(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2019년03월29일
(11) 등록번호 10-1963752
(24) 등록일자 2019년03월25일

(51) 국제특허분류(Int. Cl.)
G06F 21/57 (2013.01) *G06F 21/56* (2013.01)
(52) CPC특허분류
G06F 21/577 (2013.01)
G06F 21/563 (2013.01)
(21) 출원번호 10-2018-0133775
(22) 출원일자 2018년11월02일
심사청구일자 2018년11월02일
(56) 선행기술조사문헌
KR1020180060497 A*
강상용 외 2인, '동적 기호 실행을 이용한 윈도우 시스템 콜 Use-After-Free 취약점 자동 탐지 방법', 정보보호학회논문지, 한국정보보호학회, 2017.08, pp.803-810*
KR1020180010053 A
KR101674895 B1
*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)
최민준
인천광역시 부평구 세월천로 16, 107동 2504호(청천동, 청천푸르지오아파트)
루스타모브 파이요즈벡
서울특별시 광진구 능동로 209, 세종대학교 율곡관 702호 (군자동, 세종대학교)
(74) 대리인
두호특허법인

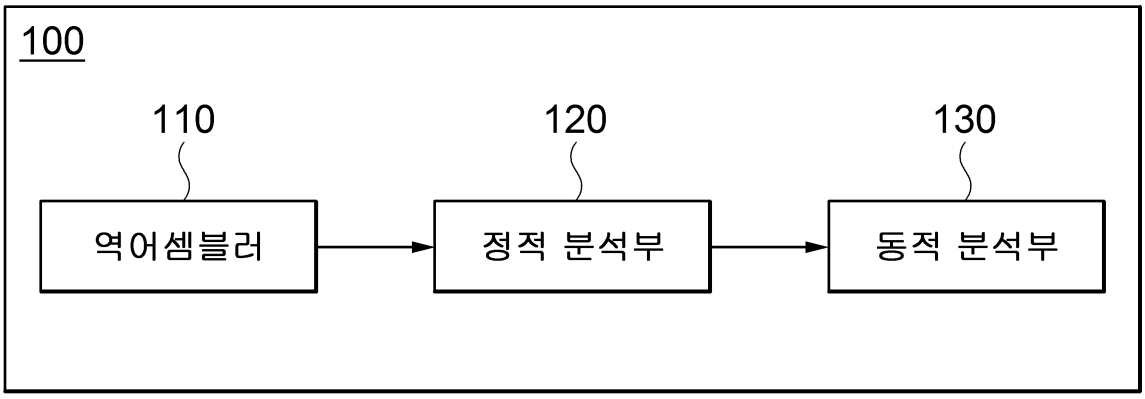
전체 청구항 수 : 총 10 항 심사관 : 구대성

(54) 발명의 명칭 **소프트웨어 취약점 분석 장치 및 방법**

(57) 요 약

소프트웨어 취약점 분석 장치 및 방법이 개시된다. 본 발명의 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, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하는 정적 분석부; 및 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에 기초하여 상기 의심 함수를 실행하기 위한 최적 실행 경로를 결정하고, 상기 소프트웨어 바이너리 파일을 이용하여 상기 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행하는 동적 분석부를 포함한다.

대 표 도 - 도1



이 발명을 지원한 국가연구개발사업
과제고유번호 1711075702
부처명 과학기술정보통신부
연구관리전문기관 정보통신기술진흥센터
연구사업명 정보통신기술인력양성(정보화)
연구과제명 지능형 비행로봇 융합기술 연구
기 여 율 7/10
주관기관 세종대학교 산학협력단
연구기간 2018.06.01 ~ 2021.12.31
이 발명을 지원한 국가연구개발사업
과제고유번호 1345281963
부처명 교육부
연구관리전문기관 한국연구재단
연구사업명 기본연구(1년~5년)
연구과제명 검색 기반 소프트웨어 보안취약점 자동 탐지 기술 연구
기 여 율 3/10
주관기관 세종대학교
연구기간 2018.06.01 ~ 2021.05.31

