통하여 수신한 것일 수 있다. 또한, 상기 판정 결과에 대한 데이터는 네트워크를 통하여 다른 서비스 서버에 송신되거나, 사용자 단말(103)에 송신될 수 있다.

[0038] 제1 데이터베이스 서버(101)는 복수의 실행 가능 파일로 구성된 학습 데이터 셋을 저장할 수 있다. 상기 실행 가능 파일은, 예를 들어 PE 파일일 수 있다. 서비스 서버(100)는 제1 데이터 베이스 서버(101)에 상기 실행 가능 파일을 송신할 수 있다. 예를 들어, 서비스 서버(100)는 기업 내 클라우드 스토리지 서비스를 제공하는 서버이면서, 클라우드 스토리지에 저장되는 실행 가능 파일에 대한 악성 코드 탐지 서비스를 제공할 수 있다. 이때, 서비스 서버(100)는 소정의 조건에 부합하는 실행 가능 파일을 선택적으로 제1 데이터베이스 서버(101)에

송신할 수 있다.

[0039] 상기 실행 가능 파일은 PE 파일로 한정되지 않는다. 예를 들어, 유닉스 계열 운영 체제의 실행 가능 파일 포맷인 ELF, macOS의 실행 가능 파일 포맷인 Mach-O, DOS 운영 체제의 실행 가능 파일 포맷인 COM 등 다양한 포맷의 실행 가능 파일에 대하여 본 개시의 실시예들이 적용될 수 있을 것이다.

[0040] 서비스 서버(100)는 주기적으로 또는 비주기적으로 제1 데이터베이스 서버(101)에 저장된 학습 데이터 셋을 이용한 사전 준비 프로세스를 수행할 수 있다. 상기 사전 준비 프로세스는 토큰 사전 구성, 토큰 사전 업데이트 및 인공 신경망으로 구성된 모델의 학습 단계(training stage) 실행 중 적어도 일부를 포함할 수 있다.

[0041] 서비스 서버(100)는 상기 사전 준비 프로세스를 수행한 결과 생성된 데이터를 자체적으로 저장하거나, 제2 데이터베이스 서버(102)에 저장할 수 있다. 예를 들어, 제2 데이터베이스 서버(102)는 토큰 사전 데이터를 저장할 수 있을 것이다.

[0042] 서비스 서버(100)는 상기 사전 준비 프로세스의 수행 결과 생성된 데이터를 이용하여, 실행 가능 파일에 대하여 정적 분석 방식의 악성 코드 탐지 로직을 실행하고, 상기 실행 가능 파일에 대한 악성 코드 여부를 출력할 수 있다. 상기 악성 코드 탐지 로직은 전처리 프로세스, CFG(Control Flow Graph)의 베이직 블록 의미 정보 기반의 탐지 프로세스, 함수 등 서브루틴의 의미 정보를 기반으로 하는 탐지 프로세스, 실행 가능 파일의 포맷 정보 기반 탐지 프로세스 및 실행 가능 파일 자체의 바이트스트림(byte stream) 기반 탐지 프로세스 중 적어도 하나를 포함할 수 있다.

[0043] 상기 사전 준비 프로세스 및 상기 악성 코드 탐지 로직에 포함되는 각 프로세스에 대하여는 본 개시의 몇몇 실시예들을 통하여 자세히 후술될 것이다. 또한, 도 1에는 제1 데이터베이스 서버(101) 및 제2 데이터베이스 서버(102)가 서비스 서버(100)와 별개의 장치로서 도시되어 있으나, 실시예에 따라 제1 데이터베이스 서버(101) 및 제2 데이터베이스 서버(102) 중 적어도 하나가 생략될 수 있을 것이다.

[0044] 다음으로, 도 2 내지 도 3을 참조하여 본 개시의 또 다른 실시예에 따른 악성 코드 탐지 방법을 설명한다. 본 실시예에 따른 악성 코드 탐지 방법은 하나 이상의 컴퓨팅 장치에 의하여 수행될 수 있다. 즉, 본 실시예에 따른 악성 코드 탐지 방법은 하나의 컴퓨팅 장치에 의하여 모든 동작이 수행될 수도 있고, 일부의 동작이 다른 컴퓨팅 장치에 의하여 수행될 수도 있다. 이하, 본 실시예에 따른 방법을 설명함에 있어서, 일부 동작의 수행 주체에 대한 기재가 생략될 수 있다. 이 때, 해당 동작의 수행 주체는 상기 컴퓨팅 장치인 것으로 이해되어야 한다. 상기 컴퓨팅 장치는, 예를 들어 도 1을 참조하여 설명한 실시예에서의 서비스 서버(100)일 수 있지만, 상기 컴퓨팅 장치가 서비스 서버(100)인 것으로 한정되어서는 아니된다.

[0045] 도 2는 상기 전처리 프로세스를 설명하기 위한 순서도로 이해될 수 있을 것이다. 이하, 도면 들에서 상기 실행 가능 파일의 예시로서 'PE 파일'이 도시될 수 있으나, 본 개시의 실시예들은 PE 파일 이외의 실행 가능 파일을 입력 받는 경우에도 적용될 수 있음을 유의 하여야 한다.

[0046] PE 파일(10)이 입력되면, 패킹(packing) 여부가 확인된다(S1). 패킹 된 PE 파일(10)에 대하여는 정적 분석이 불가능하므로, 패킹 된 PE 파일이 공지된 바이너리 언패킹(unpacking) 기술을 이용하여 언패킹 할 수 있는 형식의 것이라면(S2) 언패킹이 시도된다(S3). 언패킹을 실패하거나(S4) 패킹된 PE 파일이 언패킹 할 수 없는 형식의 것이라면(S2), 언패킹 실패를 가리키는 에러 메시지가 출력(S5)되고 악성 코드 탐지 절차는 종료될 것이다.

[0047] 몇몇 실시예에서, 언패킹을 실패하거나(S4) 패킹된 PE 파일이 언패킹 할 수 없는 형식의 것이라도, 악성 코드 탐지 절차가 종료되지 않을 수 있다. 즉, 언패킹을 실패하거나(S4) 패킹된 PE 파일이 언패킹 할 수 없는 형식의 것이라도 도 3의 S6 내지 S19 단계가 실행될 수 있는 것이다. 즉, 본 개시에서 다양한 방식의 악성 코드 탐지가 종합적으로 수행되므로, PE 파일의 언패킹이 실패하더라도 악성 코드 탐지가 가능하다.

[0048] 도 3을 참조하여 상기 전처리 프로세스 이후의 악성 코드 탐지 로직을 설명한다.

[0049] 먼저 이해되어야 하는 점은, 상기 악성 코드 탐지 로직이 다양한 방식의 악성 코드 탐지 결과들을 종합적으로 고려하여 최종적으로 악성 코드 탐지 결과를 출력할 수 있다는 것이다. 도 3에 도시된 바와 같이 다양한 방식의 악성 코드 탐지 결과들(S12 내지 S18)이 종합적으로 고려되어, 앙상블 예측(S19) 방식에 따라 최종 악성 코드 탐지 결과가 생성될 수 있다. 즉, 앙상블 예측(S19) 과정에서 악성 코드 탐지 결과들(S12 내지 S18)이 가중 합산 되고, 그 결과가 기준치를 초과하는지 여부에 따라 실행 가능 파일의 악성 코드 여부가 결정될 수 있는 것이다.

[0050] 물론, 상기 악성 코드 탐지 로직은 일부 방식의 악성 코드 탐지 프로세스 만을 실행하는 것일 수도 있다. 상기

악성 코드 탐지 로직은 특정 방식의 악성 코드 탐지 프로세스 만을 실행하는 것일 수도 있을 것이다.

[0051] 제1 방식의 악성 코드 탐지로서, PE 포맷 정보를 입력 받는 악성 코드 탐지 모델이 이용될 수 있다. 제1 타입의 PE 포맷 정보 기반 악성 코드 탐지 모델은 언패킹 된 PE 파일의 포맷 정보를 입력 받을 수 있다(S6, S12). 제2 타입의 PE 포맷 정보 기반 악성 코드 탐지 모델은 PE 파일 자체의 포맷 정보를 입력 받을 수 있다(S11, S17). 상기 제1 방식의 악성 코드 탐지와 관련하여, Samuel Kim,"PE Header Analysis for Malware Detection PE Header Analysis for Malware Detection", 2018(https://doi.org/10.31979/etd.q3dd-gp9u), Edward Raff, Jared Sylvester, Charles Nicholas, "Learning the PE Header, Malware Detection with Minimal Domain Knowledge", 2017, (https://doi.org/10.1145/3128572.3140442) 등 공지 문헌들을 참조할 수 있을 것이다.

[0052] 제2 방식의 악성 코드 탐지로서, PE 파일 자체를 입력 받는 바이트 스트림 기반 악성 코드 탐지 모델이 이용될 수 있다(S18). 이 때, PE 파일 자체가 악성 코드로 알려진 실행 가능 파일과 유사한 특성을 가지는지 여부가 판정되는 것으로 이해할 수 있을 것이다. 상기 제2 방식의 악성 코드 탐지와 관련하여, Angelo Oliveira, "MALWARE ANALYSIS DATASETS: RAW PE AS IMAGE", 2019, (https://dx.doi.org/10.21227/8brp-j220), Damin Moon, JaeKoo Lee, MyungKeun Yoon, "Compact feature hashing for machine learning based malware detection", 2021.8, (https://doi.org/10.1016/j.icte.2021.08.005) 등 공지 문헌들을 참조할 수 있을 것이다.

[0053] 제3 방식의 악성 코드 탐지로서, CFG(Control Flow Graph)의 베이직 블록 의미 정보 기반의 악성 코드 탐지가 수행될 수 있다. 이를 위해, PE 파일(10)에 대한 CFG(Control Flow Graph) 추출 및 디스어셈블(disassemble)이 시도되고(S7), 시도(S7)의 결과 성공인 경우(S8), CFG의 베이직 블록 의미 정보 기반의 악성 코드 탐지 모델을 이용한 악성 코드 탐지가 수행될 것이고(S13), 시도(S7)의 결과 실패인 경우(S8), 실패를 가리키는 데이터가 앙상블 예측(S19) 과정에서 고려될 것이다(S14). 제3 방식의 악성 코드 탐지와 관련하여, 본 개시의 몇몇 실시예들을 통하여 보다 자세히 후술될 것이다.

