(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2018-0076550
(43) 공개일자 2018년07월06일

(51) 국제특허분류(Int. Cl.)
G06F 21/57 (2013.01) *G06F 21/56* (2013.01)
(52) CPC특허분류
G06F 21/577 (2013.01)
G06F 21/562 (2013.01)
(21) 출원번호 10-2016-0180831
(22) 출원일자 2016년12월28일
심사청구일자 없음

(71) 출원인
충남대학교산학협력단
대전광역시 유성구 대학로 99 (궁동, 충남대학교)
(72) 발명자
조은선
대전광역시 유성구 어은로 57, 110동 1504호
전현구
대전광역시 유성구 문화원로 106, 408호
(뒷면에 계속)
(74) 대리인
홍성욱, 심경식

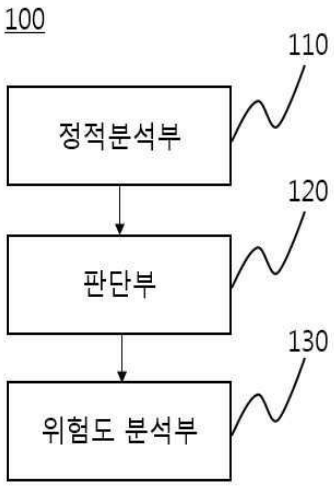
전체 청구항 수 : 총 1 항

(54) 발명의 명칭 **다중 함수 정적 분석 장치 및 방법**

(57) 요 약

입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행하는 정적분석부, 정적 분석의 결과에 기초하여 입력 데이터가 다른 함수를 호출하는지 여부를 판단하는 판단부 및 판단 결과에 기초하여 다른 함수에 영향을 주는 적어도 하나의 자유 변수를 추출하고, 적어도 하나의 자유 변수에 대한 위험도를 분석하는 위험도 분석부를 포함하는, 다중 함수 정적 분석 장치 및 그 방법에 관한 것이다.

대 표 도 - 도1



(72) 발명자

목성균

충청북도 청주시 서원구 청남로1887번길 74, A동 408호

김영철

서울특별시 강남구 개포로 310, 20동 101호

이 발명을 지원한 국가연구개발사업

과제고유번호 R2718160003000 1002

부처명 미래창조과학부

연구관리전문기관 정보통신기술진흥센터

연구사업명 대학ICT연구센터육성지원사업

연구과제명 지능정보기술을 활용한 금융서비스 보호

기 여 율 1/1

주관기관 충남대학교 산학협력단

연구기간 2016.06.01 ~ 2016.12.31

명 세 서

청구범위

청구항 1

입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행하는 정적분석부;

상기 정적 분석의 결과에 기초하여 상기 입력 데이터가 다른 함수를 호출하는지 여부를 판단하는 판단부; 및

상기 판단 결과에 기초하여 상기 다른 함수에 영향을 주는 적어도 하나의 자유 변수를 추출하고, 상기 적어도 하나의 자유 변수에 대한 위험도를 분석하는 위험도 분석부를 포함하는, 다중 함수 정적 분석 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 바이너리 프로그램에 대한 정적 분석에서 분석 범위를 확장하는 다중 함수 정적 분석 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] 프로그램의 취약점을 방어하기 위해서는 취약점이 발생하지 않게 소스코드를 작성하는 것도 중요하지만, 배포된 프로그램이 실제로 해커들에게 공격 당할 가능성이 있는지 해커의 입장에서 검증하는 것 또한 매우 중요하다.

[0003] 이러한 해커의 입장에서 프로그램의 취약점을 검증하는 프로그램 검증 도구인 퍼저(Fuzzer)는 입력데이터를 조작하여 크래시를 인위적으로 유발하며, 크래시가 발생하는지 유무로 프로그램을 검증한다.

[0004] 여기서, 소스코드가 없는 바이너리 프로그램에서 퍼징(Fuzzing)을 통해 입력 데이터를 조작하여 크래시를 발생시키고, 이 크래시를 분석하여 취약점을 찾는 해커들의 패턴이 있으며, 이 과정에서 퍼징(Fuzzing)은 자동화가 되어 있지만 발생한 크래시가 취약점인지 여부를 확인하는 과정은 수백에서 수천 개 베이지 블록(Basic Block)을 수동으로 확인 해야 하기 때문에 오랜 시간이 소요되는 문제가 있다.

[0005] 여기서, 취약점이 발생하기 위한 가장 중요한 조건은 조작한 입력데이터에 의해 공격자가 원하는 PC값이 바뀔 수 있는지 여부이며, 대부분의 경우 발생한 크래시가 바로 공격지점으로 이어지지만 최근 크래시 발생지점과 공격지점이 서로 떨어져 있는 공격 방식이 출현하고 있다.

[0006] 이 경우, 크래시 발생에 영향을 준 데이터가 어떤 경로로 전파되고, 어떤 명령어에서 공격 가능한지 판단하는 것은 매우 어려우며, 즉, 크래시가 발생한 지점에서 함수 종료까지 수백 개 이상 베이직 블록(Basic Block)과 명령어들이 있기 때문에 이러한 취약점을 수동으로 분석하는 것은 매우 어려운 문제가 있다.

선행기술문헌

특허문헌

[0007] (특허문헌 0001) 한국 등록특허공보 제10-1159365호(2012.06.18)

발명의 내용

해결하려는 과제

[0008] 본 발명의 목적은, 상기 문제점을 해결하기 위한 것으로 입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행하고, 정적 분석의 결과에 기초하여 입력 데이터가 다른 함수를 호출하는지 여부를 판단하여, 다른 함수를 호출하는 경우 추출된 적어도 하나의 자유 변수에 대한 위험도를 분석하여, 바이너리 프로그램에 대한 정적 분석에서 분석범위를 확장하기 위함이다.

[0009] 본 발명이 해결하고자 하는 과제는 이상에서 언급한 과제(들)로 제한되지 않으며, 언급되지 않은 또 다른 과제(들)은 아래의 기재로부터 당업자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0010] 상기한 목적을 달성하기 위하여, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치는 입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행하는 정적분석부, 정적 분석의 결과에 기초하여 입력 데이터가 다른 함수를 호출하는지 여부를 판단하는 판단부 및 판단 결과에 기초하여 다른 함수에 영향을 주는 적어도 하나의 자유 변수를 추출하고, 적어도 하나의 자유 변수에 대한 위험도를 분석하는 위험도 분석부를 포함한다.

발명의 효과

[0011] 본 발명의 일 실시예에 따르면, 입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행하고, 정적 분석의 결과에 기초하여 입력 데이터가 다른 함수를 호출하는지 여부를 판단하여, 다른 함수를 호출하는 경우 추출된 적어도 하나의 자유 변수에 대한 위험도를 분석하여, 바이너리 프로그램에 대한 정적 분석에서 분석범위를 확장하여, 크래시를 정적 분석 함에 있어서 다양한 정적 분석 기법을 활용하여 사용자에게 유용한 정보를 제공해 줄 수 있을 뿐 아니라, 넓은 범위를 분석할 수 있는 기술을 활용하여 사용자에게 더 정확한 정보를 제공 할 수 있다.

도면의 간단한 설명

[0012] 도 1은 본 발명의 실시예에 따른 다중 함수 정적 분석 장치를 설명하기 위한 구성도이다.

도 2는 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명하기 위한 도면이다.

도 3 및 도 4는 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명하기 위한 도면이다.

도 5 내지 도 14는 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명하기 위한 도면이다.

도 15는 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치의 분석 결과와 종래 기술의 분석 결과를 비교하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0013] 이하, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 본 발명의 기술적 사상을 용이하게 실시할 수 있을 정도로 상세히 설명하기 위하여, 본 발명의 가장 바람직한 실시예를 첨부 도면을 참조하여 설명하기로 한다. 우선 각 도면의 구성요소들에 참조부호를 부가함에 있어서, 동일한 구성요소들에 대해서는 비록 다른 도면상에 표시되더라도 가능한 한 동일한 부호를 가지도록 하고 있음에 유의해야 한다. 또한, 본 발명을 설명함에 있어, 관련된 공지 구성 또는 기능에 대한 구체적인 설명이 본 발명의 요지를 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명은 생략한다.

