(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2023년04월26일
(11) 등록번호 10-2526302
(24) 등록일자 2023년04월24일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01)
(52) CPC특허분류
G06F 11/3664 (2013.01)
G06F 11/3624 (2013.01)
(21) 출원번호 10-2021-0157211
(22) 출원일자 2021년11월16일
심사청구일자 2021년11월16일
(56) 선행기술조사문헌
KR101568224 B1*
KR101881271 B1*
US20210034753 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
권태경
서울특별시 강남구 선릉로 221, 410동 1602호(도곡동, 도곡렉슬아파트)
안도현
대구광역시 수성구 달구벌대로 3280-1, 204동 601호(신매동, 시지 효성 백년가약 2단지)
(뒷면에 계속)
(74) 대리인
민영준

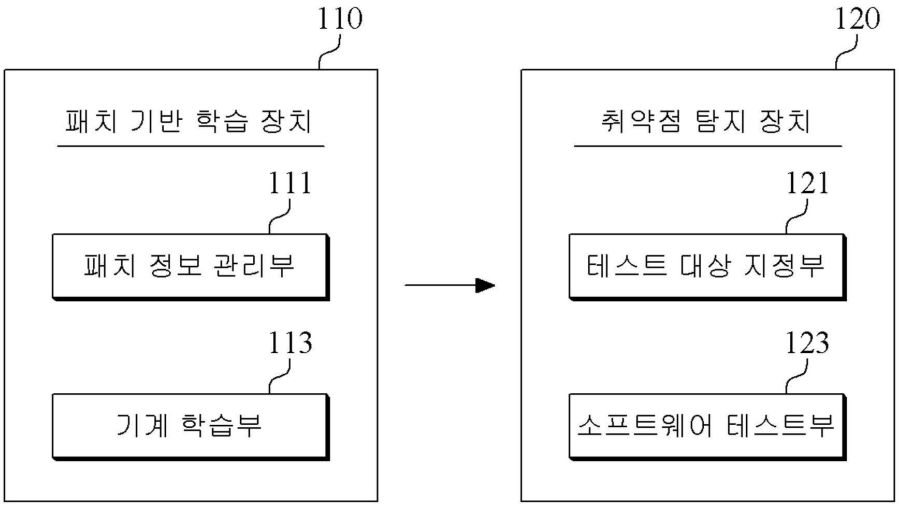
전체 청구항 수 : 총 8 항 심사관 : 장지혜

(54) 발명의 명칭 **소프트웨어 테스팅 방법 및 소프트 테스팅을 위한 취약점 분류 모델 생성 방법**

(57) 요 약

소프트웨어의 버그를 찾기 위한 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법이 개시된다. 개시된 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법은, 훈련용 소프트웨어의 버전별 소스 코드를 입력받는 단계; 상기 소스 코드의 취약점을 분석하여, 상기 소스 코드의 업데이트 전후에 대한 취약점 데이터를 생성하는 단계; 및 상기 업데이트 전후의 소스 코드의 차이점 및 상기 취약점 데이터를 이용하여, 취약점 분류 모델을 학습하는 단계를 포함한다.

대 표 도 - 도1



(52) CPC특허분류
G06F 11/368 (2013.01)
G06F 11/3684 (2013.01)
G06F 11/3688 (2013.01)
G06F 11/3692 (2013.01)

(72) 발명자
조민기
서울특별시 서대문구 증가로 26, 302호(연희동)

진호용
서울특별시 서대문구 연대동문길 113, 108호(대신동, 메이즈)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711126385
과제번호	2018-0-00513-004
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송연구개발사업
연구과제명	기계학습을 활용한 UNIX 기반 커널 취약점 탐지 자동화 연구
기 여 율	1/1
과제수행기관명	연세대학교 산학협력단
연구기간	2021.01.01 ~ 2021.12.31

명 세 서

청구범위

청구항 1

컴퓨팅 장치에 의해 수행되는, 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법에 있어서,

훈련용 소프트웨어의 버전별 소스 코드를 입력받는 단계;

상기 소스 코드의 취약점을 분석하여, 상기 소스 코드의 업데이트 전후에 대한 취약점 데이터를 생성하는 단계; 및

상기 업데이트 전후의 소스 코드의 차이점 및 상기 취약점 데이터를 이용하여, 취약점 분류 모델을 학습하는 단계를 포함하며,

상기 소스 코드의 취약점 데이터를 생성하는 단계는

상기 소스 코드에 포함된 업데이트 메시지 및 업데이트된 코드를 분석하여, 상기 취약점 데이터를 생성하는

소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법.

청구항 2

삭제

청구항 3

제 1항에 있어서,

상기 소스 코드의 취약점 데이터를 생성하는 단계는

객체 크기 검사 코드의 추가, 함수 네임의 변경 또는 메모리 오류 검출 함수 호출 여부를 이용하여, 상기 취약점 데이터를 생성하는

소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법.

청구항 4

제 1항에 있어서,

상기 취약점 분류 모델을 학습하는 단계는

상기 업데이트 전후의 소스 코드를 토큰화하여, 상기 차이점을 추출하는

소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법.

청구항 5

제 1항에 있어서,

상기 취약점 분류 모델을 학습하는 단계는

타겟 소프트웨어에 대한 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류하도록, 상기 취약점 분류 모델을 학습하는

소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법.