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, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하는 정적 분석부; 및

상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에 기초하여 상기 의심 함수를 실행하기 위한 최적 실행 경로를 결정하고, 상기 소프트웨어 바이너리 파일을 이용하여 상기 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행하는 동적 분석부를 포함하며,

상기 동적 분석부는, 상기 제어 흐름 그래프 상에서 엔트리 노드로부터 상기 의심 함수를 포함하는 타겟 노드까지의 최단 실행 경로를 상기 최적 실행 경로로 결정하는 소프트웨어 취약점 분석 장치.

청구항 2

청구항 1에 있어서,

상기 의심 함수 리스트는, 사전에 발견된 취약점과 관련된 함수를 포함하는 소프트웨어 취약점 분석 장치.

청구항 3

청구항 2에 있어서,

상기 의심 함수 리스트는, CVE(Common Vulnerabilities and Exposure) 에 등록된 취약점과 관련된 함수를 포함하는 소프트웨어 취약점 분석 장치.

청구항 4

삭제

청구항 5

청구항 1에 있어서,

상기 동적 분석부는, 상기 최단 실행 경로를 제외한 나머지 실행 경로를 프루닝(pruning)하는 소프트웨어 취약점 분석 장치.

청구항 6

청구항 5에 있어서,

상기 동적 분석부는, 상기 나머지 실행 경로 상에 존재하는 각 노드에 대응되는 블록의 시작 메모리 주소를 포함하는 회피 블록 리스트를 생성하고, 상기 회피 블록 리스트에 기초하여 상기 나머지 실행 경로를 프루닝하는 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 생성하는 단계;

문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하는 단계;

상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에 기초하여 상기 의심 함수를 실행하기 위한 최적 실행 경로를 결정하는 단계; 및

상기 소프트웨어 바이너리 파일을 이용하여 상기 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행하는 단계를 포함하며,

상기 최적 실행 경로를 결정하는 단계는, 상기 제어 흐름 그래프 상에서 엔트리 노드로부터 상기 의심 함수를 포함하는 타겟 노드까지의 최단 실행 경로를 상기 최적 실행 경로로 결정하는 소프트웨어 취약점 분석 방법.

청구항 8

청구항 7에 있어서,

상기 의심 함수 리스트는, 사전에 발견된 취약점과 관련된 함수를 포함하는 소프트웨어 취약점 분석 방법.

청구항 9

청구항 8에 있어서,

상기 의심 함수 리스트는, CVE(Common Vulnerabilities and Exposure) 에 등록된 취약점과 관련된 함수를 포함하는 소프트웨어 취약점 분석 방법.

청구항 10

삭제

청구항 11

청구항 7에 있어서,

상기 최적 실행 경로를 결정하는 단계는, 상기 최단 실행 경로를 제외한 나머지 실행 경로를 프루닝(pruning)하는 단계를 더 포함하는 소프트웨어 취약점 분석 방법.

청구항 12

청구항 11에 있어서,

상기 프루닝하는 단계는, 상기 나머지 실행 경로 상에 존재하는 각 노드에 대응되는 블록의 시작 메모리 주소를 포함하는 회피 블록 리스트를 생성하고, 상기 회피 블록 리스트에 기초하여 상기 나머지 실행 경로를 프루닝하는 소프트웨어 취약점 분석 방법.

발명의 설명

기 술 분 야

[0001] 본 발명의 실시예들은 소프트웨어 취약점 분석 기술과 관련된다.

배 경 기 술

[0002] 엄청나게 많은 소프트웨어의 보안 취약점을 전문가가 수동으로 분석하는 것은 불가능하다. 따라서, 소프트웨어에 대한 취약점을 자동으로 분석하기 위한 방법에 대해 많은 진행되고 있다.