[0014] 이하, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 첨부된 도면을 참조하여 상세하게 설명하면 아래와 같다.

[0015] 도 1은 본 발명의 실시예에 따른 다중 함수 정적 분석 장치를 설명하기 위한 구성도이다.

[0016] 도 1에 도시된 바와 같이 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 정적분석부(110), 판단부(120) 및 위험도 분석부(130)를 포함한다.

[0017] 정적분석부(110)는 입력된 바이너리 프로그램에 대한 데이터인 입력 데이터에 대한 정적 분석을 수행한다.

[0018] 판단부(120)는 정적 분석의 결과에 기초하여 입력 데이터가 다른 함수를 호출하는지 여부를 판단한다.

[0019] 위험도 분석부(130)는 판단 결과에 기초하여 다른 함수에 영향을 주는 적어도 하나의 자유 변수를 추출하고, 적어도 하나의 자유 변수에 대한 위험도를 분석한다.

[0020] 예컨대, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 바이너리 프로그램을 정적 분석 하는데 있어서 분석 범위를 다중함수(Inter-Procedure) 범위로 확장하기 위한 것일 수 있다.

[0021] 예를 들어, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 프로그램 정적 분석에 있어서 분석 범위

를 다중함수 범위로 확장하기 위해서 함수 호출에서 호출하는 함수(Caller)와 호출되는 함수(Callee)사이에 전달되는 데이터를 기준으로 분석을 이어 나가는 방법을 사용할 수 있다.

[0022] 이제 도 2를 참조하여, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명한다.

[0023] 도 2는 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명하기 위한 도면이다.

[0024] 정적분석부(110)는 입력된 바이너리 프로그램에 대한 데이터인 입력 데이터(Taint Source, 크래시 정보)에 대한 정적 분석을 수행한다.

[0025] 이때, 정적분석부(110)가 수행하는 정적 분석은 리칭-데프(Reaching Def) 분석, 메모리 영역 분석, 데프-유즈-체인(Def-Use-Chain) 분석 및 익스플로이터블(Exploitable) 분석을 포함할 수 있다.

[0026] 예컨대, 정적분석부(110)가 수행하는 정적 분석은 상술한 4가지 정적 분석 방법을 순차적으로 수행하는 것일 수 있으나, 본 발명은 이에 한정되지 않는다.

[0027] 예컨대, 정적분석부(110)가 수행하는 정적 분석은 상술한 4가지 정적 분석 외에도 각종 정적 분석 방법을 포함할 수 있으며, 본 발명은 상술한 4가지 정적 분석 방법에 한정되지 않는다.

[0028] 판단부(120)는 정적 분석 과정을 거친 후 다른 함수를 호출하는지 여부로 다중함수 분석 여부를 판단한다.

[0029] 판단 결과, 다른 함수를 호출하지 않는 경우, 위험도 분석부(130)는 입력된 바이너리 프로그램에 대한 위험도를 분석하여 위험도 등급을 산출할 수 있다.

[0030] 판단 결과, 다른 함수를 호출하는 경우, 위험도 분석부(130)는 다른 함수에 영향을 주는 적어도 하나의 자유 변수를 추출한다.

[0031] 이때, 적어도 하나의 자유 변수는 전역 변수(Global Variable), 인수(Argument) 및 리턴값(Return Value)를 포함할 수 있으나 본 발명은 이에 한정되지 않는다.

[0032] 이 경우, 위험도 분석부(130)는 전역 변수(Global Variable), 인수(Argument) 및 리턴값(Return Value) 각각에 대한 위험도를 분석한 뒤, 결과를 종합하고, 위험도 등급을 산출할 수 있다.

[0033] 이 경우, 위험도 분석부(130)는 추출된 자유 변수 중 인수(Argument)를 정적분석부(110)에 전달하여, 정정분석부(110)가 인수(Argument)를 입력 데이터(Taint Source, 함수 인자)로 활용하여 추가적인 정적 분석을 수행하여 상술한 동작을 반복하도록 할 수 있다.

[0034] 상술한 동작이 반복되어 더 이상 다른 함수를 호출하지 않는 경우, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 결과를 저장한 뒤, 외부로 출력하여 사용자에게 보고함으로써 분석을 완료할 수 있다.

[0035] 다시 말해, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 바이너리 파일과 크래시 정보로 정적 분석을 실시하고 크래시의 위험도를 자동으로 판단하여 사용자에게 보고하여, 도움을 주는 자동화 도구의 분석 범위를 다중 함수 범위로 확장하기 위한 것일 수 있으나 본 발명은 이에 한정되지 않는다.

[0036] 예를 들어, 종래의 대표적인 크래시 분석도구로는 MicroSoft에서 제작한 "!exploitable"이라는 프로그램이 있으나, 종래의 크래시 분석 도구는 동적으로 크래시를 분석하여 위험도를 판단하고, 조건에 따라 일부 정적 분석을 사용하나, 단일 블록의 분석범위를 가지고 있어 그 이후의 위험도를 판단하지 못하는 문제가 있다.

[0037] 반면, 본 발명의 실시예에 따른 다중 함수 정적 분석 장치(100)는 종래의 단일 블록을 분석할 수 있을 뿐만 아니라, 다중 블록을 분석할 수 있고, 더 나아가 다중함수까지 분석 가능한 장점이 있다.

[0038] 이제 도 3 및 도 4를 참조하여, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법에 대하여 계속 설명한다.

[0039] 퍼저(fuzzer)를 사용해 생성된 크래시는 프로그램의 결점을 의미하고 이는 취약점을 유발 할 수 있으니 반드시 수정되어야 하며, 퍼저(fuzzer)를 통해 발생한 크래시는 무작위로 입력 값이 조작되어 발생한 결과이기 때문에 입력 데이터에 영향을 받았다고 할 수 있다.

[0040] 이 데이터로 인한 취약점을 탐지 하기 위해 동적 분석을 하려면 실행 트레이스(trace)가 필요하지만 크래시 발생 지점 이후는 실행 트레이스(trace)를 뽑을 수 없기 때문에 불가능한 문제가 있으며, 이를 해결하기 위해 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 정적 분석을 통해 이를 밝혀낼 수 있다.

[0041] 예를 들어, 퍼저(Fuzzer)는 자동으로 입력데이터의 일부를 무작위로 바꾸어 실행하며, 이때 프로그램의 예외처리가 잘 되지 않는다면 운영체제에서 강제로 수행을 종료시키는 크래시가 발생하게 되고, 퍼저(Fuzzer)는 이와 같은 방식으로 크래시가 발생하는지 유무를 통하여 프로그램을 테스팅 할 때 사용된다. 이와 같은 검증은 공격자 입장에서 크래시를 만들고 취약점으로 발전시키는 패턴을 미연에 방지하기 위함이며, 대표적인 도구로는 피치 퍼저(Peach Fuzzer)가 있다.

[0042] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법의 동작 순서는 도 3에 도시된 바와 같을 수 있다.

[0043] 유저에 의해 만들어진 크래시의 데이터가 입력데이터 조작을 통해 조작 가능하다면 이 데이터를 사용해 브랜치/점프 인스트럭션(branch/jump instruction)의 목적지 주소의 값에 영향을 줄 수 있으며, 이는 공격자가 원하는 곳으로 분기를 할 수 있는 취약점이 발생 할 수 있다는 것을 의미할 수 있다.

[0044] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 익스플로이터블 분석(exploitable analysis)을 통해 크래시에 영향을 받은 데이터가 크래시 포인트 이후에 취약점으로 발생 할 가능성이 있는지를 판단하여 발생한 크래시를 통하여 취약점이 발생할 수 있는지 검사하고, 위험도를 분류한다.

[0045] 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 먼저 IDA pro를 통해 디스어셈블된 데이터와 제어 흐름 데이터(Control Flow Data)가 포함된 idb파일을 추출하고, 이 파일을 활용하여 BinNavi를 통해 REIL 중간언어로 번역하는 전처리 과정을 거치며, 이후 본격적으로 익스플로이터블 포인트(exploitable point)를 찾기 위한 정적 분석을 실시하며 그 과정은 도 3에 도시된 바와 같다.

