(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0129928
(43) 공개일자 2024년08월28일

(51) 국제특허분류(Int. Cl.)
G06F 21/56 (2013.01) *G06F 11/36* (2006.01)
(52) CPC특허분류
G06F 21/563 (2013.01)
G06F 11/3624 (2013.01)
(21) 출원번호 10-2023-0023155
(22) 출원일자 2023년02월21일
심사청구일자 2023년02월21일

(71) 출원인
울산과학기술원
울산광역시 울주군 언양읍 유니스트길 50
(72) 발명자
전유석
울산광역시 울주군 언양읍 유니스트길 50
(74) 대리인
리앤목특허법인

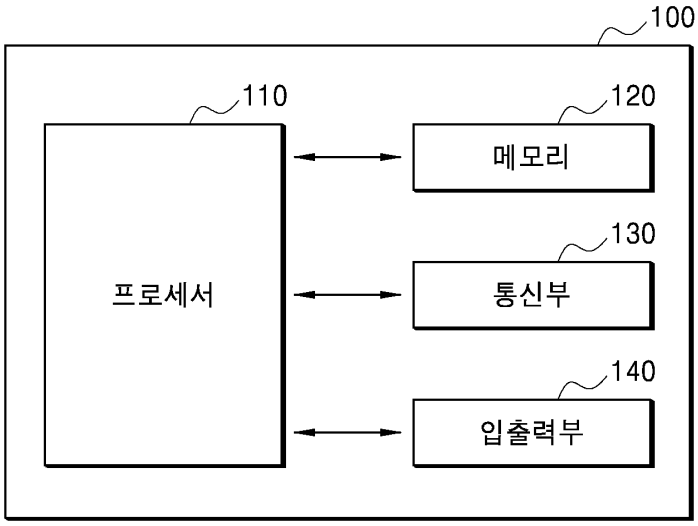
전체 청구항 수 : 총 5 항

(54) 발명의 명칭 **러스트 언어를 위한 취약점 탐지 장치, 취약점 탐지 방법 및 컴퓨터 프로그램**

(57) 요 약

본 개시의 실시예에 따르면, 취약점 탐지 장치가, 로스트 언어로 구현된 대상 프로그램을 판독하고 상기 대상 프로그램을 고레벨 중간 표현의 코드로 변환하면서 상기 대상 프로그램에 포함된 안정하지 않은 코드를 구분하고, 상기 안정하지 않은 코드에 대한 정보를 테이블로 기록하는 단계; 상기 취약점 탐지 장치가, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 단계; 및 상기 취약점 탐지 장치가, 저레벨 중간 표현의 코드 중에서, 상기 테이블을 이용하여 안정하지 않은 영역의 코드에 포함된 취약점을 검출하는 단계를 포함하는, 러스트 언어를 위한 취약점 탐지 방법을 개시한다.

대 표 도 - 도1



이 발명을 지원한 국가연구개발사업

과제고유번호	1711170475
과제번호	2022-0-00745-001
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보보호핵심원천기술개발(정진기금)
연구과제명	랜섬웨어 공격 근원지 식별 및 분석 기술 개발
기 여 율	1/1
과제수행기관명	(주)에스투더블유
연구기간	2022.04.01 ~ 2022.12.31

명 세 서

청구범위

청구항 1

취약점 탐지 장치가, 로스트 언어로 구현된 대상 프로그램을 판독하고 상기 대상 프로그램을 고레벨 중간 표현의 코드로 변환하면서 상기 대상 프로그램에 포함된 안정하지 않은 코드를 구분하고, 상기 안정하지 않은 코드에 대한 정보를 테이블로 기록하는 단계;

상기 취약점 탐지 장치가, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 단계; 및

상기 취약점 탐지 장치가, 저레벨 중간 표현의 코드 중에서, 상기 테이블을 이용하여 안정하지 않은 영역의 코드에 포함된 취약점을 검출하는 단계를 포함하는, 러스트 언어를 위한 취약점 탐지 방법.

청구항 2

제1항에 있어서,

상기 테이블로 기록하는 단계는

상기 고레벨 중간 표현의 코드를 가지고 데이터 타입의 안정성을 판단하여, 타입 안전을 위한 코드를 검출하는 단계를 더 포함하는, 러스트 언어를 위한 취약점 탐지 방법.

청구항 3

프로세서와 비일시적 메모리를 포함하고,

상기 프로세서가, 상기 메모리에 저장된 명령어들을 판독하여

로스트 언어로 구현된 대상 프로그램을 판독하고 상기 대상 프로그램을 고레벨 중간 표현의 코드로 변환하면서 상기 대상 프로그램에 포함된 안정하지 않은 코드를 구분하고, 상기 안정하지 않은 코드에 대한 정보를 테이블로 기록하고,

고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하며,

저레벨 중간 표현의 코드 중에서, 상기 테이블을 이용하여 안정하지 않은 영역의 코드에 포함된 취약점을 검출하는, 러스트 언어를 위한 취약점 탐지 장치.

청구항 4

제3항에 있어서,

상기 프로세서가,

상기 고레벨 중간 표현의 코드를 가지고 데이터 타입의 안정성을 판단하여, 타입 안전을 위한 코드를 검출하는, 러스트 언어를 위한 취약점 탐지 장치.

청구항 5

컴퓨터를 이용하여 제1항 또는 제2항의 방법을 실행시키기 위하여 컴퓨터 판독 가능한 저장 매체에 저장된 컴퓨터 프로그램.