[0054] 제4 방식의 악성 코드 탐지로서, 함수 등 서브루틴의 의미 정보를 기반으로 하는 악성 코드 탐지가 수행될 수 있다. 이를 위해, PE 파일(10)에 대한 함수 호출 그래프(call graph) 추출 및 디스어셈블(disassemble)이 시도되고(S9), 시도(S9)의 결과 성공인 경우(S10), PE 파일에서 호출되는 각 함수의 의미 정보 기반의 악성 코드 탐지 모델을 이용한 악성 코드 탐지가 수행될 것이고(S15), 시도(S9)의 결과 실패인 경우(S10), 실패를 가리키는 데이터가 앙상블 예측(S19) 과정에서 고려될 것이다(S16). 제4 방식의 악성 코드 탐지와 관련하여, 본 개시의 몇몇 실시예들을 통하여 보다 자세히 후술될 것이다.

[0055] 다음으로, 도 4 내지 도 5를 참조하여, CFG(Control Flow Graph)의 베이직 블록 의미 정보 기반의 악성 코드 탐지 방법을 설명한다.

[0056] 먼저, 사전 트레이닝 과정(S130)을 설명한다.

[0057] 악성 코드 탐지를 위하여 몇몇 동작이 사전에 준비될 필요가 있으며, 본 개시에서는 이를 사전 트레이닝으로 지칭하기로 한다. 상기 사전 트레이닝에는 토큰 사전(token dictionary)의 구성 및 인공 신경망(neural network) 기반의 몇몇 모델들에 대한 기계 학습 과정이 포함된다. 이하, 각각의 과정을 상세히 설명한다.

[0058] **1. 토큰 사전의 구축**

[0059] 상기 토큰 사전은 각 토큰 별로 그 토큰 임베딩(token embedding)을 포함하는 데이터로 이해될 수 있을 것이다. 악성 코드의 탐지 시점에서 각 토큰에 대한 토큰 임베딩을 구하는 시간을 절약하기 위해, 토큰의 최적화된 임베딩을 미리 저장해두는 것으로 이해될 수 있을 것이다.

[0060] 본 개시에서, 상기 '토큰(token)'은 어셈블리 코드의 각 인스트럭션(instruction)을 가리킨다. 즉, 상기 토큰은, 상기 토큰에 대응되는 어셈블리 코드에 포함된 명령어(opcode) 및 그 피연산자(operand)에 대응될 것이다. 또한, 토큰 임베딩은 토큰을 표현하는 하나 이상의 숫자로 이해될 수 있을 것이다.

[0061] 예를 들어, "cmp ecx, 0x408963"이라는 인스트럭션을 대상으로 토큰 임베딩이 부여되는 과정을 설명한다. 몇몇 실시예에서, 토큰 사전에 포함되는 토큰의 개수가 불필요하게 늘어나는 것을 방지하기 위하여, 피연산자에 대한 정규화(normalization)가 수행될 수 있다. 예를 들어, ecx, r8d, eax 등 4바이트 레지스터라면 'reg4'와 같이 정규화 될 수 있고, 0x408963과 같은 상수 값은 'im'과 같이 정규화 될 수 있을 것이다.

[0062] 상기 설명된 바와 같이 정규화 된 인스트럭션인 "cmp reg3, im"을 대상으로 토큰 임베딩이 부여되는 과정을 설명한다. 연산자 'cmp'에 [0.03, 0.16, 1.92, 쪈]의 임베딩이 부여되고, 피연산자인 "reg4"에 [0.62, -0.125,

0.76, 쪈]의 임베딩이 부여되며, "im"에 [1.5, 1.6, -0.92]의 임베딩이 부여된다고 가정하자. 이 때, 연산자에 대한 중요도를 반영하기 위해, TF-IDF 모델 등 키워드 선정 로직을 이용하여, 연산자 중요도가 생성될 수 있으며, 상기 연산자 중요도가 연산자에 대한 임베딩에 가중치로서 반영될 수 있을 것이다. 다음으로, 가중치가 반영된 연산자 임베딩과, 피연산자의 임베딩이 연결(concatenation) 됨으로써, 최종적으로 토큰 임베딩이 생성된다.

[0063] 학습 데이터의 각 실행 가능 파일들에 디스어셈블리(disassembly)를 적용하여 어셈블리 코드를 얻고, 상기 어셈블리 코드에 포함된 인스트럭션들 중 적어도 일부에 대하여 상술한 방식으로 토큰 임베딩이 생성된다. 각 토큰과, 상기 생성된 토큰 임베딩이 쌍(pair)을 이뤄 상기 토큰 사전에 추가될 것이다.

[0064] 몇몇 실시예에서, 상기 실행 가능 파일의 어셈블리 코드 자체에서 상기 토큰 사전에 추가될 토큰의 인스트럭션이 얻어지는 것이 아니라, 상기 실행 가능 파일에서 추출된 CFG를 랜덤 워크(random walk)한 결과로 얻어진 순차 실행문에서 상기 토큰 사전에 추가될 토큰의 인스트럭션이 얻어질 수 있다. 상기 랜덤 워크에 대하여는 Jifeng Xuan, He Jiang, Zhilei Ren, Yan Hu, Zhongxuan Luo, "A Random Walk Based Algorithm for Structural Test Case Generation", 2017(https://arxiv.org/pdf/1704.04772) 등 다양한 공지 문헌을 참조할 수 있을 것이다.

[0065] 상기 순차 실행문은 런타임(run-time)에 순차적으로 실행될 인스트럭션들을 반영한다. 상기 어셈블리 코드 자체보다는 상기 순차 실행문이 실행 가능 파일이 실행될 때의 동작을 보다 잘 표현(representation)하는 것으로 이해될 수 있을 것이며, 본 개시에서는 상기 순차 실행문을 아티클(article)으로 지칭하기로 한다.

[0066] 또한, 몇몇 실시예에서는 상기 실행 가능 파일의 어셈블리 코드 자체에서 토큰을 추출하고, 이에 더하여 상기 아티클에서 토큰을 더 추출할 수도 있을 것이다. 이 때, 학습 데이터로부터 최대한 많은 토큰을 추출할 수 있는 효과를 얻을 수 있다.

[0067] 몇몇 실시예에서, 상기 토큰 임베딩은 Word2Vec 기술을 이용하여 최적화될 수 있다. 예를 들어, 토큰 임베딩의 최적화에, Word2Vec 기술 중 CBOW(Continuous Bag of Words) 기술이 이용될 수 있다. 이 때, CBOW 기술은 인접한 주변 단어(context word)를 이용하여 대상 단어(target word)를 예측하는 기술이므로, 실행 가능 파일이 실행될 때 각 인스트럭션의 실행 순서를 반영하고 있는 아티클에서 추출된 토큰에 대하여, CBOW 기술을 이용한 토큰 임베딩 최적화가 수행될 수 있을 것이다.

[0068] CBOW 기술의 적용 시에, 요구되는 탐지 정확도에 따라 주변 단어(context word)의 범위가 달라질 수 있다. 즉, 요구되는 탐지 정확도에 따라 윈도우(window)의 사이즈가 조정될 수 있는 것이다. 즉, 높은 수준의 탐지 정확도에 대응하기 위한 토큰 사전이라면, 시간이 더 걸리더라도 큰 사이즈의 윈도우를 적용하여 토큰 임베딩이 최적화될 것이고, 낮은 수준의 탐지 정확도에 대응하기 위한 토큰 사전이라면, 작은 사이즈의 윈도우를 적용하여 토큰 임베딩이 빠르게 최적화될 것이다. 상기 탐지 정확도는 시스템 환경 설정 값으로 관리될 수 있을 것이다. 또는, 상기 탐지 정확도는 시스템의 트래픽 등을 고려하여 자동으로 조정될 수 있을 것이다.

[0069] 몇몇 실시예에서, 도 5에 도시된 바와 같이, 토큰 사전 구축에 소요되는 시간을 최소화할 수 있도록, 추가되는 토큰의 수가 미미한 시점에 학습 데이터가 남아 있더라도, 토큰 사전 구축이 중단될 수 있다. 즉, 상기 토큰 추가 프로세스가 상기 학습 데이터 셋에 포함된 각각의 실행 가능 파일에 대하여 반복되는데, 상기 반복에 따른 추가 토큰 개수가 제1 기준치에 미달하는 경우, 잔여 실행 가능 파일이 존재하더라도 상기 토큰 추가 프로세스가 종료되는 것으로 이해될 수 있다. 상기 제1 기준치 역시 상기 시스템 환경 설정 값으로 관리될 수 있을 것이다.

[0070] 또한, 몇몇 실시예에서, 상기 반복에 따른 추가 토큰 개수가 상기 제1 기준치에 미달하는 횟수가 제2 기준치 이상이어야 상기 토큰 추가 프로세스가 종료되는 것으로 중간 종료 요건이 강화될 수 있을 것이다. 상기 제2 기준치 역시 상기 시스템 환경 설정 값으로 관리 될 수 있을 것이다.

[0071] 상기 제1 기준치 및 상기 제2 기준치는 상기 탐지 정확도에 기반하여 자동 조정되거나, 시스템 트래픽을 고려하여 자동 조정될 수 있다.

[0072] **2. 인공 신경망 기반 모델들에 대한 기계 학습**

[0073] **(1) CBOW 모델의 기계 학습**

[0074] 상술한 바와 같이, 토큰 사전의 구성 과정에서 CBOW 기술을 이용한 토큰 임베딩의 최적화가 수행된다. 상기 CBOW 기술의 적용을 위한 CBOW 모델의 기계 학습이 상기 토큰 사전의 구성 이전에 선행될 수 있다. CBOW 모델의

기계 학습 과정은 다수의 공지 문헌을 참고하면 될 것이므로 상세한 설명은 생략한다.

[0075] **(2) 분류 모델의 기계 학습**

[0076] 인공 신경망 기반의 분류 모델들에 대한 기계 학습이 수행된다. 본 개시에서는 GNN 기반의 분류 모델 또는 CNN 기반의 분류 모델이 사용될 수 있다. 상기 GNN 기반의 분류 모델 및 상기 CNN 기반의 분류 모델 모두 분류기 역할을 하는 FCL(Fully-Connected Layer)를 포함할 수 있으며, 상기 FCN은 출력 층 이전에 연결될 것이다.

