(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2019년10월22일
(11) 등록번호 10-2035246
(24) 등록일자 2019년10월16일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01)
(52) CPC특허분류
G06F 11/3624 (2013.01)
G06F 11/3604 (2013.01)
(21) 출원번호 10-2019-0061038
(22) 출원일자 2019년05월24일
심사청구일자 2019년05월24일
(56) 선행기술조사문헌
KR101963752 B1*
KR1020190041912 A*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
세종대학교산학협력단
서울특별시 광진구 능동로 209 (군자동, 세종대학교)
(72) 발명자
윤주범
서울특별시 송파구 충민로4길 19, 704동 401호(장지동, 송파파인타운7단지)
김주환
서울특별시 강서구 곰달래로31다길 1-4, 3층 (화곡동)
(74) 대리인
두호특허법인

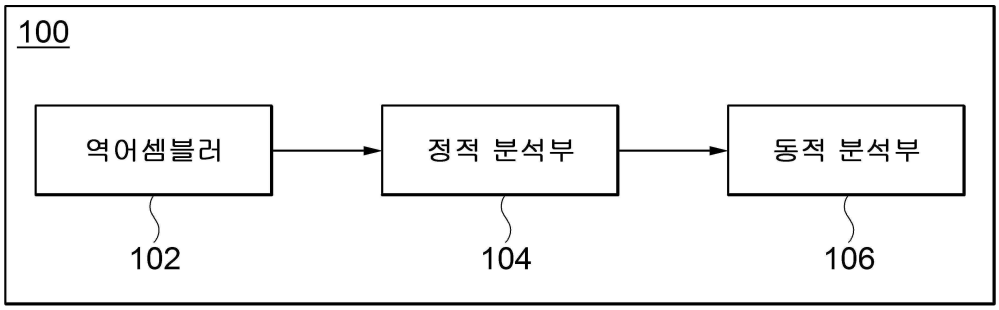
전체 청구항 수 : 총 8 항 심사관 : 이철수

(54) 발명의 명칭 **백워드 패스파인딩을 이용한 소프트웨어 취약점 분석 장치 및 방법**

(57) 요 약

소프트웨어 취약점 분석 장치 및 방법이 개시된다. 일 실시예에 따른 소프트웨어 취약점 분석 장치는, 소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성하는 역어셈블러; 상기 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집하는 정적 분석부; 및 상기 의심 노드 집합에 포함된 주소를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색하는 동적 분석부를 포함한다.

대 표 도 - 도1



(52) CPC특허분류
G06F 11/3644 (2013.01)
G06F 11/3696 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711075702
부처명 과학기술정보통신부
연구관리전문기관 정보통신기획평가원
연구사업명 대학ICT연구센터지원사업
연구과제명 지능형 비행로봇 융합기술 연구
기 여 율 1/1
주관기관 세종대학교 산학협력단
연구기간 2018.06.01 ~ 2021.12.31

명 세 서

청구범위

청구항 1

소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성하는 역어셈블러;

상기 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집하는 정적 분석부; 및

상기 의심 노드 집합에 포함된 노드를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색하는 동적 분석부를 포함하고,

상기 동적 분석부는,

임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 퍼저 모듈;

상기 제어 흐름 그래프에 포함된 노드들 중, 상기 입력값에 의하여 실행되는 노드들의 주소를 수집하는 추적 모듈; 및

상기 추적 모듈에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 기호 실행을 수행하는 기호 실행 모듈을 포함하며,

상기 기호 실행 모듈은, 상기 기호 실행 도중 분기가 발견되는 경우 상기 추적 모듈에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단하고, 판단 결과 포함되는 경우 해당 노드로 진입하기 위한 입력값을 상기 퍼저 모듈로 제공하는 것을 특징으로 하는, 소프트웨어 취약점 분석 장치.

청구항 2

삭제

청구항 3

삭제

청구항 4

청구항 1에 있어서,

상기 퍼저 모듈은, 상기 기호 실행 모듈로부터 획득한 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는, 소프트웨어 취약점 분석 장치.

청구항 5

청구항 1에 있어서,

상기 기호 실행 모듈은, 상기 판단 결과 해당 노드가 상기 의심 노드 집합에 포함되지 않는 경우 해당 노드를 기호 실행 대상에서 제외하는, 소프트웨어 취약점 분석 장치.

청구항 6

청구항 1에 있어서,

상기 기호 실행 모듈은, 상기 기호 실행 도중 상기 의심 함수를 호출한 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료하고 생성된 입력값을 상기 퍼저 모듈로 제공하는, 소프트웨어 취약점 분석 장치.

청구항 7

소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성하는 단계;

상기 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집하는 정적 분석 단계; 및

상기 의심 노드 집합에 포함된 노드를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색하는 동적 분석 단계를 포함하고,

상기 동적 분석 단계는,

임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 단계;

추적 모듈에서, 상기 제어 흐름 그래프에 포함된 노드들 중, 상기 입력값에 의하여 실행되는 노드들의 주소를 수집하는 단계; 및

상기 추적 모듈에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 기호 실행을 수행하는 단계를 포함하며,

상기 기호 실행 단계는, 상기 기호 실행 도중 분기가 발견되는 경우 상기 추적 모듈에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단하고, 판단 결과 포함되는 경우 해당 노드로 진입하기 위한 입력값을 생성하는, 소프트웨어 취약점 분석 방법.

청구항 8

삭제

청구항 9

삭제

청구항 10

청구항 7에 있어서,

생성된 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 단계를 더 포함하는, 소프트웨어 취약점 분석 방법.

청구항 11

청구항 7에 있어서,

상기 기호 실행 단계는, 상기 판단 결과 해당 노드가 상기 의심 노드 집합에 포함되지 않는 경우 해당 노드를 기호 실행 대상에서 제외하는, 소프트웨어 취약점 분석 방법.

청구항 12

청구항 7에 있어서,

상기 기호 실행 단계는, 상기 기호 실행 도중 상기 의심 함수를 호출한 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료하는, 소프트웨어 취약점 분석 방법.

발명의 설명

기 술 분 야

[0001] 개시되는 실시예들은 소프트웨어 취약점 분석 기술과 관련된다.

배 경 기 술

[0003] 소프트웨어 내의 잠재적인 모든 보안 취약점을 수동으로 분석하는 것은 사실상 불가능하다. 그래서 자동화된 소프트웨어 취약점 탐지 및 분석 방법에 대한 많은 연구가 진행되고 있다. 그러나 기존의 연구들은 취약점을 분석하기 위한 기호 실행(symbolic execution)의 탐색 과정이 지나치게 오래 걸리는 문제가 있다. 특히 소프트웨어 바이너리의 크기가 증가할수록 그만큼 탐색해야 할 경로의 개수도 증가하며 이는 경로 폭발(path explosion) 문제로 이어지게 된다.