발명의 설명

기 술 분 야

[0001] 본 개시의 실시예들은, 러스트 언어를 위한 취약점 탐지 장치, 취약점 탐지 방법 및 컴퓨터 프로그램에 관한 것으로, 보다 구체적으로는 러스트 언어로 구현된 프로그램에서 안전한 영역(safe)과 안전하지 않은 영역(unsaf

e)을 구분하고 안정하지 않은 영역에 대한 정보를 이용하여 프로그램의 취약점을 탐지할 수 있는 취약점 탐지 장치, 취약점 탐지 방법 및 컴퓨터 프로그램에 관한 것이다.

배 경 기 술

[0002] 소스 코드에 존재하는 잠재적인 오류(bug)나 취약점을 발견하기 위해 정적분석기가 널리 사용된다. 정적분석기는 각 기능별 체커의 실행에 의해 각 체커별로 미리 정의된 오류를 검출하고, 오류가 검출된 것으로 판단하면 이에 대한 경보(alarm) 메시지를 발생한다.

[0003] 그러나, 이러한 정적분석기의 분석 과정에서 오류 여부의 판단이 항상 정확하게 이뤄지지 못하는 것이 현실이며, 이로 인해 분석 대상 소스 코드에 오류가 존재하지 않음에도 불구하고 오류가 존재하는 것으로 잘못 판단하여 오경보(false alarm)를 발생하는 문제가 존재하였다.

[0004] 일반적으로 소스 코드의 오류 검출 경보가 발생되면, 오류를 검토 및 처리하기 위해 다양한 리소스가 투입되어야 하므로, 이러한 오경보는 프로그램 개발 및 검수 과정에서 다양한 리소스의 낭비 요인이 되었다.

[0005] [선행문헌]

[0006] 대한민국 등록특허공보 제10-1694778호

발명의 내용

해결하려는 과제

[0007] 본 발명은 상기와 같은 문제점을 포함하여 여러 문제점들을 해결하기 위한 것으로서, 러스트 언어(RUST)로 구현된 프로그램에 대해서는, 안전한 영역(safe)과 안전하지 않은 영역(unsafe)을 구분하고 안정하지 않은 영역에 대한 정보를 이용하여 프로그램의 취약점을 탐지할 수 있다.

과제의 해결 수단

[0008] 본 개시의 실시예들에 따른 방법은, 취약점 탐지 장치가, 로스트 언어로 구현된 대상 프로그램을 판독하고 상기 대상 프로그램을 고레벨 중간 표현의 코드로 변환하면서 상기 대상 프로그램에 포함된 안정하지 않은 코드를 구분하고, 상기 안정하지 않은 코드에 대한 정보를 테이블로 기록하는 단계; 상기 취약점 탐지 장치가, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 단계; 및 상기 취약점 탐지 장치가, 저레벨 중간 표현의 코드 중에서, 상기 테이블을 이용하여 안정하지 않은 영역의 코드에 포함된 취약점을 검출하는 단계를 포함할 수 있다.

[0009] 상기 테이블로 기록하는 단계는 상기 고레벨 중간 표현의 코드를 가지고 데이터 타입의 안정성을 판단하여, 타입 안전을 위한 코드를 검출하는 단계를 더 포함할 수 있다.

[0010] 본 발명의 실시예에 따른 컴퓨터 프로그램은 컴퓨터를 이용하여 본 발명의 실시예에 따른 방법 중 어느 하나의 방법을 실행시키기 위하여 매체에 저장될 수 있다.

[0011] 이 외에도, 본 발명을 구현하기 위한 다른 방법, 다른 시스템 및 상기 방법을 실행하기 위한 컴퓨터 프로그램을 기록하는 컴퓨터 판독 가능한 기록 매체가 더 제공된다.

[0012] 전술한 것 외의 다른 측면, 특징, 이점이 이하의 도면, 특허청구범위 및 발명의 상세한 설명으로부터 명확해 질 것이다.

발명의 효과

[0013] 상기한 바와 같이 이루어진 본 발명의 실시예에 따르면, 러스트 언어(RUST)로 구현된 프로그램에 대해서는, 안전한 영역(safe)과 안전하지 않은 영역(unsafe)을 구분하고 안정하지 않은 영역에 대한 정보를 이용하여 프로그램의 취약점을 탐지할 수 있는 러스트 언어를 위한 취약점 탐지 장치, 취약점 탐지 방법 및 컴퓨터 프로그램을 구현할 수 있다.

도면의 간단한 설명

[0014] 도 1은, 본 개시의 실시예들에 따른 러스트 언어를 위한 취약점 탐지 장치의 블록도이다.

도 2는 메모리(120)의 블록도이다.

도 3은, 본 개시의 실시예들에 따른 취약점 탐지 방법의 흐름도이다.

도 4는, 러스트 언어로 구현된 프로그램의 취약점을 검출하는 과정을 설명하는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0015] 이하 첨부된 도면들에 도시된 본 발명에 관한 실시예를 참조하여 본 발명의 구성 및 작용을 상세히 설명한다.

[0016] 본 발명은 다양한 변환을 가할 수 있고 여러 가지 실시예를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 상세한 설명에 상세하게 설명하고자 한다. 본 발명의 효과 및 특징, 그리고 그것들을 달성하는 방법은 도면과 함께 상세하게 후술되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 다양한 형태로 구현될 수 있다.

[0017] 이하, 첨부된 도면을 참조하여 본 발명의 실시예들을 상세히 설명하기로 하며, 도면을 참조하여 설명할 때 동일하거나 대응하는 구성 요소는 동일한 도면부호를 부여하고 이에 대한 중복되는 설명은 생략하기로 한다.