[0077] 1) GNN 기반 분류 모델

[0078] 상기 GNN 기반의 분류 모델은 CFG의 베이직블록 별 의미 정보 기반 악성 코드 탐지 과정 또는 함수의 의미 정보 기반 악성 코드 탐지 과정에 사용될 수 있다. CFG의 베이직블록 별 의미 정보 기반 악성 코드 탐지 과정에 이용되는 GNN 기반의 분류 모델을 편의상 제1 GNN 기반 분류 모델으로 지칭하고, 함수의 의미 정보 기반 악성 코드 탐지 과정에 이용되는 GNN 기반의 분류 모델을 편의상 제2 GNN 기반 분류 모델으로 지칭할 것이다.

[0079] 상기 제1 GNN 기반 분류 모델은 CFG의 베이직블록 별 의미 정보 기반 악성 코드 탐지 과정에서 CFG의 표현(representation)을 가리키는 입력 특징 데이터(feature data)를 입력 받고, 악성 코드 여부에 대한 데이터를 출력한다. 상기 CFG의 표현은 그래프의 노드인 베이직 블록의 특징 벡터를 포함한다. 상기 베이직 블록의 특징 벡터는 상기 베이직 블록의 의미 정보(semantic information)를 가리킨다.

[0080] 상기 베이직 블록의 특징 벡터는, 상기 베이직 블록에 속한 각각의 토큰의 토큰 임베딩을 이용하여 생성될 수 있다. 상기 베이직 블록에 속한 각각의 토큰은 상기 베이직 블록에 대응되는 상기 아티클에서 추출된 것일 수 있다. 예를 들어, 상기 베이직 블록의 특징 벡터는 상기 베이직 블록에 대응되는 상기 아티클의 모든 토큰의 토큰 임베딩의 대표값(예를 들어, 평균값)이거나, 토큰의 명령어(opcode)의 중요도를 기준으로 선정된 일부 토큰의 토큰 임베딩의 대표 값일 수 있다.

[0081] 상기 베이직 블록에 속한 각각의 토큰에 대한 토큰 임베딩은, 상술한 방식으로 구성된 토큰 사전에서 읽어오는 값이다. 상기 베이직 블록에 속한 토큰이 상기 토큰 사전에 포함되지 않은 것이라면, 상기 토큰 임베딩의 값은 영(zero)이 될 수 있다.

[0082] 즉, 상기 제1 GNN 기반 분류 모델은 실행 가능 파일의 각 베이직 블록의 의미 정보와 베이직 블록 사이의 연결 관계에 대한 정보를 입력 받게 되는 것이다. 즉, 상기 제1 GNN 기반 분류 모델은 실행 가능 파일의 전체적인 실행 흐름을 가리키는 CFG 정보를 기반으로 악성 코드 여부를 탐지하는데, CFG의 각 노드인 베이직 블록의 특징 벡터는 상기 베이직 블록의 의미 정보를 표현하고 있으므로, 단순한 변형 만으로 악성 코드 탐지를 회피하는 것이 쉽지 않게 될 것이다.

[0083] 상기 제1 GNN 기반 분류 모델의 학습 과정에서, 학습 데이터에 포함된 각각의 실행 가능 파일에 대한 CFG 표현을 가리키는 입력 특징 데이터가 상기 제1 GNN 기반 분류 모델에 입력되고 상기 제1 GNN 기반 분류 모델의 출력이 상기 실행 가능 파일의 악성 여부를 맞추는 방향으로 상기 제1 GNN 기반 분류 모델의 가중치가 업데이트 될 것이다.

[0084] 다음으로 상기 제2 GNN 기반 분류 모델의 학습 과정을 설명한다.

[0085] 상기 제2 GNN 기반 분류 모델은 함수의 의미 정보 기반 악성 코드 탐지 과정에서 함수 호출 그래프의 표현을 가리키는 입력 특징 데이터를 입력 받고, 악성 코드 여부에 대한 데이터를 출력한다. 상기 함수 호출 그래프의 표현에는 그래프의 노드인 함수의 특징 벡터를 포함한다. 상기 함수의 특징 벡터는 상기 함수의 의미 정보를 가리킨다.

[0086] 상기 함수의 특징 벡터는, 상기 함수에 속한 각각의 토큰의 토큰 임베딩과, 상기 함수 자체의 임베딩을 이용하여 생성될 수 있다. 상기 함수에 속한 각각의 토큰은 상기 함수에 대응되는 아티클에서 추출된 것일 수 있다. 예를 들어, 상기 함수의 특징 벡터는 상기 함수에 대응되는 상기 아티클의 모든 토큰의 토큰 임베딩의 대표값(예를 들어, 평균값)이거나, 토큰의 명령어(opcode)의 중요도를 기준으로 선정된 일부 토큰의 토큰 임베딩의 대표 값일 수 있다.

[0087] 상기 함수에 대응되는 상기 아티클에 속한 각각의 토큰에 대한 토큰 임베딩은, 상술한 방식으로 구성된 토큰 사전에서 읽어오는 값이다. 상기 베이직 블록에 속한 토큰이 상기 토큰 사전에 포함되지 않은 것이라면, 상기 토큰 임베딩의 값은 영(zero)이 될 수 있다.

[0088] 또한, 상기 함수 자체의 임베딩은 함수에 부여된 ID 등 식별정보를 기반으로 초기값이 부여된 후, Word2Vec 기

술을 이용하여 최적화 될 수 있다. Word2Vec 기술의 적용 과정에서, 상기 함수 자체의 임베딩은 상기 함수의 아티클에 포함된 토큰의 토큰 임베딩을 반영하여 업데이트 된다. 상기 Word2Vec 기술 중, PV-DM(Paragraph Vector - Distributed Memory) 기술이 적용될 수 있다. 즉, 상기 제2 GNN 기반 분류 모델의 학습을 위하여는, PV-DM 모델의 학습이 선행되는 것으로 이해될 것이다. PV-DM 모델의 학습과 관련하여 자세히 후술한다.

[0089] 상기 제2 GNN 기반 분류 모델은 실행 가능 파일의 각 함수의 의미 정보와 함수 사이의 호출 관계에 대한 정보를 입력 받게 되는 것이다. 즉, 상기 제2 GNN 기반 분류 모델은 실행 가능 파일의 전체적인 실행 흐름을 가리키는 함수 호출 정보를 기반으로 악성 코드 여부를 탐지하는데, 함수 호출 그래프의 각 노드인 함수의 특징 벡터는 상기 함수의 의미 정보를 표현하고 있으므로, 단순한 변형 만으로 악성 코드 탐지를 회피하는 것이 쉽지 않게 될 것이다.

[0090] 상기 제2 GNN 기반 분류 모델의 학습 과정에서, 학습 데이터에 포함된 각각의 실행 가능 파일에 대한 함수 호출 그래프의 표현을 가리키는 입력 특징 데이터가 상기 제2 GNN 기반 분류 모델에 입력되고 상기 제2 GNN 기반 분류 모델의 출력이 상기 실행 가능 파일의 악성 여부를 맞추는 방향으로 상기 제2 GNN 기반 분류 모델의 가중치가 업데이트 될 것이다.

[0091] 상기 제1 GNN 기반 분류 모델 및 상기 제2 GNN 기반 분류 모델 중 적어도 일부는 GIN 레이어(Graph Isomorphism Network layer)를 포함할 수 있다. 상기 GIN 레이어는, GCN(Graph Convolutional Network)에 비하여 그래프 간의 구조적 차이를 임베딩에 보다 정확히 반영할 수 있다.

[0092] 몇몇 실시예에서, 상기 GIN 레이어의 레이어 수는 N개(N은 자연수) 이하로 제한된다. 상기 N은 5이하일 수 있으며, 예를 들어 3일 수 있다. 악성 코드의 경우 악성 코드의 성질을 나타내는 특징은 매우 작은 영역의 명령어 혹은 적은 수의 노드 간 연결 구조에서 나타날 것으로 생각된다. 악성 코드의 특징이 넓은 영역에 존재하면 탐지가 용이해지기 때문이다. 따라서, 다수의 레이어로 GIN 레이어를 구성하는 것은 자원의 낭비를 가져올 수 있을 것이다. 물론, 상기 GIN 레이어의 수는 탐지 정확도에 따라 조정될 수 있다. 상기 탐지 정확도가 높아질수록 상기 GIN 레이어의 수가 커질 수 있을 것이다.

[0093] 2) PV-DM 모델

[0094] 이하, PV-DM 모델의 학습과 관련하여 설명한다.

[0095] 상기 PV-DM 모델은 함수 임베딩이 함수에 속한 토큰들의 토큰 임베딩을 반영할 수 있도록 상기 함수 임베딩을 최적화 시킨다. 상기 함수를 하나의 문서라면, 상기 함수에 대응되는 아티클의 토큰들은 문성 포함된 word들이 될 것이다. 상기 문서의 임베딩이 문서에 속한 word들을 반영하여 최적화되는 것처럼, 상기 함수 임베딩이 함수에 속한 토큰들의 토큰 임베딩을 반영할 수 있도록 최적화되는 것으로 이해될 수 있을 것이다. 상기 PV-DM 모델의 기계 학습 과정에서, 학습 데이터에 포함된 각각의 실행 가능 파일들이 이용될 수 있다.

[0096] 함수에 대응되는 아티클은 상기 함수의 CFG와 상기 함수의 어셈블리 코드를 이용하여 구성될 수 있다. 함수에 대응되는 아티클은 상기 함수의 CFG에 대한 랜덤 워크의 결과로 선택된 토큰들과, 상기 함수의 CFG에서 랜덤 샘플링 된 엣지(edge)에 연결된 2개의 명령어에 대응되는 토큰들을 선정하는 것을 반복하는 과정에서 선택된 토큰들 중 적어도 하나를 포함하여 구성될 수 있다.

