



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2025-0080722
(43) 공개일자 2025년06월05일

- | | |
|---|---|
| <p>(51) 국제특허분류(Int. Cl.)
 <i>G06F 21/56</i> (2013.01) <i>G06F 11/36</i> (2025.01)
 <i>G06F 21/57</i> (2013.01) <i>G06F 21/64</i> (2013.01)
 <i>H04L 9/00</i> (2022.01)</p> <p>(52) CPC특허분류
 <i>G06F 21/563</i> (2013.01)
 <i>G06F 11/3688</i> (2013.01)</p> <p>(21) 출원번호 10-2024-0052599
 (22) 출원일자 2024년04월19일
 심사청구일자 2024년04월19일</p> <p>(30) 우선권주장
 1020230168081 2023년11월28일 대한민국(KR)</p> | <p>(71) 출원인
 고려대학교 산학협력단
 서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)</p> <p>(72) 발명자
 인호
 서울특별시 용산구 서빙고로 35 용산시티파크1단지 102동 1705호</p> <p>전인성
 세종특별자치시 새롬중앙로 90 새뜸마을10단지 1001동 1502호</p> <p>(74) 대리인
 윤귀상</p> |
|---|---|

전체 청구항 수 : 총 10 항

(54) 발명의 명칭 **취약점 분석 방법, 이를 수행하기 위한 기록 매체 및 장치**

(57) 요약

본 발명은 블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 방법에 관한 것으로, 입력부가, 미 완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받는 단계; 추정부가, 사전에 마련되는 매핑 테이블에 기초하여 상기 소스 코드의 전체 맥락 중 상기 타겟 문장의 맥락을 추정하는 단계; 파서부가, 상기 추정된 맥락에 따라 상기 미완성된 스마트 컨트랙트를 파싱하여 상기 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성하는 단계; 및 검출부가, 상기 AST에 기초하여 상기 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하는 단계를 포함한다. 이에 의해 스마트 컨트랙트의 개발 도중에 인터프리터 방식으로 코드를 분석하고, 문제를 파악하여 수정이 필요한 경우 수정에 소모되는 비용을 감소시킬 수 있다.

대 표 도 - 도2



(52) CPC특허분류

G06F 11/3692 (2013.01)

G06F 11/3698 (2025.01)

G06F 21/57 (2013.01)

G06F 21/64 (2013.01)

H04L 9/50 (2022.05)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711193387
과제번호	2022-0-01198-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송혁신인재양성
연구과제명	융합보안대학원(고려대학교)
과제수행기관명	고려대학교산학협력단
연구기간	2019.05.01 ~ 2024.12.31

명세서

청구범위

청구항 1

블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 방법에 있어서,

입력부가, 미완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받는 단계;

추정부가, 사전에 마련되는 매핑 테이블에 기초하여 상기 소스 코드의 전체 맥락 중 상기 타겟 문장의 맥락을 추정하는 단계;

파서부가, 상기 추정된 맥락에 따라 상기 미완성된 스마트 컨트랙트를 파싱하여 상기 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성하는 단계; 및

검출부가, 상기 AST에 기초하여 상기 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하는 단계를 포함하는, 취약점 분석 방법.

청구항 2

제1항에 있어서,

상기 타겟 문장의 맥락을 추정하는 것은,

사전에 마련되는 매핑 테이블을 이용하여 상기 타겟 문장의 형태에 대응되는 라인 넘버에 따라 상기 맥락을 추정하는 것이며,

상기 사전에 마련되는 매핑 테이블은,

상기 스마트 컨트랙트를 구성하는 적어도 하나 이상의 하위 컨트랙트, 상기 하위 컨트랙트를 구성하는 적어도 하나 이상의 함수 및 상기 함수를 구성하는 적어도 하나 이상의 명령문인 성명(statement) 각각에 대한 라인 넘버를 키(key) 값으로 갖고, 상기 라인 넘버에 대응하는 맥락을 밸류(value) 값으로 갖는 테이블인, 취약점 분석 방법.

청구항 3

제1항에 있어서,

상기 스마트 컨트랙트는,

솔리디티 언어 기반의 스마트 컨트랙트이고,

상기 취약점이 존재하는 문장을 검출하는 단계는,

사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 상기 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트하는 것이고,

상기 상태는,

상기 변수에 대한 범위를 의미하는 매핑변수로, 상기 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값인, 취약점 분석 방법.

청구항 4

제3항에 있어서,

상기 사전에 마련되는 밸류 범위 검증 함수는,

상기 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 상기 정수형 변수가 포함되는 문장에 대한 상기 상태를 업데이트하되,

업데이트된 상태와 상기 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련되는 것인, 취약점 분석 방법.

청구항 5

제4항에 있어서,

상기 취약점 분석 방법은,

출력부가, 상기 미완성된 스마트 컨트랙트를 디스플레이하되, 상기 오류를 반환한 문장을 상기 취약점이 존재하는 문장으로 식별 가능하도록 표시하는 상기 미완성된 스마트 컨트랙트를 디스플레이하는 단계를 더 포함하는, 취약점 분석 방법.

청구항 6

제1항에 따른 상기 취약점 분석 방법을 수행하기 위한 컴퓨터 프로그램이 기록된 컴퓨터로 판독가능한 기록 매체.

청구항 7

블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 장치에 있어서,

미완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받는 입력부;

사전에 마련되는 매핑 테이블에 기초하여 상기 소스 코드의 전체 맥락 중 상기 타겟 문장의 맥락을 추정하는 추정부;

상기 추정된 맥락에 따라 상기 미완성된 스마트 컨트랙트를 파싱하여 상기 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성하는 파서부; 및

상기 AST에 기초하여 상기 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하는 검출부를 포함하는, 취약점 분석 장치.

청구항 8

제7항에 있어서,

상기 추정부는,

사전에 마련되는 매핑 테이블을 이용하여 상기 타겟 문장의 형태에 대응되는 라인 넘버에 따라 상기 맥락을 추정하고,

상기 사전에 마련되는 매핑 테이블은,

상기 스마트 컨트랙트를 구성하는 적어도 하나 이상의 하위 컨트랙트, 상기 하위 컨트랙트를 구성하는 적어도 하나 이상의 함수 및 상기 함수를 구성하는 적어도 하나 이상의 명령문인 성명(statement) 각각에 대한 라인 넘버를 키(key) 값으로 갖고, 상기 라인 넘버에 대응하는 맥락을 밸류(value) 값으로 갖는 테이블인, 취약점 분석 장치.

청구항 9

제7항에 있어서,

상기 스마트 컨트랙트는,

솔리디티 언어 기반의 스마트 컨트랙트이고,

상기 검출부는,

사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 상기 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트하고,

상기 상태는,

상기 변수에 대한 범위를 의미하는 매핑변수로, 상기 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값인, 취약점 분석 장치.

청구항 10

제9항에 있어서,

상기 사전에 마련되는 밸류 범위 검증 함수는,

상기 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 상기 정수형 변수가 포함되는 문장에 대한 상기 상태를 업데이트하되,

업데이트된 상태와 상기 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련되는 것인, 취약점 분석 장치.

발명의 설명

기술 분야

[0001] 본 발명은 미완성된 스마트 컨트랙트(Smart Contract)의 취약점을 분석할 수 있는, 취약점 분석 방법, 이를 수행하기 위한 기록 매체 및 장치에 관한 것이다.

[0002] 본 발명은 국가연구개발사업인 과학기술정보통신부의 정보통신방송혁신인재양성사업(과제고유번호 : 1711193387)의 지원에 의한 연구 성과이다.

배경 기술

[0003] NFT(Non-Fungible Token)는 스마트 컨트랙트(Smart Contract) 기반으로 구현된 디지털 자산으로, 토큰 간 교환 기능을 제공하지 않아 연동된 실제 자산의 소유권과 유일성을 보장한다. 이러한 특성으로 인해 자산의 고유성이 중요한 예술, 경매 등에서 주로 활용되며 최근에는 게임, 스포츠, 메타버스로 그 사용처가 확장되고 있다.

[0004] 하지만 NFT를 기반으로 한 프로젝트가 증가함에 따라 보안 관련 문제도 많이 발생하고 있다. 특히 토큰의 발생, 전송, 구매 기능 등의 구현 시 정량적인 요소가 많아 필연적으로 정수형 변수와 관련된 취약 사례가 문제되고 있다.

[0005] 대표적인 정수형 변수 관련 취약점으로는 정수 넘침(Integer Overflow/Underflow)이 있으나, 이를 해결하기 위해 컨트랙트 구현단계에서 정수 넘침을 방지할 수 있는 SafeMath 라이브러리, 정적 분석 기반의 정수 넘침 취약점 분석기 등이 개발되었다.

[0006] 그러나 정수 넘침이 아닌 특정 변수의 값이 0이 되는 등 사용자가 의도했던 변수 값의 범위와 실제 변수의 값의 범위가 다름으로 인한 취약사례는 해결하지 못한다.

[0007] 보다 구체적으로 도 1은 종래의 정수 넘침이 아닌 NFT 정수형 변수의 취약사례를 도시한 도면이다.

[0008] 도 1에 도시된 소스 코드 중 함수 "buyItem"은 해당 토큰에 대해 마켓 플레이스(marketplace)에 등록된 사람 수 만큼 토큰 발행량을 정하고, 그 가격을 설정하여 토큰을 발행한다. 그러나 라인 37에서 "_quantity" 변수의 값을 검사하는 코드가 없어 공격자가 buyItem 함수 호출 시 "_quantity" 값을 0으로 초기화하여 최종 토큰 가격이 0원으로 발행되는 문제가 발생했다.