[0018] 본 명세서에서 “학습”, “러닝” 등의 용어는 인간의 교육 활동과 같은 정신적 작용을 지칭하도록 의도된 것이 아닌 절차에 따른 컴퓨팅(computing)을 통하여 기계 학습(machine learning)을 수행함을 일컫는 용어로 해석한다.

[0019] 이하의 실시예에서, 제1, 제2 등의 용어는 한정적인 의미가 아니라 하나의 구성 요소를 다른 구성 요소와 구별하는 목적으로 사용되었다.

[0020] 이하의 실시예에서, 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다.

[0021] 이하의 실시예에서, 포함하다 또는 가지다 등의 용어는 명세서상에 기재된 특징, 또는 구성요소가 존재함을 의미하는 것이고, 하나 이상의 다른 특징들 또는 구성요소가 부가될 가능성을 미리 배제하는 것은 아니다.

[0022] 도면에서는 설명의 편의를 위하여 구성 요소들이 그 크기가 과장 또는 축소될 수 있다. 예컨대, 도면에서 나타난 각 구성의 크기 및 두께는 설명의 편의를 위해 임의로 나타내었으므로, 본 발명이 반드시 도시된 바에 한정되지 않는다.

[0023] 어떤 실시예가 달리 구현 가능한 경우에 특정한 공정 순서는 설명되는 순서와 다르게 수행될 수도 있다. 예를 들어, 연속하여 설명되는 두 공정이 실질적으로 동시에 수행될 수도 있고, 설명되는 순서와 반대의 순서로 진행될 수 있다.

[0024] 본 명세서에서, 러스트 언어는, 현대적인 시스템 프로그래밍 언어로서, C/C++과 동등한 수준의 속도를 달성하면서 안전성, 동시성을 가지는 언어를 말한다. 안전하지 않은 코드를 사용하여 안전한 코드로 추상화할 수 있는 도구가 제공될 수 있다. 취약점 탐지 장치(100)는, 러스트 언어로 구현된 프로그램이 컴파일 하는 동안에, 객체와 변수의 생성과 소멸 시기를 컴파일 타임에 탐지할 수 있다. 컴파일 타임에 탐지한다는 것은, 컴파일을 수행하는 동안에 객체의 생성 시기와 객체의 소멸 시기, 변수의 생성 시기, 변수의 소멸 시기 등을 검출함을 의미한다. 컴파일 단계와 별개의 단계에서 수행하는 것이 아닌 컴파일을 수행하면서 동시에 객체 또는 변수에 대한 정보인 생성 시간값와 소멸 시간값 등을 검출할 수 있다.

[0025] 프로그래밍 언어를 이용하여 프로그램의 개발을 완료하게 되면, 프로그램을 몇 개의 단계를 거쳐 실행 파일(바이너리 코드)을 생성할 수 있다. 프로그램을 실행 파일로 변환한다는 것은, 프로그램을 프로세서에 의해 수행되는 코드로 변환하는 것을 의미하며, 프로그램을 전처리하고, 전처리한 프로그램을 컴파일하며, 컴파일된 프로그램을 정해진 형식인 어셈블리 어로 변환하고, 변환된 프로그램의 오브젝트 파일을 연결하는 과정을 포함할 수 있다. 프로그램의 실행 파일로 변환되는 과정에서는, 형식적 에러를 포함하는 코드는 컴파일 되지 않는다.

[0026] 본 개시의 실시예들에 따른 취약점 탐지 장치(100)는, 러스트 언어로 구현된 프로그램에 대해서 안전한 영역과 안전하지 않은 영역으로 구분하고, 안전한 영역과 안전하지 않은 영역에 대해서 각각의 오버헤드로 취약점을 검출할 수 있다.

[0027] 취약점 탐지 장치(100)는, 프로그램에 포함된 객체 사이의 할당 규칙 중에서, 기 정해진 규칙을 위반한 코드를 탐지할 수 있다. 취약점 탐지 장치(100)는, 프로그램에 포함된 비효율적인 코드를 제거하고 효율적인 코드로 변경함으로써, 프로그램을 최적화할 수 있다.

[0028] 할당 규칙은, 대여 확인 규칙을 포함할 수 있다. 대여 확인 규칙은, 선언된 변수가 런타임의 가비지에 할당되어 사용되는데 선언된 변수가 사용이 완료된 이후에도 할당된 영역을 반환하는지 여부로 판단되는 것이다. 프로그램에서 제1 시점에 선언된 변수는 프로그램의 실행으로 제2 시점에 사용이 완료될 수 있다. 이때, 제2 시점 이후의 임의의 시점에 변수에 할당된 메모리 공간(영역) 등이 반환되지 않는 경우, 대여 확인 규칙의 위반에 해당한다고 판단할 수 있다.

[0029] 프로그램의 최적화는, 프로그램에 포함된 불필요한 코드를 제거하면서 프로그램을 변환하는 것을 말할 수 있으나 이에 한정되지 않고 다양한 방법으로 최적화하는 것을 포함할 수 있다. 여기서, 불필요한 코드는 프로그램에 포함되어 있어 실행되는 부분이지만, 해당 코드를 통해서 산출된 결과값이 추가적으로 다른 코드에 전혀 사용되지 않는 부분, 사용될 수 없는 부분을 말한다(섹션, 포션). 예시로는, 프로그램에 포함되어 있어 컴파일의 대상에 해당하지만 실제로는 실행되지 않는 코드, 프로그램의 실행되는 동안에 실행되어 실행되는 동안에 리소스(컴퓨터의 프로세싱 능력)를 사용하게 되지만 실행된 이후에는 추가적인 연산에 사용되지 않는 선언문, 프로그램 내에서 실행되어 실행되지만 출력 값을 반환하지 않거나 반환된 출력값을 다른 연산에서 사용하지 않는 함수의 호출 등이 있을 수 있다.