[0004] 한편, 소프트웨어 취약점을 발견하기 위한 기술 중 하나로 퍼저(fuzzer)가 있다. 퍼저는 임의의 입력값을 자동으로 생성하여 충돌 또는 예기치 않은 동작을 발생시켜 크래시(crash)를 유발함으로써 잠재적인 소프트웨어 취약점을 찾아도록 구성된다. 그러나 이러한 퍼저 또한 단순히 무작위 값을 반복하여 입력하는 방식이므로 소프트웨어의 크기가 증가할수록 보안 취약점을 발견할 가능성이 점점 낮아지게 되는 문제가 존재한다.

선행기술문헌

특허문헌

[0006] (특허문헌 0001) 대한민국 등록특허 제10-1796369호 (2017.12.01. 공고)

발명의 내용

해결하려는 과제

[0007] 본 발명의 실시예들은 소프트웨어 바이너리 파일과 안전하지 않은 함수 정보를 활용하여 소프트웨어 취약점을 효과적으로 분석하기 위한 기술적인 수단을 제공하기 위한 것이다.

과제의 해결 수단

[0009] 일 실시예에 따르면, 소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성하는 역어셈블러; 상기 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하고, 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집하는 정적 분석부; 상기 의심 노드 집합에 포함된 노드를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색하는 동적 분석부를 포함하는, 소프트웨어 취약점 분석 장치가 제공된다.

[0010] 상기 동적 분석부는, 임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 퍼저 모듈; 상기 제어 흐름 그래프에 포함된 노드들 중, 상기 입력값에 의하여 실행되는 노드들의 주소를 수집하는 추적 모듈; 및 상기 추적 모듈에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 기호 실행을 수행하는 기호 실행 모듈을 더 포함할 수

있다.

[0011] 상기 기호 실행 모듈은, 상기 기호 실행 도중 분기가 발견되는 경우 상기 추적 모듈에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단하고, 판단 결과 포함되는 경우 해당 노드로 진입하기 위한 입력값을 상기 퍼저 모듈로 제공할 수 있다.

[0012] 상기 퍼저 모듈은, 상기 기호 실행 모듈로부터 획득한 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지할 수 있다.

[0013] 상기 기호 실행 모듈은, 상기 판단 결과 해당 노드가 상기 의심 노드 집합에 포함되지 않는 경우 해당 노드를 기호 실행 대상에서 제외할 수 있다.

[0014] 상기 기호 실행 모듈은, 상기 기호 실행 도중 상기 의심 함수를 호출한 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료하고 생성된 입력값을 상기 퍼저 모듈로 제공할 수 있다.

[0015] 다른 예시적인 실시예에 따르면, 소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성하는 단계; 상기 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하고, 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집하는 정적 분석 단계; 상기 의심 노드 집합에 포함된 주소를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색하는 동적 분석 단계를 포함하는, 소프트웨어 취약점 분석 방법이 제공된다.

[0016] 상기 동적 분석 단계는, 임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 단계; 상기 제어 흐름 그래프에 포함된 노드들 중, 상기 입력값에 의하여 실행되는 노드들의 주소를 수집하는 단계; 및상기 추적 모듈에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 기호 실행을 수행하는 단계를 더 포함할 수 있다.

[0017] 상기 기호 실행 단계는, 상기 기호 실행 도중 분기가 발견되는 경우 상기 추적 모듈에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단하고, 판단 결과 포함되는 경우 해당 노드로 진입하기 위한 입력값을 생성할 수 있다.

[0018] 상기 방법은, 생성된 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하는 단계를 더 포함할 수 있다.

[0019] 상기 기호 실행 단계는, 상기 판단 결과 해당 노드가 상기 의심 노드 집합에 포함되지 않는 경우 해당 노드를 기호 실행 대상에서 제외할 수 있다.

[0020] 상기 기호 실행 단계는, 상기 기호 실행 도중 상기 의심 함수를 호출한 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료할 수 있다.

발명의 효과

[0022] 개시되는 실시예들에 따르면, 소프트웨어의 취약점 존재 여부를 분석하는 과정에서 발생할 수 있는 경로 폭발(path explosion)을 방지하고, 소프트웨어의 취약점을 신속하고 효과적으로 탐지할 수 있다.

도면의 간단한 설명

[0024] 도 1은 일 실시예에 따른 소프트웨어 취약점 분석 장치(100)의 구성을 설명하기 위한 블록도

도 2는 일 실시예에 따른 동적 분석부(106)의 상세 구성을 설명하기 위한 블록도

도 3은 일 실시예에 따른 정적 분석부(104)에서 의심 노드 집합을 수집하는 예를 나타낸 예시도

도 4는 일 실시예에 따른 추적 모듈(204)에서 퍼저 모듈(202)의 입력값에 의하여 실행되는 노드를 수집하는 예를 나타낸 예시도

도 5는 일 실시예에 따른 소프트웨어 취약점 분석 방법(500)를 설명하기 위한 흐름도

도 6은 일 실시예에 따른 소프트웨어 취약점 분석 방법(500)의 동적 분석 과정(506)을 상세히 설명하기 위한 흐

름도

도 7은 예시적인 실시예들에서 사용되기에 적합한 컴퓨팅 장치를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도

발명을 실시하기 위한 구체적인 내용

[0025] 이하, 도면을 참조하여 본 발명의 구체적인 실시형태를 설명하기로 한다. 이하의 상세한 설명은 본 명세서에서 기술된 방법, 장치 및/또는 시스템에 대한 포괄적인 이해를 돕기 위해 제공된다. 그러나 이는 예시에 불과하며 본 발명은 이에 제한되지 않는다.

[0026] 본 발명의 실시예들을 설명함에 있어서, 본 발명과 관련된 공지기술에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략하기로 한다. 그리고, 후술되는 용어들은 본 발명에서의 기능을 고려하여 정의된 용어들로서 이는 사용자, 운용자의 의도 또는 관례 등에 따라 달라질 수 있다. 그러므로 그 정의는 본 명세서 전반에 걸친 내용을 토대로 내려져야 할 것이다. 상세한 설명에서 사용되는 용어는 단지 본 발명의 실시예들을 기술하기 위한 것이며, 결코 제한적이어서는 안 된다. 명확하게 달리 사용되지 않는 한, 단수 형태의 표현은 복수 형태의 의미를 포함한다. 본 설명에서, "포함" 또는 "구비"와 같은 표현은 어떤 특성들, 숫자들, 단계들, 동작들, 요소들, 이들의 일부 또는 조합을 가리키기 위한 것이며, 기술된 것 이외에 하나 또는 그 이상의 다른 특성, 숫자, 단계, 동작, 요소, 이들의 일부 또는 조합의 존재 또는 가능성을 배제하도록 해석되어서는 안 된다.