청구항 6

컴퓨팅 장치에 의해 수행되는, 소프트웨어 테스팅 방법에 있어서,

타겟 소프트웨어의 버전별 소스 코드를 입력받는 단계;

취약점 분류 모델을 이용하여, 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류하는 단계; 및

업데이트 후의 소스 코드에서, 상기 취약점이 존재하는 것으로 분류된 코드를 포함하는 테스트 목표 지점에 대한 테스트를 진행하는 단계를 포함하며,

상기 취약점 분류 모델은

훈련용 소프트웨어의 소스 코드의 업데이트 전후에 대한 취약점 데이터로부터 학습된 모델이며,

상기 취약점 데이터는, 상기 훈련용 소프트웨어의 소스 코드에 포함된 업데이트 메시지에 대한 분석을 통해 생성된 데이터인,

소프트웨어 테스팅 방법.

청구항 7

제 6항에 있어서,

상기 업데이트 후의 소스 코드에 포함된 업데이트 메시지를 분석하여, 상기 테스트 목표 지점를 선정하는 단계를 더 포함하는 소프트웨어 테스팅 방법.

청구항 8

제 6항에 있어서,

상기 테스트 목표 지점은

상기 취약점이 존재하는 것으로 분류된 코드를 포함하는 기본 블록 또는 함수인

소프트웨어 테스팅 방법.

청구항 9

제 6항에 있어서,

상기 테스트 목표 지점은

상기 취약점이 존재하는 것으로 분류된 코드에 포함된 객체 또는 변수와 동일한 객체 또는 변수를 포함하는 코드를 더 포함하는

소프트웨어 테스팅 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 소프트웨어의 버그를 찾기 위한 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류

모델 생성 방법에 관한 것으로서, 보다 상세하게는 미리 학습된 취약점 분류 모델을 이용하는 소프트웨어 테스팅 방법 및 소프트 테스팅을 위한 취약점 분류 모델 생성 방법에 관한 것이다.

배 경 기 술

[0003] 소프트웨어의 버그를 찾기 위해 취약 코드 패턴 매칭과 같은 정적 분석 기술과 퍼징(fuzzing), 콘콜릭(concolic) 실행과 같은 동적 분석 기술이 널리 사용되고 있다. 취약 코드 패턴 매칭은 사전에 알려진 취약점 코드들의 패턴을 분석하고 수집하여 이를 데이터베이스화한 뒤, 취약점 데이터베이스를 활용하여 새로운 코드에서의 취약점 존재 여부를 판단한다.

[0004] 하지만 취약점(vulnerability)이 존재하는 코드와 존재하지 않는 코드의 구분이 애매모호한 경우가 많고, 같은 코드에서도 프로그램의 흐름에 따라 취약점 유무가 달라질 수 있어 패턴 매칭과 같은 정적 분석 기술은 많은 수의 거짓 양성(false positive)이 발생한다. 이 한계점을 보완하여 탐지된 취약점 유무를 판단하기 위해 퍼징과 콘콜릭 실행과 같은 동적 분석 기술이 정적 분석 기술과 함께 활용된다.

[0005] 또 다른 취약 코드 패턴 매칭 방법의 한계점으로 취약점 패턴을 수집하기 위해 분석가의 많은 노력이 필요한 점이 있다. 데이터베이스를 생성하기 위한 취약점 정보는 주로 CVE(common vulnerabilities and exposures) 번호가 할당된 취약점에서 수집한다. 하지만 CVE 번호가 할당된 모든 취약점의 취약 코드가 공개되는 것이 아니고, CVE 번호가 할당되지 않은 취약점도 존재하기 때문에 분석가가 소프트웨어의 패치 기록을 분석하여 취약 코드를 직접 수집해야 하는 어려움이 있다.

[0006] 퍼징과 콘콜릭 실행은 새로운 테스트 입력을 생성하고 만약 생성된 테스트 입력을 사용하여 프로그램을 실행시켰을 때 프로그램이 비정상 동작을 일으킨다면 소프트웨어 내의 버그가 있다고 판단한다. 이때 프로그램의 비정상 동작은 프로그램이 강제로 종료되는 것을 말한다.

[0007] 이 같은 동적 분석 방법은 취약한 부분까지 도달하기 위하여 커버리지를 최대화하는 방향으로 발전하고 있으며 이를 위해 심볼릭, 콘콜릭 실행과 같이 제약 조건을 해결하는 방법론들이 제시되었다. 하지만 취약점이 존재하는 위치를 사전에 알 수 없으므로 단순히 동적 테스트의 코드 커버리지를 최대화하는 것이 취약점 발견율의 최대화로 이어지지는 않으며, 결과적으로 취약점의 존재를 알 수 없는 상태에서 비효율적인 동적 분석을 수행하게 된다.

[0008] 효율적인 퍼징을 수행하기 위해 목표 함수를 지정하여 퍼징을 수행하는 방법이 제시되었다. 2017년 ACM CCS에서 Marcel Bohme 등은 직접 퍼징(directed fuzzing)을 수행하는 방법인 AFLGo를 공개하였다. 기존 AFL과는 다르게 AFLGo는 테스트 대상과 관련 없는 프로그램 구성 요소를 구성하는데 리소스를 낭비하지 않고, 특정 대상 위치에 도달하는데 대부분의 자원을 사용한다. 하지만 AFLGo와 같은 직접 퍼징 방법을 사용하기 위해서는 테스트 목표 지점을 사용자가 대상 소프트웨어를 직접 분석하여 지정해야 하는 한계점이 존재한다. 소프트웨어의 크기가 커질수록 분석 난이도가 높아지고, 분석에 긴 시간이 소요되므로 크기가 큰 소프트웨어에는 적용하기 어려워진다.