[0030] 취약점 탐지 장치(100)는, 프로그램을 컴파일하는 동안에 획득, 분석된 데이터를 이용하여 프로그램에 대한 안전한 영역에 대한 데이터와 안전하지 않은 데이터를 기록할 수 있다. 안전한 영역에 대한 데이터와 안전하지 않은 데이터는 테이블로 기록될 수 있다. 예를 들어, 안전하지 않은 영역이라는 것은, 선언된 변수가 사용된 이후의 시점에 해당 변수에 할당된 메모리가 반환되지 않는 코드(Use after free)를 포함하는 영역을 포함할 수 있다.

[0031] 취약점 탐지 장치(100)는, 생성된 테이블을 판독하여 프로그램에 포함된 안전하지 않은 영역에 대한 정보를 획득하고 이러한 정보에 기초하여 안전하지 않은 영역에 대해서만 취약점을 검출하도록 구현될 수 있다.

[0032] 취약점 탐지 장치(100)는, 프로그램을, 파일, 함수, 블록 등의 단위로 분리하고 안전하지 않은 영역에 해당하는 블록들에 대한 정보(저장된 영역에 대한 주소, 오프셋 등)를 기록한 테이블을 생성하고 테이블을 판독하여 안전한 영역과 안전하지 않은 영역을 식별할 수 있다.

[0033] 도 1은, 본 개시의 실시예들에 따른 러스트 언어를 위한 취약점 탐지 장치의 블록도이다.

[0034] 취약점 탐지 장치(100)는 러스트 언어로 구현된 프로그램을 실행 파일로 변환하는 과정에서, 러스트 언어에 포함된 취약점을 탐지함에 있어서, 러스트 언어에 포함된 안전하지 않은 영역을 별도로 분리하고 분리된 안전하지 않은 영역에 대해서만 취약점을 탐지할 수 있다. 이를 통해, 취약점을 탐지하는 시간을 효과적으로 단축할 수 있다. 또한, 컴파일 타임 동안에 획득된 정보를 이용하여 안전하지 않은 코드에 대한 정보를 획득함으로써, 프로그램의 코드에 포함된 취약점 탐지의 시간을 효과적으로 단축할 수 있다. 이때, 프로그램은 러스트(LUST) 언어로 구현되는 것에 한정되지 않고 고차원 언어로 구현된 모든 것을 포함할 수 있다.

[0035] 취약점 탐지 장치(100)는, 러스트 언어로 구현된 프로그램(소스코드)을 고레벨 중간 표현의 코드로 변환하는 제1 단계, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 제2 단계, 저레벨 중간 표현의 코드를 머신 코드로 변환하는 제3 단계를 수행하여 실행 파일(바이너리 코드)을 생성할 수 있다.

[0036] 제1 단계에서는, 프로그램의 소스 코드를 고레벨 중간 표현의 코드로 변환하고 변환하는 과정에서 파일, 함수, 블록의 단위로 분할하고 분할된 각 영역(portion)를 안전하지 않은 코드와 안전한 코드 중 하나로 구분할 수 있다. 취약점 탐지 장치(100)는 제1 단계에서 획득된 안전하지 않은 코드에 대한 정보(저장 영역에 대한 주소, 오프셋 값 등)를 별도의 테이블로 기록할 수 있다.

[0037] 추가적으로 취약점 탐지 장치(100)는 변환된 고레벨 중간 표현의 코드를 이용하여 데이터 타입의 안전성을 판단할 수 있다. 취약점 탐지 장치(100)는 코드에서의 타입 안전을 위한 코드를 검출할 수 있다. 예를 들어, 취약점 탐지 장치(100)는 메모리 할당 코드에서, 제1 포인터에 제2 포인터가 할당되는 경우, 제1 포인터의 타입과 제2 포인터의 타입이 일치하지 않는 코드 등을 타입 안전성이 없는 코드로 검출할 수 있다.

[0038] 제2 단계에서는, 취약점 탐지 장치(100)는, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환할 수 있다. 고레벨 중간 표현의 코드는 프로세싱의 시간이 단축되는 저레벨 중간 표현의 코드로 변환될 수 있다.

[0039] 제3 단계에서는, 취약점 탐지 장치(100)는, 키워드의 맞춤법이 틀리거나, 필요한 문장 부호를 빠뜨리거나, if, end if 등의 잘못된 사용, 선언되지 않은 변수의 사용, 무한 루프에 빠지거나, Null pointer 에러, 존재하지 않는 메모리 위치에 접근하는 등의 유효하지 않은 코드를 검출할 수 있다. 취약점 탐지 장치(100)는, 제1 단계에

서 기록한 테이블(UNSI, 도 4 참조)을 이용하여 안전한 코드에 대해서는 취약점을 탐지하는 것을 스킵하고, 안전하지 않은 코드에 대해서는 상기에서 언급한 취약점을 탐지할 수 있다.

[0040] 취약점 탐지 장치(100)는, 상기의 과정을 통해 탐지한 취약점에 대한 정보를 생성할 수 있다. 취약점 탐지 장치(100)는, 취약점에 대한 데이터를 출력하는 출력 데이터를 생성할 수 있다. 여기서, 취약점이라는 것은, 프로그램이 코딩된 의도와 다른 의도로 실행될 수 있는 부분을 말할 수 있다. 취약점은 컴파일 과정에는 에러로 검출되지 않아 저레벨 코드로 변환되는 부분이지만 실행 시에는 버그 등의 형태로 나타나는 것을 말할 수 있다.