[0003] 그러나, 기존 연구들은 취약점을 100% 탐지하지 못하거나 점검 시간이 오래 걸린다는 단점이 존재하며, 수많은 프로그램 분기문 등으로 인해 발생하는 경로 폭발(path explosion) 문제가 해결되지 않고 있다.

선행기술문헌

특허문헌

[0004] (특허문헌 0001) 대한민국 등록특허 제10-1796369호 (2017.12.01. 공고)

발명의 내용

해결하려는 과제

[0005] 본 발명의 실시예들은 소프트웨어 바이너리 파일과 안전하지 않은 함수 정보를 활용하여 소프트웨어 취약점을 분석하기 위한 장치 및 방법을 제공하기 위한 것이다.

과제의 해결 수단

[0006] 본 발명의 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, 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하는 정적 분석부; 및 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에 기초하여 상기 의심 함수를 실행하기 위한 최적 실행 경로를 결정하고, 상기 소프트웨어 바이너리 파일을 이용하여 상기 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행하는 동적 분석부를 포함한다.

[0007] 상기 의심 함수 리스트는, 사전에 발견된 취약점과 관련된 함수를 포함할 수 있다.

[0008] 상기 의심 함수 리스트는, CVE(Common Vulnerabilities and Exposure)에 등록된 취약점과 관련된 함수를 포함할 수 있다.

[0009] 상기 동적 분석부는, 상기 제어 흐름 그래프 상에서 엔트리 노드로부터 상기 의심 함수를 포함하는 타겟 노드까지의 최단 실행 경로를 상기 최적 실행 경로로 결정할 수 있다.

[0010] 상기 동적 분석부는, 상기 최단 실행 경로를 제외한 나머지 실행 경로를 프루닝(pruning)할 수 있다.

[0011] 상기 동적 분석부는, 상기 나머지 경로 상에 존재하는 각 노드에 대응되는 블록의 시작 메모리 주소를 포함하는 회피 블록 리스트를 생성하고, 상기 회피 블록 리스트에 기초하여 상기 나머지 실행 경로를 프루닝할 수 있다.

[0012] 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법은, 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 생성하는 단계; 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단하는 단계; 상기 어셈블리어 코드에 상기 의심 함수가 포함되어 있는 경우, 상기 제어 흐름 그래프에 기초하여 상기 의심 함수를 실행하기 위한 최적 실행 경로를 결정하는 단계; 및 상기 소프트웨어 바이너리 파일을 이용하여 상기 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행하는 단계를 포함한다.

[0013] 상기 의심 함수 리스트는, 사전에 발견된 취약점과 관련된 함수를 포함할 수 있다.

[0014] 상기 의심 함수 리스트는, CVE(Common Vulnerabilities and Exposure) 에 등록된 취약점과 관련된 함수를 포함할 수 있다.

[0015] 상기 최적 실행 경로를 결정하는 단계는, 상기 제어 흐름 그래프 상에서 엔트리 노드로부터 상기 의심 함수를

포함하는 타겟 노드까지의 최단 실행 경로를 상기 최적 실행 경로로 결정할 수 있다.

[0016] 상기 최적 실행 경로를 결정하는 단계는, 상기 최단 실행 경로를 제외한 나머지 실행 경로를 프루닝(pruning)하는 단계를 더 포함할 수 있다.

[0017] 상기 프루닝하는 단계는, 상기 나머지 경로 상에 존재하는 각 노드에 대응되는 블록의 시작 메모리 주소를 포함하는 회피 블록 리스트를 생성하고, 상기 회피 블록 리스트에 기초하여 상기 나머지 실행 경로를 프루닝할 수 있다.

발명의 효과

[0018] 본 발명의 실시예들에 따르면, 안전하지 않은 함수 정보를 활용한 문자열 검색을 통해 소프트웨어 바이너리 파일 내에 취약점이 존재할 만한 부분을 식별한 후 식별된 부분을 타겟으로 한 최적 실행 경로에 따라 동적 기호 실행(concolic execution)을 실행하도록 함으로써, 소프트웨어의 소스 코드 없이도 정확하고 효율적인 취약점 점검이 가능하며, 경로 폭발 문제를 해결할 수 있다.