[0097] 상술한 바와 같이 구성된, 함수 대응 아티클에 포함된 토큰들을 기반으로, PV-DM 모델을 이용한 함수 임베딩 최적화가 수행된다. 상기 최적화는 상기 함수 임베딩의 업데이트가 반복되는 것을 의미할 수 있으며, 이러한 업데이트는 아티클에 포함된 각각의 토큰들에 대한 윈도우(window)를 쉬프트 해가면서 수행될 수 있다. 예를 들어, 상기 윈도우는 상기 아티클의 시작 지점부터 끝지점까지 쉬프트 될 것이다. 상기 윈도우가 상기 아티클의 끝지점에 도달하여 상기 업데이트가 마무리 되면, 상기 함수 임베딩 값이 최적화 된 것으로 이해될 수 있을 것이다. 몇몇 실시예에서, epoch가 1을 초과하도록 파라미터 설정된 경우, 상기 아티클에 대하여 epoch 만큼의 윈도우 순회가 발생될 수 있다.

[0098] 상기 윈도우의 사이즈가 3인 경우를 예시적으로 설명한다. 상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록 상기 함수 임베딩 및 상기 윈도우의 각 토큰의 토큰 임베딩이 업데이트될 수 있다. 즉, PV-DM 모델에 포함된 매트릭스가 업데이트 됨으로써, 상기 윈도우의 두 번째 토큰과 매치되도록 상기 함수 임베딩 및 상기 윈도우의 각 토큰의 토큰 임베딩이 업데이트될 수 있을 것이다.

[0099] 요구되는 탐지 정확도에 따라 윈도우의 사이즈가 조정될 수 있다. 즉, 높은 수준의 탐지 정확도에 대응하기 위

하여는 시간이 더 걸리더라도 큰 사이즈의 윈도우를 적용하여 PV-DM 모델이 학습될 것이고, 낮은 수준의 탐지 정확도에 대응하기 위하여는 작은 사이즈의 윈도우를 적용하여 PV-DM 모델이 학습될 것이다.

[0100] 3) CNN 기반 분류 모델

[0101] 상기 CNN 기반의 분류 모델은 함수의 의미 정보 기반 악성 코드 탐지 과정에 사용될 수 있다. 상기 CNN 기반의 분류 모델의 학습을 위하여, 학습 데이터에 포함된 각각의 실행 가능 파일에 속한 각각의 함수 마다 PV-DM 모델을 이용한 함수 임베딩 최적화가 수행될 수 있으며, 최적화된 함수 임베딩이 함수의 특징 벡터로 이용된다.

[0102] 상기 각각의 함수의 특징 벡터들을 이용하여 상기 CNN 기반 분류 모델의 입력 특징 벡터가 얻어진다. 예를 들어, 상기 함수의 특징 벡터가 n 차원으로 구성되고, 상기 실행 가능 파일의 함수의 개수가 m개라면, 상기 CNN 기반 분류 모델의 입력 특징 벡터는 각각의 함수 특징 벡터를 포함하는 (m X n) 매트릭스 데이터로 구성될 수 있을 것이다.

[0103] 이 때, 상기 입력 특징 벡터에 각각의 함수 특징 벡터가 포함되는 순서는 각 함수의 주소 값에 따를 수 있다. 예를 들어, 각 함수의 주소가 증가하는 순서로 함수 특징 벡터가 배열됨으로써 상기 입력 특징 벡터가 구성될 수 있을 것이다. 이 때, 먼저 실행될 가능성이 높은 함수의 특징 벡터를 CNN 레이어가 초기에 학습하게 됨으로써, 상기 CNN 기반의 분류 모델의 학습 속도가 증가되는 효과를 얻을 수 있다.

[0104] 물론, 몇몇 실시예에서는, 각 함수의 주소가 감소하는 순서로 함수 특징 벡터가 배열됨으로써 상기 입력 특징 벡터가 구성될 수도 있다.

[0105] 상기 CNN 기반의 분류 모델의 학습 과정에 수행되는 컨볼루션 연산 과정에서, 커널(kernel)은 상기 입력 특징 벡터를 1차원 상에서 이동(traverse)할 수 있다. 상기 입력 특징 벡터를 구성하는 2개의 축(단일 함수의 특징 벡터의 인덱스 이동, 서로 다른 함수로의 이동) 중에서 단일 함수의 특징 벡터에서의 인덱스 이동은 그 의미가 크지 않기 때문이다. 즉, 상기 CNN 기반의 분류 모델은 1D 컨볼루션 연산에 의하여 학습된 1D CNN 기반의 분류 모델일 수 있다.

[0106] 지금까지 사전 트레이닝 과정(S130)을 설명하였다. 도 4에 도시된 방법은 추론 과정에서 수행되는 방법이며, 트레이닝 과정과 추론 과정의 유사성에 따라 트레이닝 과정에서 수행된 동작들이 동일하게 수행될 수 있는 바, 중복된 설명은 생략해 가면서, 도 4의 악성 코드 탐지 방법을 마저 설명하기로 한다.

[0107] PE 파일이 입력되면(S131), PE 파일을 대상으로 한 디스어셈블리가 수행되어 상기 PE 파일의 어셈블리 코드가 얻어지고, 상기 PE 파일의 CFG가 추출된다(S132).

[0108] 다음으로, CFG의 각 베이직 블록에 대한 특징 벡터가 상술한 방식으로 추출된다. 즉, 베이직 블록의 아티클이 구성되고, 상기 아티클에 포함된 토큰 각각의 토큰 임베딩이 상기 토큰 사전에서 조회된다. 이 때, 상기 토큰은 정규화 된 후 상기 토큰 사전에서 조회될 수 있다. 상기 토큰이 상기 토큰 사전에서 조회 되지 않은 경우, 그 토큰 임베딩은 0으로 결정될 것이다. 또한, 상기 베이직 블록의 특징 벡터는 최적화된 토큰 임베딩들의 대표값을 이용하여 결정될 수 있다.

[0109] 다음으로, 각 베이직 블록에 대한 특징 벡터를 반영한, CFG의 표현(representation)이 생성되고(S134), 상기 CFG의 표현에서 입력 특징 벡터가 생성된다(S135). 또한, 생성된 입력 특징 벡터가 상기 제1 GNN 기반 그래프 분류 모델에 입력된다(S136).

[0110] 마지막으로, 상기 제1 GNN 기반 그래프 분류 모델에서 출력된 데이터를 이용하여 상기 PE 파일의 악성 코드 여부가 판정된다(S137).

[0111] 다음으로, 도 6을 참조하여 함수의 의미 정보에 기반한 제1 악성 코드 탐지 방법을 설명한다. 사전 트레이닝 단계(S150)는 도 4를 참조한 설명을 참조한다.

[0112] PE 파일이 입력되면(S151), PE 파일을 대상으로 한 디스어셈블리가 수행되어 상기 PE 파일의 어셈블리 코드가 얻어지고, 상기 PE 파일의 함수 단위 CFG가 추출된다(S152). 또한, 상기 PE 파일에 대하여 함수의 호출 그래프(Call Graph)가 추출될 수 있다.

[0113] 다음으로, 상기 PE 파일에 포함된 각각의 함수에 대한 특징 벡터가 상술한 방식으로 추출된다(S153). 즉, 함수의 아티클이 구성되고, 상기 아티클에 포함된 토큰 각각의 토큰 임베딩이 상기 토큰 사전에서 조회된다. 상기 아티클은 상기 함수의 어셈블리 코드, 상기 함수의 CFG에 대하여 랜덤 워크(random walk)를 수행한 결과, 상기 함수의 CFG에서 샘플링된 엣지에 연결된 인스트럭션들을 이용하여 구성될 수 있는 점은 이미 설명한 바 있다.

이 때, 상기 토큰은 정규화 된 후 상기 토큰 사전에서 조회될 수 있다. 상기 토큰이 상기 토큰 사전에서 조회되지 않은 경우, 그 토큰 임베딩은 0으로 결정될 것이다.

[0114] 또한, 상기 함수의 특징 벡터는 함수의 식별 정보를 이용하여 초기 세팅 된 함수 임베딩이 PV-DM과 같은 Word2Vec 모델에 의하여 최적화된 것일 수 있다. 상기 최적화 과정에서, 상기 함수 임베딩이 함수의 아티클 내 토큰의 토큰 임베딩들을 반영하게 됨으로써, 함수의 아티클에 속한 토큰에 대한 정보를 반영하게 되는 점은 이미 설명한 바 있다. 즉 상기 아티클에 포함된 3개의 연속적인 토큰으로 구성되는 윈도우가 세팅되고, 상기 함수 임베딩, 상기 윈도우의 첫 번째 토큰 및 상기 윈도우의 세 번째 토큰의 평균치가, 상기 윈도우의 두 번째 토큰과 매치되도록, 상기 함수 임베딩이 업데이트 될 수 있으며, 상기 아티클의 마지막에 도달할 때까지 상기 윈도우를 쉬프트하면서 상기 세팅하는 단계 및 상기 업데이트 하는 단계가 반복될 수 있을 것이다. 이 때, 상기 PV-DM 모델의 파라미터 및 상기 윈도우의 토큰이 고정된 상태에서 상기 함수 임베딩이 업데이트 될 것이다.

[0115] 다음으로, 각 함수의 특징 벡터를 반영한, 함수 호출 그래프의 표현(representation)이 생성되고(S154), 상기 함수 호출 그래프의 표현에서 입력 특징 벡터가 생성된다(S155). 또한, 생성된 입력 특징 벡터가 상기 제2 GNN 기반 그래프 분류 모델에 입력된다(S156).

[0116] 마지막으로, 상기 제2 GNN 기반 그래프 분류 모델에서 출력된 데이터를 이용하여 상기 PE 파일의 악성 코드 여부가 판정된다(S157).

[0117] 다음으로, 도 7을 참조하여 함수의 의미 정보에 기반한 제2 악성 코드 탐지 방법을 설명한다. 사전 트레이닝 단계(S160)는 도 4를 참조한 설명을 참조한다.

[0118] PE 파일이 입력되면(S161), PE 파일을 대상으로 한 디스어셈블리가 수행되어 상기 PE 파일의 어셈블리 코드가 얻어지고, 상기 PE 파일의 함수 단위 CFG가 추출된다(S162).