[0046] 예컨대, Zynamics에서 개발한 Binnavi는 프로그램 분석 및 디버그를 할 수 있는 도구로, 프로그램 분석을 위하여 기본적으로 패스 파인더(path finder)와 같은 플러그인을 제공할 뿐만 아니라 다양한 API와 REIL중간언어를 제공할 수 있다.

[0047] 여기서 REIL(Reversing engineering Intermediate Language)은 x86, ARM, PowerPC-32와 같은 다양한 명령어에 대한 변환이 가능하며, 명령어는 총 17개로 구성되어 있어 수백 개의 명령어를 가지고 있는 x86 및 ARM과 같은 복잡한 아키텍처보다 정적 분석을 수행하기 용이한 장점이 있다.

[0048] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 리칭 정의법(Reaching definition), 데프-유즈 체이닝(Def-use chaining) 및 익스프로이터블 확인(Exploitable checking)을 활용하여 정적 분석을 실행할 수 있다.

[0049] 예를 들어, IDA pro(Interactive Disassembler)는 바이너리 프로그램으로부터 어셈블리어 소스코드를 생성해주는 디스어셈블러를 의미할 수 있으며, 윈도우, ARM, 리눅스, Mac OS에서 사용가능하며 인텔x86, ARM 아키텍처 등 다양한 명령어 집합을 표현할 수 있는 프로그램을 의미하나 본 발명은 IDA pro 외에도 기 공지된 각종 프로그램을 활용하여 데이터를 디스어셈블 할 수 있으며, 본 발명은 IDA pro에 한정되지 않는다.

[0050] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 컴파일러의 정적 분석 기법인 리칭 정의법(Reaching definition) 분석을 통해 각 인스트럭션(instruction)에 도달 가능한 인스트럭션(instruction)이 어떤 것이 있는지 추출하고, 이 결과를 이용하여 데프-유즈 체이닝(Def-use chaining) 분석을 할 수 있다.

[0051] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 입력 데이터(Taint Source, 크래시 포인트)를 시작으로, 데프-유즈 체이닝(Def-use chaining)은 어떤 명령어 i1에 도달 가능한 정의(Definition)들 중에서 i1에서 사용(Use)하는 경우에 그 관계를 그래프로 반영한다.

[0052] 결과적으로 크래시에 영향을 받은 모든 명령어들의 관계가 분석되어 데프-유즈 그래프가 완성된다. 데프-유즈 그래프에 포함된 노드(vertex)는 크래시 포인트의 영향을 받은 인스트럭션(instructions)이고 이 중 공격자의 코드로 분기할 수 있는 가능성이 있는 명령어인 익스플로이터블 포인트(exploitable point) 가 있는지 탐지하는 익스프로이터블 확인(Exploitable checking)작업을 거친다. 이에 해당하는 경우를 인스트럭션(instruction)의 크래시의 위험도 판단 기준에 의해 결정하고, 분석이 종료됨과 함께 오염된 데이터의 경로와 분석 결과를 저장하고 사용자에게 보여줄 수 있으며, 위험도 분류 정책은 도 4에 도시된 바와 같을 수 있다.

[0053] 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 크래시 분류를 보수적으로 하기 위해 더 위험한 크래시 위험도로 최종 분류하여 사용자에게 알릴 수 있으며, 오염된 데이터는 다양한 경로로 진행

될 가능성이 있고, 둘 이상의 분류기준에 해당될 수도 있다.

[0054] 이 경우 보수적인 분석결과를 사용자에게 제공하기 위하여 더 위험한 분류기준으로 최종 분류하여 보고한다. 예컨대, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법의 결과를 활용하여 프로그램을 보완 입장에서 NE(not exploitable)로 분류된 크래시를 공격가능성이 없는 것으로 간주하여 보완대상에서 제외할 수 있어 분석 범위를 줄일 수 있는 장점이 있다.

[0055] 반대로 E(exploitable)의 경우 위험한 부분을 찾아 수정해야 하기 때문에 중요할 수 있으며, PE(probably exploitable)로 분류된 크래시의 경우 데이터 또는 목적지 주소만 제어 가능한 경우를 의미하여 공격 가능성이 상당히 낮은 경우를 의미할 수 있으며, 사용자에게 더 유용한 정보를 제공하기 위해서는 PE(probably exploitable)로 분류된 크래시 부분을 특히 더 정교하게 분석하여 정보를 제공할 필요가 있다.

[0056] 예를 들어, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 익스플로이터블 포인트(exploitable point)를 찾기 위한 정적 분석을 수행하기 위해 컴파일러의 최적화 기법을 사용하여 구현될 수 있으며, 사용자로 하여금 분석범위를 줄여주기 위해 NE(not exploitable)라는 결과를 제공할 수 있으며, 정확도를 보장하기 위해 보수적인 분석을 실시하기 때문에 메모리를 추상화하여 분석하고 이 때문에 분석결과에 노이즈가 더해지는 문제가 있을 수 있으며, 내부 단계(inner procedure) 범위에서 분석하여 생기는 문제도 있을 수 있으며, 이는, 크래시에 영향을 받은 오염된 데이터가 다른 함수를 호출하여 호출되는 함수(callee)에서 익스플로이터블 포인트(exploitable point)가 발생할 가능성은 탐지하지 못한다는 문제가 발생할 수 있다.

[0057] 이제 도 5 내지 도14를 참조하여, 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치 및 방법을 설명한다.

[0058] 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 상술한 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법에서의 메모리 영역 추상화와 내부 단계 분석(Inner Procedure Analysis)의 한계를 개선하기 위한 것일 수 있다.

[0059] 이하에서는, 추상화된 메모리영역을 더 정교하게 분석하기 위해 밸류-셋-분석(value-set-analysis)를 도입한 메모리 위치 분석(Memory Location Analysis, MLA), 내부 단계 분석(Inner Procedure Analysis)의 분석범위를 확장하기 위한 다중 함수 분석(Inter Procedure Analysis, IPA)에 대해 설명한다.

[0060] 그 결과, 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법과 같이 크래시 정보와 바이너리 파일을 입력으로 받아 본래 기능을 그대로 사용하면서 메모리 위치 분석과 인터 프로시져 분석을 선택적으로 추가하여 정교화하는 장치 및 방법을 의미할 수 있다.

[0061] 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치 및 방법의 구성은 도 5에 도시된 바와 같을 수 있다.

[0062] 메모리 위치 분석(Memory Location Analysis, MLA)은, 익스플로이터블 포인트(Exploitable Point)를 찾기 위해 데이터의 데프-유즈(def-use) 관계 분석을 하여 명령어들이 사용하는 변수의 상관관계를 분석한다.

[0063] 예컨대, 각 PC에서 명령어를 수행하기 위해서 데이터를 읽고 쓰는 과정에서, 레지스터와 메모리 영역을 사용하며, 메모리에 있는 데이터의 오염 전파 여부를 판단할 때, 실제로는 서로 다른 메모리 공간에 있기 때문에 오염되지 않은 메모리 영역의 값을 사용했음에도 불구하고 오염된 데이터를 사용한 것으로 분석 할 수 있다.

[0064] 여기서, 메모리 추상화의 단점은 도 6에 도시된 바와 같을 수 있다.

[0065] 이러한 현상이 나타나는 이유는 다음과 같다. 레지스터의 경우 인스트럭션(instruction)의 오퍼랜드(operand)에서 대상이 명확하지만, 메모리 영역의 경우 명령어 수행에 따라 스택의 모양이 달라지고 esp, ebp, SP 같은 특수한 레지스터를 사용해 접근한다.

[0066] 그 결과 오퍼랜드(operand)가 같아도 각 명령어에서 가리키는 대상은 다를 수 있고, 명령어의 오퍼랜드(operand)만 보고는 메모리영역의 대상을 특정 할 수 없기 때문에, 본 발명의 제1 실시예에 따른 다중 함수 정적 분석 장치 및 방법에서는 모든 메모리 영역을 하나의 레지스터(a register)처럼 하나의 공간으로 추상화하여 분석하였고 그 결과 오버 테인트(over taint)가 발생 할 수 있다. 이 경우 오염되지 않은 데이터를 오염되었다고 분석하여 NE(not exploitable)로 분류되어야 할 크래시가 E(exploitable)나 PE(probably exploitable)로 분류될 수 있는 원인이 된다.