[0009] 관련 선행문헌으로 대한민국 공개특허 제2021-0061893호, 대한민국 등록특허 제10-1963756호가 있다.

발명의 내용

해결하려는 과제

[0011] 본 발명은 소스 코드 관리 도구를 이용하는 모든 소프트웨어에 적용할 수 있는 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법을 제공하기 위한 것이다.

[0012] 또한 본 발명은 관리자의 개입없이 소스와 업데이트 로그만을 이용하여 소프트웨어의 취약점을 검증하고 취약점을 탐지할 수 있는 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법을 제공하기 위한 것이다.

[0013] 또한 본 발명은 패치 검증과 패치의 새로운 취약점 탐지를 동시에 수행할 수 있는 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법을 제공하기 위한 것이다.

과제의 해결 수단

[0015] 상기한 목적을 달성하기 위해 본 발명의 일실시예에 따르면, 훈련용 소프트웨어의 버전별 소스 코드를 입력받는 단계; 상기 소스 코드의 취약점을 분석하여, 상기 소스 코드의 업데이트 전후에 대한 취약점 데이터를 생성하는

단계; 및 상기 업데이트 전후의 소스 코드의 차이점 및 상기 취약점 데이터를 이용하여, 취약점 분류 모델을 학습하는 단계를 포함하는 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법이 제공된다.

[0016] 또한 상기한 목적을 달성하기 위해 본 발명의 다른 실시예에 따르면, 타겟 소프트웨어의 버전별 소스 코드를 입력받는 단계; 취약점 분류 모델을 이용하여, 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류하는 단계; 및 업데이트 후의 소스 코드에서, 상기 취약점이 존재하는 것으로 분류된 코드를 포함하는 테스트 목표 지점에 대한 테스트를 진행하는 단계를 포함하는 소프트웨어 테스팅 방법이 제공된다.

발명의 효과

[0018] 본 발명의 일실시예에 따르면, 테스트가 이루어지는 테스트 목표 지점을 자동으로 설정하여 테스트가 수행됨으로써, 테스트 목표 지점을 설정하는데 소요되는 인력과 비용이 감소될 수 있다.

[0019] 또한 본 발명의 일실시예에 따르면, 취약점 수정 패치에 대한 검증과 취약점 수정 패치에 대한 새로운 취약점의 발견이 동시에 이루어질 수 있다.

도면의 간단한 설명

[0021] 도 1은 본 발명의 일실시예에 따른 소프트웨어 테스팅 시스템을 설명하기 위한 도면이다.

도 2는 본 발명의 일실시예에 따른 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법을 설명하기 위한 도면이다.

도 3은 업데이트 메시지로부터 생성된 취약점 데이터를 나타내는 도면이다.

도 4는 업데이트된 코드로부터 생성된 취약점 데이터를 나타내는 도면이다.

도 5는 본 발명의 일실시예에 따른 소프트웨어 테스팅 방법을 설명하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0022] 본 발명은 다양한 변경을 가할 수 있고 여러 가지 실시예를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 상세한 설명에 상세하게 설명하고자 한다. 그러나, 이는 본 발명을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다. 각 도면을 설명하면서 유사한 참조부호를 유사한 구성요소에 대해 사용하였다.

[0024] 소프트웨어의 버그는 사용자의 신뢰를 무너뜨리고 해킹 공격에 악용될 수 있어 국가 기관 및 여러 기업들에서 발견한 버그에 대해 포상을 하는 버그 바운티를 실행하고 있다. 해외의 경우 구글, 페이스북 등의 기업이 버그 바운티를 실시하고 있고 국내의 경우 삼성, 한국인터넷진흥원 등에서 버그 바운티를 실시하고 있다. 그리고 각 기업들은 자체적으로 자사 제품에 대해 소프트웨어 버그를 찾기 위해 많은 투자를 하고 있다. 다만 이러한 제품이나 공개 소프트웨어의 규모는 매우 크며, 이 때문에 취약점 탐지를 자동화하는 것은 쉽지 않다. 또한 현재의 소프트웨어 테스팅 기술은 인간이 취약점 예상 지점을 지정하는 등 수동적인 작업이 반드시 포함되어 있어야 하므로 완전히 자동화된 방법이라 할 수 없고, 방대한 규모의 소프트웨어의 버그를 자동화하지 않고 찾는 것은 많은 인력과 비용이 들기 때문에 다양한 소프트웨어에서 버그를 비교적 빠른 시간 내에 탐지할 수 있는 것은 소프트웨어 보안 시장 전체에 기여할 수 있는 바가 크다.

[0025] 이에 본 발명은 소프트웨어에서 취약성이 존재할 것으로 예상되는 지점을 자동으로 탐지하고, 탐지된 해당 지점을 테스트하는 소프트웨어 테스팅 방법을 제안한다.

[0026] 본 발명의 일실시예에 따른 소프트웨어 테스팅 방법은, 미리 학습된 인공 신경망 기반의 취약점 분류 모델을 이용하여, 취약성이 존재할 것으로 예상되는 테스트 목표 지점을 결정하고, 테스트 목표 지점에 대해 테스트를 진행한다.