[0009] 이러한 취약점은 구현 단계에서 사용자가 의도했던 변수 값의 범위와 실제 변수의 값의 범위가 다름으로 인한 취약 사례이며, 기존의 정수 넘침 취약점과 달라 도 1에 도시된 바와 같은 취약사례에 대한 다양한 해결 방안이 필요하다.

[0010] 특히 스마트 컨트랙트도 프로그래밍 언어로 작성된 소프트웨어로 프로그래머에 의해 개발되므로 스마트 컨트랙트에 필연적으로 보안 취약점이 존재할 수 밖에 없는데, 현재까지 개발된 스마트 컨트랙트 취약점 분석기들은 완성된 스마트 컨트랙트를 대상으로 분석하게 된다. 이 경우 이와 같은 다양한 문제가 발생할 수 있다.

[0011] 먼저 스마트 컨트랙트는 함수 간, 컨트랙트 간 의존성이 높은 프로그램 중 하나이다. 이러한 특성으로 인해 스마트 컨트랙트 기반으로 만들어진 탈중앙화 금융(DeFi, Decentralized Finance) 생태계는 'money lego'라는 별

명이 붙었다.

- [0012] 여기서 money lego는, 모든 Defi 애플리케이션(application)들은 독립적으로 개발되지 않고, 기존의 스마트 컨트랙트를 활용해서 새로운 형태의 애플리케이션 들을 만든다고 해서 생겨난 용어이다.
- [0013] 이에 분석기를 통해 완성된 스마트 컨트랙트에서 보안 취약점이 발견된 경우, 단순히 발견된 코드 라인의 로직만을 수정하는 것이 아닌, 해당 스마트 컨트랙트와 의존 관계에 있는 모든 스마트 컨트랙트를 수정해야 하는 경우가 발생한다.
- [0014] 한편 종래의 분석 도구들은 코드의 실행 의미를 분석하고 대부분 이미 알려진 취약점을 다루는 데 초점을 맞추고 있어 새로운 취약점을 분석하고 이에 대한 대응이 어렵다는 문제가 있다.
- [0015] 또한 종래의 분석 도구들은 모든 형태의 보안 취약점을 탐지하지 못하고 특정 보안 취약점만 탐지할 수 있어 효율성이 떨어진다는 문제가 있다.
- [0016] 따라서 완성된 스마트 컨트랙트를 분석하는 것이 아닌, 개발 도중에 인터프리터 방식으로 코드를 분석하고, 문제를 파악하여 수정이 필요한 경우 수정에 소모되는 비용을 감소시킬 수 있는 방안이 필요하다.

선행기술문헌

특허문헌

- [0017] (특허문헌 0001) 한국공개특허공보 제10-2022-0067521호

발명의 내용

해결하려는 과제

- [0018] 본 발명은 상기와 같은 문제를 해결하기 위해 안출된 것으로, 본 발명의 목적은 완성된 스마트 컨트랙트를 분석하는 것이 아닌, 개발 도중에 인터프리터 방식으로 코드를 분석하고, 문제를 파악하여 수정이 필요한 경우 수정에 소모되는 비용을 감소시킬 수 있는 취약점 분석 방법, 이를 수행하기 위한 기록 매체 및 장치를 제공하는 것이다.

과제의 해결 수단

- [0019] 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른 취약점 분석 방법은, 블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 방법으로, 입력부가, 미완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받는 단계; 추정부가, 사전에 마련되는 매핑 테이블에 기초하여 상기 소스 코드의 전체 맥락 중 상기 타겟 문장의 맥락을 추정하는 단계; 파서부가, 상기 추정된 맥락에 따라 상기 미완성된 스마트 컨트랙트를 파싱하여 상기 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성하는 단계; 및 검출부가, 상기 AST에 기초하여 상기 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하는 단계를 포함한다.
- [0020] 그리고 상기 타겟 문장의 맥락을 추정하는 것은, 사전에 마련되는 매핑 테이블을 이용하여 상기 타겟 문장의 형태에 대응되는 라인 넘버에 따라 상기 맥락을 추정하는 것이며, 상기 사전에 마련되는 매핑 테이블은, 상기 스마트 컨트랙트를 구성하는 적어도 하나 이상의 하위 컨트랙트, 상기 하위 컨트랙트를 구성하는 적어도 하나 이상의 함수 및 상기 함수를 구성하는 적어도 하나 이상의 명령문인 성명(statement) 각각에 대한 라인 넘버를 키(key) 값으로 갖고, 상기 라인 넘버에 대응하는 맥락을 밸류(value) 값으로 갖는 테이블일 수 있다.
- [0021] 또한 상기 스마트 컨트랙트는, 솔리디티 언어 기반의 스마트 컨트랙트이고, 상기 취약점이 존재하는 문장을 검출하는 단계는, 사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 상기 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트하는 것이고, 상기 상태는, 상기 변수에 대한 범위를 의미하는 매핑변수로, 상기 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값일 수 있다.
- [0022] 그리고 상기 사전에 마련되는 밸류 범위 검증 함수는, 상기 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 상기 정수형 변수가 포함되는 문장에 대한 상기 상태를 업데이트하되, 업데이트된 상

태와 상기 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련되는 것일 수 있다.

[0023] 또한 상기 취약점 분석 방법은, 출력부가, 상기 미완성된 스마트 컨트랙트를 디스플레이하되, 상기 오류를 반환한 문장을 상기 취약점이 존재하는 문장으로 식별 가능하도록 표시하는 상기 미완성된 스마트 컨트랙트를 디스플레이하는 단계를 더 포함할 수 있다.

[0024] 한편 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른 기록 매체는, 상기 취약점 분석 방법을 수행하기 위한 컴퓨터 프로그램이 기록된 컴퓨터로 판독가능한 기록 매체이다.

[0025] 한편 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른 취약점 분석 장치는, 블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 장치로서, 미완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받는 입력부; 사전에 마련되는 매핑 테이블에 기초하여 상기 소스 코드의 전체 맥락 중 상기 타겟 문장의 맥락을 추정하는 추정부; 상기 추정된 맥락에 따라 상기 미완성된 스마트 컨트랙트를 파싱하여 상기 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성하는 파서부; 및 상기 AST에 기초하여 상기 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하는 검출부를 포함할 수 있다.

[0026] 그리고 상기 추정부는, 사전에 마련되는 매핑 테이블을 이용하여 상기 타겟 문장의 형태에 대응되는 라인 넘버에 따라 상기 맥락을 추정하고, 상기 사전에 마련되는 매핑 테이블은, 상기 스마트 컨트랙트를 구성하는 적어도 하나 이상의 하위 컨트랙트, 상기 하위 컨트랙트를 구성하는 적어도 하나 이상의 함수 및 상기 함수를 구성하는 적어도 하나 이상의 명령문인 성명(statement) 각각에 대한 라인 넘버를 키(key) 값으로 갖고, 상기 라인 넘버에 대응하는 맥락을 밸류(value) 값으로 갖는 테이블일 수 있다.

[0027] 또한 상기 스마트 컨트랙트는, 솔리디티 언어 기반의 스마트 컨트랙트이고, 상기 검출부는, 사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 상기 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트하고, 상기 상태는, 상기 변수에 대한 범위를 의미하는 매핑변수로, 상기 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값일 수 있다.

[0028] 그리고 상기 사전에 마련되는 밸류 범위 검증 함수는, 상기 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 상기 정수형 변수가 포함되는 문장에 대한 상기 상태를 업데이트하되, 업데이트된 상태와 상기 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련되는 것일 수 있다.

발명의 효과

[0029] 상술한 본 발명의 일측면에 따르면, 취약점 분석 방법, 이를 수행하기 위한 기록 매체 및 장치를 제공함으로써, 완성된 스마트 컨트랙트를 분석하는 것이 아닌, 개발 도중에 인터프리터 방식으로 코드를 분석하고, 문제를 파악하여 수정이 필요한 경우 수정에 소모되는 비용을 감소시킬 수 있다.