[0041] 도 1에 도시된 바와 같이, 취약점 탐지 장치(100)는, 프로세서(110), 메모리(120), 통신부(130), 및 입출력부(140)를 포함할 수 있다.

[0042] 프로세서(110)는, 러스트 언어로 구현된 프로그램을 실행 파일로 변환하는 과정에서, 러스트 언어에 포함된 취약점을 탐지함에 있어서, 러스트 언어에 포함된 안전하지 않은 영역을 별도로 분리하고 분리된 안전하지 않은 영역에 대해서만 취약점을 탐지할 수 있다.

[0043] 프로세서(110)는, 러스트 언어로 구현된 프로그램(소스코드)을 고레벨 중간 표현의 코드로 변환하는 제1 단계, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 제2 단계, 저레벨 중간 표현의 코드를 머신 코드로 변환하는 제3 단계를 수행하여 실행 파일(바이너리 코드)을 생성할 수 있다.

[0044] 제1 단계에서는, 소스 코드를 고레벨 중간 표현의 코드로 변환하고 변환하는 과정에서 파일, 함수, 블록의 단위로 분할하고 분할된 각 영역(portion)에 대해서 안전하지 않은 코드와 안전한 코드를 구분할 수 있다. 프로세서(110)는, 제1 단계에서 획득된 안전하지 않은 코드에 대한 정보를 별도의 테이블로 기록할 수 있다. 프로세서(110)는, 고레벨 중간 표현의 코드를 가지고 데이터 타입의 안전성을 판단할 수 있다. 예를 들어, 프로세서(110)는, 메모리 할당 코드에서, 제1 포인터에 제2 포인터가 할당되는 경우, 제1 포인터의 타입과 제2 포인터의 타입이 일치하지 않는 코드 등을 검출할 수 있다.

[0045] 제2 단계에서는, 프로세서(110)는, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환할 수 있다.

[0046] 제3 단계에서는, 프로세서(110)는, 키워드의 맞춤법이 틀리거나, 필요한 문장 부호를 빠뜨리거나, if, end if 등의 잘못된 사용, 선언되지 않은 변수의 사용, 무한 루프에 빠지거나, Null pointer 에러, 존재하지 않는 메모리 위치에 접근하는 등의 유효하지 않은 코드 등을 검출할 수 있다. 프로세서(110)는, 제1 단계에서 기록한 테이블을 이용하여 안전한 코드에 대해서는 취약점을 탐지하는 것을 스킵하고, 안전하지 않은 코드에 대해서는 상기에서 언급한 취약점을 탐지할 수 있다.

[0047] 프로세서(110)는, 상기의 과정을 통해 탐지한 취약점에 대한 정보를 생성할 수 있다. 프로세서(110)는, 취약점에 대한 데이터를 출력하는 출력 데이터를 생성할 수 있다.

[0048] 이를 통해, 취약점을 탐지하는 시간을 효과적으로 단축할 수 있다. 또한, 컴파일 타임 동안에 획득된 정보를 이용하여 안전하지 않은 코드에 대한 정보를 획득함으로써, 프로그램의 코드에 포함된 취약점 탐지의 시간을 효과적으로 단축할 수 있다. 이때, 프로그램은 러스트(LUST) 언어로 구현되는 것에 한정되지 않고 고차원 언어로 구현된 모든 것을 포함할 수 있다.

[0049] 본 개시의 일 실시예에 따르면, 메모리(120)는 프로세서(110)의 처리 및 제어를 위한 프로그램을 저장할 수 있고, 취약점 탐지 장치(100)로 입력되거나 취약점 탐지 장치(100)로부터 출력되는 데이터를 저장할 수도 있다. 본 개시의 일 실시예에 따르면, 메모리(120)는 러스트 언어로 구현된 프로그램의 취약점 검출을 판단하는데 필요한 데이터, 구성요소를 저장할 수도 있고, 취약점 검출에 대한 정보를 저장할 수도 있다. 메모리(120)는 취약점의 탐지가 필요한 프로그램을 저장할 수 있고, 프로그램을 정적 분석하는 구성, 프로그램의 실행을 모니터링하는 구성 등을 저장할 수 있다. 메모리(120)는 상기의 정보들을 저장한 데이터베이스를 포함할 수 있다.

[0050] 본 개시의 일 실시예에 따르면, 메모리(120)는 플래시 메모리 타입(flash memory type), 하드디스크 타입(hard disk type), 멀티미디어 카드 마이크로 타입(multimedia card micro type), 카드 타입의 메모리(예를 들어 SD 또는 XD 메모리 등), 램(RAM, Random Access Memory) SRAM(Static Random Access Memory), 롬(ROM, Read-Only Memory), EEPROM(Electrically Erasable Programmable Read-Only Memory), PROM(Programmable Read-Only Memory), 자기 메모리, 자기 디스크, 광디스크 중 적어도 하나의 타입의 저장매체를 포함할 수 있다. 또한 본 개시의 일 실시예에 따르면, 메모리(120)에 저장된 프로그램들은 그 기능에 따라 복수 개의 모듈들로 분류할 수 있다.