[0027] 본 발명의 일실시예에 따른 소프트웨어 테스팅 방법 및 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법은 프로세서 및 메모리를 포함하는 컴퓨팅 장치에서 수행될 수 있다.

[0028] 이하에서, 본 발명에 따른 실시예들을 첨부된 도면을 참조하여 상세하게 설명한다.

[0030] 도 1은 본 발명의 일실시예에 따른 소프트웨어 테스팅 시스템을 설명하기 위한 도면이다.

[0031] 도 1을 참조하면 본 발명의 일실시예에 따른 소프트웨어 테스팅 시스템은 패치 기반 학습 장치(110) 및 취약점

탐지 장치(120)를 포함한다.

[0032] 패치 기반 학습 장치(110)는 패치 정보 관리부(111) 및 기계 학습부(113)를 포함한다.

[0033] 패치 정보 관리부(111)는 기계 학습 모델인 취약점 분류 모델을 학습시키기 위해, 훈련 데이터를 수집한다. 패치 정보 관리부(111)는 여러 소프트웨어의 패치 기록을 수집하고, 수집된 패치를 취약점 수정 패치와, 취약점 수정용이 아닌 일반 패치로 분류하여 관리한다. 일반 패치는 예컨대, 소프트웨어의 기능을 업데이트하기 위한 패치일 수 있다.

[0034] 기계 학습부(113)는 수집된 훈련 데이터를 이용하여, 취약점 분류 모델을 학습한다. 기계 학습부(113)는 업데이트 전 소스 코드와 비교하여 업데이트 후 소스 코드, 즉 패치에서 수정된 코드와 수정된 코드에 대해 대한 라벨을 이용하여, 취약점 분류 모델을 학습한다. 여기서 업데이트 후 소스 코드는 업데이트 전 소스 코드로부터 업데이트된 코드로서, 업데이트 전 소스 코드의 일부와 업데이트된 코드를 포함한다. 그리고 라벨은, 수정된 코드에 대한 취약점 데이터이다. 취약점 분류 모델은 업데이트된 코드를 포함하는 패치를 취약점 수정 패치 또는 일반 패치로 분류하며, 취약점 수정 패치에서 업데이트된 코드에 대응되는 취약점의 클래스를 분류할 수 있다.

[0035] 취약점 탐지 장치(120)는 테스트 대상 지정부(121) 및 소프트웨어 테스트부(123)를 포함한다.

[0036] 테스트 대상 지정부(121)는 취약점 분류 모델을 이용하여, 타겟 소프트웨어의 패치에서 테스트를 수행할 지점인 테스트 목표 지점을 결정한다. 테스트를 수행할 지점은, 패치의 소스 코드에서 취약점이 존재하는 것으로 예상되는 코드를 포함하는 지점으로서, 테스트 목표 지점은 소스 코드의 기본 블록(basic block) 또는 함수일 수 있다.

[0037] 소프트웨어 테스트부(123)는 테스트 목표 지점에 대해 테스트를 수행한다. 소프트웨어 테스트부(123)는 퍼징을 이용하여 테스트를 수행할 수 있다.

[0039] 도 2는 본 발명의 일실시예에 따른 소프트웨어 테스팅을 위한 취약점 분류 모델 생성 방법을 설명하기 위한 도면이다. 그리고 도 3은 업데이트 메시지로부터 생성된 취약점 데이터를 나타내는 도면이며, 도 4는 업데이트된 코드로부터 생성된 취약점 데이터를 나타내는 도면이다.

[0040] 도 2를 참조하면 본 발명의 일실시예에 따른 컴퓨팅 장치는 훈련용 소프트웨어의 버전별 소스 코드를 입력받는다(S210). 소스 코드가 업데이트될수록 버전이 갱신되며, 따라서 버전별 소스 코드는 업데이트 전후의 소스 코드에 대응된다.

[0041] 그리고 컴퓨팅 장치는 소스 코드의 취약점을 분석하여, 소스 코드의 업데이트 전후에 대한 취약점 데이터를 생성(S220)한다. 취약점 데이터는 업데이트된 코드 즉, 업데이트 전후의 소스 코드의 차이점 별로 생성될 수 있으며, 업데이트된 코드에 존재하는 취약점의 클래스, 취약점 관련 함수 및 객체 등을 포함할 수 있다. 취약점의 클래스는 취약점의 타입에 대응될 수 있으며, 소스 코드의 분석을 통해 획득된 데이터는 소스 코드 분석 데이터베이스에 저장될 수 있다.

[0042] 컴퓨팅 장치는 일실시예로서, 소스 코드에 포함된 업데이트 로그를 분석하여, 취약점 데이터를 생성할 수 있다. 업데이트 로그에는, 소스 코드의 업데이트에 대한 정보를 포함하는 commit 메시지와 같은 업데이트 메시지가 포함되며, 컴퓨팅 장치는 업데이트 메시지에 대한 자연어 분석을 통해 업데이트된 코드에 대한 취약점 데이터를 생성할 수 있다.

[0043] 도 3과 같이 업데이트 메시지(310)가 주어진 경우, 컴퓨팅 장치는 자연어 분석을 통해 취약점 클래스가 “Buffer overflow” 타입이며, 취약점 관련 함수 및 객체가 vuln_func 및 obj임을 포함하는 취약점 데이터(320)을 생성할 수 있다.