도면의 간단한 설명

[0030] 도 1은 종래의 NFT 정수형 변수 취약사례를 도시한 도면,
 도 2는, 본 발명의 일 실시예에 따른 취약점 분석 장치의 구성을 설명하기 위한 도면,
 도 3은, 종래의 파서에서 파싱을 시작하는 방법을 설명하기 위한 도면,
 도 4는 종래의 파서가 파싱하는 솔리디티 언어 기반의 컨트랙트를 설명하기 위한 예시도,
 도 5는 도 3에 도시된 하위 규칙들 중 하나를 사용한 파싱 규칙을 구체적으로 설명하기 위한 도면,
 도 6 내지 도 8은 도 5에 도시된 규칙의 세부 규칙을 설명하기 위한 도면,
 도 9 및 도 10은 본 실시예에 따른 장치(100)가 실시간으로 분석하기 위한 타겟 문장을 예시적으로 도시한 도면,
 도 11은 본 실시예에 따른 장치(100)가 타겟 문장의 맥락을 파악하는 과정을 설명하기 위한 예시 코드,
 도 12는 본 발명의 일 실시예에 따른 장치(100)가 사용하는 매핑 테이블을 설명하기 위한 도면,
 도 13은 스마트 컨트랙트의 구조를 설명하기 위한 도면,
 도 14 내지 도 16은 각각의 함수를 구성하는 성명의 유형을 설명하기 위한 도면,

도 17은 본 실시예에 따른 장치가 사용하는 매핑 테이블에 대한 실제 자료 구조를 설명하기 위한 도면,
 도 18 및 도 19는 주석을 작성하는 방식을 설명하기 위한 예시도,
 도 20 및 도 21은 파싱 규칙을 만들 때와 같이 새롭게 주석을 파싱할 수 있는 규칙을 도시한 도면,
 도 22는 종래 파서에서의 파싱 규칙을 도시한 도면,
 도 23은 주석을 파싱하고 오류를 나타내는 방법과 관련된 스마트 컨트랙트의 예시,
 도 24는 본 실시예에 따른 장치가 사용하는 밸류 검증 함수를 설명하기 위한 도면,
 도 25는 도 24의 밸류 검증 함수의 의사코드를 설명하기 위한 도면,
 도 26은 본 실시예에 따른 요약 해석을 위한 스마트 컨트랙트의 예시도,
 도 27은 본 실시예에 따른 장치(100)가 취약점이 존재하는 문장을 출력하는 모습을 설명하기 위한 예시도, 그리고
 도 28은 본 발명의 일 실시예에 따른 취약점 분석 방법을 설명하기 위한 흐름도이다.

발명을 실시하기 위한 구체적인 내용

- [0031] 후술하는 본 발명에 대한 상세한 설명은, 본 발명이 실시될 수 있는 특정 실시예를 예시로서 도시하는 첨부 도면을 참조한다. 이들 실시예는 당업자가 본 발명을 실시할 수 있기에 충분하도록 상세히 설명된다. 본 발명의 다양한 실시예는 서로 다르지만 상호 배타적일 필요는 없음이 이해되어야 한다. 예를 들어, 여기에 기재되어 있는 특정 형상, 구조 및 특성은 일 실시예와 관련하여 본 발명의 정신 및 범위를 벗어나지 않으면서 다른 실시예로 구현될 수 있다. 또한, 각각의 개시된 실시예 내의 개별 구성요소의 위치 또는 배치는 본 발명의 정신 및 범위를 벗어나지 않으면서 변경될 수 있음이 이해되어야 한다. 따라서, 후술하는 상세한 설명은 한정적인 의미로서 취하려는 것이 아니며, 본 발명의 범위는, 적절하게 설명된다면, 그 청구항들이 주장하는 것과 균등한 모든 범위와 더불어 첨부된 청구항에 의해서만 한정된다. 도면에서 유사한 참조부호는 여러 측면에 걸쳐서 동일하거나 유사한 기능을 지칭한다.
- [0032] 본 발명에 따른 구성요소들은 물리적인 구분이 아니라 기능적인 구분에 의해서 정의되는 구성요소들로써 각각이 수행하는 기능들에 의해서 정의될 수 있다. 각각의 구성요소들은 하드웨어 또는 각각의 기능을 수행하는 프로그램 코드 및 프로세싱 유닛으로 구현될 수 있을 것이며, 두 개 이상의 구성요소의 기능이 하나의 구성요소에 포함되어 구현될 수도 있을 것이다. 따라서 이하의 실시예에서 구성요소에 부여되는 명칭은 각각의 구성요소를 물리적으로 구분하기 위한 것이 아니라 각각의 구성요소가 수행되는 대표적인 기능을 암시하기 위해서 부여된 것이며, 구성요소의 명칭에 의해서 본 발명의 기술적 사상이 한정되지 않는 것임에 유의하여야 한다.
- [0033] 이하에서는 도면들을 참조하여 본 발명의 바람직한 실시예들을 보다 상세하게 설명하기로 한다.
- [0034] 도 2는 본 발명의 일 실시예에 따른 취약점 분석 장치(100)를 설명하기 위한 블록도이다.
- [0035] 본 실시예에 따른 취약점 분석 장치(100, 이하 장치)는 블록체인을 위한 솔리디티 언어 기반의 스마트 컨트랙트의 취약점을 분석하기 위해 마련될 수 있다.
- [0036] 특히 본 실시예에 따른 장치(100)는 완성된 스마트 컨트랙트가 아닌 개발 중인 미완성된 상태의 스마트 컨트랙트의 취약점을 분석하기 위해 마련된다.
- [0037] 이를 위해 본 실시예에 따른 장치(100)는 입력부(110), 추정부(130), 파서부(150), 검출부(170) 및 출력부(190)를 포함할 수 있다. 그리고 장치(100)는 취약점 분석 방법을 수행하기 위한 소프트웨어(어플리케이션)가(이) 설치되어 실행될 수 있으며, 입력부(110), 추정부(130), 파서부(150), 검출부(170) 및 출력부(190)는 취약점 분석 방법을 수행하기 위한 소프트웨어(어플리케이션)에 의해 제어될 수 있다.
- [0038] 이때 장치(100)는 별도의 단말이거나 또는 단말의 일부 모듈일 수 있다. 또한, 입력부(110), 추정부(130), 파서부(150), 검출부(170) 및 출력부(190)의 구성은 통합 모듈로 형성되거나, 하나 이상의 모듈로 이루어질 수 있다. 그러나, 이와 반대로 각 구성은 별도의 모듈로 이루어질 수도 있다.
- [0039] 또한 장치(100)는 이동성을 갖거나 고정될 수 있다. 이러한 장치(100)는, 서버(server) 또는 엔진(engine) 형태일 수 있으며, 디바이스(device), 기구(apparatus), 단말-terminal), UE(user equipment), MS(mobile

station), 무선기기(wireless device), 휴대기기(handheld device) 등 다른 용어로 불릴 수 있다. 그리고 장치(100)는 운영체제(Operation System; OS), 즉 시스템을 기반으로 다양한 소프트웨어를 실행하거나 제작할 수 있다. 여기서 운영체제는 소프트웨어가 장치의 하드웨어를 사용할 수 있도록 하기 위한 시스템 프로그램으로서, 안드로이드 OS, iOS, 윈도우 모바일 OS, 바다 OS, 심비안 OS, 블랙베리 OS 등 모바일 컴퓨터 운영체제 및 윈도우 계열, 리눅스 계열, 유닉스 계열, MAC, AIX, HP-UX 등 컴퓨터 운영체제를 모두 포함할 수 있다.

[0040] 또한 도면에는 미도시되었으나, 본 실시예에 따른 장치는 저장부는 취약점 분석 방법을 수행하기 위한 프로그램이 기록된 저장부를 더 포함할 수 있다. 이러한 저장부는 입력부(110), 추정부(130), 파서부(150), 검출부(170) 및 출력부(190)가 처리하는 데이터를 일시적 또는 영구적으로 저장하며, 휘발성 저장매체 또는 비휘발성 저장매체를 포함할 수 있으나, 본 발명의 범위가 이에 한정되는 것은 아니다.

[0041] 그리고 저장부는 취약점 분석 방법을 수행하면서 누적되는 데이터가 저장된다.

[0042] 본 실시예에 따른 입력부(110)는 프로그래머와 같은 사용자로부터 타겟 문장을 입력받기 위해 마련된다.

[0043] 이러한 타겟 문장은, 미완성된 스마트 컨트랙트를 구성하는 소스 코드의 적어도 일부를 의미할 수 있다. 그리고 입력부(110)는 타겟 문장이 추정부(130)로 전달되도록 할 수 있다.

[0044] 한편 추정부(130)는 입력부(110)로부터 전달받은 타겟 문장의 맥락을 추정하기 위해 마련될 수 있다.

[0045] 이를 위해 본 실시예에 따른 추정부(130)는 사전에 마련되는 매핑 테이블에 기초하여 소스 코드의 전체 맥락 중 타겟 문장의 맥락을 추정할 수 있다.

[0046] 이하에서는 도 3 내지 도 13을 참고하여 본 발명에 따른 추정부(130)가 미완성된 코드의 타겟 문장의 맥락을 추정하는 과정에 대해 보다 구체적으로 설명하기로 한다.

[0047] 도 3은 종래의 파서에서 파싱을 시작하는 방법을 설명하기 위한 도면이다.

[0048] 먼저 파서는 입력으로 받는 파일에 대해서 코드의 각 단어들이 어떤 의미를 가지는지 분석하는 도구로, 단어 단위로 의미를 찾을 수 있다.

[0049] 이러한 파서의 파싱 규칙은 상위(부모) 규칙과 하위(자식) 규칙이 존재하며 상위 규칙은 여러 개의 하위 규칙으로 구성될 수 있다.

[0050] 그리고 파서는, 모든 단어들을 어떠한 하위 규칙으로도 구분할 수 없을 정도로 최대한 나누고, 단어들의 의미 파악이 모두 완료되면 파싱을 종료하게 된다.

[0051] 만약 예컨대 기호가 누락되었거나, 특정 키워드가 누락된 경우와 같이 규칙에 위배되는 문장, 즉 코드 문법의 오류가 있는 경우에도 파서는 파싱을 중간에 중단하게 된다.

[0052] 이를 위해 코드 편집기에서 문법적으로 오류가 없는 완성된 코드를 파일 형태(.sol 파일)로 추출한 후, 종래의 파서는 해당 파일을 입력받으면 요약 구문 트리인 AST(Abstract Syntax Tree)를 생성하게 된다.