[0028] 도 1은 일 실시예에 따른 소프트웨어 취약점 분석 장치(100)의 구성을 설명하기 위한 블록도이다. 도시된 바와 같이, 일 실시예에 따른 소프트웨어 취약점 분석 장치(100)는 역어셈블러(102), 정적 분석부(104) 및 동적 분석부(106)를 포함한다.

[0029] 역어셈블러(102)는 취약점 분석 대상인 소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 해당 바이너리 파일에 대한 어셈블리어 코드를 생성한다. 이때 상기 소프트웨어 바이너리 파일은 소프트웨어 에뮬레이터 내의 가상 주소 공간에 로딩된 파일일 수 있다.

[0030] 정적 분석부(104)는 생성된 어셈블리 코드에 대한 제어 흐름 그래프(Control Flow Graph, CFG)를 생성한다. 이때, 제어 흐름 그래프의 생성은 공지된 다양한 방식을 이용하여 수행될 수 있다.

[0031] 한편, 정적 분석부(104)는 문자열 검색을 통해 어셈블리 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하고 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집한다.

[0032] 구체적으로, 정적 분석부(104)는 문자열 검색을 통해 어셈블리 코드 내에서 의심 함수 리스트에 포함된 의심 함수를 포함하고 있는 노드인 기본 블록(basic block)을 식별할 수 있다. 이때, 의심 함수 리스트는 사전에 발견된 취약점과 관련된 함수들에 대한 리스트를 의미할 수 있다. 구체적으로, 본 발명의 일 실시예에 따르면, 의심 함수 리스트는 예를 들어, CVE(Common Vulnerabilities and Exposure)에 등록된 취약점 과 관련된 함수를 포함할 수 있다.

[0033] 구체적인 예로, 의심 함수 리스트는 strcpy(), memcpy(), strcat(), getwd(), gets(), fscanf(), scanf(), realpath(), sprint(), fprintf(), snprintf(), wcscpy(), wcxcat(), wmemcpy(), sscanf(), vscanf(), vfscanf(), vsscanf(), wscanf(), fwscanf(), swscanf(), vwscanf(), vfwscanf(), ip_cmsg_recv_checksum 등과 같은 함수들을 포함할 수 있다. 한편, 의심 함수 리스트는 상술한 예 외에도 사전 발견된 취약점과 관련된 다양한 함수들을 포함할 수 있다.

[0034] 정적 분석부(104)는, 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집한다. 구체적으로, 정적 분석부(104)는 전술한 의심 함수 리스트에 포함된 함수를 호출한 노드를 의심 노드로 식별하고, 제어 흐름 그래프의 시작점(entry point)로부터 상기 의심 노드로 진입 가능한 모든 경로상의 노드(해당 의심 노드의 부모 노드)를 추가로 식별하여 상기 의심 노드 집합을 구성한다.

[0035] 도 3은 일 실시예에 따른 정적 분석부(104)에서 의심 노드 집합을 수집하는 예를 나타낸 예시도이다. 도 3에 도시된 제어 흐름 그래프에서 모서리가 둥근 사각형으로 표시된 부분은 노드를 나타내며, 각 노드는 어셈블리 코

드에 포함된 기본 블록을 나타낸다. 또한, 각 노드 사이의 화살표는 실행 경로를 나타내며, 노드에 기재된 0x80484cf 등의 문자열은 해당 노드의 가상 주소 공간 내 주소를 의미한다.

[0036] 도 3에 도시된 제어 흐름 그래프에서, 정적 분석부(104)는 의심 함수 리스트에 포함된 함수 중 하나인 "gets" 함수를 호출하고 있는 노드인 0x9000008 노드를 의심 노드로 식별할 수 있다. 이후 백워드 패스파인딩 모듈(202)은 제어 흐름 그래프의 시작점인 0x80484cf로부터 상기 의심 노드로 진입 가능한 모든 경로상의 노드를 추가로 식별할 수 있다. 도시된 예시도에서, 상기 의심 노드로 진입 가능한 의심 노드의 부모 노드들을 기재하면 다음과 같다(도면에서 굵은 테두리로 표시).

[0037] 0x80484cf

[0038] 0x80484dc

[0039] 0x8048518

[0040] 0x804848b

[0041] 0x8048350

[0042] 정적 분석부(104)는 식별된 상기 의심 노드 및 이의 부모 노드들을 의심 노드 집합으로 구성할 수 있다.

[0043] 다음으로, 동적 분석부(106)는 정정 분석부(104)에서 구성된 의심 노드 집합에 포함된 노드를 기반으로 기호 실행(symbolic execution)을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색한다.

[0045] 도 2는 일 실시예에 따른 동적 분석부(106)의 상세 구성을 설명하기 위한 블록도이다. 도시된 바와 같이, 일 실시예에 따른 동적 분석부(106)는 퍼저 모듈(202), 추적 모듈(204), 및 기호 실행 모듈(206)을 포함할 수 있다.

[0046] 퍼저(fuzzer) 모듈(202)은 임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하기 위한 모듈이다.

[0047] 추적(tracer) 모듈(204)은 상기 제어 흐름 그래프에 포함된 노드들 중, 상기 퍼저 모듈(202)이 생성한 입력값에 의하여 실행되는 노드들의 주소를 수집한다. 일 실시예에서, 추적 모듈(204)은 퍼저 모듈(202)이 단독으로 찾은 입력값에 의한 제어 흐름 그래프 내의 실행 경로를 동적 추적(dynamic tracing)하여 해당 입력값으로 인해 거쳐지는 노드들의 주소를 수집할 수 있다. 퍼저 모듈(202)은, 예를 들어 QEMU(Quick EMUlator)를 이용하여 상기 동적 추적을 수행할 수 있다.

[0048] 도 4는 일 실시예에 따른 추적 모듈(204)에서 퍼저 모듈(202)의 입력값에 의하여 실행되는 노드를 수집하는 예를 나타낸 예시도이다. 예를 들어, 퍼저 모듈(202)이 소프트웨어 바이너리 파일의 입력값으로 "!@#$xxx"를 입력할 경우의 제어 흐름 그래프 내의 실행 경로가 도 4의 두꺼운 화살표와 같다고 가정하자. 그러면 추적 모듈(204)은 동적 추적을 통해 해당 경로들을 추적하여 위 입력값으로 인해 거쳐지는 노드들(도면에서 빗금으로 표시)의 주소를 수집할 수 있다. 도시된 예시도에서, 퍼저 모듈(202)의 입력값으로 인해 거쳐지는 노드들의 주소를 기재하면 다음과 같다.