[0044] 또는 컴퓨팅 장치는 업데이트된 코드를 분석하여, 취약점 데이터를 생성할 수 있으며, 업데이트된 코드를 분석하기 위해, 일반적으로 이용되는 코드 관리 프로그램이 활용될 수 있다. 컴퓨팅 장치는 일실시예로서, 객체 크기 검사 코드의 추가, 함수 네임의 변경 또는 메모리 오류 검출 함수 호출 여부 등을 이용하여, 취약점 데이터를 생성할 수 있다.

[0045] 도 4와 같은 업데이트 후 소스 코드(411, 412, 413)에서, 컴퓨팅 장치는 초록색으로 마킹된 업데이트된 코드를 분석하여, 취약점 데이터(421, 422, 423)를 생성할 수 있다. 제1소스 코드(411)와 같이, 객체 크기를 검사하는 코드가 추가된 경우, 컴퓨팅 장치는 취약점 클래스가 “Miss Size check” 타입이며, 취약점 관련 함수 및 객체가 vuln_func 및 new_obj임을 포함하는 취약점 데이터(421)을 생성할 수 있다. 그리고 제2소스 코드(412)와 같

이, 메모리 오염을 방지하는 함수의 보편적인 네임인 memcpy가 memcpy_s로 변경된 경우, 컴퓨팅 장치는 취약점 클래스가 “Changed to safe function” 타입이며, 취약점 관련 함수 및 객체가 vuln_func 및 new_obj임을 포함하는 취약점 데이터(422)을 생성할 수 있다. 그리고 제3소스 코드(413)와 같이, Address Sanitizer와 같은 메모리 오류 검출 장치와 관련된 함수가 호출된 경우 컴퓨팅 장치는 취약점 클래스가 “Address sanitizer allocation” 타입이며, 취약점 관련 함수 및 객체가 vuln_func 및 new_obj임을 포함하는 취약점 데이터(423)을 생성할 수 있다.

[0046] 다시 도 2로 돌아가, 컴퓨팅 장치는 업데이트 전후의 소스 코드의 차이점 및 취약점 데이터를 이용하여, 취약점 분류 모델을 학습(S230)한다. 업데이트 전후의 소스 코드의 차이점에 대해 취약점 데이터가 라벨링되며, 컴퓨팅 장치는 업데이트 전후의 소스 코드를 토큰화하여, 업데이트 전후의 소스 코드 사이의 차이점을 추출한다. 컴퓨팅 장치는 예컨대, AST(Abstract Syntax Tree)와 같은 토큰 형태로 소스 코드를 전처리하여, 바이너리 코드 형태로 업데이트 전후의 소스 코드 사이의 차이점을 추출할 수 있다.

[0047] 취약점 데이터가 라벨링된 업데이트 전후의 소스 코드의 차이점을 통해 학습이 이루어짐으로써, 타겟 소프트웨어에 대한 업데이트 전후의 소스 코드의 차이점이 취약점 분류 모델로 입력되면, 입력된 차이점에 대응되는 취약점의 클래스가 분류될 수 있다.

[0048] 이와 같이, 취약점 분류 모델은 업데이트 전후의 소스 코드를 입력받아, 소스 코드 변경 패턴이, 훈련 데이터로 이용된 소스 코드 변경 패턴과 얼마나 유사한지를 비교할 수 있고, 소스 코드 변경 내역과 취약점 정보를 입력받아, 소스 코드 변경 내역 중에서 또 다른 유사한 취약점이 존재하는지 확인할 수 있다. 학습된 취약점 분류 모델은 기계 학습 데이터 베이스에 저장될 수 있다.

[0050] 도 5는 본 발명의 일실시예에 따른 소프트웨어 테스팅 방법을 설명하기 위한 도면이다.

[0051] 도 5를 참조하면 본 발명의 일실시예에 따른 컴퓨팅 장치는 타겟 소프트웨어의 버전별 소스 코드를 입력받고(S510), 취약점 분류 모델을 이용하여, 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류(S520)한다. 취약점 분류 모델은 업데이트된 코드를 포함하는 업데이트 후의 소스 코드가 취약점 수정 패치인지 판단하고, 취약점 수정 패치인 경우 취약점이 존재하는 업데이트된 코드에 대한 취약점의 클래스를 분류한다. 취약점 분류 모델 학습 과정에서 소스 코드를 토큰화하여 학습이 이루어진만큼, 타겟 소프트웨어의 업데이트 전후의 소스 코드의 차이점 역시 토큰화된 형태로 취약점 분류 모델로 입력된다.

[0052] 그리고 컴퓨팅 장치는 업데이트 후의 소스 코드에서, 취약점의 클래스가 분류되어, 취약점이 존재하는 것으로 분류된 코드를 포함하는 테스트 목표 지점에 대한 테스트를 진행(S530)한다. 테스트 목표 지점은, 소스 코드에서 테스트 대상이 되는 코드에 대응되며, 테스트 목표 지점은 일실시예로서, 전술된 바와 같이, 취약점이 존재하는 것으로 분류된 코드를 포함하는 기본 블록 또는 함수일 수 있다. 그리고 테스트는 일예로서 퍼징일 수 있으며, 이 때 사용되는 퍼저는 사용자에 의해 선택될 수 있으며, 직접 퍼징을 수행하는 퍼저나 심볼릭, 콘콜릭 실행을 수행하는 퍼저 등이 이용될 수 있다.