[0053] AST는 프로그래머가 작성한 코드들에 포함되는 단어 하나 하나가 어떤 의미를 가지는지를 나타내주는 구조화된 자료를 의미할 수 있다.

[0054] 도 3은 종래의 파서가 완성된 솔리디티 파일(.sol)을 입력받았을 때 파싱을 시작하기 위한 문법 규칙을 도시한 도면이다.

[0055] 도 3에 도시된 바와 같이 "sourceUnit"은 솔리디티 파일을 파싱하기 위한 최상위 규칙 이름이자, 파싱의 시작 규칙인데, 종래의 파서는 이러한 시작 규칙이 "sourceUnit" 한 개이다.

[0056] 따라서 파서는 최상위 규칙 아래의 하위규칙들 중에 하나로는 무조건 파싱을 시작할 수 있어야 하며, 파싱을 시작하지 못한다는 것은 코드의 문법적 오류가 있다는 것을 의미한다.

[0057] 도 3에서와 같이 시작 규칙의 형태는 "시작규칙 이름:(하위규칙들)* EOF;"이며, "|"는 '또는'을 의미한다. 즉, "|"의 왼쪽 규칙 또는 오른쪽 규칙 중에 하나라는 의미이다.

[0058] 그리고 "*"은 '*의 왼쪽 괄호 안에 있는 규칙들이 0개 이상 나올 수 있다'를 의미한다.

[0059] 또한 "EOF"는 '코드의 가장 끝에 있는 빈 문자, 즉 더 이상 파싱할 문자가 없을 때 까지'를 의미한다. 따라서 시작 규칙에만 "EOF"가 존재할 수 있다.

- [0060] 따라서 도 3을 정리하면, 솔리디티 언어 기반의 스마트 컨트랙트는 "sourceUnit"이라는 최상위 규칙의 하위 규칙들인 "pragmaDirective"라는 규칙부터 "errorDefinition" 규칙 사이의 규칙 중에 하나로는 모두 파싱되어야 한다.
- [0061] 이는 즉, 상위 규칙을 건너뛰고, 하위 규칙에 해당하는 문장이 먼저 나오는 경우 해당 코드는 문법적으로는 문제가 존재하는 코드이므로 파싱이 될 수 없다.
- [0062] 도 4는 종래의 파서가 파싱하는 솔리디티 언어 기반의 컨트랙트를 설명하기 위한 예시도이고, 도 5는 도 3에 도시된 하위 규칙들 중 하나를 사용한 파싱 규칙을 구체적으로 설명하기 위한 도면이다.
- [0063] 도 5에서 초록색 및 보라색으로 표시된 단어는 더 이상 나뉘지지 않는 하위 규칙을 의미하고, 노란색으로 표시된 단어는 나뉘질 수 있는 하위 규칙을 의미한다.
- [0064] 먼저 도 5에 도시된 하위 규칙은 도 3에 도시된 코드를 파싱하기 위한 규칙으로 중 하나로서, "sourceUnit"의 하위 규칙 중 4번째 하위 규칙인 "contractDefine"에 대한 하위 규칙부터 설명하기로 한다.
- [0065] 도 5에 도시된 "'abstract'?"를 살펴보면, 문법 규칙 중 "?"는 '?' 왼쪽에 있는 단어 또는 ()로 되어 있는 규칙의 집합이 1개 있을 수도 있고, 아예 없을 수도 있다"는 의미이다. 즉 "abstract"는 있을 수도 있고, 없을 수도 있다.
- [0066] 그리고 "'contract'"라는 단어는 코드 상에 무조건 있어야 한다. 만약에 어떤 코드 문장의 시작하는 단어가 "abstract contract"또는 "contract"로 시작한다면 파싱을 계속 수행하고, 아니라면 다음 파싱 규칙을 검사하게 된다.
- [0067] 도 6은 도 5에 도시된 규칙 중 하나인 "identifier"규칙을 설명하기 위한 도면이다.
- [0068] 이러한 "identifier"는 주로 변수 이름과 같은 사용자가 정의하는 이름에 대한 규칙 이름이다.
- [0069] 그리고 "identifier"규칙의 세부 규칙을 보면 보라색으로 표시된 "Identifier"이기 때문에 더 이상 하위 규칙으로 나눌 수 없다. 따라서 "Identifier"는 도 4에 도시된 "contract"다음에 사용자가 정의한 컨트랙트 이름, 'test'를 의미한다.
- [0070] 다음으로 도 5에 도시된 "('is' inheritanceSpecifier (',' inheritanceSpecifier)*)?"를 해석해보면, 첫 번째 괄호 안에 있는 "'is' inheritanceSpecifier (',' inheritanceSpecifier)*"가 1개 있을 수도 있고 없을 수도 있다'는 의미이다.
- [0071] 만약에 도 4에 도시된 예시 코드에 해당 규칙이 적용되었다면 "is" 키워드는 무조건 있어야 하는데, 예시 코드에는 "contract test {" 로 작성되었을 뿐, "is" 키워드가 존재하지 않으니, 해당 규칙은 적용되지 않게 된다.
- [0072] 그리고 나서 다음으로 있어야 하는 것은 기호 "{"인데, 예시 코드에서 "contract test" 다음에 "{"에 존재하니 파싱은 그 다음 규칙으로 넘어갈 수 있다.
- [0073] "contractBodyElement*"에서 "*"는 앞서 설명한 바와 같이 '* 왼쪽에 있는 단어나 규칙이 0개 이상 있다'를 의미한다.
- [0074] 이에 도 4의 예시 코드 중 "uint public var1=0;"에 대한 규칙을 "contractBodyElement"의 하위 규칙들에서 다시 찾아야 한다.
- [0075] 여기서 "contractBodyElement"는 "sourceUnit"의 하위 규칙이 아닌 "contractDefinition"의 하위 규칙이지만 "contractDefinition"의 세부 규칙이므로 파싱을 계속 이어 나갈 수 있다.
- [0076] 한편 도 7은 도 5의 규칙 중 "contractBodyElement" 규칙의 세부 규칙을 도시한 도면이다.
- [0077] 도 5에 도시된 "contractBodyElement" 규칙의 세부 규칙 중에서 도 4의 예시 코드의 "uint public var1=0;"에 대한 규칙은, 도 7에서 9번째 하위 규칙인 "stateVariableDeclaration" 규칙이다.
- [0078] 즉, 파서는 만약 9번째 하위 규칙의 앞에 존재하는 8개의 하위 규칙을 모두 살펴보고, 해당 코드에 맞는 규칙이 없을 때 9번째 하위 규칙을 검사하게 된다.
- [0079] 도 8은 이러한 도 5의 규칙 중 "contractBodyElement" 규칙의 세부 규칙 중 하나인 "stateVariableDeclaration"을 도시한 도면이다.

- [0080] 이러한 "stateVariableDeclaration" 규칙의 첫 번째 규칙은 도 8에 도시된 바와 같이 "typeName"이다.
- [0081] "typeName"의 세부 규칙을 본 발명에서 설명하기에는 바람직하지 않으며, 간단히 설명하자면 "typeName"은 솔리디티 언어에서 지원하는 변수의 타입 종류로, 예컨대 "int", "uint", "bool", "address" 등일 수 있다.
- [0082] 이러한 "typeName"에 해당되는 코드는 도 4의 "uint public var1=0;" 중 "unit"에 해당한다.
- [0083] 그 다음 규칙은 "(PublicKeyword | InternalKeyword | PrivateKeyword | ConstantKeyword | ImmutableKeyword | overrideSpecifier)*"이며 괄호 안의 여러 개의 키워드들이 0개 이상 있어야 함을 의미한다.
- [0084] "uint public var1=0;"코드 상으로 "Public"에 해당하는 건 "PublicKeyword" 라는 보라색으로 표시된 하위 규칙이며, "Public"에 대해서는 더 이상 파싱되지 않는다.
- [0085] 그 다음 앞서 설명한 바와 같이 사용자가 정의한 변수 이름인 "identifier" 규칙은 "var1"에 해당된다.
- [0086] 그 다음 규칙은 "('=' expression)?"이다. 이는 '괄호 안의 '=' 과 'expression' 규칙이 있을 수도 있고 없을 수도 있다'라는 의미이다.
- [0087] 도 4의 예시 코드에는 "="이 있기 때문에 그 뒤의 expression은 무조건 있어야 한다. 만약 "="이 있는데 expression이 없다면 이는 문법적 오류이며 파싱은 그 즉시 종료될 수 있다.
- [0088] 한편 표현식인 "expression"은 어떤 숫자 값이 될 수도 있고, a+b 라는 식이 될 수도 있는데, 꼭 이에 한정되지 않으며 다양하게 표현될 수 있다.
- [0089] 따라서 "expression"은 도 4의 예시 코드에서 숫자 "0"이 해당된다.
- [0090] 그 다음으로 "stateVariableDeclaration" 규칙상 세미콜론인 ";"은 코드에 무조건 있어야 하는데, 예시 코드에서도 ";"가 존재하는 것을 확인할 수 있다.
- [0091] 이렇게 "stateVariableDeclaration" 규칙이 끝났고, "contractBodyElement"도 끝났으니, 그림 4의 "contractDefinition" 규칙에서 "}" 부분이 남는다.
- [0092] 그리고 예시 코드에도 "}"가 있으니 예시 코드 전체는 "contractDefinition" 규칙에 부합할 수 있다.
- [0093] 그리고 "}" 뒤에는 아무것도 없으니 즉, EOF에 해당한다. 이는 "sourceUnit" 규칙과 부합하며, "sourceUnit"의 "EOF"를 만나면 파서는 파싱을 성공적으로 마치게 된다.
- [0094] 하지만 도 3 내지 도 8에 따라 상술한 종래 파서의 파싱 규칙은 사용자가 실시간으로 입력하는 코드에 대해 분석이 필요한 경우에는, 적합하지 않다.
- [0095] 도 9 및 도 10은 본 실시예에 따른 추정부(130)가 실시간으로 분석하기 위한 타겟 문장을 예시적으로 도시한 도면이다.
- [0096] 본 실시예에 따른 추정부(130)는 사용자가 실시간으로 입력하는 코드에 대해서 분석하기 때문에 실시간으로 분석하고자 하는 코드 범위에 대한 기준을 정해야 한다.
- [0097] 이에 입력부(110)를 통해 "{"가 입력되거나, ";"이 입력되었을 때 해당 코드를 타겟 문장으로 하여 실시간으로 분석되도록 할 수 있다.
- [0098] 따라서 사용자는 도 9에서와 같이 입력부(110)를 통해 타겟 문장을 입력하는데, 입력부(110)는 사용자가 "{"를 입력하면 "}"는 사용자의 편의를 위해 자동 완성되도록 마련될 수 있다.
- [0099] 이러한 도 9에 도시된 예시 코드는 최상위 규칙인 "sourceUnit"의 하위 규칙인 "contractDefinition" 규칙과 완전히 부합하기 때문에 종래의 파서로도 분석이 가능하다.
- [0100] 이후 사용자는 도 10에 도시된 바와 같은 타겟 문장을 입력부(110)를 통해 입력하게 된다.
- [0101] 즉, 도 9에 도시된 첫번째 타겟 문장에 대해서는 이미 분석이 완료되었으므로, 이후 도 10에 도시된 두번째 타겟 문장에 대한 파싱을 수행해야 한다.
- [0102] 하지만 종래 파서에서의 파싱 시작 규칙으로는 도 10에 도시된 두번째 타겟 문자를 파싱할 수 없다는 문제가 있다.