[0119] 다음으로, 상기 PE 파일에 포함된 각각의 함수에 대한 특징 벡터가 도 6을 참조하여 상술한 방식으로 추출된다(S163). 다음으로, 각 함수의 특징 벡터들로 입력 특징 벡터가 생성된다(S164). 이 때, 각 함수의 특징 벡터들을 함수의 주소 값을 기준으로 정렬된 순서로 배치함으로써 상기 입력 특징 벡터가 생성될 수 있음은 상술한 바와 같다. 생성된 입력 특징 벡터는 상기 CNN 기반 그래프 분류 모델에 입력된다(S165). 마지막으로, 상기 CNN 기반 그래프 분류 모델에서 출력된 데이터를 이용하여 상기 PE 파일의 악성 코드 여부가 판정된다(S166).

[0120] 몇몇 실시예에서, 도 8에 도시된 바와 같이, 함수 의미 정보 기반의 악성 코드 탐지 방법은 도 6을 참조하여 설명된 제1 방법과 도 7을 참조하여 설명된 제2 방법이 선택적으로 수행될 수 있다. 즉, 입력된 PE 파일(S161)에 대하여, 함수 호출 그래프의 추출 성공시(S162-1) 도 6을 참조하여 설명된 제1 방법(S1500)에 따라 GNN 기반의 악성 코드 탐지 모델이 이용되고, 함수 호출 그래프의 추출 실패시(S162-1) 도 7을 참조하여 설명된 제2 방법(S1600)에 따라 CNN 기반의 악성 코드 탐지 모델이 이용될 수 있을 것이다.

[0121] 지금까지 도 1 내지 도 8을 참조하여 설명된 본 개시의 기술적 사상은 컴퓨터가 읽을 수 있는 매체 상에 컴퓨터가 읽을 수 있는 코드로 구현될 수 있다. 상기 컴퓨터로 읽을 수 있는 기록 매체는, 예를 들어 이동형 기록 매체(USB 저장 장치, 이동식 하드 디스크)일 수 있다. 상기 컴퓨터로 읽을 수 있는 기록 매체에 기록된 상기 컴퓨터 프로그램은 인터넷 등의 네트워크를 통하여 다른 컴퓨팅 장치에 전송되어 상기 다른 컴퓨팅 장치에 설치될 수 있고, 이로써 상기 다른 컴퓨팅 장치에서 사용될 수 있다.

[0122] 이하, 본 개시의 몇몇 실시예들에 따른 예시적인 컴퓨팅 장치의 하드웨어 구성을 도 9를 참조하여 설명하기로 한다. 상기 컴퓨팅 장치는, 예를 들어 도 1을 참조하여 설명한 서비스 서버(100)일 수 있다.

[0123] 도 9는 본 개시의 다양한 실시예에서 컴퓨팅 장치를 구현할 수 있는 예시적인 하드웨어 구성도이다. 본 실시예에 따른 컴퓨팅 장치(1000)는 프로세서(1100), 시스템 버스(1600), 통신 인터페이스(1200), 프로세서(1100)에 의하여 수행되는 컴퓨터 프로그램(1500)을 로드(load)하는 메모리(1400)와, 컴퓨터 프로그램(1500)을 저장하는 스토리지(1300)를 포함할 수 있다. 도 9에는 본 개시의 실시예와 관련 있는 구성요소들 만이 도시되어 있다. 따라서, 본 개시에 속한 기술분야의 통상의 기술자라면 도 9에 도시된 구성요소들 외에 다른 범용적인 구성 요소들이 더 포함될 수 있음을 알 수 있다.

[0124] 프로세서(1100)는 컴퓨팅 장치(2000)의 각 구성의 전반적인 동작을 제어한다. 프로세서(1100)는 CPU(Central Processing Unit)로 이해될 수 있을 것이다. 또한, 프로세서(1100)는 본 개시의 다양한 실시예들에 따른 방법/동작을 실행하기 위한 적어도 하나의 애플리케이션 또는 프로그램에 대한 연산을 수행할 수 있다. 몇몇 실시예에서, 컴퓨팅 장치(1000)는 GPU(Graphics Processing Unit)(1150)를 더 포함할 수 있다. 기계 학습(machine

learning)과 관련된 연산 등은 프로세서(1100)가 아닌 GPU(1150)를 통하여 실행될 수 있다.

[0125] 메모리(1400)는 각종 데이터, 명령 및/또는 정보를 저장한다. 메모리(1400)는 본 개시의 다양한 실시예들에 따른 방법/동작들을 실행하기 위하여 스토리지(1300)로부터 하나 이상의 프로그램(190)을 로드(load) 할 수 있다. 메모리(1400)의 예시는 RAM(Random Access Memory)이 될 수 있으나, 이에 한정되는 것은 아니다.

[0126] 시스템 버스(1600)는 컴퓨팅 장치(1000)의 구성 요소 간 통신 기능을 제공한다. 시스템 버스(1600)는 주소 버스(Address Bus), 데이터 버스(Data Bus) 및 제어 버스(Control Bus) 등 다양한 형태의 버스로 구현될 수 있다. 통신 인터페이스(1200)는 컴퓨팅 장치(1000)의 유무선 인터넷 통신을 지원한다. 통신 인터페이스(1200)는 인터넷 통신 외의 블루투스(Bluetooth) 등 근거리 무선 통신 방식을 지원할 수도 있다.

[0127] 컴퓨팅 장치(1000)는 PE 파일 등으로 구성된 학습 데이터가 저장된 데이터베이스 서버(미도시) 및 상기 학습 데이터에 대한 분석 결과로 추출된 각 토큰에 대한 데이터를 포함하는 토큰 사전의 데이터가 저장된 데이터베이스 서버(미도시) 중 적어도 하나와 통신 인터페이스(1200)를 통하여 연결될 수 있다. 또한, 컴퓨팅 장치(1000)는 사용자 단말과 통신 인터페이스(1200)를 통하여 연결될 수 있다.

[0128] 스토리지(1300)는 하나 이상의 컴퓨터 프로그램(1500)을 비임시적으로 저장할 수 있다. 스토리지(1300)는 플래시 메모리 등과 같은 비휘발성 메모리, 하드 디스크, 착탈형 디스크, 또는 본 개시가 속하는 기술 분야에서 잘 알려진 임의의 형태의 컴퓨터로 읽을 수 있는 기록 매체를 포함하여 구성될 수 있다.

[0129] 컴퓨터 프로그램(1500)은 본 개시의 다양한 실시예들에 따른 방법/동작들이 구현된 하나 이상의 인스트럭션들을 포함할 수 있다. 컴퓨터 프로그램(1500)이 메모리(1400)에 로드 되면, 프로세서(1100)는 상기 하나 이상의 인스트럭션들을 실행시킴으로써 본 개시의 다양한 실시예들에 따른 방법들을 수행할 수 있다.

[0130] 컴퓨터 프로그램(1500)은 악성 코드 탐지 대상인 실행 가능 파일을 정적 분석 방식으로 분석하고, 상기 분석 결과를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 판정하는 악성 코드 탐지 프로그램이다. 상기 실행 가능 파일은 스토리지(1300)에 저장된 파일이거나, 통신 인터페이스(1200)를 통하여 수신된 파일일 수 있다.

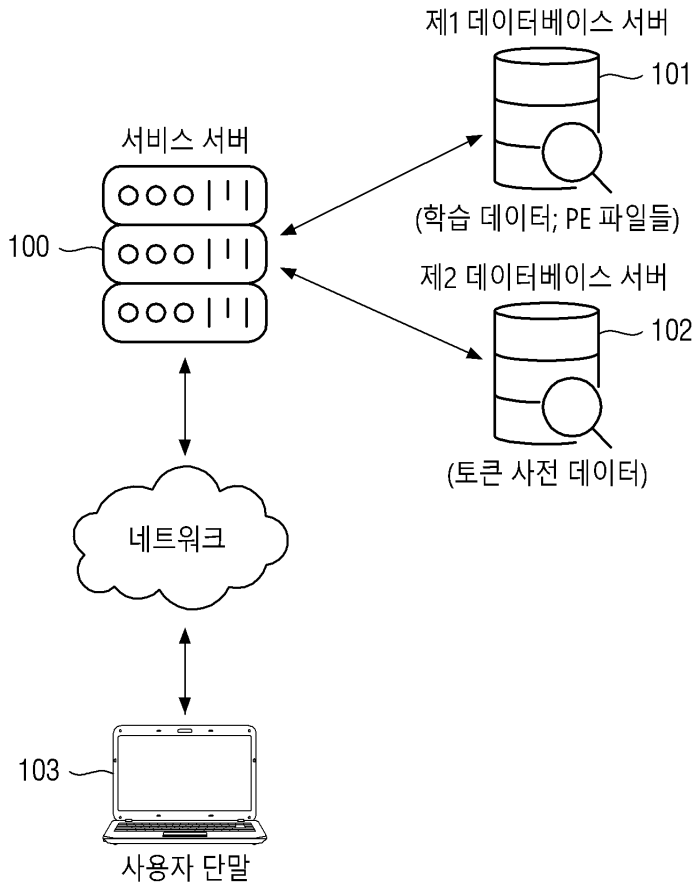
[0131] 컴퓨터 프로그램(1500)은, 악성 코드 탐지 대상인 실행 가능 파일의 CFG(Control Flow Graph)에 포함된 베이직 블록 각각에 대하여, 토큰 임베딩(token embedding) 기반 제1 특징 벡터를 얻고, 상기 CFG의 각 베이직 블록 간 연결 관계와 각 베이직 블록의 상기 제1 특징 벡터를 이용하여 상기 CFG를 표현하는 제1 입력 특징 벡터를 생성하는 제1 인스트럭션(instruction)과, 상기 실행 가능 파일에 포함된 함수 각각에 대한 제2 특징 벡터를 얻는 제2 인스트럭션과, 상기 실행 가능 파일의 함수 호출 그래프(function call graph)에 포함된 각 함수 간 연결 관계와, 각 함수의 상기 제2 특징 벡터를 이용하여 상기 함수 호출 그래프를 표현하는 제2 입력 특징 벡터를 생성하는 제3 인스트럭션과, 상기 실행 가능 파일에 포함된 함수 각각에 대한 상기 제2 특징 벡터들을 포함하는 제3 입력 특징 벡터를 생성하는 제4 인스트럭션과, 상기 제1 입력 특징 벡터를 입력 받은 제1 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제5 인스트럭션과, 상기 제2 입력 특징 벡터를 입력 받은 제2 GNN(Graph Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제6 인스트럭션과, 상기 제3 입력 특징 벡터를 입력 받은 CNN(Convolutional Neural Network) 기반 그래프 분류 모델의 출력 데이터를 이용하여 상기 실행 가능 파일의 악성 코드 여부를 결정하는 제7 인스트럭션을 포함할 수 있다.