도면의 간단한 설명

[0019] 도 1은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치의 구성도

도 2는 본 발명의 일 실시예에 따른 제어 흐름 그래프의 일 예를 도시한 도면

도 3은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법의 순서도

도 4는 예시적인 실시예들에서 사용되기에 적합한 컴퓨팅 장치를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도

발명을 실시하기 위한 구체적인 내용

[0020] 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.

[0021] 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.

[0022] 도 1은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치의 구성도이다.

[0023] 도 1을 참조하면, 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치(100)은 역 어셈블러(110), 정적 분석부(120) 및 동적 분석부(130)를 포함한다.

[0024] 역어셈블러(110)는 취약점 분석 대상이 소프트웨어 바이너리 파일을 역어셈블(disassemble)하여 해당 바이너리 파일에 대한 어셈블리어 코드를 생성한다.

[0025] 정적 분석부(120)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.

[0026] 한편, 정적 분석부(120)는 문자열 검색을 통해 어셈블리 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단한다.

[0027] 구체적으로, 정적 분석부(120)는 문자열 검색을 통해 어셈블리 코드 내에서 의심 함수 리스트에 포함된 의심 함수를 포함하고 있는 기본 블록(basic block)을 식별할 수 있다.

[0028] 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.

[0029] 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.

[0030] 동적 분석부(130)는 어셈블리어 코드에서 하나 이상의 의심 함수가 검색된 경우, 제어 흐름 그래프에 기초하여 각 의심 함수에 대한 최적 실행 경로를 결정한다.

[0031] 구체적으로, 동적 분석부(130)는 제어 흐름 그래프 상에서 엔트리(entry) 노드로부터 의심 함수를 포함하는 블록에 해당하는 타겟(target) 노드까지의 최단 실행 경로를 해당 의심 함수에 대한 최적 실행 경로로 결정할 수 있다. 이때, 최단 실행 경로는 예를 들어, 다익스트라(dijkstra) 알고리즘을 이용하여 결정될 수 있으나, 이 외에도 공지된 다양한 기술을 이용하여 결정될 수 있다.

[0032] 한편, 본 발명의 일 실시예에 따르면, 동적 분석부(130)는 최적 실행 경로가 결정된 경우, 최적 실행 경로를 제외한 나머지 실행 경로들을 프루닝(pruning)할 수 있다.

[0033] 구체적으로, 도 2는 본 발명의 일 실시예에 따른 제어 흐름 그래프의 일 예를 도시한 도면이다.

[0034] 도 2에 도시된 제어 흐름 그래프(200)에서 사각형으로 표시된 부분은 노드를 나타내며, 각 노드는 어셈블리 코드에 포함된 기본 블록을 나타낸다. 또한, 각 노드 사이의 화살표는 실행 경로를 나타내며, 사선(/)으로 표시된 화살표는 프루닝된 실행 경로를 나타낸다.

[0035] 한편, 'bug' 노드(240)에 해당하는 블록에서 의심 함수를 포함하고 있는 것으로 가정하면, 동적 분석부(130)는 제어 흐름 그래프 상에서 엔트리 노드인 'main() start' 노드(210)로부터 타겟 노드인 'bug' 노드(240)까지의 모든 실행 경로 중 최단 실행 경로를 최적 실행 경로로 결정할 수 있다. 구체적으로, 도시된 예에서, 최적 실행 경로는 'main() start' 노드(210) =>'file open' 노드(220)=>'check handle' 노드(230)=>'bug' 노드(240)로 이루어진 실행 경로이다.

[0036] 한편, 동적 분석부(130)는 최적 실행 경로 상에 있는 각 노드에 대응되는 블록의 시작 메모리 주소를 필수 블록 리스트에 포함시키고, 나머지 노드들에 대응되는 기본 블록의 시작 메모리 주소를 회피 블록 리스트에 포함시킴으로써 최적 실행 경로를 제외한 나머지 실행 경로를 프루닝 시킬 수 있다.