[0067] 이를 정교화 하려면 서로 다른 메모리를 정의하고 사용하는 관계를 명확하게 할 필요가 있다. 즉 데프-유즈(def-use) 관계를 분석할 때 메모리 분석을 정교화 함으로서 해결할 수 있다. 이를 위해 본 발명의 제2 실시예

에 따른 다중 함수 정적 분석 장치 및 방법에서는 각 명령어에서 특정 메모리영역 또는 레지스터가 가질 수 있는 값의 집합을 저장한 테이블을 작성하여 메모리 이미지(memory image)를 만든다. 데프(Definition) 명령어가 가질 수 있는 값의 집합과 유즈(Use) 명령어의 것을 비교하여 전혀 겹치는 부분이 없다면, 같은 메모리 주소를 활용한 것이 아니므로 데프-유즈 체인(def-use chain)을 끊는 방법으로 각각의 메모리를 구분할 수 있으며, 이러한 메모리 정교화의 목적은 도 7에 도시된 바와 같다.

[0068] 밸류-셋-분석(value-set-analysis)은 어떤 명령어에서 레지스터나 메모리가 가질 수 있는 값이 어떤 것이 있는지 파악하여 메모리 이미지(memory image)를 만드는 분석이다. 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법에서 각 명령어에서 메모리 이미지(memory image)는 2개의 맵으로 구성된다. 하나는 register - set of 'Values' (이하 rTable로 정의함) 그리고 다른 하나는 abstract memory location - set of 'Values' (이하 Env로 정의함)이다.

[0069] 이때,'Values'는 크게 4가지로 구성되며, 그 항목은 1) 오리지널 머신 코드 레지스터(original machine code register), 임시 레지스터(temporary register) 2) 상수값(Constant Values) 3) 메모리 위치(Memory location) 4) 탑 & 바텀 심볼(Top & bottom symbol)일 수 있다.

[0070] 임시 레지스터(temporary register)는 어셈블리어에서 REIL중간언어로 번역되는 과정에서 삽입되는 임시 레지스터이다. 메모리영역은 "*(reg1 + c1) + reg2+c2"와 같이 표현될 수 있으며, 여기서, "reg1"과 "reg2"는 임시 레지스터(temporary register)가 될 수 없다.

[0071] 이는 메모리 공간을 표현하기 위한 방법으로 "reg2+ c2"를 통해 레지스터(register) ± 오프셋(offset)을 표현하고, 인다이렉트(indirect)된 값을 표현하기 위한 포인터를 결합하여 사용한다. 마지막으로 탑(Top), 바텀(Bottom) 심볼 은 정적분석을 위해 추가된 요소일 수 있으며, 탑 심볼은 어느 하나의 도메인으로 표현 할 수 없는 경우에 나타낸다. 예를 들면 "register * memory location"같은 경우이다. 바텀 심볼은 아직 값이 할당되지 않아 어떤 값인지 모르는 경우이다.

[0072] 탑 심볼과 바텀 심볼을 제외한 다른 요소들은 아래 도 8에 도시된 바와 같이 수평적으로 구성될 수 있으며, 값들은 계층의 윗부분 또는 수평으로 바뀌는 방향만 가질 수 있고 역방향으로는 진행되지 않을 수 있다.

[0073] 예를 들어, 메모리 정교화 분석은 Binnavi에서 제공하는 MonoREIL 프레임워크를 사용하여 앱스트랙트 인터프리테이션(abstract interpretation)을 기반으로 3단계로 진행될 수 있으며, 먼저 테이블과 각종 특수 레지스터를 초기화시키는 단계를 거친다. 이후 명령어가 진행됨에 따라 레지스터나 메모리 영역이 가질 수 있는 값들이 달라지고 이를 테이블에 반영하기 위해 전달 함수(transfer function)를 적용한다. 또 다양한 제어 흐름(control flow)에 의한 가능성을 하나로 합치기 위한 미트 오퍼레이션(meet operation)을 유니온(union)연산을 사용하여 구현하고, 기타로 반복에 필요한 요소들을 적용한다. 마지막으로 완성된 테이블을 통해 메모리를 구분하고, 이를 적용하여 정교성을 높일 수 있다.

[0074] 예를 들어, MonoREIL 프레임워크의 경우에는 정적분석을 하기 위한 앱스트랙트 인터프리테이션(abstract interpretation)을 제공하는데, 초기화(initialize), 전달 함수(transfer function), 미트 오퍼레이션(meet operation)등 분석에 필요한 필수적인 부분만 분석하는 형태로 사용 가능할 수 있으나, 본 발명은 이에 한정되지 않는다.

[0075] 첫 번째 초기화 작업에서 rTable과 Env를 생성하는 과정은 함수시작 부분에서 실시하며, 함수가 시작할 때 스택(Stack)영역의 기준점을 "stack-0"로 삼는다. 이때 ARM 바이너리의 경우 SP레지스터, x86의 경우 ESP를 기준으로 할 수 있으며, 기존 함수에서 사용했던 레지스터 값이나 함수종료 후 호출하는 함수(caller)로 돌아가기 위한 명령어의 주소를 저장한 LR레지스터 등을 초기화할 수 있다.

[0076] 그리고 전달 함수(transfer function)를 적용하여 각 명령어에 대해 변화된 테이블을 적용하며, 전달 함수(transfer function)는 도 9에 도시된 바와 같은 공식을 적용할 수 있다.

[0077] 위와 같은 과정을 통해 각 인스트럭션(instruction)별 메모리 이미지(memory image)가 나오게 되면 그 인스트럭션(instruction)에서 사용하는 오퍼랜드(operand)가 사용하는 메모리가 어떤 영역인지 더 구체적으로 구별 할 수 있다.

[0078] 따라서 이전에 오버 테인트(over taint)로 인해 연관이 있다고 분석된 연결들을 끊어 더 효과적인 결과물을 사용자에게 알려 줄 수 있으며, 그 과정은 도 10에 도시된 바와 같다.

[0079] 도 10에 도시된 바와 같이, 정교화를 통해 왼쪽 2개 경로가 끊어진다면 실제로 위험하다고 판단했던 경로가 사

라지게 되며, 기존에 익스플로이터블 포인트(exploitable point)로 인해 E(exploitable)로 분류되었던 등급이 익스플로이터블 포인트(exploitable point)가 사라지면서 NE(not exploitable)로 재분류 된다. 바이너리 파일과 크래시 종류에 따라 위험도 분류 등급이 하향되는 방향으로 바뀌거나, 그렇지 않더라도 분석해야 되는 경로의 수가 줄어드는 효과가 있다.

[0080] 이제, 다중 함수 분석(Inter Procedure Analysis, IPA)에 대해 설명한다.

[0081] 프로그램 분석에서 분석 범위는 프로그램 분석에서 빠질 수 없는 중요한 요소 중 하나이다. 이러한 이유 때문에 동적 분석에서는 범위(coverage)를 중요한 지표로 여기고 있고, 크래시 발생 후에는 분석을 할 수 없다는 단점을 정적분석로 채우는 것도 이와 같은 이유이다.

[0082] 대표적인 정적 분석 도구인 "!exploitable[!exploitable]"은 분석 범위가 크래시가 발생한 내부 베이직 블록(Inner Basic Block)으로 그 범위를 벗어난다면 알 수 없음(Unknown)이라는 분석결과를 보고한다는 점도 분석 범위의 한계로 나오는 단점이다.

[0083] 따라서, 본 발명의 제2 실시예에 따른, 다중 함수 정적 분석 장치 및 방법은 상술한 함수의 익스플로잇(exploit) 탐지 기술을 확장하여 다중 함수(Inter-Procedure) 범위에서 익스플로이터블 포인트(exploitable point)를 탐지하는 방법을 포함할 수 있다.

[0084] 도 11은 전역변수와 함수 인자를 통해서 크래시에 영향을 받은 데이터가 전파되어 익스플로이터블 포인트(exploitable point)까지 연결되는 예시코드이다.

[0085] 사용자의 입력을 입력으로 받아 전역변수와 함수 인자로 넘길 데이터에 저장하는 과정에서 0으로 나누는 연산을 하게 된다면 크래시가 발생 할 수 있다. 이 데이터는 호출되는 함수(callee)에서 함수포인터를 바꾸어 원하는 코드로 분기 할 수 있는 익스플로이터블 포인트(exploitable point)가 된다.