[0132] 상기 악성 코드 탐지 프로그램은, 상기 실행 가능 파일에서 상기 함수 호출 그래프를 추출하는 것이 성공하면 상기 제3 인스트럭션과 상기 제6 인스트럭션으로 구성된 제1 그룹 인스트럭션을 실행하고, 상기 함수 호출 그래프를 추출하는 것이 실패하면 상기 제4 인스트럭션과 상기 제7 인스트럭션으로 구성된 제2 그룹 인스트럭션을 실행하는 것일 수 있다. 즉, 상기 악성 코드 탐지 프로그램은 상기 제1 그룹 인스트럭션과 상기 제2 그룹 인스트럭션 중 어느 하나를 선택적으로 실행할 수 있는 것이다.

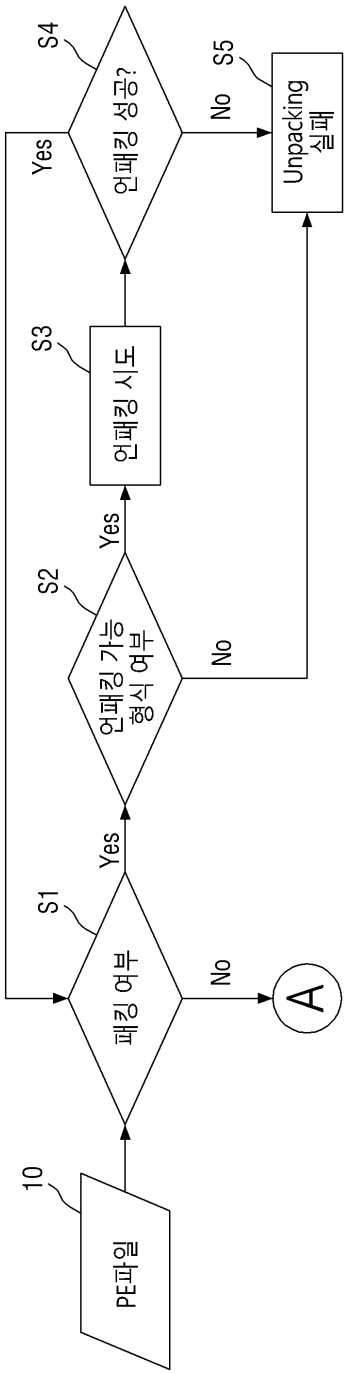
[0133] 이상 첨부된 도면을 참조하여 본 개시의 실시예들을 설명하였지만, 본 개시의 실시예들이 속하는 기술분야에서 통상의 지식을 가진 자는 그 기술적 사상이나 필수적인 특징을 변경하지 않고서 본 개시의 실시예들이 다른 구체적인 형태로도 실시될 수 있다는 것을 이해할 수 있다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적인 것이 아닌 것으로 이해해야만 한다. 본 발명의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 개시에 의해 정의되는 기술적 사상의 권리 범위에 포함되는 것으로 해석되어야 할 것이다.

도면

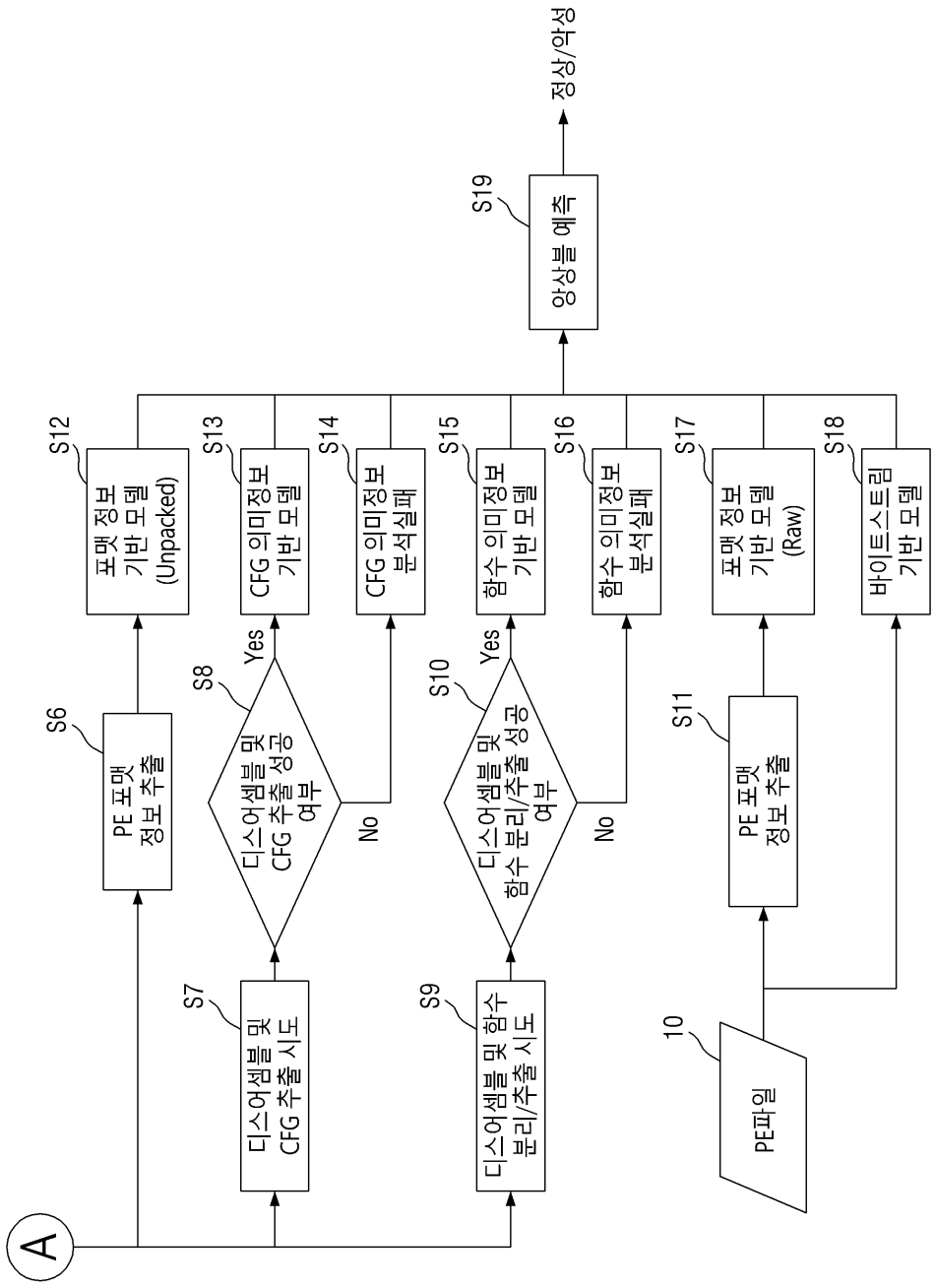
도면1


제1 데이터베이스 서버
101
서비스 서버
100
(학습 데이터; PE 파일들)
제2 데이터베이스 서버
102
(토큰 사전 데이터)
네트워크
103
사용자 단말