[0037] 한편, 최적 실행 경로가 결정된 경우, 동적 분석부(130)는 소프트웨어 바이너리 파일을 이용하여 최적 실행 경로에 대한 동적 기호 실행(concolic execution)을 수행한다.

[0038] 이때, 동적 기호 실행은 실제 수행(concrete execution)과 기호 실행(symbolic execution)을 모두 이용하여 경로 제약조건(path constraint)을 해결하기 위한 기술이다.

[0039] 구체적으로, 동적 분석부(130)는 회피 블록 리스트에 포함된 블록으로의 실행 흐름을 배제하고, 필수 블록 리스트에 포함된 블록으로 실행 흐름이 진행되도록 입력 값 조건을 결정할 수 있으며, 이를 이용하여 최적 실행 경로에 대한 동적 기호 실행이 수행되도록 할 수 있다.

[0040] 도 3은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법의 순서도이다.

[0041] 도 3에 도시된 방법은 예를 들어, 도 1에 도시된 소프트웨어 취약점 분석 장치(100)에 의해 수행될 수 있다.

[0042] 도 3을 참조하면, 우선, 소프트웨어 취약점 분석 장치(100)는 취약점 분석 대상인 소프트웨어 바이너리 파일을 역어셈블하여 상기 소프트웨어 바이너리 파일에 대한 어셈블리어 코드를 생성한다(310).

[0043] 이후, 소프트웨어 취약점 분석 장치(100)는 어셈블리어 코드에 대한 제어 흐름 그래프(control flow graph)를 생성한다(320).

[0044] 이후, 소프트웨어 취약점 분석 장치(100)는 문자열 검색을 통해 상기 어셈블리어 코드가 사전 정의된 의심 함수 리스트에 포함된 의심 함수들 중 적어도 하나를 포함하고 있는지 여부를 판단한다(330).

[0045] 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.

[0046] 한편, 어셈블리어 코드에 의심 함수가 포함되어 있는 경우, 소프트웨어 취약점 분석 장치(100)는 제어 흐름 그래프에 기초하여 검색된 의심 함수를 실행하기 위한 최적 실행 경로를 결정한다(340).

[0047] 이때, 본 발명의 일 실시예에 따르면, 소프트웨어 취약점 분석 장치(100)는 제어 흐름 그래프 상에서 엔트리 노드로부터 의심 함수를 포함하는 타겟 노드까지의 최단 실행 경로를 상기 최적 실행 경로로 결정할 수 있다.

[0048] 또한, 본 발명의 일 실시예에 따르면, 소프트웨어 취약점 분석 장치(100)는 최단 실행 경로가 결정된 경우, 제외한 나머지 실행 경로를 프루닝(pruning)할 수 있다. 구체적으로, 소프트웨어 취약점 분석 장치(100)는 최적 실행 경로를 제외한 나머지 경로 상에 존재하는 각 노드에 대응되는 블록의 시작 메모리 주소를 포함하는 회피 블록 리스트를 생성하고, 생성된 회피 블록 리스트에 기초하여 상기 나머지 실행 경로를 프루닝할 수 있다.

[0049] 이후, 소프트웨어 취약점 분석 장치(100)는 소프트웨어 바이너리 파일을 이용하여 최적 실행 경로에 대한 동적 기호 실행 실행(concolic execution)을 수행한다(350).

[0050] 한편, 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.

[0051] 도 4는 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.

[0052] 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.

[0053] 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.

[0054] 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.

[0055] 통신 버스(18)는 프로세서(14), 컴퓨터 판독 가능 저장 매체(16)를 포함하여 컴퓨팅 장치(12)의 다른 다양한 컴포넌트들을 상호 연결한다.

[0056] 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.