[0086] 다중 함수 분석을 하기 전에는 분석 범위를 넘어선 곳에서 익스플로잇(exploit)이 발생할 가능성을 탐지 할 수 없었기 때문에 내부 단계(inner-procedure) 만 가능했다고 볼 수 있다. 다중 함수 분석은 크래시를 분석 할 수 있게 분석 범위를 넓히기 위한 목적으로, 크래시가 발생한 함수에서 다른 함수로 오염된 데이터가 전파 되어 취약점이 발생할 수 있는 가능성을 분석한다.

[0087] 다중 함수 분석(Inter Procedure Analysis)을 하는 방법에는 크게 두 가지 방법이 있을 수 있다.

[0088] 먼저 인라이닝(inlining)의 경우 호출하는 함수(caller)에서 다른 함수를 호출할 때, 호출 명령어 대신 호출되는 함수(callee)의 함수코드 부분을 그대로 삽입하여 코드를 수정하는 방식이며, 도 12에 도시된 바와 같이 메인 함수에서 "plus()"를 호출 했을 경우 인라이닝(Inlining)을 적용한 모습을 확인할 수 있다.

[0089] 이러한 방법은, 코드를 단순히 삽입하면 되기 때문에 구현이 쉽다는 장점이 이 있지만 호출하는 함수(caller)의 코드와 호출되는 함수(callee)의 코드가 합쳐지게 되므로, 코드의 크기가 상당히 커져서 생기는 단점이 있다.

[0090] 정적 분석에서는 각 명령어 마다 특징을 저장하는데, 이때 명령어의 길이가 길어질수록 각 명령어에서 수행해야 하는 데이터나 계산 횟수가 급격히 늘어나게 된다. 예를 들면 리칭-데프(Reaching-Definition) 분석에서는 각 명령어에서 도달 가능한 명령어들의 집합, 메모리 위치 분석(Memory Location Analysis)에서는 가질 수 있는 값들의 집합이 된다. 뿐만 아니라 호출되는 함수(callee)의 코드를 그대로 삽입하는 것이기 때문에 되부름(recursion)으로 구현된 코드의 경우 적용할 수 없다.

[0091] 다음으로 본 발명의 제2 실시예에 따른, 다중 함수 정적 분석 장치 및 방법에서 다중 함수 분석을 하기 위해서 도입한 구성 분석(Compositional Analysis)은 호출하는 함수(caller)와 호출되는 함수(callee) 사이에서 영향을 미치는 요인에 관하여 분석할 수 있으며, 그 분석 방법은 도 13에 도시된 바와 같다.

[0092] 다른 함수로 오염된 데이터가 전파되는 인수(argument), 전역 변수(global variable)를 비롯한 요인을 파악하고 분석한다. 이 분석의 핵심은 호출하는 함수(caller)로부터 호출되는 함수(callee)에 도달하게 된 데이터인 자유변수(free variable)를 입력 데이터(taint source)로 익스플로잇 분석(exploit analysis)을 하는 것이다.

[0093] 이는 인라이닝(Inlining)과는 다르게 호출되는 함수(callee)가 자기 자신을 다시 호출(recursion)하는 경우도 분석 가능하며, 한번 분석한 함수가 다른 함수에서 호출 되는 경우 이미 저장해 놓은 결과를 활용하여 (dynamic programing) 시간을 단축 할 수 있다.

[0094] 이때, 자유 변수는 호출하는 함수(caller)로부터 온 인수(argument), 호출되는 함수(callee)에서 정의하지 않고

사용하는 레지스터, 호출하는 함수(caller) 및 호출되는 함수(callee)에서 모두 활용되는 전역 변수(global variable)를 포함할 수 있다.

[0095] 이때, 호출되는 함수(callee)에서 정의하지 않고 사용하는 레지스터는 인수(argument)로 사용되었을 가능성이 매우 높으며, 예를 들면 x86에서 빠른 호출(fast call)로 함수를 호출 하는 경우와 ARM에서 r1 내지 r4의 레지스터는 인수(argument)로 사용하였을 가능성이 매우 크다.

[0096] 예컨대, 호출 규약에 따라 호출되는 함수(callee)에서 호출하는 함수(caller)가 사용하던 레지스터 데이터(register data)를 백업하기 위해 레지스터를 푸쉬하는 부분은 제외할 수도 있다.

[0097] 일 실시예에 따르면, 속도 향상 및 효율성을 위해 다중 함수 분석을 하지 않는 경우가 발생할 수도 있다.

[0098] 예컨대, 다중 함수 분석은 더 위험한 요소들이 있는지 추가적으로 파악하는 것으로 이미 E(exploitable)로 분류된 크래시의 경우 분석 대상이 아니다. 또한 지금 분석하고 있는 대상이 되는 대상 함수에 함수 호출 명령어가 없는 경우도 다른 함수로 전파되는 않는 것으로 간주할 수 있다.

[0099] 대상 함수가 전역 변수(global variable)를 사용하지 않는 경우에는 자유 변수(free variable)를 분석할 때 전역 변수 분석(global variable analysis)은 수행되지 않을 수도 있다.

[0100] 예를 들어, 호출되는 함수(callee)에서 위험도 분석은 크게 3부분으로 나누어 분석하는데, 먼저 호출되는 함수(callee) 내에서 익스플로이터블 포인트(exploitable point)가 있는 경우에는 제1 실시예에서와 동일한 방법으로 익스플로이터블 분석(exploitable analysis)의 기준에 따라 크래시를 분류할 수 있으며, 그 방법은 도 14에 도시된 바와 같을 수 있다.

[0101] 이때 다른 함수를 호출하게 된다면 이때 호출된 함수에 익스플로이터블 포인트(exploitable point)가 있는지 분석하기 위해 다중 함수 분석을 또 다시 실시한다. 마지막으로 함수가 끝날 때 오염된 데이터가 리턴되는지를 파악한다.

[0102] 분석은 입력 데이터(taint source)의 특성에 따라 전역 변수 분석(global variable analysis)과 함수 호출 분석(function call analysis)으로 나누어 분석할 수 있다.

[0103] 마지막으로 각각의 요소들을 분석 후 이를 취합한다. 입력 값의 영향을 받아 익스플로이터블 포인트(exploitable point)에 도달하는 경로가 많을 수 있고, 다른 함수까지 영향을 줄 수 있다.

[0104] 예를 들어 본 발명의 실시예에 따른 다중 함수 정적 분석 장치 및 방법은 사용자에게 크래시의 위험도를 분류하여 알려줄 때 보수적인 결과를 알려주기 때문에 상술한 익스플로이터블 분석(exploitable Analysis), 함수 호출 분석(function call analysis), 전역 변수 분석(global variable analysis)의 결과 중 가장 위험하다고 판단되는 등급을 최종 등급으로 보여줄 수도 있다.

[0105] 여기서, 다중 함수 분석에 해당하는 부분은 도 14의 붉은 선의 범위 안쪽을 의미할 수 있다.

[0106] 예컨대, 함수 호출 분석(function call analysis)은 함수 인자를 통하여 전파된 오염된 데이터가 취약점을 발생시킬 수 있는 가능성을 탐지하기 위한 분석이다. 앞서 크래시 지점(crash point)을 입력 데이터(taint source)로 놓고 분석한 것과 마찬가지로 호출되는 함수(callee)의 함수인자, 정의하지 않고 사용하는 레지스터와 같은 자유 변수(free variable)를 찾아내어 익스플로이터블 분석(exploitable analysis)을 수행하고 결과를 저장한다. 이 때, 현재 분석 중이던 대상 함수인 호출되는 함수(callee) 내부에 호출 명령어가 또 존재 할 수 있고, 마찬가지로 취약 할 수 있기 때문에 같은 방법으로 익스플로이터블 분석(exploitable analysis) 및 다중 함수 분석을 수행해야 할 수 있다.

[0107] 예컨대, 한번 호출이 되어 분석 대상이 된 함수는 따로 저장하여 같은 함수를 여러 번 호출 한 경우에도 반복해서 분석 할 필요가 없어 시간 비용을 줄일 수 있을 뿐 아니라, 회귀와 같이 무한히 계속 될 수 있는 상황도 피할 수 있다.