[0053] 실시예에 따라서, 컴퓨팅 장치는 테스트 목표 지점을 결정하기 위해, 전술된 취약점 분류 모델을 이용하는 한편, 취약점 분류 모델 생성 과정에서 활용된 업데이트 메시지 및 업데이트된 코드 분석 방법을 이용하여 예상 취약 지점을 결정한 후, 테스트 목표 지점으로 활용할 수 있다.

[0054] 컴퓨팅 장치는 업데이트 후의 소스 코드에 포함된 업데이트 메시지에 대한 자연어 분석을 통해, 업데이트된 코드 중에서 취약점이 존재하는 코드와 해당 코드에 대한 취약점 데이터를 획득할 수 있으며, 취약점이 존재하는 코드를 테스트 목표 지점으로 선정할 수 있다. 또한 컴퓨팅 장치는 코드 관리 프로그램 등을 통해, 업데이트된 코드 중에서 취약점이 존재하는 코드와 해당 코드에 대한 취약점 데이터를 획득할 수 있다.

[0055] 또한 실시예에 따라서 테스트 목표 지점을 결정하기 위한 방법들은, 병렬적으로 진행되거나 또는 업데이트 메시지와 업데이트된 코드에 대한 분석이 이루어진 후에 취약점이 존재하는 코드가 검출되지 않은 경우, 취약점 분류 모델을 이용하는 방법이 적용될 수 있다.

[0056] 컴퓨팅 장치는 소스 코드를 정적 분석하여 테스트 목표 지점에 대한 인자 정보를 추론하고, 테스트 목표 지점까지 도달하는 기본 블록 경로를 확인하며, 해당 경로까지 도달하기 위한 초기 시드를 생성한다. 이후 테스트 목표 지점까지 도달하여 취약점을 트리거하기 위해 퍼징을 진행한다. 테스트 목표 지점에 도달하였을 경우 테스트 목표 지점에서 취약점이 트리거 되었다면, 컴퓨팅 장치는 취약점을 트리거한 입력 파일과 정적 분석 결과, 크래시가 발생했을 때 산출된 퍼징 로그를 최종 출력으로 산출한다. 퍼징의 특성상 문맥정보를 완벽하게 고려하고 실행하는 것이 어렵기 때문에, 목표 지점에서 크래시가 발생하지 않는 경우를 고려하여, 사용자가 목표 지점 최

대 도달 횟수를 설정할 수 있다.

[0057] 본 발명의 일실시예에 따르면, 컴퓨팅 장치는 관리자에 의해 수동으로 설정되지 않고, 자동으로 설정된 테스트 목표 지점에 대해 테스트를 수행하며, 따라서 테스트 목표 지점 설정에 따른 인력과 비용이 감소될 수 있다.

[0058] 한편, 컴퓨팅 장치는 취약점이 존재하는 것으로 분류된 코드에 포함된 객체 또는 변수와 동일한 객체 또는 변수를 포함하는 코드를 추가로 테스트 목표 지점으로 결정하여 테스트를 진행할 수 있다. 즉, 컴퓨팅 장치는 업데이트 후의 소스 코드에 대한 코드 분석을 통해, 업데이트된 코드 뿐만 아니라, 취약점이 존재하는 업데이트된 코드와 유사한 코드가, 업데이트되지 않은 코드에 존재하는지를 판단하여, 테스트 목표 지점을 결정할 수 있다.

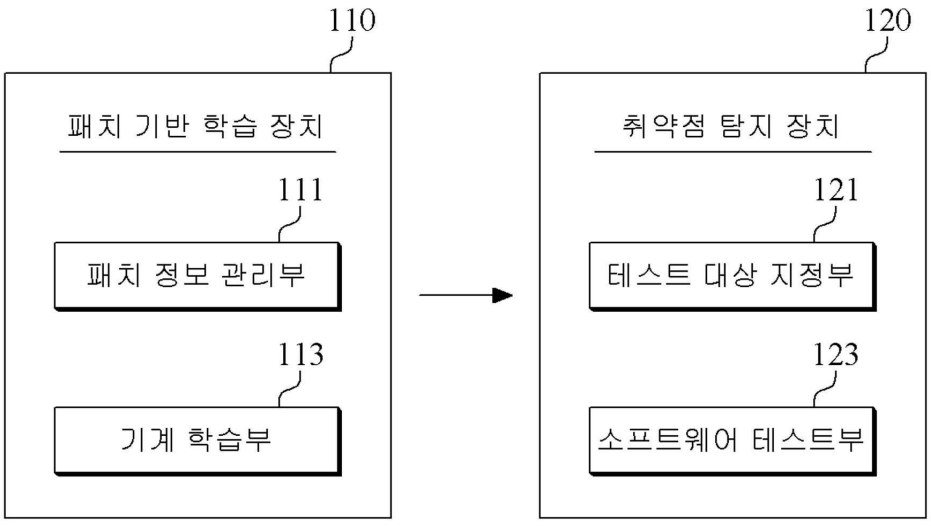
[0059] 따라서 본 발명의 일실시예에 따르면, 최초 취약점이 존재할 것으로 판단된 패치에 대한 검증과 동시에, 취약점이 존재하는 것으로 분류된 코드와 유사한 코드에 유사한 취약점이 존재하는 것으로 판단하여 테스트를 진행할 수 있다.