[0057] 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, 후

술하는 특허청구범위뿐만 아니라 이 특허청구범위와 균등한 것들에 의해 정해져야 한다.

부호의 설명

[0058] 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

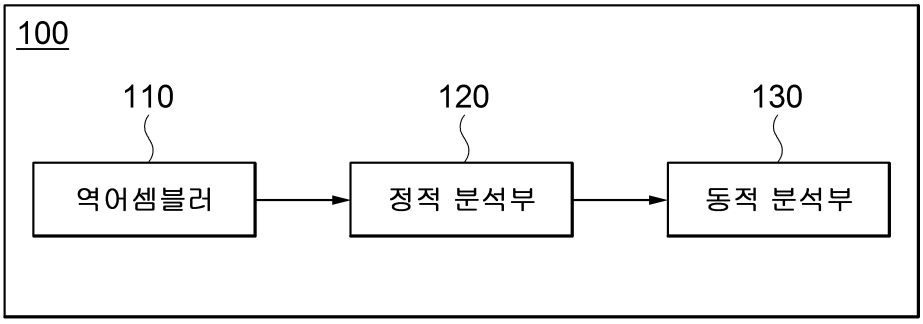
110: 역어셈블러

120: 정적 분석부

130: 동적 분석부

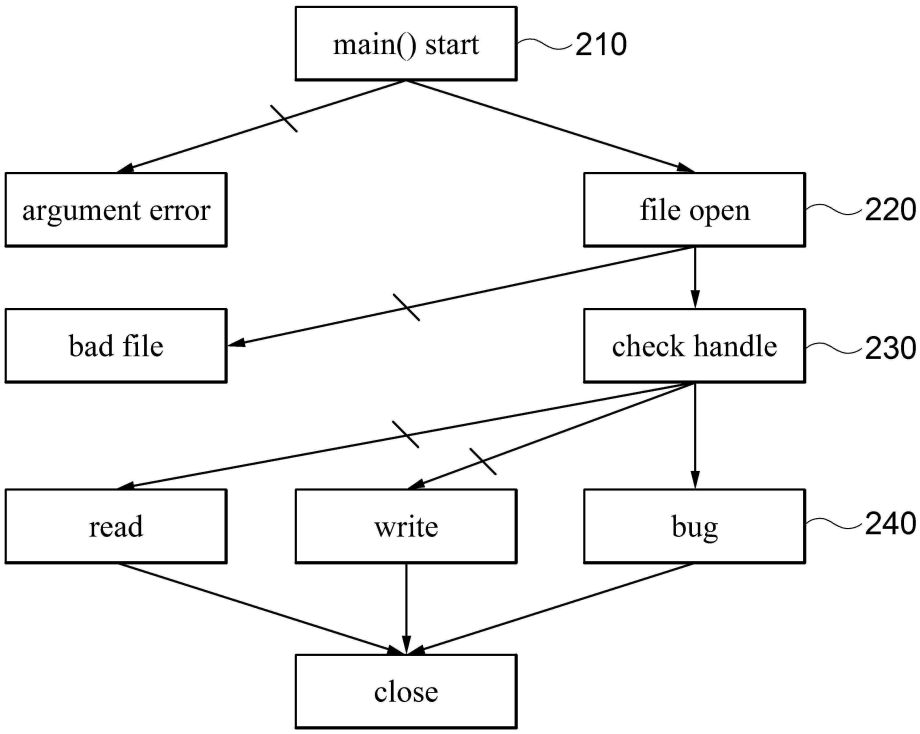
도면

도면1