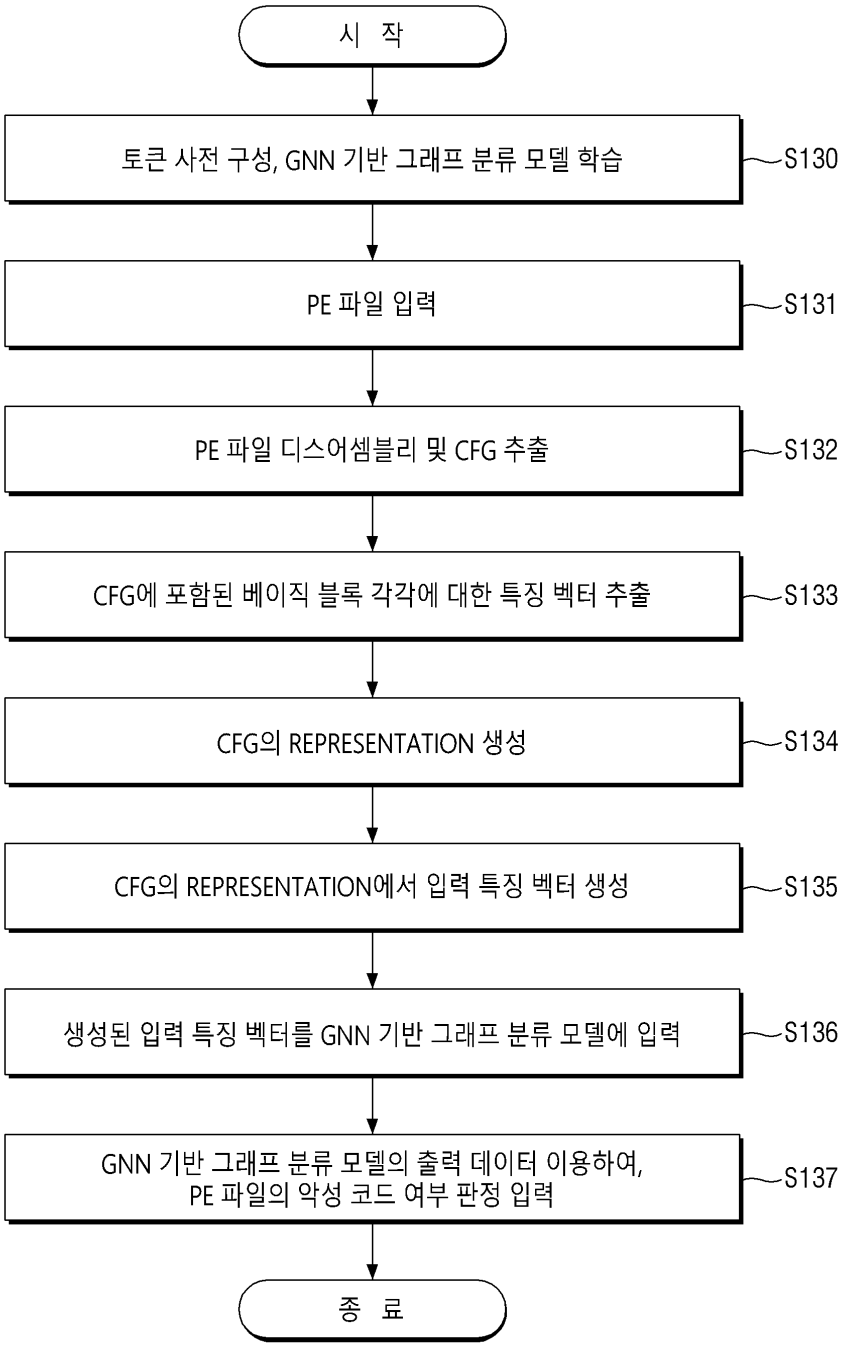
도면2


10
PE파일
S1
패킹 여부
Yes
No
A
S2
언패킹 가능 형식 여부
Yes
No
S3
언패킹 시도
S4
언패킹 성공?
Yes
No
S5
Unpacking 실패

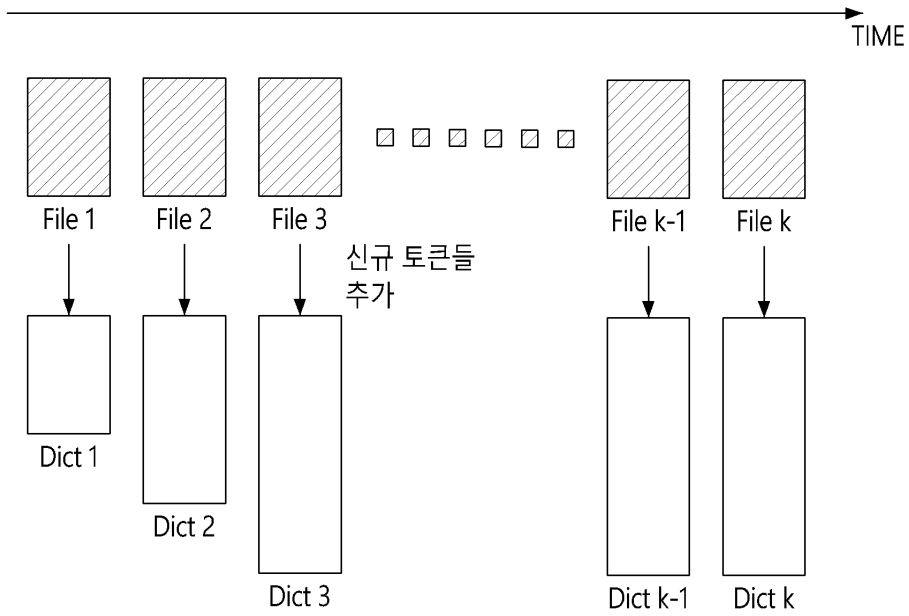
도면3


정상/악성
S19
앙상블 예측
S12
포맷 정보 기반 모델 (Unpacked)
S13
CFG 의미정보 기반 모델
S14
CFG 의미정보 분석실패
S15
함수 의미정보 기반 모델
S16
함수 의미정보 분석실패
S17
포맷 정보 기반 모델 (Raw)
S18
바이트스트림 기반 모델
S6
PE 포맷 정보 추출
S8
디스어셈블 및 CFG 추출 성공 여부
Yes
No
S10
디스어셈블 및 함수 분리/추출 성공 여부
Yes
No
S11
PE 포맷 정보 추출
S7
디스어셈블 및 CFG 추출 시도
S9
디스어셈블 및 함수 분리/추출 시도
10
PE파일
A

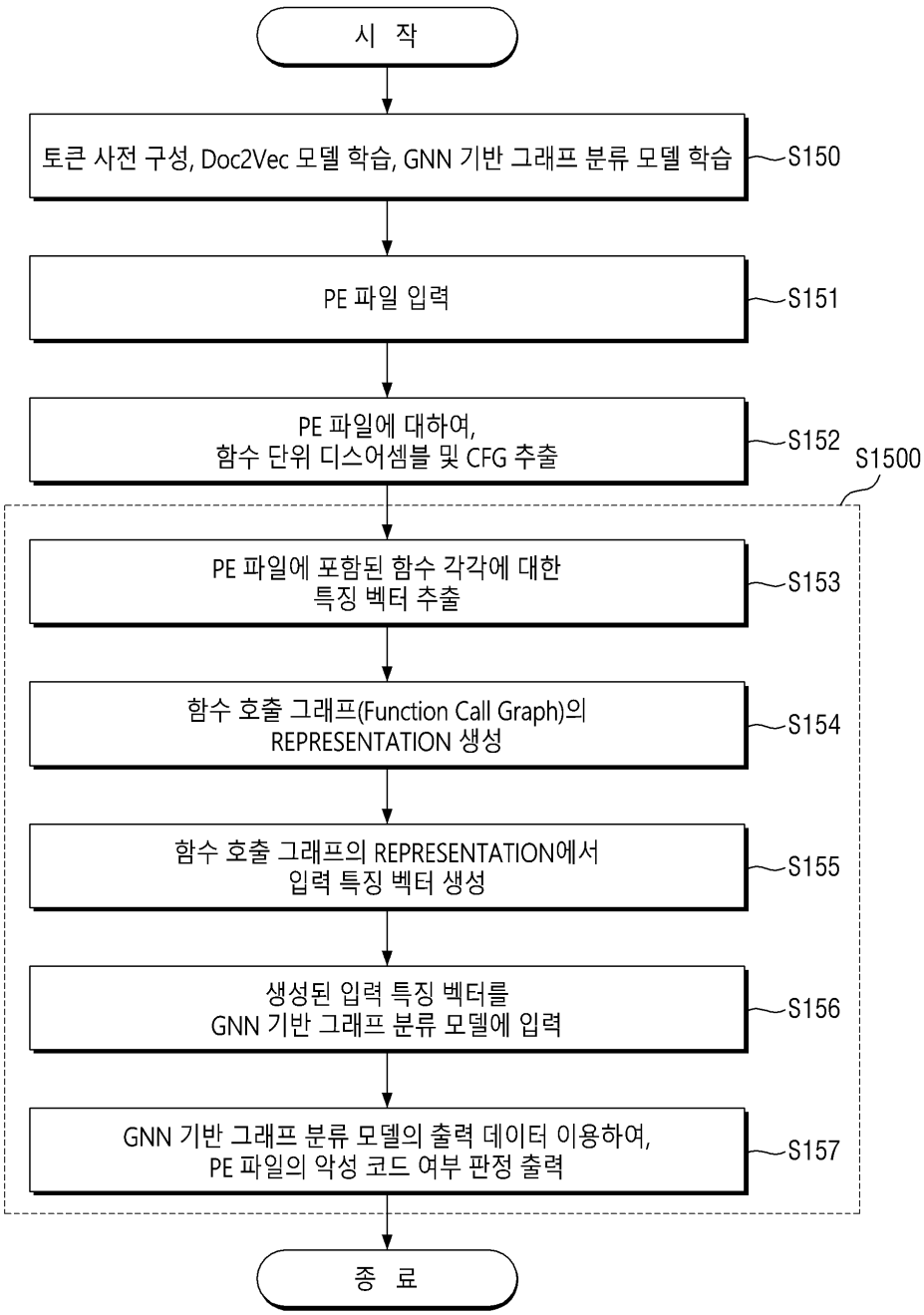
도면4


시 작
토큰 사전 구성, GNN 기반 그래프 분류 모델 학습
S130
PE 파일 입력
S131
PE 파일 디스어셈블리 및 CFG 추출
S132
CFG에 포함된 베이직 블록 각각에 대한 특징 벡터 추출
S133
CFG의 REPRESENTATION 생성
S134
CFG의 REPRESENTATION에서 입력 특징 벡터 생성
S135
생성된 입력 특징 벡터를 GNN 기반 그래프 분류 모델에 입력
S136
GNN 기반 그래프 분류 모델의 출력 데이터 이용하여,
PE 파일의 악성 코드 여부 판정 입력
S137
종 료