[0051] 본 개시의 일 실시예에 따르면, 통신부(130)는 프로세서(110)의 외부 장치와 통신을 수행할 수 있다. 예를

들어, 통신부(130)는 프로세서(110)의 제어에 의해 보안이 필요한 프로그램이 저장된 전자 장치와 통신을 수행할 수 있다. 또한, 통신부(130)는 외부 인터페이스와의 통신을 통해 보안이 필요한 프로그램이 저장된 전자 장치에 접근하여 프로그램을 분석하고 프로그램에 대한 보안 정책을 설정하도록 통신할 수 있다.

[0052] 입출력부(140)는 입출력 장치와의 인터페이스를 위한 수단일 수 있다. 예를 들어, 입력 장치는 키보드 또는 마우스 등의 장치를, 그리고 출력 장치는 어플리케이션의 통신 세션을 표시하기 위한 디스플레이와 같은 장치를 포함할 수 있다. 다른 예로 입출력부(140)는 터치스크린과 같이 입력과 출력을 위한 기능이 하나로 통합된 장치와의 인터페이스를 위한 수단일 수도 있다. 보다 구체적인 예로, 취약점 탐지 장치(100)의 프로세서(110)는 메모리(120)에 로딩되거나, 저장된 컴퓨터 프로그램의 명령을 처리함에 있어서 서비스 화면이나 컨텐츠가 입출력부(140)를 통해 디스플레이에 표시될 수 있다.

[0053] 또한, 다른 실시예들에서 취약점 탐지 장치(100)는 도 1의 구성요소들보다 더 많은 구성요소들을 포함할 수도 있다. 예를 들어, 상술한 입출력 장치 중 적어도 일부를 포함하도록 구현되거나 또는 내부 구성요소들에 전력을 공급하는 배터리 및 충전 장치, 각종 센서, 데이터베이스 등과 같은 다른 구성요소들을 더 포함할 수도 있다.

[0054] 도 2는 메모리(120)의 블록도이다.

[0055] 메모리(120)는, 대상 프로그램(121)과 취약점 탐지부(122)를 포함할 수 있다. 대상 프로그램(121)는 복수의 명령어들을 포함하는 소프트웨어로서, 타입 안전 위반 등의 취약점을 탐지하고자 하는 대상 프로그램을 말한다.

[0056] 취약점 탐지부(122)는 러스트 언어로 구현된 프로그램을 실행 파일로 변환하는 과정에서, 러스트 언어에 포함된 취약점을 탐지함에 있어서, 러스트 언어에 포함된 안전하지 않은 영역을 별도로 분리하고 분리된 안전하지 않은 영역에 대해서만 취약점을 탐지할 수 있다.

[0057] 취약점 탐지부(122)는 러스트 언어로 구현된 프로그램(소스코드)을 고레벨 중간 표현의 코드로 변환하는 제1 단계, 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환하는 제2 단계, 저레벨 중간 표현의 코드를 머신 코드로 변환하는 제3 단계를 수행하여 실행 파일(바이너리 코드)을 생성할 수 있다.

[0058] 제1 단계에서는, 소스 코드를 고레벨 중간 표현의 코드로 변환하고 변환하는 과정에서 파일, 함수, 블록의 단위로 분할하고 분할된 각 영역(portion)에 대해서 안전하지 않은 코드와 안전한 코드를 구분할 수 있다. 취약점 탐지부(122)는 제1 단계에서 획득된 안전하지 않은 코드에 대한 정보를 별도의 테이블로 기록할 수 있다. 취약점 탐지부(122)는, 타입의 안전성을 평가할 수 있다. 보다 구체적으로는, 취약점 탐지부(122)는, 메모리 할당 코드에서, 제1 포인터에 제2 포인터가 할당되는 경우, 제1 포인터의 타입과 제2 포인터의 타입이 일치하지 않는 코드 등을 검출할 수 있다.

[0059] 제2 단계에서는, 취약점 탐지부(122)는 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환할 수 있다.

[0060] 제3 단계에서는, 취약점 탐지부(122)는 키워드의 맞춤법이 틀리거나, 필요한 문장 부호를 빠뜨리거나, if, end if 등의 잘못된 사용, 선언되지 않은 변수의 사용, 무한 루프에 빠지거나, Null pointer 에러, 존재하지 않는 메모리 위치에 접근하는 등의 유효하지 않은 코드를 검출할 수 있다. 취약점 탐지부(122)는 상기의 취약점에 한정되지 않고 다양한 취약점을 검출할 수 있다.

[0061] 취약점 탐지부(122)는 제1 단계에서 기록한 테이블을 이용하여 안전한 코드에 대해서는 취약점을 탐지하는 것을 스킵(skip)하고, 안전하지 않은 코드에 대해서는 상기에서 언급한 취약점을 탐지할 수 있다. 여기서, 취약점이라는 것은, 프로그램이 코딩된 의도와 다른 의도로 실행될 수 있는 부분을 말할 수 있다. 취약점은 컴파일 과정에는 에러로 검출되지 않아 저레벨 코드로 변환되는 부분이지만 실행 시에는 버그 등의 형태로 나타나는 것을 말할 수 있다.

[0062] 도 3은, 본 개시의 실시예들에 따른 취약점 탐지 방법의 흐름도이다.

[0063] S110에서는, 취약점 탐지 장치(100)는, 대상 프로그램을 판독한다.

[0064] S120에서는, 취약점 탐지 장치(100)는, 대상 프로그램을 파일, 함수, 블록으로 구분하면서 고레벨 중간 표현의 코드로 변환하되, 대상 프로그램에 포함된 안전하지 않은 코드에 대한 정보를 별도의 테이블로 기록할 수 있다.