[0049] 0x80484cf

[0050] 0x8048518

[0051] 0x8048558

[0052] 0x900010

[0053] 0x80485ad

[0054] 기호 실행(symbolic execution) 모듈(208)은 추적 모듈(204)에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 기호 실행을 수행한다. 기호 실행은 컴퓨터 과학 및 컴퓨터 공학 분야에서 사용하는 용어로써, 컴퓨터 프로그램의 입력값에 대한 실행 경로를 분석하기 위한 기법이다. 입력값에 대한 실행 경로 취득이 가능하면, 역으로 실행 경로를 위한 입력값 생성도 가능하기 때문에 다양한 방법으로 활용 가능하다.

[0055] 일 실시예에서, 기호 실행 모듈(206)은 기호 실행 도중 분기가 발견되는 경우 추적 모듈(204)에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단하고, 판단 결과 포함되는 경우 해당 노드로 진입하기 위한 입력값을 상기 퍼저 모듈로 제공한다. 그러면 퍼저 모듈(202)은, 기호 실행 모

듈(206)로부터 획득한 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하게 된다. 이를 좀 더 상세히 설명하면 다음과 같다.

[0056] 도 4에 도시된 실시예에서, 기호 실행 모듈(206)은 추적 모듈(204)에서 수집된 주소를 기반으로 제어 흐름 그래프의 진입점인 0x80484cf부터 기호 실행을 수행한다. 진입점인 0x80484cf은 분기를 포함하며, 이중 추적 모듈(204)에서 수집된 경로인 0x8048518과 반대되는 경로는 0x80484dc이다. 0x80484dc는 정적 분석부(104)가 수집한 의심 노드 집합에 포함된 주소이므로, 기호 실행 모듈(206)은 해당 노드(0x80484dc)로 진입하기 위한 새로운 입력값을 생성하고 이를 퍼저 모듈(202)에 전달한다.

[0057] 한편, 기호 실행 모듈(206)은, 추적 모듈(204)에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되지 않는 경우, 해당 노드를 기호 실행 대상에서 제외하게 된다. 도 4에 도시된 실시예에서 0x8048518 노드에 포함된 분기 중, 추적 모듈(204)에서 수집된 경로인 0x8048558과 반대되는 경로는 0x8048522이다. 그러나 0x8048522는 정적 분석부(104)가 수집한 의심 노드 집합에 포함된 주소에 해당하지 않으므로, 기호 실행 모듈(206)은 해당 노드(0x80484dc)로 진입하기 위한 새로운 입력값을 생성하지 않는다(기호 실행 대상에서 제외).

[0058] 기호 실행 모듈(206)은, 상기 기호 실행 도중 상기 의심 함수를 호출한 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료하고 생성된 입력값을 퍼저 모듈(202)로 제공한다. 예를 들어, 기호 실행 모듈(206)이 0x9000008로 진입하기 위한 입력값을 생성한 경우, 기호 실행 모듈(206)은 기호 실행을 종료하고 해당 입력값을 퍼저 모듈(202)로 제공한다. 그러면 퍼저 모듈(202)은, 기호 실행 모듈(206)로부터 획득한 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지하게 된다.

[0059] 일반적인 동적 기호 실행을 이용한 바이너리의 탐색의 경우, 바이너리의 크기가 커지고 분기문의 개수가 많아질수록 전체 경로를 탐색하는 것이 어려워진다. 제어 흐름 그래프에서 조건문 등과 같은 분기문을 만날 때마다 탐색해야 하는 경로의 개수가 2배 이상 증가하며, 계속해서 아래로 탐색이 진행이 될 경우 상태 폭발(state explosion)의 가능성이 높아지게 되기 때문이다. 이는 시스템의 자원을 고갈하는 현상을 초래할 수 있다. 이에 따라, 개시되는 실시예들에서는 안전하지 않은 함수를 호출하는 노드를 우선 찾고 거꾸로 진입점(entry point)까지 부모 노드를 모두 수집하는 백워드 패스파인딩(Backward Pathfinding) 기법을 사전에 적용함으로써 바이너리의 크래시 유발 가능성을 높이도록 구성된다. 또한, 개시되는 실시예들은 추적 모듈(204)에 의해 추적된 된 주소를 기반으로 기호 실행을 수행하고, 현 상태와 반대되는 분기문을 위한 입력 값만 퍼저에게 전달하고 정상적으로 기호 실행이 종료되기 때문에 상태 폭발 및 경로 폭발(path explosion)의 가능성을 효과적으로 감소시킬 수 있다.

[0061] 도 5는 개시되는 실시예에 따른 소프트웨어 취약점 분석 방법(500)을 설명하기 위한 흐름도이다. 도 5에 도시된 방법은 예를 들어, 도 1에 도시된 소프트웨어 취약점 분석 장치(100)에 의해 수행될 수 있다. 도 3에 도시된 순서도에서는 상기 방법을 복수 개의 단계로 나누어 기재하였으나, 적어도 일부의 단계들은 순서를 바꾸어 수행되거나, 다른 단계와 결합되어 함께 수행되거나, 생략되거나, 세부 단계들로 나뉘어 수행되거나, 또는 도시되지 않은 하나 이상의 단계가 부가되어 수행될 수 있다.

[0062] 단계 502에서, 소프트웨어 취약점 분석 장치(100)의 역어셈블러(102)는 취약점 분석 대상인 소프트웨어 바이너리 파일을 역어셈블하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성한다.

[0063] 단계 504에서, 소프트웨어 취약점 분석 장치(100)의 정적 분석부(104)는 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성하고, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단한다. 이때, 의심 함수 리스트는 예를 들어, CVE(Common Vulnerabilities and Exposure) 에 등록된 취약점과 관련된 함수와 같이 사전에 발견된 취약점과 관련된 함수를 포함할 수 있다. 어셈블리어 코드에 의심 함수가 포함되어 있는 경우, 정적 분석부(104)는 상기 제어 흐름 그래프에서 상기 의심 함수를 호출한 노드 및 상기 노드의 부모 노드들의 주소를 의심 노드 집합으로 수집한다.

[0064] 어셈블리어 코드에 의심 함수가 포함되어 있는 경우, 단계 506에서 소프트웨어 취약점 분석 장치(100)의 동적 분석부(106)는 상기 의심 노드 집합에 포함된 노드를 기반으로 기호 실행을 수행함으로써 상기 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색한다.