- [0103] 즉, 상술한 바와 같이 타겟 문장인 "uint public var1=0;"이라는 코드와 매칭되는 규칙은 기존 파서의 파싱 규칙 상으로는 "stateVariableDeclaration" 규칙으로 파싱해야 하나, "stateVariableDeclaration"은 파싱 시작 규칙인 "sourceUnit" 규칙의 하위 규칙이 아니기 때문이다.
- [0104] 구체적으로 도 9 및 도 10에 도시된 예시 코드가 합쳐져 있는 도 4의 예시 코드의 분석이 가능했던 이유는, "stateVariableDeclaration" 규칙이 도 7에 도시된 바와 같이 "contractBodyElement" 규칙의 하위 규칙이며, 이는 "contractDefinition" 규칙 분석 상에서 이루어지기 때문이다.
- [0105] 즉 도 5에 도시된 "contractBodyElement*" 문법 때문이며, 컨트랙트 내부 코드의 모든 문장은 이 "contractBodyElement*" 문법 상에 속한다. 이는 "*" 왼쪽에 있는 규칙이 0개 이상 무한히 발생할 수 있기 때문이다.
- [0106] 따라서 "contractBodyElement" 규칙부터 바로 파싱하는 규칙은 "sourceUnit" 규칙에는 존재하지 않기 때문에 도 10에 도시된 두번째 타겟 문장만 입력하는 경우에는 두번째 타겟 문장을 분석할 수 없다.
- [0107] 다시 말해 컨트랙트 중간에 있는 코드를 파싱하기 위해서는 컨트랙트 전체를 파싱해야만 분석이 가능하며, 중간 코드만을 따로 파싱하는 것은 종래의 파서로는 수행할 수 없다. 이에 중간 코드만을 따로 파싱하기 위해서는 새로운 문법 규칙이 필요하게 된다.
- [0108] 종래의 파서는 코드를 처음부터 순차적으로 파싱해 나가기 때문에 입력되는 코드 순서대로 코드의 맥락을 순차적으로 파악해 나간다.
- [0109] 하지만 실시간으로 분석이 필요한 경우에는, 실시간으로 입력되는 타겟 문장이 어느 맥락에 속한 부분인지 파악하는 것이 필요하다.
- [0110] 이에 본 실시예에 따른 추정부(130)는 사전에 코드 범위에 대한 기준이 정해지고, 도9 또는 도 10에서와 같이 해당 기준을 충족하는 타겟 문장이 입력부(110)를 통해 입력되면, 해당 타겟 문장의 맥락을 추정할 수 있다.
- [0111] 도 11은 본 실시예에 따른 추정부(130)가 타겟 문장의 맥락을 파악하는 과정을 설명하기 위한 예시 코드이다.
- [0112] 도 11에서 "int public stateVariable;"(이하 1번문장)과 "int localVariable = 20;"(이하 2번문장) 문장 모두 변수 선언과 관련된 코드 문장이다.
- [0113] 여기서 1번문장은 함수(function) 바깥에 선언되어 블록체인 상에 값이 계속 남아 있는 "상태 변수"를 나타내는 반면, 2번문장은 함수 안에 선언되어, 함수 수행이 끝나면 값이 블록체인에 남지 않고 사라져버리는 "지역 변수"를 의미한다.
- [0114] 이에 종래의 파서는 첫 줄부터 순차적으로 파싱해 나가기에, 1번문장은 함수 안에서 파싱되지 않고, 함수 바깥에서 파싱 되기에 상태변수로 인지하고, 2번문장은 함수가 선언되고 나서 "{ }" 안, 즉 함수 안에서 선언되었기에 지역변수로 인지하게 된다.
- [0115] 하지만 본 실시예에서의 입력부(110)는 실시간으로 타겟 문장을 입력 받으므로, 1번문장 및 2번문장의 분석을 서로 개별적으로 수행하기 위해서는 해당 타겟 문장이 함수 안에서 선언되었는지, 바깥에서 선언되었는지 파악하는 맥락 파악이 필요하다.
- [0116] 또한 맥락 관점에서 완성된 코드가 아닌 실시간으로 받는 코드의 파싱에서 중요하게 고려되어야 하는 부분은, 입력부(110)를 통해 타겟 문장이 순차적으로 입력되지 않을 가능성이 있다. 예컨대 2번문장이 1번문장보다 먼저 입력될 수 있다. 따라서 이러한 상황에서 각 타겟 문장이 어떤 맥락에 속한 것인지 파악이 필요하다.
- [0117] 이에 본 실시예에 따른 추정부(130)는, 사전에 마련되는 매핑 테이블을 이용하여 타겟 문장의 형태에 대응되는 라인 넘버에 따라 맥락을 추정할 수 있다.
- [0118] 도 12는 본 발명의 일 실시예에 따른 추정부(130)가 사용하는 매핑 테이블을 설명하기 위한 도면이고, 도 13은 스마트 컨트랙트의 구조를 설명하기 위한 도면, 도 14 내지 도 16은 각각의 함수를 구성하는 성명의 유형을 설명하기 위한 도면이다.
- [0119] 매핑 테이블은, 키(key) 값이 있고, 해당 키 값에 해당하는 밸류(value) 값을 표 형태로 정리한 것을 의미할 수 있다.
- [0120] 이에 추정부(130)가 사용하는 사전에 마련되는 매핑 테이블은, 도 12에 도시된 바와 같이 라인 넘버가 키 값이

고, 라인 넘버에 대응하는 맥락(Context)을 밸류 값으로 갖는 테이블이다.