[0065] S130에서는, 취약점 탐지 장치(100)는, 대상 프로그램의 고레벨 중간 표현의 코드를 저레벨 중간 표현의 코드로 변환할 수 있다.

[0066] S140에서는, 취약점 탐지 장치(100)는, 저레벨 중간 표현의 코드 중에서, 안전하지 않은 영역에 대한 정보를 포함하는 테이블을 이용하여 안전하지 않은 영역과 안전한 영역을 구분할 수 있다. 취약점 탐지 장치(100)는, 안전하지 않은 영역의 코드에 포함된 취약점을 검출할 수 있다. 여기서, 취약점이라는 것은, 프로그램이 코딩된 의도와 다른 의도로 실행될 수 있는 부분을 말할 수 있다. 취약점은 컴파일 과정에는 에러로 검출되지 않아 저레벨 코드로 변환되는 부분이지만 실행 시에는 버그 등의 형태로 나타나는 것을 말할 수 있다.

[0067] 도 4는, 러스트 언어로 구현된 프로그램의 취약점을 검출하는 과정을 설명하는 도면이다.

[0068] 취약점 탐지 장치(100)는, 소정의 방법을 이용하여 소스 코드를 고레벨 중간 표현의 코드로 변환할 수 있다. 취약점 탐지 장치(100)는, 프로그램을 HIR(High level intermediate representation)의 코드로 변환하고, THIR(Typed High level intermediate representation)의 코드로 변환하는 2단계로 코드를 변환할 수 있다.

[0069] 취약점 탐지 장치(100)는, HIR(High level intermediate representation)의 코드로 변환된 이후에, 해당 코드를 분석하여 타입의 안전성을 평가할 수 있다. 취약점 탐지 장치(100)는, THIR(Typed High level intermediate representation)의 코드에 대해서, -Z thir-unsafeck 등의 플래그를 사용하여 안전한 영역과 안전하지 않은 영역을 분리할 수 있다. 상기 -Z thir-unsafeck의 플래그는 하나의 예시에 불과하며 안전하지 않은 영역과 안전한 영역을 분리하도록 구현된 다른 플래그를 사용할 수 있음은 자명하다. 취약점 탐지 장치(100)는, 안전하지 않은 영역에 대한 데이터를 별도로 테이블로 저장할 수 있다(UNSI).

[0070] 제2 단계에서는, 고레벨 중간 표현의 코드를 중간레벨 중간 표현(Mid level intermediate representation)으로 변환하고, 중간레벨 중간 표현의 코드를 저레벨 중간 표현(IR)의 코드로 변환할 수 있다. 제2 단계에서는, 변환된 코드에 대해서 대여 확인 규칙(Borrowing check) 및 최적화를 진행할 수 있다. 취약점 탐지 장치(100)는, 변환된 코드에서 대여 확인 규칙을 위반하는 코드가 존재하는지 여부를 판단할 수 있다. 또한, 취약점 탐지 장치(100)는, 변환된 코드에서 불필요한 코드를 검출하고 불필요한 코드를 제거하는 등으로 최적화할 수 있다.

[0071] 제3 단계에서는, 취약점 탐지 장치(100)는, 안전하지 않은 영역에 대한 데이터를 이용하여 저레벨 중간 표현의 코드 중에서 안전하지 않은 영역의 코드에 대해서 메모리 안전성을 판단하여 상기 코드가 기 정의된 취약점을 가지는지 여부를 판단할 수 있다. 취약점은 미리 설정된 정보에 기초하여 결정되는 것으로, 관리자에 의해 변경 가능할 수 있다.

[0072] 저레벨 중간 표현의 코드는 안전한 저레벨 중간 표현의 코드와 안전하지 않은 저레벨 중간 표현의 코드를 포함할 수 있다.

[0073] 이를 통해, 러스트 언어로 구현된 프로그램에 포함된 취약점을 판단함에 있어서, 안전한 영역의 코드까지 분석하지 않고 안전하지 않은 영역의 코드 만을 분석할 수 있다. 프로그램을 전체적으로 분석하여 취약점을 검출하지 않고 프로그램의 일부인 안전하지 않은 영역의 코드 만을 분석하게 되면 불필요한 오버헤드가 발생하지 않아 더 짧은 시간, 더 적은 리소스를 사용하고 프로그램을 분석할 수 있다. 종래의 기술에 따르면, 컴파일 과정에서 생성된 데이터는 컴파일 수행 이후에 소멸될 수 있다. 그에 반해, 취약점 탐지 장치는 이전 단계를 수행하면서 획득된 정보를 기록하여 이후의 단계에서 사용할 수 있게 된다. 안전하지 않은 영역의 비율은, 프로그램 언어에 따라 달라질 수 있는데 러스트 언어의 경우, 10프로 미만일 수 있다.

[0074] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 어플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[0075] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로

(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(signal wave)에 영구적으로, 또는 일시적으로 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

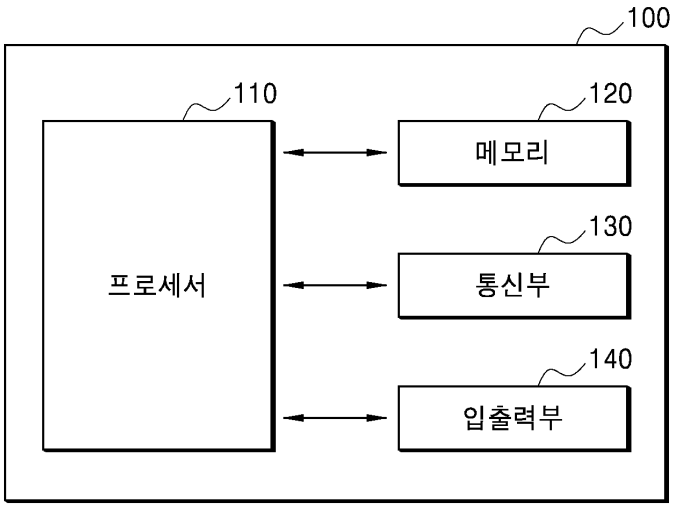
[0076] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic 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. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0077] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