[0066] 도 6은 일 실시예에 따른 소프트웨어 취약점 분석 방법(500)의 동적 분석 과정(506)을 상세히 설명하기 위한 흐

름도이다.

[0067] 단계 602에서, 퍼저 모듈(202)은 임의의 입력값을 생성하고, 이를 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지한다.

[0068] 단계 604에서, 추적 모듈(204)은 상기 제어 흐름 그래프에 포함된 노드들 중, 상기 602 단계에서 퍼저 모듈(202)에 의해 생성된 입력값에 의하여 실행되는 노드들의 주소를 수집한다.

[0069] 단계 606에서, 기호 실행 모듈(206)은 상기 추적 모듈에서 수집된 주소를 이용하여 상기 제어 흐름 그래프의 진입점부터 순차적으로 기호 실행(symbolic execution)을 수행한다.

[0070] 단계 608에서, 기호 실행 모듈(206)은 상기 기호 실행 도중 분기가 존재하는지 여부를 판단한다.

[0071] 만약 상기 608 단계의 판단 결과 분기가 발견된 경우, 단계 610에서 기호 실행 모듈(206)은 추적 모듈(204)에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는지 여부를 판단한다. 만약 분기가 발견되지 않은 경우, 기호 실행 모듈(206)은 분기를 발견할 때까지 계속해서 상기 기호 실행을 수행한다.

[0072] 만약 상기 610 단계의 판단 결과 추적 모듈(204)에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되는 경우, 단계 614에서 기호 실행 모듈(206)은 해당 노드로 진입하기 위한 입력값을 생성하여 퍼저 모듈(202)로 제공한다. 그러면 단계 614에서 퍼저 모듈(202)은 생성된 상기 입력값을 이용하여 상기 소프트웨어 바이너리 파일을 실행함으로써 상기 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지한다.

[0073] 이와 달리, 상기 610 단계의 판단 결과 추적 모듈(204)에서 수집된 경로와 반대되는 경로에 해당하는 노드가 상기 의심 노드 집합에 포함되지 않는 경우, 단계 616에서 기호 실행 모듈은 해당 노드를 기호 실행 대상에서 제외한다.

[0074] 한편, 기호 실행 모듈(206)은 상기 기호 실행 도중 상기 의심 함수를 호출한 의심 노드로 진입하기 위한 입력값이 생성되는 경우, 상기 기호 실행을 종료하고 해당 의심 노드로 진입하기 위한 입력값을 퍼저 모듈(202)로 제공함으로써 퍼저 모듈(202)에 해당 입력값을 통해 소프트웨어 바이너리 파일의 이상 발생 여부를 탐지할 수 있도록 한다.

[0076] 도 7은 예시적인 실시예들에서 사용되기에 적합한 컴퓨팅 장치를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도이다. 도시된 실시예에서, 각 컴포넌트들은 이하에 기술된 것 이외에 상이한 기능 및 능력을 가질 수 있고, 이하에 기술되지 않은 것 이외에도 추가적인 컴포넌트를 포함할 수 있다.

[0077] 도시된 컴퓨팅 환경(10)은 컴퓨팅 장치(12)를 포함한다. 일 실시예에서, 컴퓨팅 장치(12)는 소프트웨어 취약점 분석 장치(100)에 포함되는 하나 이상의 컴포넌트일 수 있다.

[0078] 컴퓨팅 장치(12)는 적어도 하나의 프로세서(14), 컴퓨터 판독 가능 저장 매체(16) 및 통신 버스(18)를 포함한다. 프로세서(14)는 컴퓨팅 장치(12)로 하여금 앞서 언급된 예시적인 실시예에 따라 동작하도록 할 수 있다. 예컨대, 프로세서(14)는 컴퓨터 판독 가능 저장 매체(16)에 저장된 하나 이상의 프로그램들을 실행할 수 있다. 상기 하나 이상의 프로그램들은 하나 이상의 컴퓨터 실행 가능 명령어를 포함할 수 있으며, 상기 컴퓨터 실행 가능 명령어는 프로세서(14)에 의해 실행되는 경우 컴퓨팅 장치(12)로 하여금 예시적인 실시예에 따른 동작들을 수행하도록 구성될 수 있다.

[0079] 컴퓨터 판독 가능 저장 매체(16)는 컴퓨터 실행 가능 명령어 내지 프로그램 코드, 프로그램 데이터 및/또는 다른 적합한 형태의 정보를 저장하도록 구성된다. 컴퓨터 판독 가능 저장 매체(16)에 저장된 프로그램(20)은 프로세서(14)에 의해 실행 가능한 명령어의 집합을 포함한다. 일 실시예에서, 컴퓨터 판독 가능 저장 매체(16)는 메모리(랜덤 액세스 메모리와 같은 휘발성 메모리, 비휘발성 메모리, 또는 이들의 적절한 조합), 하나 이상의 자기 디스크 저장 디바이스들, 광학 디스크 저장 디바이스들, 플래시 메모리 디바이스들, 그 밖에 컴퓨팅 장치(12)에 의해 액세스되고 원하는 정보를 저장할 수 있는 다른 형태의 저장 매체, 또는 이들의 적합한 조합일 수 있다.

[0080] 통신 버스(18)는 프로세서(14), 컴퓨터 판독 가능 저장 매체(16)를 포함하여 컴퓨팅 장치(12)의 다른 다양한 컴포넌트들을 상호 연결한다.

[0081] 컴퓨팅 장치(12)는 또한 하나 이상의 입출력 장치(24)를 위한 인터페이스를 제공하는 하나 이상의 입출력 인터페이스(22) 및 하나 이상의 네트워크 통신 인터페이스(26)를 포함할 수 있다. 입출력 인터페이스(22) 및 네트워크 통신 인터페이스(26)는 통신 버스(18)에 연결된다. 입출력 장치(24)는 입출력 인터페이스(22)를 통해 컴퓨팅

장치(12)의 다른 컴포넌트들에 연결될 수 있다. 예시적인 입출력 장치(24)는 포인팅 장치(마우스 또는 트랙패드 등), 키보드, 터치 입력 장치(터치패드 또는 터치스크린 등), 음성 또는 소리 입력 장치, 다양한 종류의 센서 장치 및/또는 촬영 장치와 같은 입력 장치, 및/또는 디스플레이 장치, 프린터, 스피커 및/또는 네트워크 카드와 같은 출력 장치를 포함할 수 있다. 예시적인 입출력 장치(24)는 컴퓨팅 장치(12)를 구성하는 일 컴포넌트로서 컴퓨팅 장치(12)의 내부에 포함될 수도 있고, 컴퓨팅 장치(12)와는 구별되는 별개의 장치로 컴퓨팅 장치(12)와 연결될 수도 있다.