[0108] 익스플로이터블(exploitable) 가능성을 분석하는 것 외에도 오염된 데이터가 리턴값까지 전파된다면 호출하는 함수(caller)에 다시 영향을 미치는 경우도 생각해 볼 수 있다. 호출하는 함수(caller)에서 이 오염된 값을 저장한다면 또한 입력 데이터(taint source)로 간주되어야 하기 때문에 리턴값의 오염 유무를 판단하는 리턴값 분석(return value analysis)를 시행하고 결과를 저장한다. 리턴값은 x86일 경우는 eax, ARM인 경우는 R0 register를 의미할 수 있으나 본 발명은 이에 한정되지 않는다.

[0109] 이제 전역 변수 분석(Global Variable Analysis)을 설명한다.

[0110] 함수 인자로 전파되는 것 외에 전역 변수를 통해서도 크래시가 다른 함수의 데이터에 영향을 줄 수 있다. 오염된 데이터가 전역 변수(global variable)에 저장되면 크래시 지점(crash point)를 포함하고 있는 함수의 범위(scope)에서 벗어난 곳에 있는 데이터도 조작 가능하다.

[0111] 호출하는 함수(caller)에서 전역 변수(global variable)를 사용 한다면, 다른 함수에서 이 변수를 사용하여 취약점이 발생할 가능성이 있다. 같은 전역 변수를 호출되는 함수(callee)에서 사용하게 되는 경우, 이 변수를 통해 PC값을 바꿀 수 있는 익스플로이터블(exploitable) 가능성이 존재하기 때문에 이 전역 변수를 자유 변수로 간주할 수 있다.

[0112] 이때, 위험도 분석부(130)의 분석 결과는 E(exploitable), PE(probably exploitable), PNE(probably not explotable), NE(not exploitable)의 총 4단계를 의미할 수 있다.

[0113] 이제 도 15를 참조하여, 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치의 분석 결과와 종래 기술의 분석 결과를 비교한다.

[0114] 도 15에 도시된 바와 같이, 본 발명의 제2 실시예에 따른 다중 함수 정적 분석 장치는 종래의 기술에 비하여 더욱 많은 NE(not exploitable)를 분류할 수 있는 것을 확인할 수 있다.

[0115] 이상에서 본 발명에 따른 바람직한 실시예에 대해 설명하였으나, 다양한 형태로 변형이 가능하며, 본 기술분야에서 통상의 지식을 가진 자라면 본 발명의 특허청구범위를 벗어남이 없이 다양한 변형예 및 수정예를 실시할 수 있을 것으로 이해된다.

부호의 설명

[0116] 100: 다중 함수 정적 분석 장치

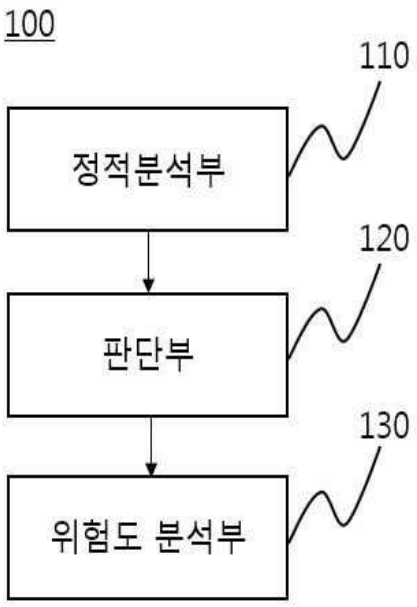
110: 정적분석부

120: 판단부

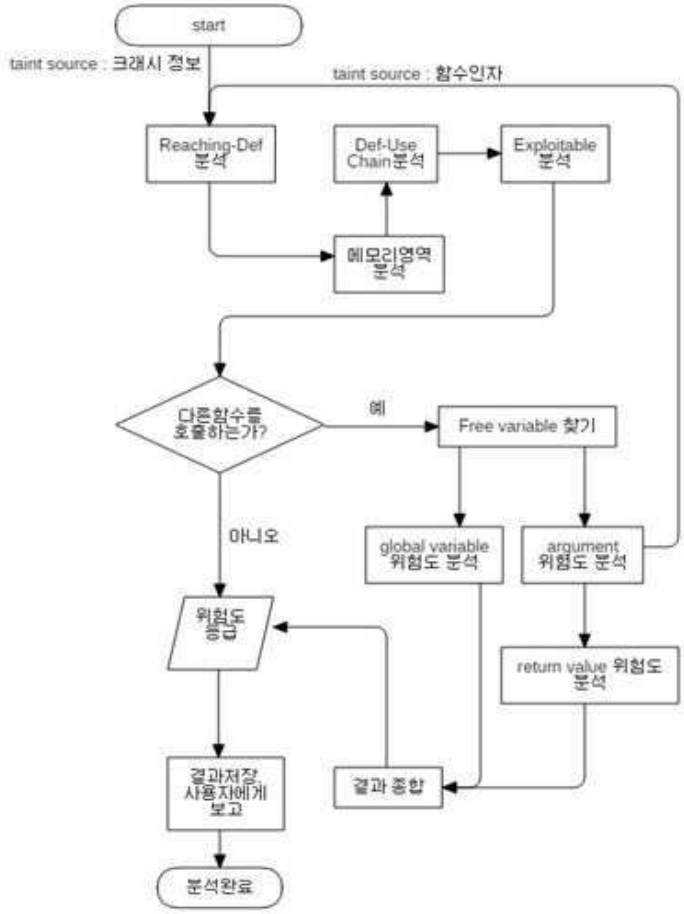
130: 위험도 분석부

도면

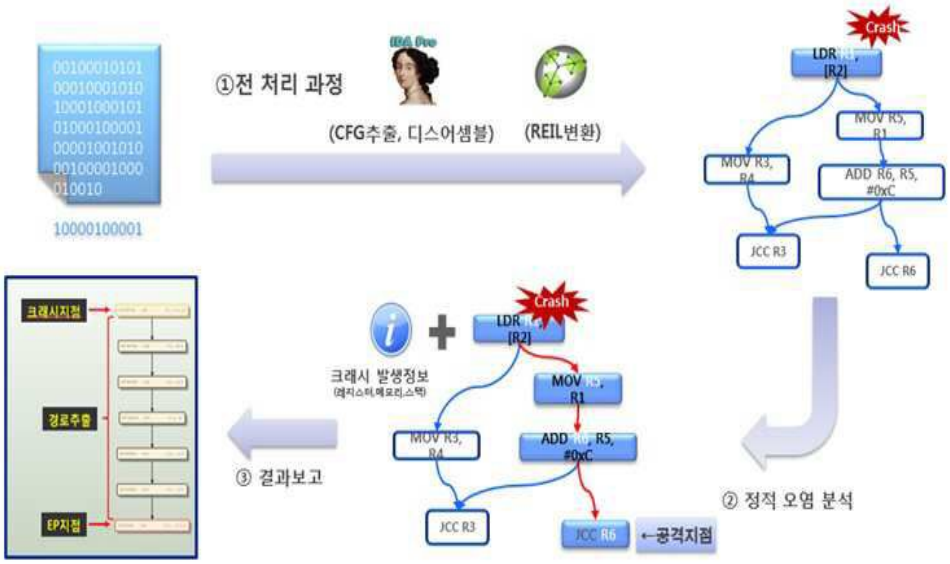
도면1



도면2


start
taint source : 크래시 정보
taint source : 함수인자
Reaching-Def 분석
Def-Use Chain분석
Exploitable 분석
메모리영역 분석
다른함수를 호출하는가?
예
Free variable 찾기
아니오
global variable 위험도 분석
argument 위험도 분석
위험도 등급
return value 위험도 분석
결과저장, 사용자에게 보고
결과 종합
분석완료