[0061] 앞서 설명한 기술적 내용들은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예들을 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 하드웨어 장치는 실시예들의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

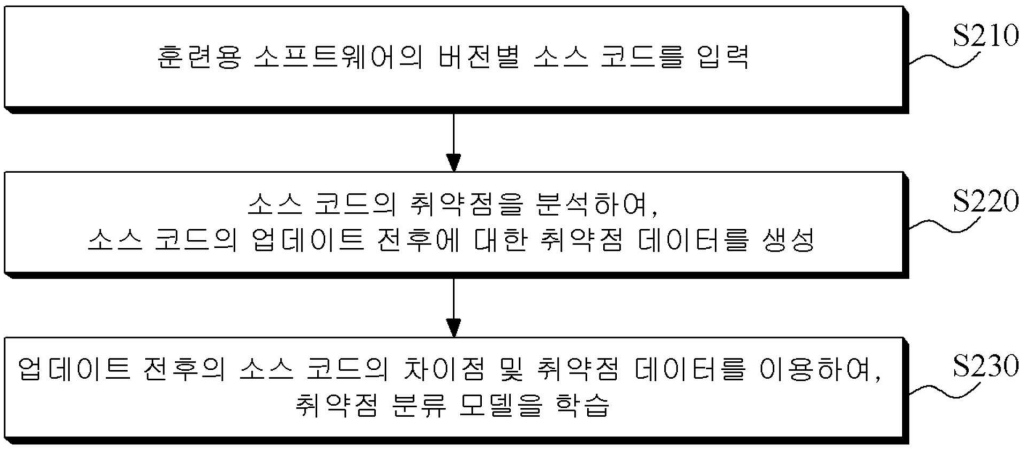
[0063] 이상과 같이 본 발명에서는 구체적인 구성 요소 등과 같은 특정 사항들과 한정된 실시예 및 도면에 의해 설명되었으나 이는 본 발명의 보다 전반적인 이해를 돕기 위해서 제공된 것일 뿐, 본 발명은 상기의 실시예에 한정되는 것은 아니며, 본 발명이 속하는 분야에서 통상적인 지식을 가진 자라면 이러한 기재로부터 다양한 수정 및 변형이 가능하다. 따라서, 본 발명의 사상은 설명된 실시예에 국한되어 정해져서는 아니되며, 후술하는 특허청구범위뿐 아니라 이 특허청구범위와 균등하거나 등가적 변형이 있는 모든 것들은 본 발명 사상의 범주에 속한다고 할 것이다.