- [0121] 구체적으로 도 13에 도시된 바와 같이 솔리디티 언어 기반의 스마트 컨트랙트 파일의 확장자는 ".sol"이다. 이러한 한 개의 .sol 파일, 즉 스마트 컨트랙트는 적어도 하나 이상의 하위 컨트랙트의 집합으로 구성된다.
- [0122] 그리고 하위 컨트랙트는 적어도 하나 이상의 함수의 집합으로 구성되며, 각각의 함수는 성명(statement)의 집합으로 구성된다.
- [0123] 이때 성명은 코드 문장으로 도 13의 최우측에 도시된 바와 같이 총 13개의 종류일 수 있는데, 이러한 코드 문장은 ";"으로 끝나는 형태의 성명 및 "{"형태로 나타나는 성명으로 분류될 수 있다.
- [0124] 도 13에 도시된 코드 문장들 중에서 ";"으로 끝나는 형태의 성명은 "simpleStatement", "continueStatement", "breakStatement", "returnStatement", "emitStatement" 및 "revertStatement"일 수 있다.
- [0125] 한편 코드 문장들 중에서 블록인 "{"형태로 나타나는 성명은, "ifStatement", "forStatement", "whileStatement", "doWhileStatement", "tryStatement", "assemblyStatement"일 수 있다.
- [0126] 이러한 함수를 구성하는 성명의 집합은 도 14 내지 도 16에 도시된 바와 같을 수 있다.
- [0127] 도 17은 추정부(130)가 사용하는 매핑 테이블에 대한 실제 자료 구조를 설명하기 위한 도면으로, 도 12에 도시된 매핑 테이블을 실제 파이썬(python) 언어의 딕셔너리(dictionary) 형태로 표현한 것이다.
- [0128] 이에 추정부(130)는 딕셔너리 형태를 사용하여 타겟 문장의 맥락을 추정할 수 있다.
- [0129] 즉, 매핑 테이블에는 타겟 문장이 어떤 컨트랙트, 어떤 함수, 어떤 성명에 속하는지와 함께 뒤에 나올 과실 시작 규칙이 함께 저장될 수 있다.
- [0130] 컨트랙트, 함수, 성명은 모두 코드 라인의 범위를 가지고 있기 때문에 추정부(130)는 매핑 테이블을 사용하여 입력부(110)로부터 전달받은 타겟 문장의 코드 라인을 파악하고, 해당 코드 라인이 어느 맥락에 속할지 찾아갈 수 있다.
- [0131] 이하에서는 추정부(130)가 매핑 테이블을 업데이트하는 과정에 대해 설명하기로 한다. 여기서 업데이트라 함은 생성, 변경 및 삭제를 의미할 수 있다.
- [0132] 먼저 추정부(130)는 전달받은 타겟 문장의 성명이 "{"형태로 나타나는 성명인지, ";"으로 끝나는 형태의 성명인지 확인하고, 범위 형태의 라인 넘버를 확인할 수 있다.
- [0133] 만약 타겟 문장이 "{"형태로 나타나는 성명이면, 추정부(130)는 타겟 문장의 첫 키워드를 확인할 수 있다. 예컨대 첫 키워드가 "contract", "function", "while", "do", "if" 등일 수 있다.
- [0134] 추정부(130)는 확인된 첫 키워드가 "contract"이면, 그에 해당하는 범위(range)가 있는 지 확인한 후, 해당하는 범위가 있으면 매핑 테이블 내용을 변경 또는 삭제할 수 있다. 단, 삭제의 경우에는 삭제에 대한 입력이 있어야만 가능할 수 있다. 추정부(130)는 만약 해당하는 범위가 없으면 해당 컨트랙트에 대한 범위를 생성하여 매핑 테이블을 업데이트할 수 있다. (이하에서는 제1 과정이라고 함)
- [0135] 그리고 추정부(130)는 확인된 첫 키워드가 "function"이면, 그에 해당하는 범위가 컨트랙트의 범위 안에 속하는 지 검사한 후, 해당 범위가 컨트랙트의 범위 안에 속하면 해당 함수에 대해서 제1 과정과 같이 생성, 변경 및 삭제를 수행할 수 있다. 이때 추정부(130)는 해당 범위가 컨트랙트의 범위 안에 속하지 않으면 다음 컨트랙트를 검사하고, 만약 속하는 컨트랙트가 없다면 오류인 것으로 판단할 수 있다. (이하 제2 과정이라고 함)
- [0136] 또한 추정부(130)는 확인된 첫 키워드가 "{"에 해당하는 성명, 즉 "while", "do", "if" 등인 경우, 그에 해당하는 범위가 컨트랙트와 함수의 범위 안에 속하는지 검사할 수 있다.
- [0137] 이에 추정부(130)는 해당 범위가 컨트랙트와 함수의 범위 안에 속하면 해당 성명에 대해서 제2 과정과 같이 생성, 변경 및 삭제를 수행할 수 있다. 이때 추정부(130)는 해당 범위가 컨트랙트와 함수의 범위 안에 속하지 않으면, 다음 함수 및 컨트랙트를 검사하고, 만약 모든 컨트랙트 및 함수에 대해 속하지 않는다면 오류인 것으로 판단할 수 있다. (이하 제3 과정이라고 함)
- [0138] 한편 타겟 문장이 ";"으로 끝나는 형태의 성명이면, 추정부(130)는 해당 범위가 컨트랙트, 함수 및 성명의 범위 안에 속하는지 검사하고, 제1 내지 제3 과정과 동일한 과정을 수행하여, 매핑 테이블을 업데이트하거나 오류를 판단할 수 있다.

- [0139] 즉 이상의 과정을 통해 추정부(130)가 매핑 테이블을 업데이트하여 맥락을 추정하면, 업데이트된 매핑 테이블의 라인 넘버는 파서부(150)의 입력으로 사용될 수 있다.
- [0140] 한편 파서부(150)는 추정부(130)에서 추정된 타겟 문장의 맥락에 따라 미완성된 스마트 컨트랙트를 파싱하여 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성할 수 있다.
- [0141] 파서부(150)는 상술한 종래의 "sourceUnit" 시작 규칙의 하위 규칙은 그대로 파싱할 수 있다.
- [0142] 반면 컨트랙트 중간에 입력되어 현재 "sourceUnit" 규칙에 없는 코드들의 유형을 정리하면 크게 2가지로 분류될 수 있다.
- [0143] 먼저 하나는 상술한 도 3 및 도 7에 도시된 바와 같이 "sourceUnit" 시작 규칙의 하위에는 없는 규칙이면서, "contractBodyElement" 규칙의 하위 규칙인 경우일 수 있다.
- [0144] 나머지 하나는 "sourceUnit" 시작 규칙의 하위에는 없는 규칙이면서, "contractBodyElement" 규칙의 하위 규칙의 하위 규칙, 즉 세부 규칙인 경우일 수 있다. 이러한 하위 규칙은 상술한 성명들을 의미할 수 있다.
- [0145] 따라서 이러한 경우에도 파싱을 수행할 수 있도록 하기 위해 본 실시예에 따른 파서부(150)는 이러한 규칙들을 "sourceUnit" 시작 규칙의 하위 규칙으로 올리거나, "sourceUnit" 시작 규칙 말고 새로운 시작 규칙을 만드는 방법으로 미완성된 코드를 파싱하도록 마련될 수 있다.
- [0146] 파서를 통해 AST를 생성하는 이미 널리 알려진 기술인 바, 자세한 설명은 생략하기로 한다.
- [0147] 따라서 파서부(150)는 이상의 과정을 통해 미완성된 스마트 컨트랙트의 파싱을 통해 생성된 AST를 검출부(170)로 전달할 수 있다.
- [0148] 또한 솔리디티 언어는 일반적인 프로그래밍 언어들과 마찬가지로 주석을 나타내는 방법은 다음과 같이 2가지 방식을 사용할 수 있다.
- [0149] 하나는 "/" 표시 후 주석을 작성하는 Line_Comment 방식과 "/*/" 표시 후 "/"과 "*/" 사이에 주석을 작성하는 Comment 방식일 수 있다.
- [0150] 구체적으로 도 18은 Line_Comment 방식으로 주석을 작성한 예시도이고, 도 19는 Comment 방식으로 주석을 작성한 예시도이다.
- [0151] Line_Comment 방식으로 주석을 작성하는 경우, , 도 18에 도시된 바와 같이 "/"와 같은 라인 상에 존재하는 글자를 주석으로 인식할 수 있는 반면, 다른 라인에 존재하는 경우 주석으로 인식하지 않을 수 있다.
- [0152] 한편 Comment 방식으로 주석을 작성하는 경우, 도 19에 도시된 바와 같이 "/"와 "*/" 사이에 작성되는 모든 글자를 주석으로 인식하고, 이러한 주석은 여러 줄로 작성될 수 있다.
- [0153] 이에 본 실시예에 따른 파서부(150)는 Line_Comment 방식에 따라 주석을 분석하도록 마련될 수 있으나, 꼭 이에 한정되는 것은 아니다. 이하에서는 본 실시예에 따른 파서부(150)가 Line_Comment 방식에 따라 주석을 파싱하는 경우에 대해 설명하기로 한다.
- [0154] 구체적으로 본 실시예에 따른 파서부(150)는 하나의 코드 문장이 끝날 때마다 기재되는 ";" 옆에 Line_Comment 방식으로 작성되는 주석을 파싱하여 후술할 검출부(170)가 해당 코드 라인의 값의 범주와 주석이 나타내는 범위가 다른 경우 오류를 출력하도록 할 수 있다.
- [0155] 종래의 Line_Comment 방식으로는 모든 글의 작성이 가능하지만 본 실시예에서는 파서부(150)가 주석을 파싱하기 위해 조건을 제한할 수 있다.
- [0156] 예컨대 본 실시예에 따른 입력부(110)를 통해 입력되는 주석은 ";" 옆에 작성되는 Line_Comment는 범위 표현이 가능한 사전에 마련된 6가지 기호가 포함되는 수학적 수식만 입력되도록 제한될 수 있다. 6가 기호는 ">", "<", ">=", "<=", "==", "!="로 마련될 수 있다.
- [0157] 도 20 및 도 21은 파싱 규칙을 만들 때와 같이 새롭게 주석을 파싱할 수 있는 규칙을 도식한 도면이다.
- [0158] 즉, 본 발명의 파서부(150)에서는 "expression" 같은 경우 ";"로 끝나는 솔리디티의 모든 문장이 "expression"에 포함된다.
- [0159] 도 22는 종래 파서에서의 파싱 규칙을 도식한 도면이다.