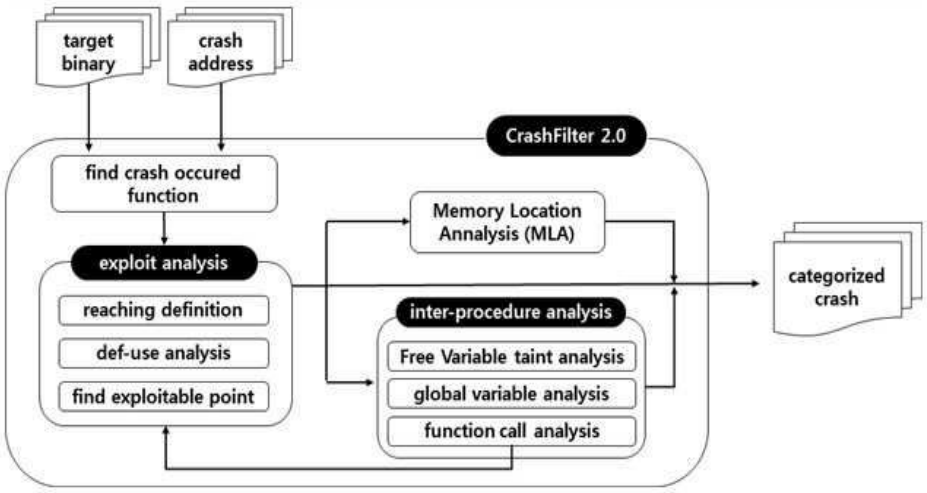
도면3


①전 처리 과정
(CFG추출, 디스어셈블)
(REIL변환)
10000100001
Crash
LDR R3, [R2]
MOV R5, R1
MOV R3, R4
ADD R6, R5, #0xC
JCC R3
JCC R6
② 정적 오염 분석
크래시 발생정보
←공격지점
③ 결과보고
크래시지점
경로추출
EP지점

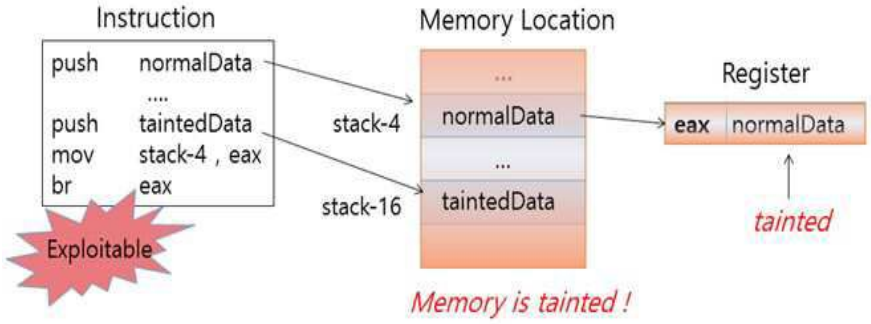
도면4

구분	조건	위험도
분기명령어	목적지주소 제어가능	E
데이터저장명령어	데이터와 목적지주소 모두제어가능	E
	데이터만 제어가능	PE
	목적지주소만 제어가능	PE
위조건에 해당하지 않는 경우		NE