도면

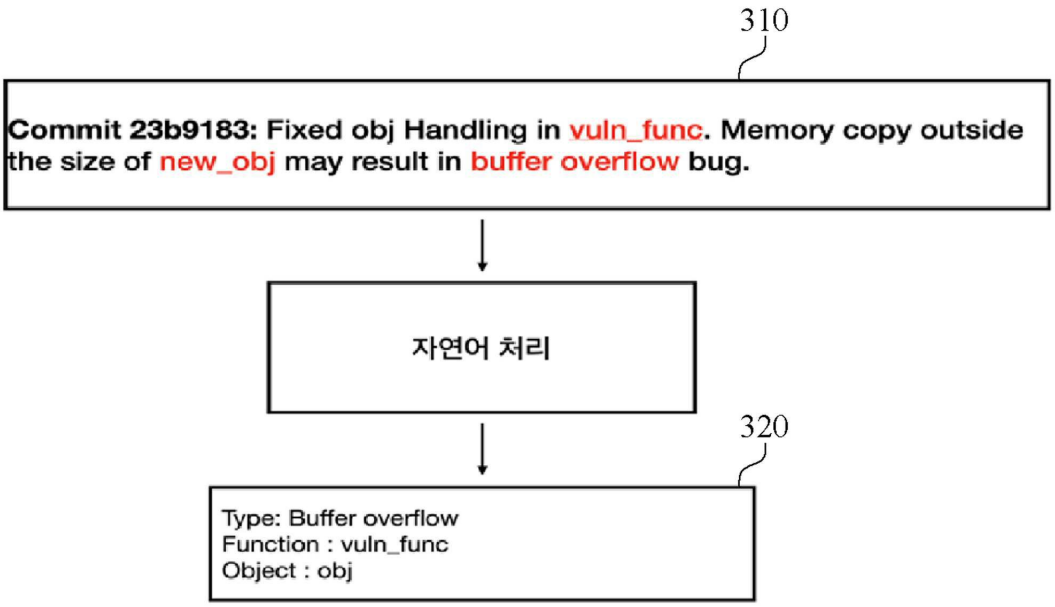
도면1



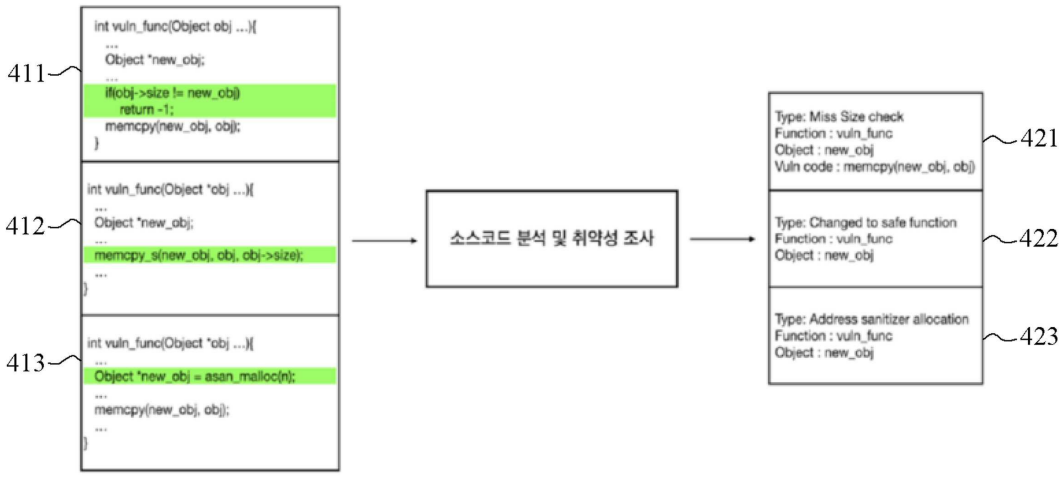
도면2


훈련용 소프트웨어의 버전별 소스 코드를 입력
S210
소스 코드의 취약점을 분석하여,
소스 코드의 업데이트 전후에 대한 취약점 데이터를 생성
S220
업데이트 전후의 소스 코드의 차이점 및 취약점 데이터를 이용하여,
취약점 분류 모델을 학습
S230

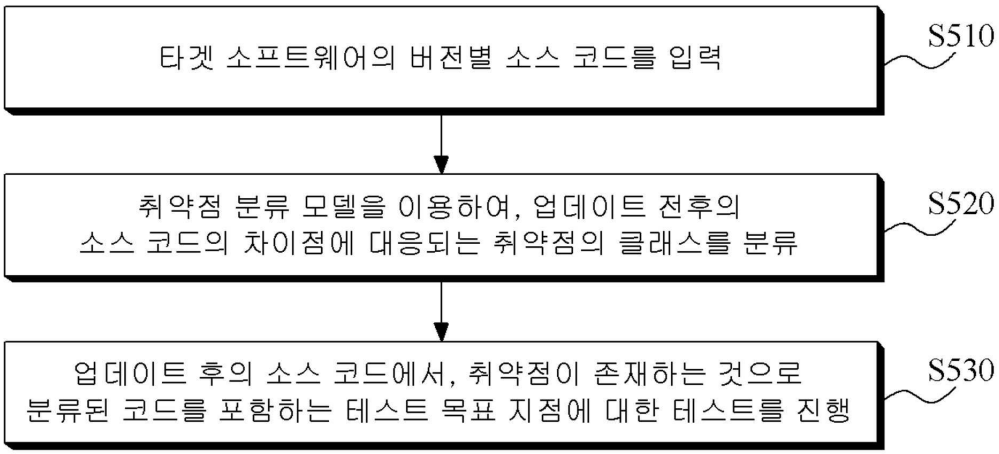
도면3


310
Commit 23b9183: Fixed obj Handling in vuln_func. Memory copy outside the size of new_obj may result in buffer overflow bug.
자연어 처리
320
Type: Buffer overflow
Function : vuln_func
Object : obj

도면4


411
412
413
int vuln_func(Object obj ...){
Object *new_obj;
if(obj->size != new_obj)
return -1;
memcpy(new_obj, obj);
}
int vuln_func(Object *obj ...){
Object *new_obj;
memcpy_s(new_obj, obj, obj->size);
}
int vuln_func(Object *obj ...){
Object *new_obj = asan_malloc(n);
memcpy(new_obj, obj);
}
소스코드 분석 및 취약성 조사
Type: Miss Size check
Function : vuln_func
Object : new_obj
Vuln code : memcpy(new_obj, obj)
421
Type: Changed to safe function
Function : vuln_func
Object : new_obj
422
Type: Address sanitizer allocation
Function : vuln_func
Object : new_obj
423

도면5



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 6

【변경전】

컴퓨팅 장치에 의해 수행되는, 소프트웨어 테스팅 방법에 있어서,

타겟 소프트웨어의 버전별 소스 코드를 입력받는 단계;

취약점 분류 모델을 이용하여, 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류하는 단계; 및

업데이트 후의 소스 코드에서, 상기 취약점이 존재하는 것으로 분류된 코드를 포함하는 테스트 목표 지점에 대한 테스트를 진행하는 단계를 포함하며,

상기 취약점 분류 모델은

상기 훈련용 소프트웨어의 소스 코드의 업데이트 전후에 대한 취약점 데이터로부터 학습된 모델이며,

상기 취약점 데이터는, 상기 훈련용 소프트웨어의 소스 코드에 포함된 업데이트 메시지에 대한 분석을 통해 생성된 데이터인,

소프트웨어 테스팅 방법.

【변경후】

컴퓨팅 장치에 의해 수행되는, 소프트웨어 테스팅 방법에 있어서,

타겟 소프트웨어의 버전별 소스 코드를 입력받는 단계;

취약점 분류 모델을 이용하여, 업데이트 전후의 소스 코드의 차이점에 대응되는 취약점의 클래스를 분류하는 단계; 및

업데이트 후의 소스 코드에서, 상기 취약점이 존재하는 것으로 분류된 코드를 포함하는 테스트 목표 지점에 대한 테스트를 진행하는 단계를 포함하며,

상기 취약점 분류 모델은

훈련용 소프트웨어의 소스 코드의 업데이트 전후에 대한 취약점 데이터로부터 학습된 모델이며,

상기 취약점 데이터는, 상기 훈련용 소프트웨어의 소스 코드에 포함된 업데이트 메시지에 대한 분석을 통해 생성된 데이터인,

소프트웨어 테스팅 방법.