[0082] 이상에서 대표적인 실시예를 통하여 본 발명에 대하여 상세하게 설명하였으나, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자는 전술한 실시예에 대하여 본 발명의 범주에서 벗어나지 않는 한도 내에서 다양한 변형이 가능함을 이해할 것이다. 그러므로 본 발명의 권리범위는 설명된 실시예에 국한되어 정해져서는 안 되며, 후술하는 특허청구범위뿐만 아니라 이 특허청구범위와 균등한 것들에 의해 정해져야 한다.

부호의 설명

[0084] 10: 컴퓨팅 환경

12: 컴퓨팅 장치

14: 프로세서

16: 컴퓨터 판독 가능 저장 매체

18: 통신 버스

20: 프로그램

22: 입출력 인터페이스

24: 입출력 장치

26: 네트워크 통신 인터페이스

100: 소프트웨어 취약점 분석 장치

102: 역어셈블러

104: 정적 분석부

106: 동적 분석부

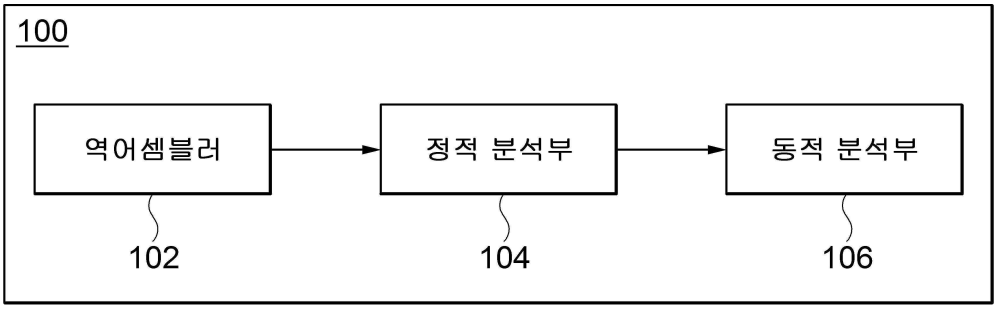
202: 퍼저 모듈