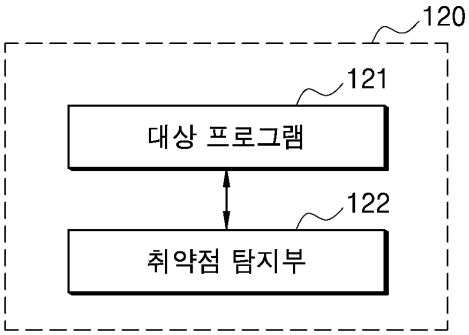
[0078] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

도면

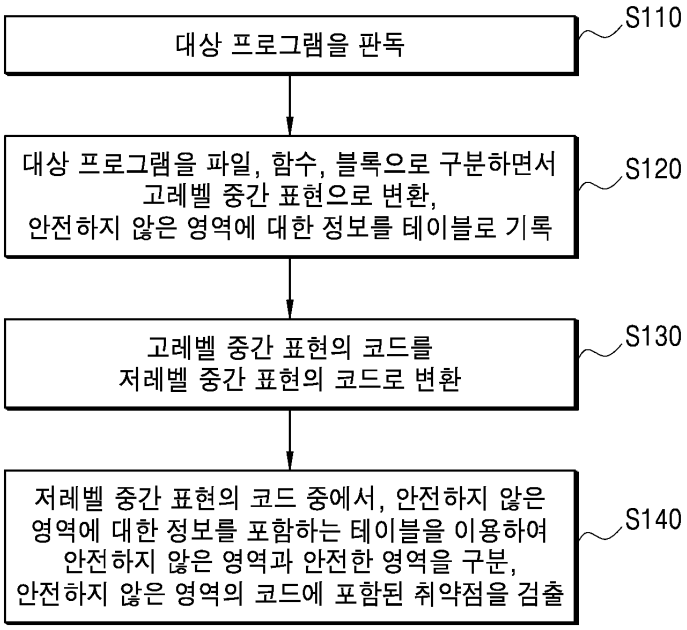
도면1



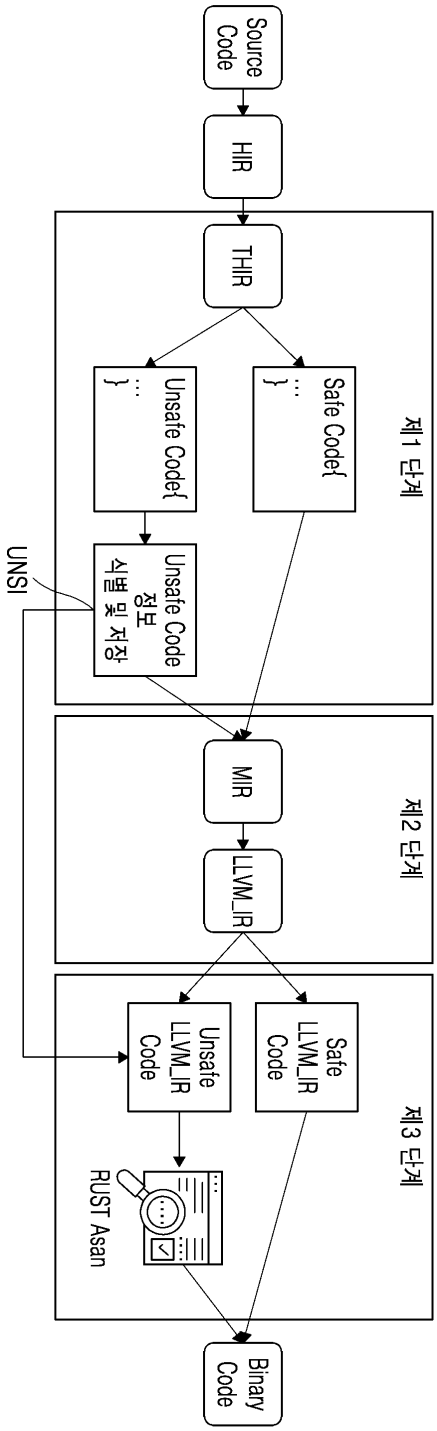
도면2


120
121
대상 프로그램
122
취약점 탐지부

도면3


대상 프로그램을 판독
S110
대상 프로그램을 파일, 함수, 블록으로 구분하면서
고레벨 중간 표현으로 변환,
안전하지 않은 영역에 대한 정보를 테이블로 기록
S120
고레벨 중간 표현의 코드를
저레벨 중간 표현의 코드로 변환
S130
저레벨 중간 표현의 코드 중에서, 안전하지 않은
영역에 대한 정보를 포함하는 테이블을 이용하여
안전하지 않은 영역과 안전한 영역을 구분,
안전하지 않은 영역의 코드에 포함된 취약점을 검출
S140

도면4


Source Code
HIR
THIR
Unsafe Code{ … }
Safe Code{ … }
제1 단계
UNSI
Unsafe Code 정보 식별 및 저장
MIR
LLVM_IR
제2 단계
Unsafe LLVM_IR Code
Safe LLVM_IR Code
RUST Asan
제3 단계
Binary Code