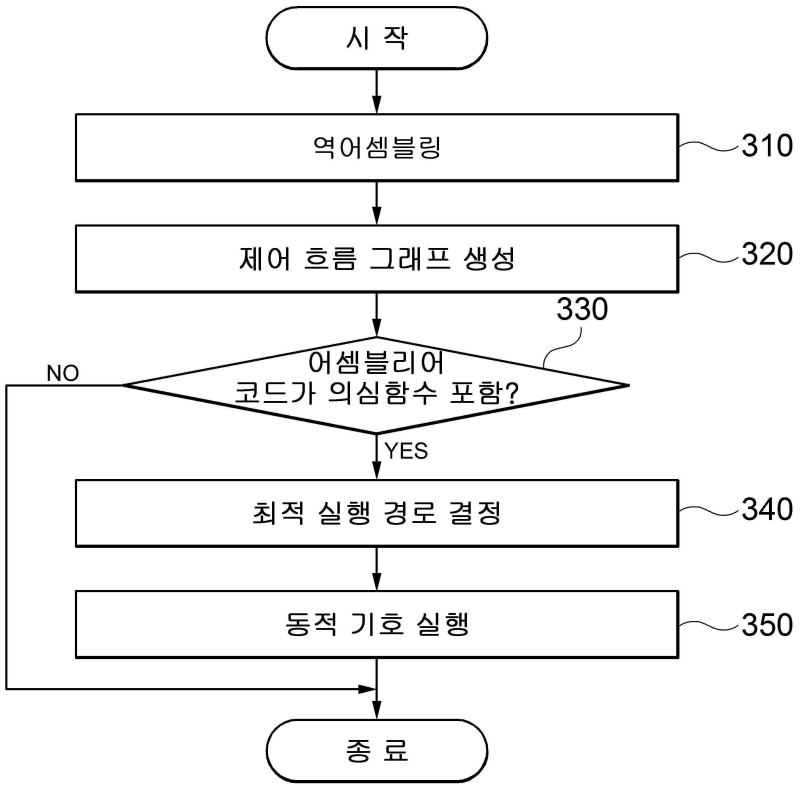


도면2

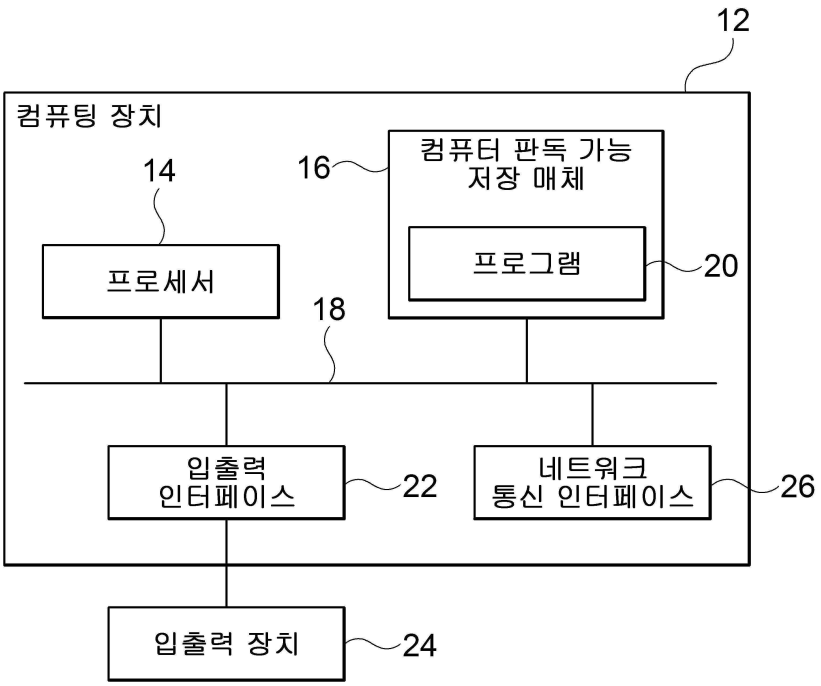
200


main() start
210
argument error
file open
220
bad file
check handle
230
read
write
bug
240
close

도면3


시 작
역어셈블링
310
제어 흐름 그래프 생성
320
330
어셈블리어
코드가 의심함수 포함?
NO
YES
최적 실행 경로 결정
340
동적 기호 실행
350
종 료

도면4


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