204: 추적 모듈

206: 기호 실행 모듈

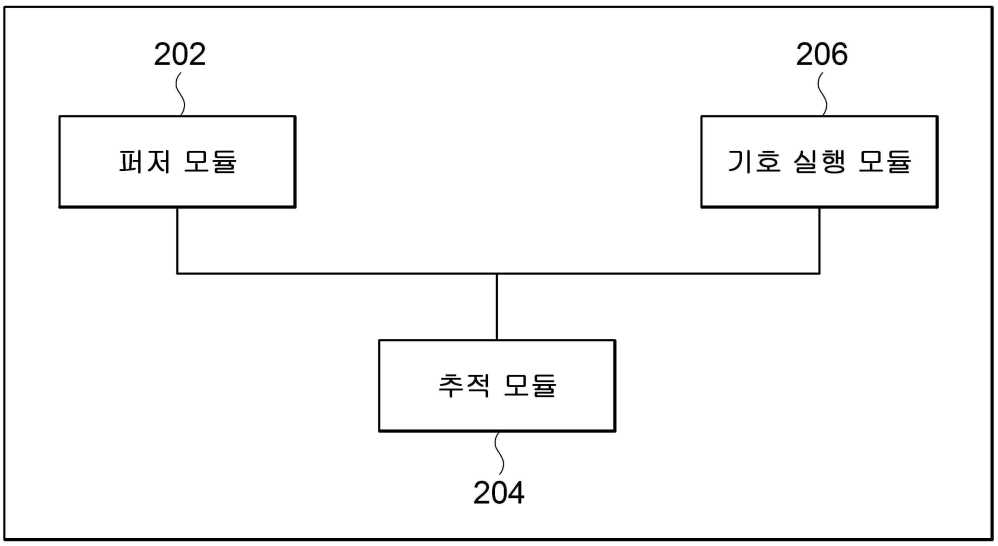
도면

도면1

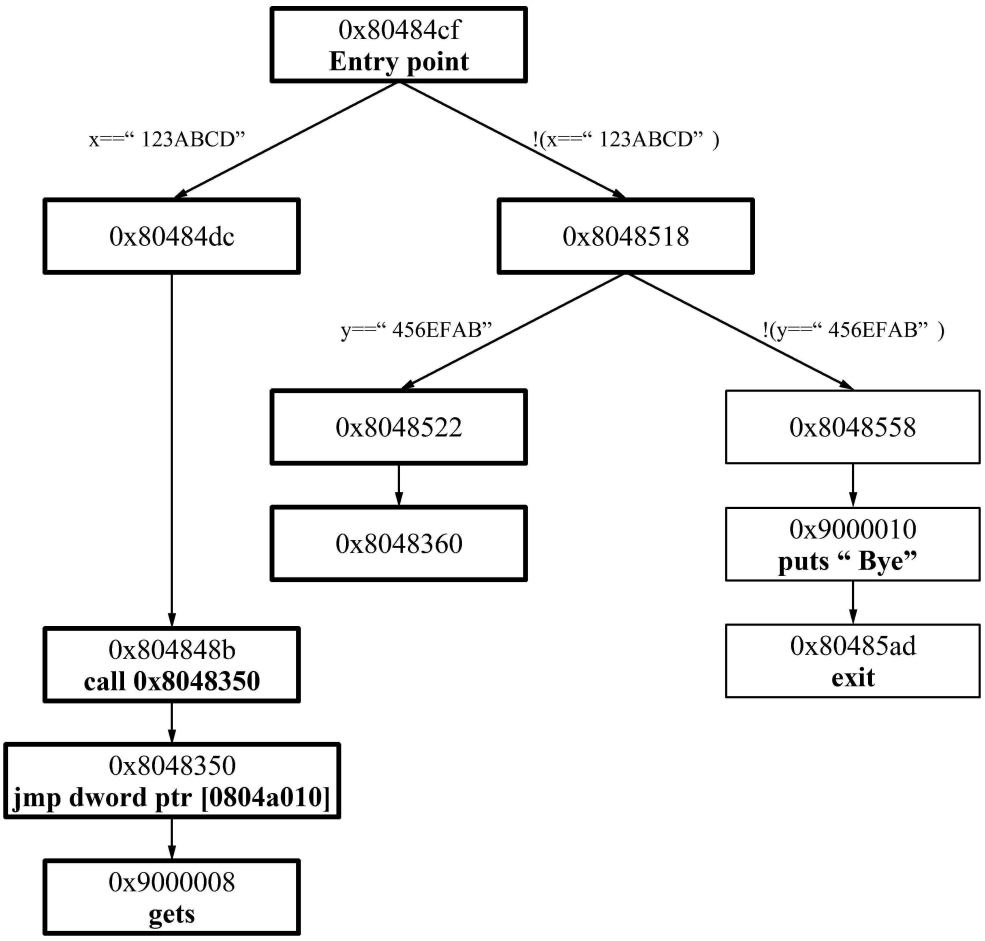


도면2

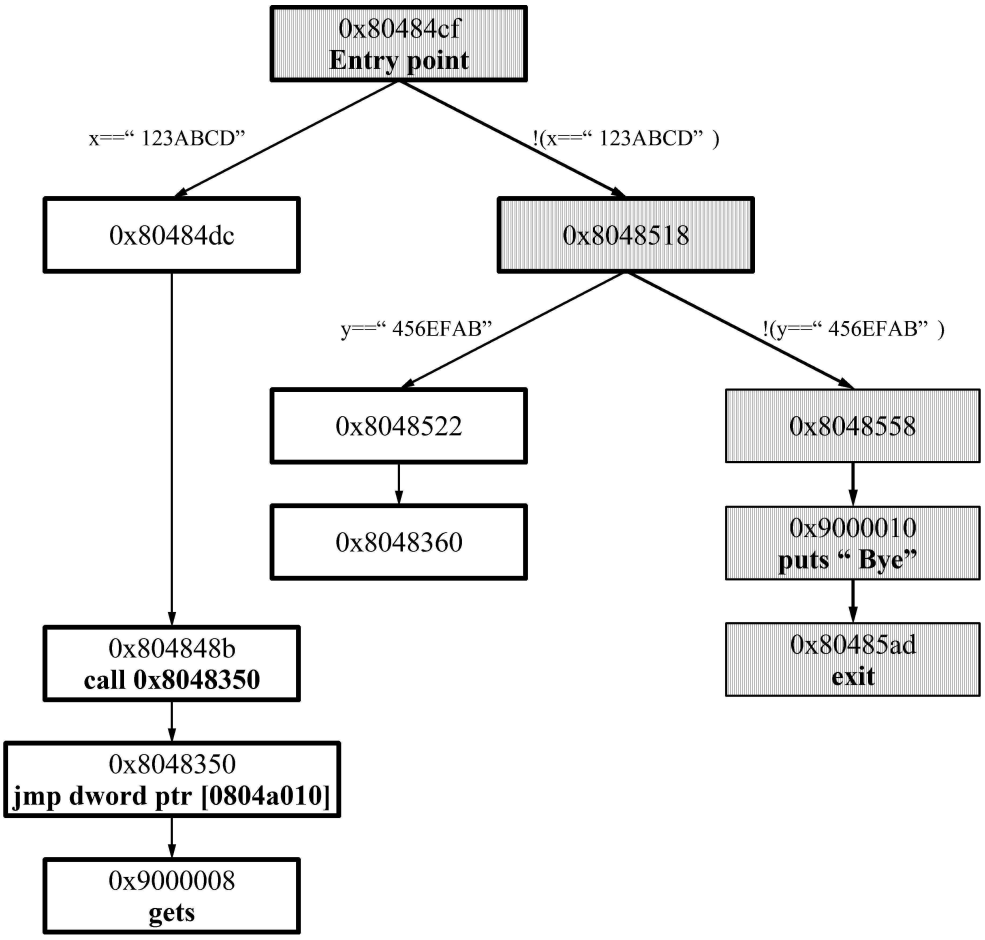
106


202
206
퍼저 모듈
기호 실행 모듈
추적 모듈
204

도면3

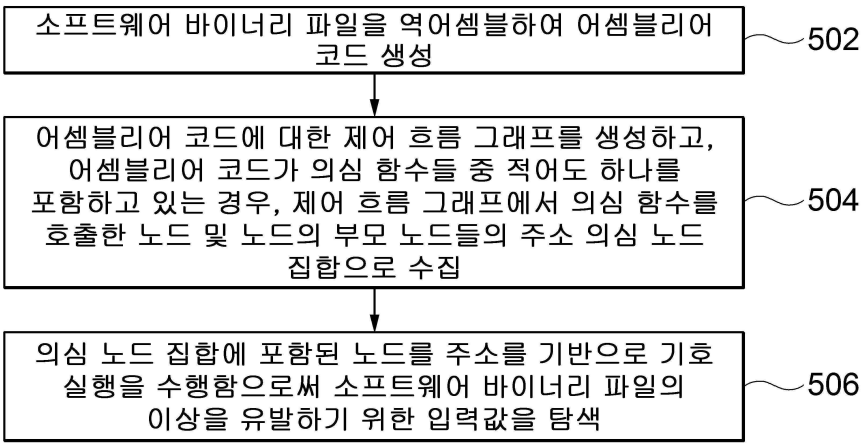

0x80484cf
Entry point
x==“ 123ABCD”
!(x==“ 123ABCD”)
0x80484dc
0x8048518
y==“ 456EFAB”
!(y==“ 456EFAB”)
0x8048522
0x8048558
0x8048360
0x9000010
puts “ Bye”
0x804848b
call 0x8048350
0x80485ad
exit
0x8048350
jmp dword ptr [0804a010]
0x9000008
gets