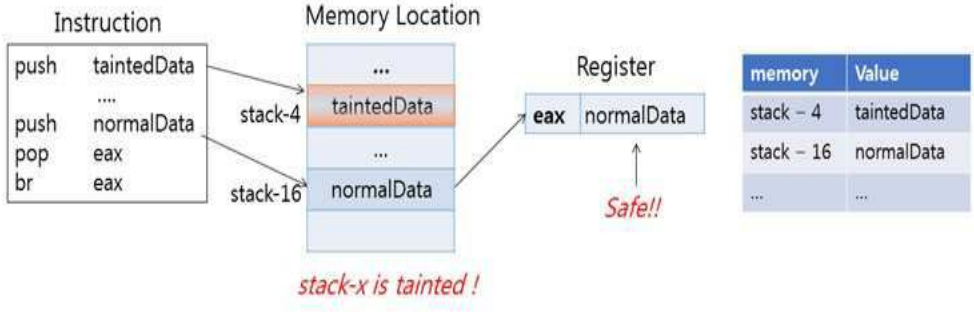
도면5



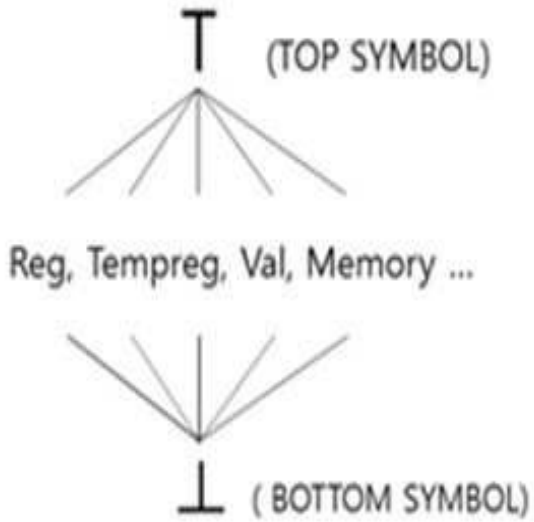
도면6



도면7



도면8



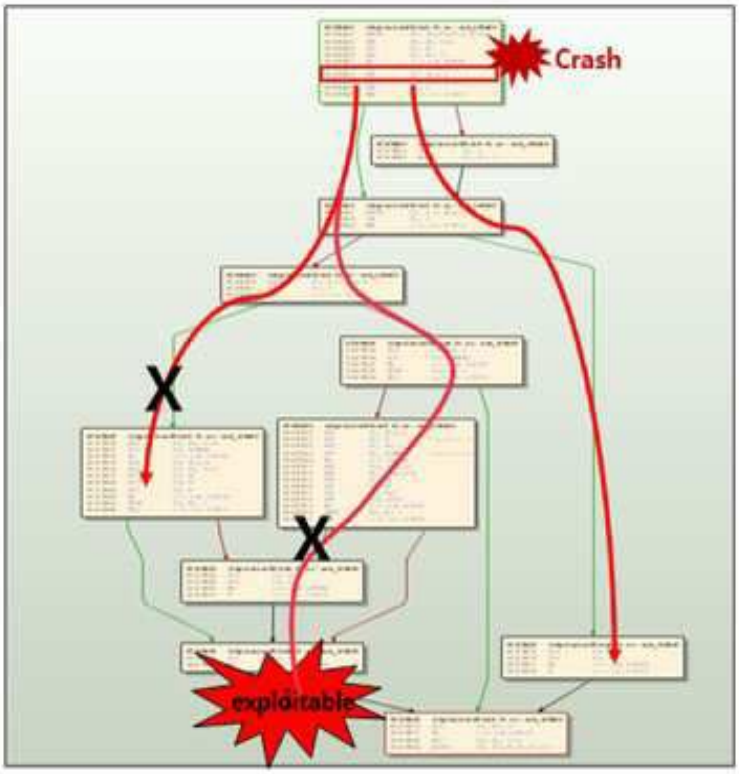
도면9

T [STM x, y] *rtable*

1. *rtable* [{x} / {y' ¦ y' ∈ *rtable*(y)}] if x ∈ Val

2. *rtable* [{v ¦ ∀ v ∈ *rtable*(x)} / {y' ¦ ∀ y' ∈ *rtable*(y)}] if x ∈ reg

도면10

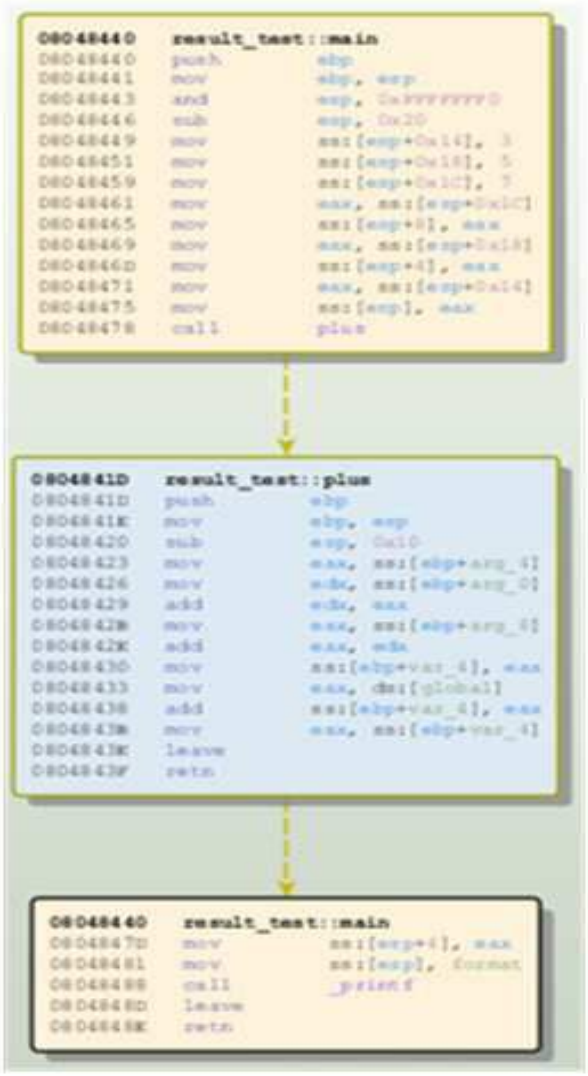


도면11

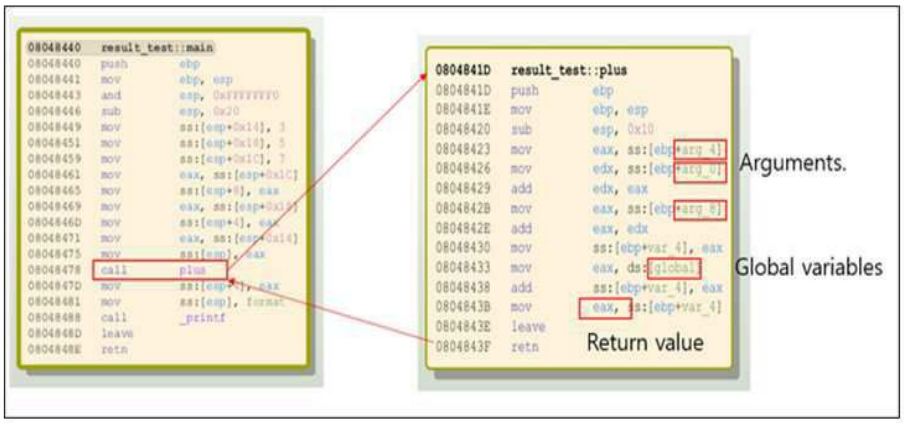
```
#include <stdio.h>

int globalVariable;

void one(int i)   { printf("one %d₩n", i);   }
void two(int i)   { printf("two %d₩n",i );   }
void three(int i) { printf("three %d₩n",i );   }

void exploitableFunction()
{
  void (*addr[3])(int);
  addr[0] = one;   addr[1] = two;  addr[2] = three;
  (*addr[globalVariable-1])(globalVariable+1);
}

int main(void) {
  puts("main function");
  int baseValue = 120;
  int inputValue;
  scanf("%x", &inputValue);
  globalVariable = baseValue/inputValue;

  exploitableFunctionFromArgument();
```

```
#include <stdio.h>

void one(int i)   { printf("one %d₩n", i);   }
void two(int i)   { printf("two %d₩n", i);   }
void three(int i) { printf("three %d₩n", i);   }

void exploitableFunctionFromArgument( int argument)
{
  void (*addr[3])(int);
  addr[0] = one;   addr[1] = two;  addr[2] = three;
  (*addr[argument-1])(argument+1);
}

int main(void) {
  puts("main function");
  int baseValue = 120;
  int inputValue;
  scanf("%x", &inputValue);
  int argument = baseValue/inputValue;

  exploitableFunction(argument);
}
```

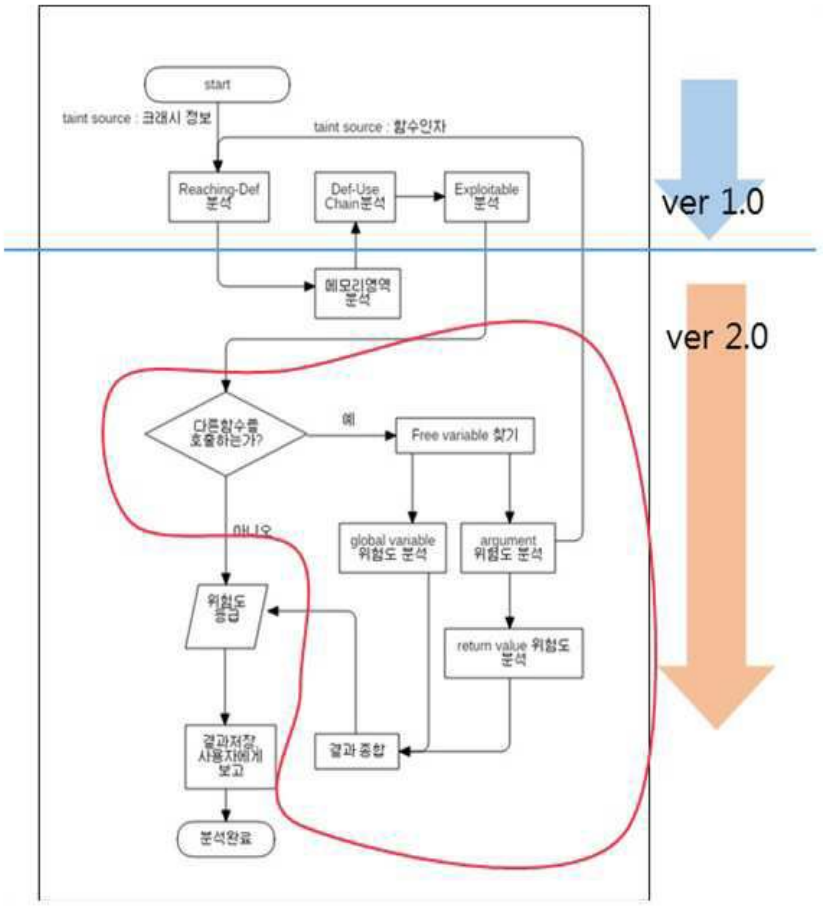
도면12


result_test::main
result_test::plus
result_test::main

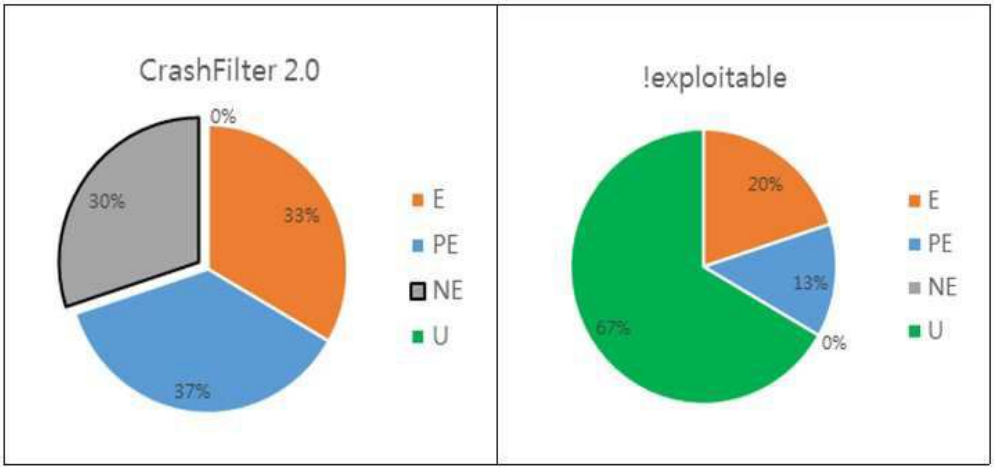
도면13


result_test::main
result_test::plus
Arguments.
Global variables
Return value

도면14


start
taint source : 크래시 정보
taint source : 함수인자
Reaching-Def 분석
Def-Use Chain분석
Exploitable 분석
ver 1.0
메모리영역 분석
ver 2.0
다른함수를 호출하는가?
예
Free variable 찾기
global variable 위험도 분석
argument 위험도 분석
return value 위험도 분석
위험도 등급
결과저장 사용자에게 보고
결과 종합
분석완료

도면15


CrashFilter 2.0
0%
33%
30%
37%
E
PE
NE
U
!exploitable
20%
13%
67%
0%
E
PE
NE
U