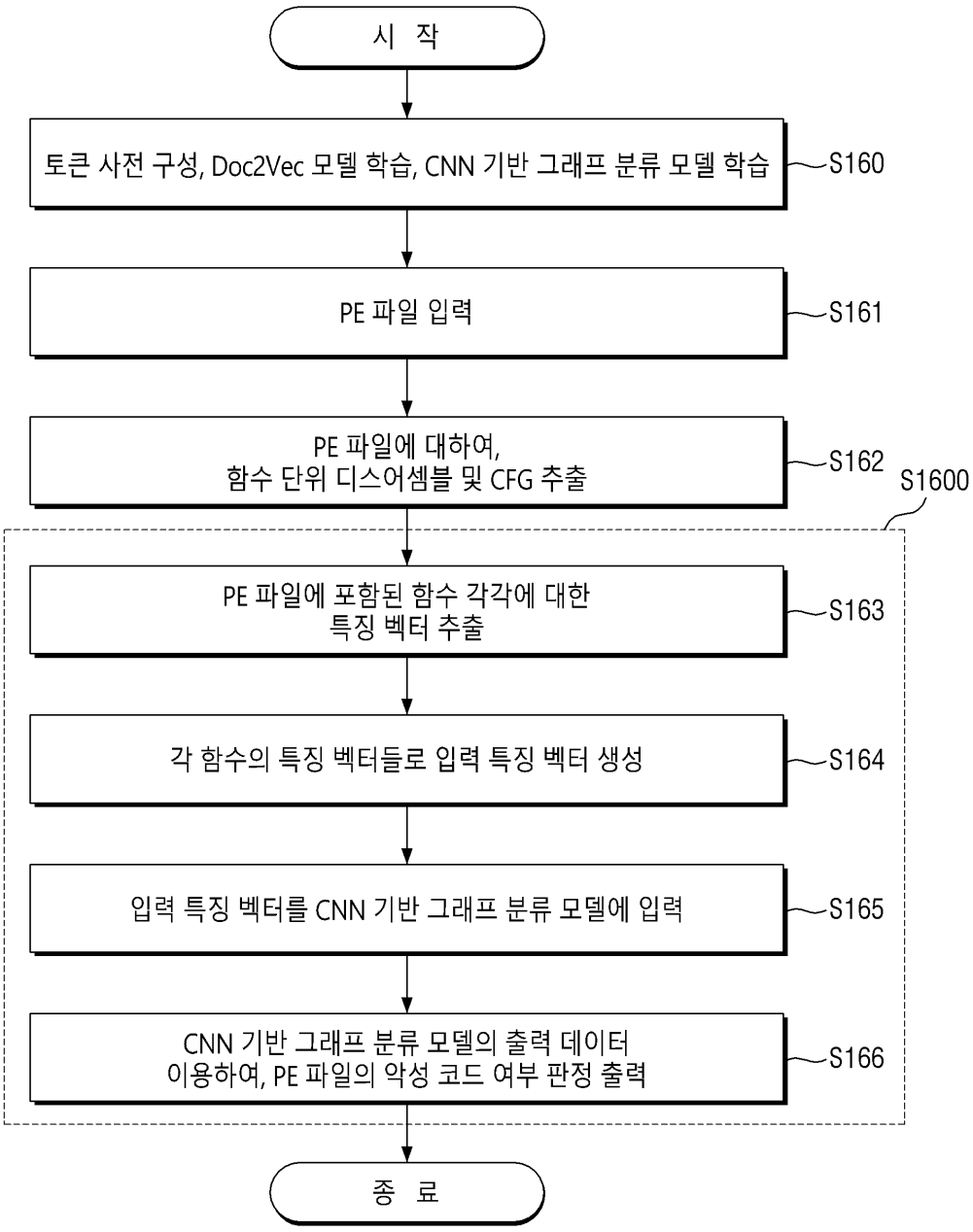
도면5


TIME
File 1
File 2
File 3
신규 토큰들
추가
File k-1
File k
Dict 1
Dict 2
Dict 3
Dict k-1
Dict k

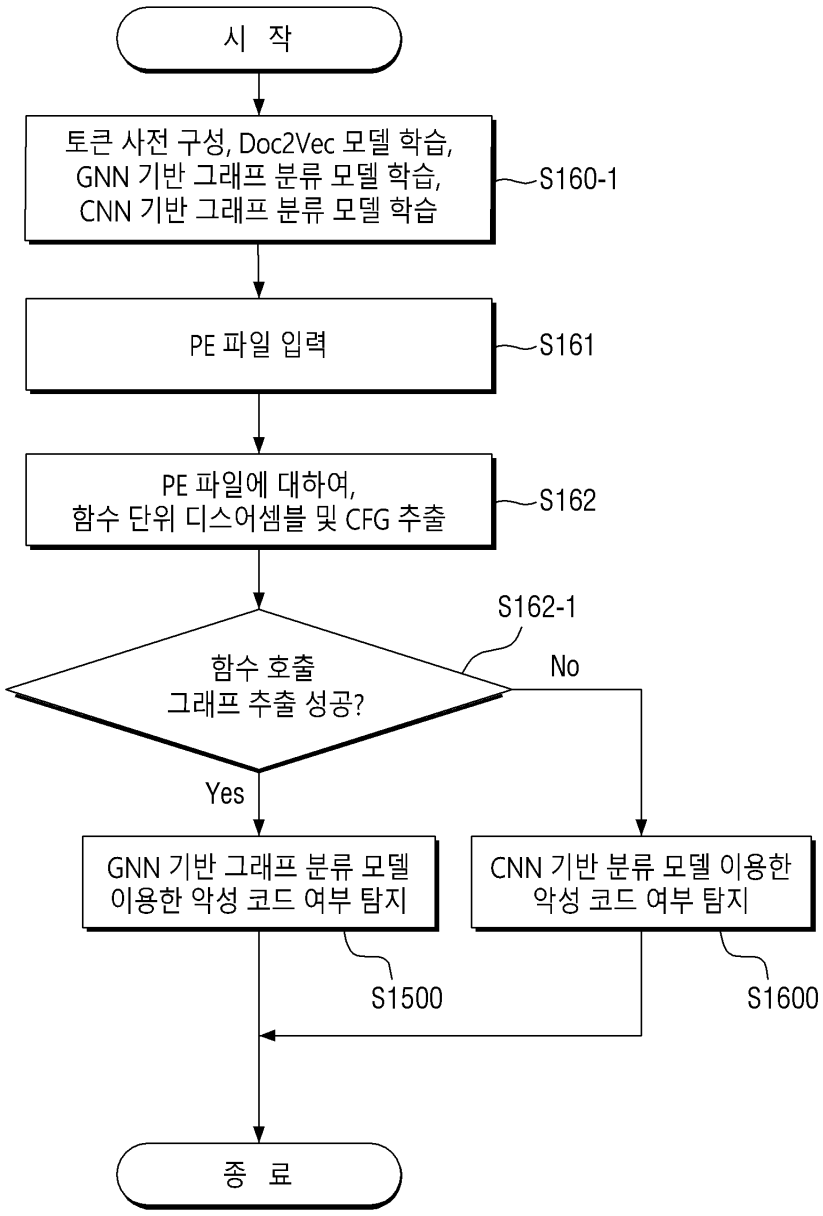
도면6


시 작
토큰 사전 구성, Doc2Vec 모델 학습, GNN 기반 그래프 분류 모델 학습
S150
PE 파일 입력
S151
PE 파일에 대하여,
함수 단위 디스어셈블 및 CFG 추출
S152
S1500
PE 파일에 포함된 함수 각각에 대한
특징 벡터 추출
S153
함수 호출 그래프(Function Call Graph)의
REPRESENTATION 생성
S154
함수 호출 그래프의 REPRESENTATION에서
입력 특징 벡터 생성
S155
생성된 입력 특징 벡터를
GNN 기반 그래프 분류 모델에 입력
S156
GNN 기반 그래프 분류 모델의 출력 데이터 이용하여,
PE 파일의 악성 코드 여부 판정 출력
S157
종 료

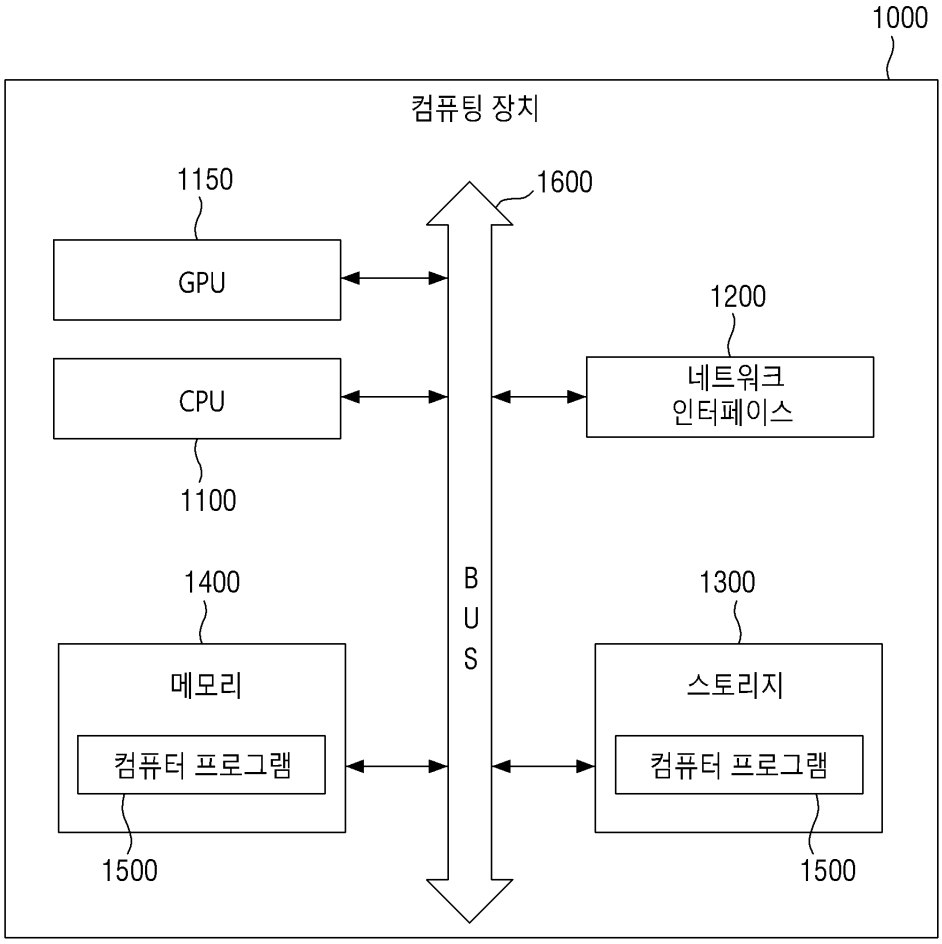
도면7


시 작
토큰 사전 구성, Doc2Vec 모델 학습, CNN 기반 그래프 분류 모델 학습
S160
PE 파일 입력
S161
PE 파일에 대하여,
함수 단위 디스어셈블 및 CFG 추출
S162
S1600
PE 파일에 포함된 함수 각각에 대한
특징 벡터 추출
S163
각 함수의 특징 벡터들로 입력 특징 벡터 생성
S164
입력 특징 벡터를 CNN 기반 그래프 분류 모델에 입력
S165
CNN 기반 그래프 분류 모델의 출력 데이터
이용하여, PE 파일의 악성 코드 여부 판정 출력
S166
종 료

도면8


시 작
토큰 사전 구성, Doc2Vec 모델 학습,
GNN 기반 그래프 분류 모델 학습,
CNN 기반 그래프 분류 모델 학습
S160-1
PE 파일 입력
S161
PE 파일에 대하여,
함수 단위 디스어셈블 및 CFG 추출
S162
S162-1
함수 호출
그래프 추출 성공?
No
Yes
GNN 기반 그래프 분류 모델
이용한 악성 코드 여부 탐지
CNN 기반 분류 모델 이용한
악성 코드 여부 탐지
S1500
S1600
종 료

도면9


1000
컴퓨팅 장치
1150
GPU
CPU
1100
1600
BUS
1200
네트워크
인터페이스
1400
메모리
컴퓨터 프로그램
1500
1300
스토리지
컴퓨터 프로그램
1500