- [0160] 종래의 솔리디티 파싱 규칙은 도 22에 도시된 바와 같이 expression 뒤에 ";"이 작성되면 파싱이 종료되었으나, 본 발명의 파서부(150)는 도 2-에 도시된 바와 같이 ";" 뒤에 "Line_Comment?" 규칙을 추가하였다.
- [0161] 즉, 상술한 바와 같이 "?"때문에 Line_Comment를 파싱할 수 있는 무언가가 있을 수도 있고, 없을 수도 있는데, 있다면 도 21에서와 같이 파싱을 수행하겠다는 의미이다.
- [0162] 그리고 도 21에 도시된 "Line_Comment" 규칙을 살펴보면, 먼저 "/"라는 문자는 무조건 있어야 하고, 그 다음 규칙은 "*"때문에 사전에 마련되는 6가지의 기호(">", "<", ">=", "<=", "==", "!=")를 사이에 두고 변수나 값으로 구성되는 무언가가 0개 이상 있어야 한다는 의미이다.
- [0163] 한편 도 23은 주석을 파싱하고 오류를 나타내는 방법과 관련된 스마트 컨트랙트의 예시이다.
- [0164] 도 23의 27번째 줄에 도시된 바와 같이 "int c = a+b;"라는 코드 옆에 Line_Comment 방식의 기호(//)가 있고 그 옆에 주석으로 "c>0"가 존재한다.
- [0165] 이에 파서부(150)는 "int c = a+b; // c > 0" 라는 코드 문장 한줄을 받아들여 주석을 포함하여 전체적으로 파싱을 수행할 수 있다.
- [0167] 한편 검출부(170)는 AST에 기초하여 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출하기 위해 마련된다.
- [0168] 도 24는 본 발명의 일 실시예에 따른 검출부(170)가 사용하는 밸류 검증 함수를 설명하기 위한 도면, 도 25는 도 24의 밸류 검증 함수의 의사코드를 도시한 도면이다.
- [0169] 검출부(170)는 사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트할 수 있다.
- [0170] 여기서 상태는, 변수에 대한 범위를 의미하는 매핑변수로, 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값을 의미할 수 있다.
- [0171] 검출부(170)가 사용하는 사전에 마련되는 밸류 범위 검증 함수는, 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 정수형 변수가 포함되는 문장에 대한 상태를 업데이트할 수 있다.
- [0172] 이때 밸류 범위 검증 함수는, 업데이트된 상태와 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련될 수 있다.
- [0173] 이하에서는 검출부(170)가 밸류 범위 검증 함수를 사용하여 취약점을 분석하는 과정에 대해 구체적으로 설명하기로 한다.
- [0174] 본 실시예에 따른 밸류 범위 검증 함수는 도 24의 첫번째 줄에 도시된 바와 같이 "Command", "State", "AnConstraint"를 받아 새로운 상태를 생성하는 함수이다.
- [0175] 여기서 "Command"는 도 24의 두번째 줄에 도시된 바와 같이 솔리디티 언어 기반의 각각의 코드 문장을 4개의 문장, 즉 nonint, assignint, consint, funcint 중 하나로 요약한 것일 수 있다.
- [0176] Nonint는 해당 문장에 대한 표현식을 나타내는 정수형 변수와 관련이 없는 문장을 의미할 수 있다.
- [0177] assignint는 정수형 변수에 대한 변수 선언을 포함하는 문장으로, 실제 값을 할당하는 문장을 의미할 수 있다.
- [0178] consint는 정수형 변수가 포함된 조건문을 의미할 수 있고, funcint는 정수형 변수가 다른 함수 호출의 인자로 넘어가는 문장을 의미할 수 있다.
- [0179] 한편 "State", 즉 상태는 도 24의 세번째 줄에 도시된 바와 같이 변수(var)에 대한 범위(Range)를 의미하는 매핑변수이다.
- [0180] 그리고 이러한 범위는 도 24의 네번째 줄에 도시된 바와 같다. 즉, 솔리디티 언어의 정수 값 범위 내의 정수는 -2^{256} ~ $2^{256}-1$ 사이의 값을 의미할 수 있다. 또한 사전에 설정된 특정 값은 범위를 특정할 수 없는 경우로, any 값을 의미할 수 있다.
- [0181] 한편 "AnConstraint"는 프로그래머가 주석(annotation)으로 정의한 조건식의 집합을 의미할 수 있다. 이러한

"AnConstraint"는 주석 입력이 없을 경우에는 공집합으로 간주될 수 있다.

- [0182] 이상의 내용에 기초하여 도 25에 도시된 밸류 범위 검증 함수의 알고리즘을 설명하면, 검출부(170)는 먼저 현재 코드 문장의 요약, 즉 "Command"가 nonint 문장일 경우, 해당 문장에 대한 검증은 수행하지 않고 현재 상태를 리턴할 수 있다.
- [0183] 그리고 검출부(170)는 도 25의 6 내지 13 라인에 도시된 바와 같이, 현재 코드 문장의 종류에 따라 상태를 새로운 상태로 업데이트할 수 있다. 현재 코드 문장의 "Command"가 assignint 또는 consint일 경우 문장에 따라 상태, 즉 범위가 새로 업데이트되지만, 문장의 종류가 funcint인 경우 반환되는 변수 값의 범위를 특정할 수 없으므로 해당 변수의 범위는 'any'값이 된다.
- [0184] 또한 검출부(170)는 도 25의 7 내지 11 라인에 도시된 바와 같이, 프로그래머가 작성한 주석이 존재할 경우, 현재 코드 문장에 의해 새로 변경된 상태와 주석으로 입력한 범위에 대해 만족하는 입력의 존재 여부를 SMT Solver를 통해 판단할 수 있다.
- [0185] 만약 만족하는 입력이 없을 경우 검출부(170)는 실제 코드, 즉 타겟 문장이 프로그래머의 의도와 다르다고 판단하여 도 25의 11 라인에 도시된 바와 같이 오류를 반환할 수 있다.
- [0186] 이를 위해 검출부(170)는, 엔진 형태의 밸류 검증 함수와 SMT Solver를 포함하도록 마련될 수 있다. 이를 통해 검출부(170)는 편집기와 컴파일러(Compiler) 사이에 밸류 검증 함수 엔진과 밸류 검증 함수 알고리즘에서 사용하는 SMT Solver를 통해 모든 정수형 관련 취약사례에 대응할 수 있다.
- [0188] 이하에서는 도 26을 참조하여 검출부(170)가 정수 값의 범위를 검증하는 과정에서 대해 보다 구체적으로 설명하기로 한다.
- [0189] 도 26은 솔리디티 범위(interval) 도메인 상에서의 요약 해석을 위한 스마트 컨트랙트 예시도이다.
- [0190] 요약 해석은, 종래 기술로 코딩된 프로그램의 실행 의미를 요약이 가능한 특정 도메인의 값으로 요약한 후, 요약된 도메인으로 프로그램을 해석하겠다는 것을 의미한다.
- [0191] 그리고 도 26에 도시된 예시 코드 상에서 취약점 검증에 직접적으로 영향을 미치는 스마트 컨트랙트의 모든 코드 문장은 assign문, if문, while문 중 하나로 바꿀 수 있으므로, assign문, if문, while문을 중심으로 설명하기로 한다.
- [0192] 참고로 이러한 코드 문장의 변수의 범위는 [최솟값, 최댓값]으로 표현될 수 있다.
- [0193] 먼저 정수형 변수에 대한 실제 값을 할당하는 assign문은 도 26에서 line 5, 9, 13, 15, 20, 23번일 수 있다.
- [0194] 이러한 assign문에서 왼쪽 변수의 범위는 오른쪽에 있는 표현식의 범위 값과 같다.
- [0195] 예컨대 도 26의 line 9에서 localResult의 범위는 오른쪽에 있는 표현식이 단순히 상수 값이기 때문에 최솟값, 최댓값이 모두 동일한 [0.0]으로 표현할 수 있다.
- [0196] 또 다른 예로 도 26의 line 23번의 변수 result의 범위는 변수 localResult의 범위와 같고, localResult의 범위는 요약 해석을 통해 line 9-21까지 분석한 결과로, 결론적으로는 [4, 5]가 된다.
- [0197] 도 26에서 result의 범위인 [4, 5]가 생성되는 과정을 다음과 같다.
- [0198] 한편 정수형 변수에 대한 조건문인 if문은 도 26의 line 12-16이다.
- [0199] 이러한 if문 범위 분석의 핵심은 if문 전의 범위를 기반으로 각 변수의 if block 시작 범위 / else block 시작 범위 / if문 후의 범위를 따져보는 것이다.
- [0200] 즉, 각 변수의 if block의 시작 범위는 if문 전의 범위와 if 조건식의 범위의 교집합일 수 있다.
- [0201] 예컨대 input이라는 변수의 초기 범위(line 8)는 input의 타입이 uint (부호 없는 정수)이기 때문에 [0, 무한대]이다.
- [0202] 이후, 도 26의 line 13이 속하는 if block의 시작 범위는 [0, 무한대]와 if문 안의 조건인 '>100'의 교집합인 [101, 무한대]가 된다.