도면4

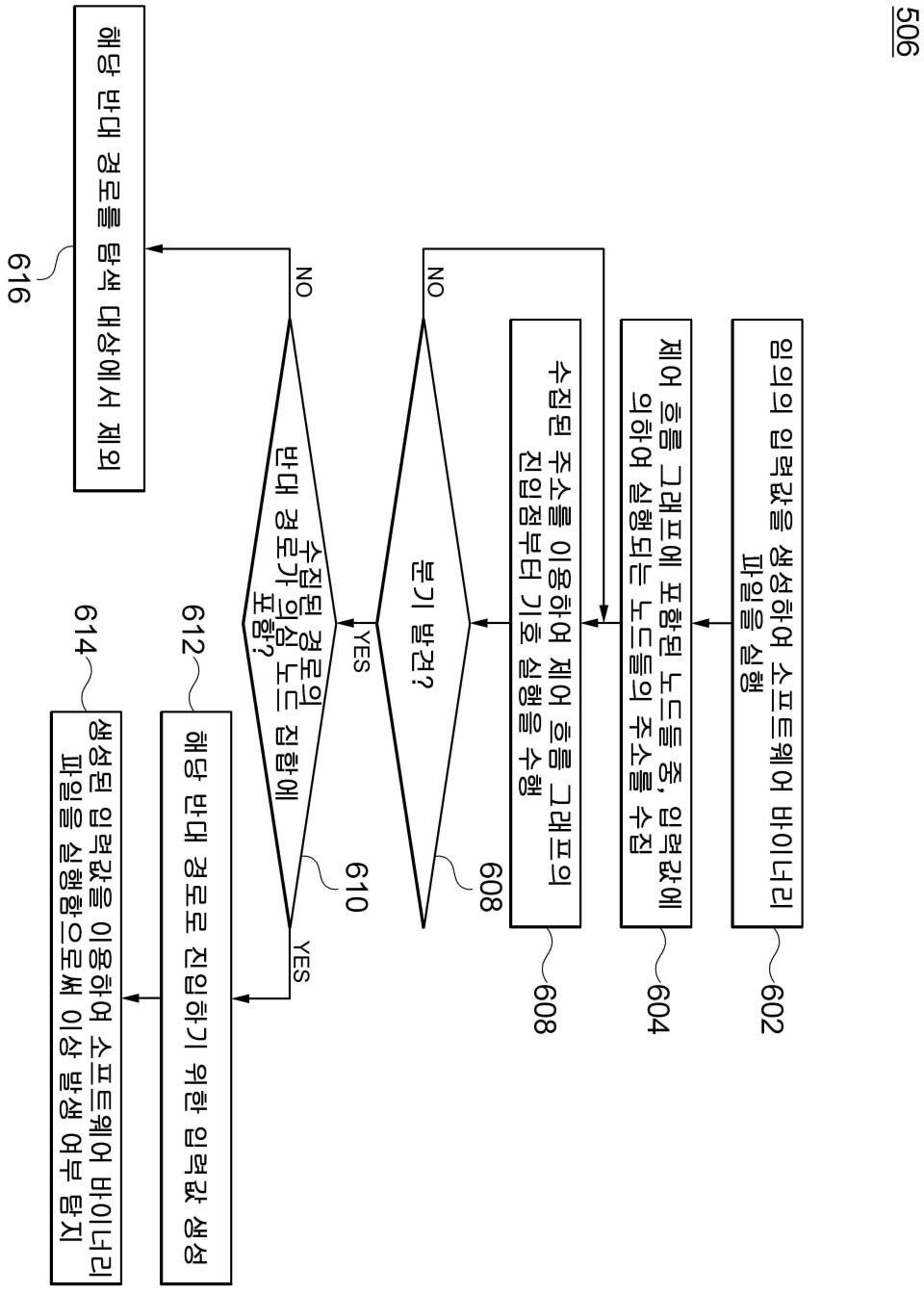

0x80484cf
Entry point
x==“ 123ABCD”
!(x==“ 123ABCD”)
0x80484dc
0x8048518
y==“ 456EFAB”
!(y==“ 456EFAB”)
0x8048522
0x8048558
0x8048360
0x9000010
puts “ Bye”
0x804848b
call 0x8048350
0x80485ad
exit
0x8048350
jmp dword ptr [0804a010]
0x9000008
gets

도면5

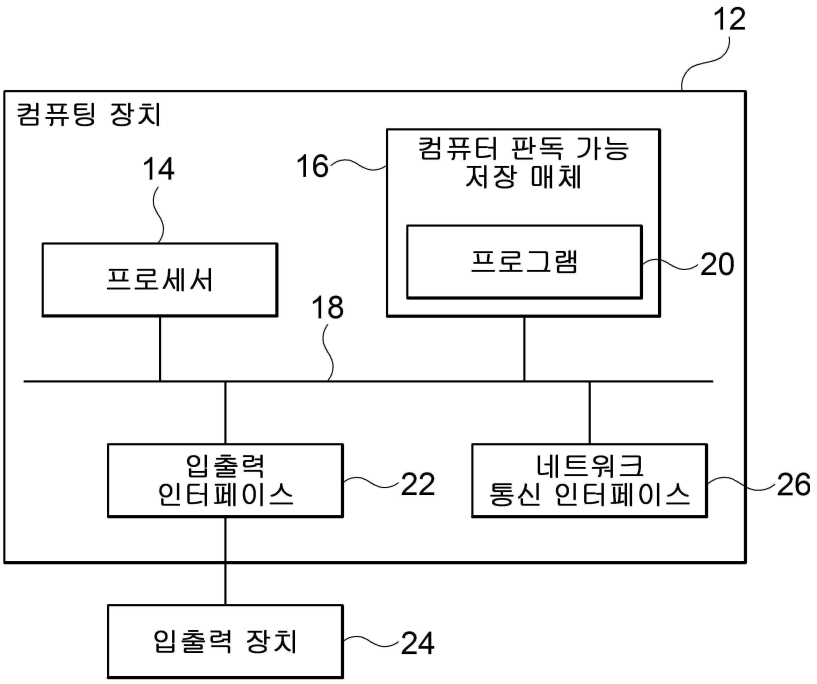
500


소프트웨어 바이너리 파일을 역어셈블하여 어셈블리어 코드 생성
502
어셈블리어 코드에 대한 제어 흐름 그래프를 생성하고, 어셈블리어 코드가 의심 함수들 중 적어도 하나를 포함하고 있는 경우, 제어 흐름 그래프에서 의심 함수를 호출한 노드 및 노드의 부모 노드들의 주소 의심 노드 집합으로 수집
504
의심 노드 집합에 포함된 노드를 주소를 기반으로 기호 실행을 수행함으로써 소프트웨어 바이너리 파일의 이상을 유발하기 위한 입력값을 탐색
506

도면6


506
임의의 입력값을 생성하여 소프트웨어 바이너리 파일을 실행
602
제어 흐름 그래프에 포함된 노드들 중, 입력값에 의하여 실행되는 노드들의 주소를 수집
604
수집된 주소를 이용하여 제어 흐름 그래프의 진입점부터 기호 실행을 수행
608
분기 발견?
608
NO
YES
수집된 경로의 반대 경로가 의심 노드 집합에 포함?
610
NO
YES
해당 반대 경로로 진입하기 위한 입력값 생성
612
생성된 입력값을 이용하여 소프트웨어 바이너리 파일을 실행함으로써 이상 발생 여부 탐지
614
해당 반대 경로를 탐색 대상에서 제외
616

도면7


10
12
컴퓨팅 장치
14
프로세서
16
컴퓨터 판독 가능
저장 매체
프로그램
20
18
입출력
인터페이스
22
네트워크
통신 인터페이스
26
입출력 장치
24