- [0203] 그리고 각 변수의 else block의 시작 범위는 if문 전의 범위와 else 조건식 범위의 교집합일 수 있다.
- [0204] 예컨대 else 조건식은 if 조건식의 반대로, '>100'의 반대 조건인 '<=100'이 된다. 따라서 else 분기에 대한 변수 input의 범위는 if문 전의 input의 범위 [0,무한대]와 else 조건식의 범위 '<=100'의 교집합인 [0,100]이 된다.
- [0205] 그리고 if문 후의 범위는, 각 변수의 if block의 끝 범위와 else block의 끝 범위의 합집합이다.
- [0206] 예컨대 input 변수의 경우에는 if block, else block 내에서 값이 변경되거나 하는 코드가 없으므로 이상에서 설명한 것처럼 if block에선 [101, 무한대]였고, else block에서는 [0,100]이므로, 이 둘의 합집합은 다시 [0,무한대]가 된다.
- [0207] localhost의 변수의 경우에는 먼저 도 26의 line 9에서 상수 값 0을 할당 받았으므로 범위는 [0,0]이다.
- [0208] 그 다음 if block 안에서 line 13에서 값이 5로 바뀌기 때문에 그 범위는 [5,5]가 되며, else block 안에서는 line 15에서 값이 3으로 바뀌기 때문에 그 범위는 [3,3]이 된다.
- [0209] 따라서 if문 후의 localhost의 값의 범위는 [3,3], [5,5]의 합집합인 [3,5]가 된다.
- [0210] 한편 정수형 변수에 대한 반복문인 while문은 도 26의 line 19-21이다.
- [0211] 이러한 while문 분석의 핵심은 반복적으로 분석을 수행하면서 특정 값으로 분석 결과가 수렴되는 가를 확인하는 것이다. 수렴의 여부는 이전 반복의 상태와 다음 반복의 상태가 같은가, 즉 특정 지점으로 수렴되는가(fixpoint algorithm)로 판단할 수 있다.
- [0212] 검출부(170)는 if문 분석을 통해 변수 input의 범위는 [0,무한대]이고, 변수 localResult의 범위는 [3,5]라는 결과를 획득하였다.
- [0213] 이에 while문 조건식은 localResult가 4보다 작을 때에만 수행되기 때문에 localResult가 [3, 3]인 경우와 [4, 5]인 경우로 분리되며, 검출부(170)는 추후 [3, 3], [4, 5]가 while문을 돌면서 수행되는 결과와 합집합을 while문의 결과 값으로 획득할 수 있다.
- [0214] 이때 [4, 5]의 경우에는 while문 조건에 부합하지 않으므로 그대로 [4, 5]가 되며, 이 [4, 5] 값으로 한 번 더 수행했을 때 그대로 [4, 5]기 때문에 이전 상태와 한 번 더 수행한 상태의 값이 같으므로 고정점이 되므로 검출부(170)는 검증을 종료할 수 있다.
- [0215] 반면 [3,3]인 경우, 검출부(170)는 while문 안으로 들어와 "localResult +=1;" 문장을 검증하게 된다. 이 문장은 localResult = localResult + 1;과 의미적으로 완전히 동일하며 이전 값의 범위에 1을 더하는 것이므로 [4, 4]가 된다. 이후 검출부(170)는 while문을 반복하면 localResult가 4보다 작거나 같을 때만 while문 안이 수행된다. 하지만 현재 변경된 상태 값이 [4, 4]이므로 검출부(170)는 while문을 빠져나오게 되고, [4, 4]를 반복 수행했을 때에도 그 값이 동일하므로 고정점이 되므로 검출부(170)는 검증을 종료할 수 있다.
- [0216] 따라서 이상의 과정을 통해 최종적으로 변수 localResult의 while문을 통한 수행 결과는 [3, 3]의 수행결과인 [4, 4]와 [4, 5]의 수행 결과인 [4, 5]의 합집합은 [4, 5]가 된다.
- [0217] 이에 검출부(170)는 변수 result에 범위를 localResult 변수의 최종 범위인 [4, 5]를 할당하고 분석을 마치게 된다.
- [0218] 한편 검출부(170)는 파싱 결과에 따라 주석이 존재하는 경우, 이상의 과정을 통해 계산된 각 변수의 범위와 주석을 동시에 만족하는지 검증할 수 있다.
- [0219] 상술한 도 23을 참고하여 설명하면, 검출부(170)는 주석 전에 위치하는"int c = a+b;"라는 문장을 상술한 요약 해석 기법을 통해 각 변수의 범위를 계산할 수 있다. 결론적으로 도 23의 27번째 줄을 통해 계산된 각 변수의 범위는 a는 값 1만 가질 수 있고, b는 값 3만 가질 수 있으며, c는 값 4만 가질 수 있다.
- [0220] 그 다음 검출부(170)는 파서부(150)를 통해 파싱 된 주석을 통해 변수 c의 값이 주석과 같이 0보다 큰지를 검증할 수 있다.
- [0221] 따라서 검출부(170)는 계산된 c의 값의 범위는 4만 가질 수 있는데 이는 주석과 같이 0보다 크므로 오류를 반환하지 않는다.
- [0222] 반면 주석이 예컨대">5"로 작성된 경우라면 검출부(170)는 c 값의 범위와 주석이 일치하지 않으므로 오류를 반

환할 수 있다.

- [0223] 한편 출력부(190)는 미완성된 스마트 컨트랙트, 즉 프로그래머가 작성중인 코드를 실시간으로 디스플레이하되, 오류를 반환한 문장을 취약점이 존재하는 문장으로 식별 가능하도록 표시할 수 있다.
- [0224] 도 27은 본 실시예에 따른 출력부(190)가 오류를 반환한 문장을 취약점이 존재하는 문장으로 식별한 결과의 예시도이다.
- [0225] 도 27에 도시된 바와 같이 출력부(190)는 검출부(170)로부터 오류를 반환한 문장에 대해서는 오류를 의미하는 빨간 줄을 포함시켜 출력할 수 있다.
- [0226] 즉, 도 27은 취약 사례인 도 1에 도시된 buyItem 함수를 본 장치(100)에 입력한 경우에 출력된 결과이다.
- [0227] 도 27에 도시된 바와 같이 본 실시예에 따른 장치(100)에서는 취약점이 발생한 지점인 도 27의 line 15에 대해 변수 "_quantity"의 값의 범위가 특정되지 않아 그 범위가 any 이지만, 프로그래머가 주석으로 작성한 조건식은 "_quantity>0"이므로 오류가 발생되어 해당 문장에 식별가능한 표시를 출력하였다.
- [0228] 그리고 임의로 작성된 "vi_test" 함수의 경우 관심 지점인 도 27의 line 26에 대해 a값과 b값의 범위에 따라 결정된 변수 c의 범위가 주석으로 작성한 조건식 범위 안에 존재하므로 프로그래머의 의도대로 작성되어 오류를 나타내지 않았다.
- [0229] 이와 같이 본 실시예에 따른 장치(100)는 완성된 코드가 아닌, 미완성된 스마트 컨트랙트를 개발하는 과정에서 인터프리터 방식으로 코드를 분석하고 문제를 평가하여 수정이 필요할 경우 소모되는 비용을 감소시킬 수 있다.
- [0230] 또한 본 실시예에 따른 장치(100)는 모든 코드에 대해 일반적으로 적용할 수 있는 분석 방식을 구현하여 발견되지 않은 새로운 형태의 취약점을 검증할 수 있으며, 해당 코드 라인에 대한 프로그래머의 개발 의도를 주석 형태로 입력받아 분석할 수 있으므로, 프로그래머의 개발 의도를 파악한 분석이 가능하다.
- [0232] 한편 도 28은 본 발명의 일 실시예에 따른 취약점 분석 방법을 설명하기 위한 흐름도로써, 본 발명의 일 실시예에 따른 취약점 분석 방법은 도 2에 도시된 취약점 분석 장치(100)와 실질적으로 동일한 구성 상에서 진행되므로, 도 2의 취약점 분석 장치(100)와 동일한 구성요소에 대해 동일한 도면 부호를 부여하고, 반복되는 설명은 생략하기로 한다.
- [0233] 본 실시예에 따른 블록체인을 위한 스마트 컨트랙트(Smart Contract)의 취약점 분석 방법은, 타겟 문장을 입력하는 단계(S110), 타겟 문장의 맥락을 추정하는 단계(S130), AST를 생성하는 단계(S150) 및 취약점이 존재하는 문장을 검출하는 단계(S170)를 포함한다.
- [0234] 타겟 문장을 입력하는 단계(S110)에서는 입력부(110)가 미완성된 스마트 컨트랙트를 구성하는 소스 코드의 일부를 타겟 문장으로서 입력받을 수 있다.
- [0235] 여기서 스마트 컨트랙트는, 솔리디티 언어 기반의 스마트 컨트랙트일 수 있으나, 꼭 이에 한정되는 것은 아니다.
- [0236] 타겟 문장의 맥락을 추정하는 단계(S130)에서는 추정부(130)가, 사전에 마련되는 매핑 테이블에 기초하여 소스 코드의 전체 맥락 중 타겟 문장의 맥락을 추정할 수 있다.
- [0237] 타겟 문장의 맥락을 추정하는 단계(S130)에서, 타겟 문장의 맥락을 추정하는 것은, 사전에 마련되는 매핑 테이블을 이용하여 상기 타겟 문장의 형태에 대응되는 라인 넘버에 따라 상기 맥락을 추정하는 것일 수 있다.
- [0238] 그리고 사전에 마련되는 매핑 테이블은, 스마트 컨트랙트를 구성하는 적어도 하나 이상의 하위 컨트랙트, 하위 컨트랙트를 구성하는 적어도 하나 이상의 함수 및 함수를 구성하는 적어도 하나 이상의 명령문인 성명(statement) 각각에 대한 라인 넘버를 키(key) 값으로 갖고, 라인 넘버에 대응하는 맥락을 밸류(value) 값으로 갖는 테이블일 수 있다.
- [0239] AST를 생성하는 단계(S150)에서는 파서부(150)가, 추정된 맥락에 따라 미완성된 스마트 컨트랙트를 파싱하여 미완성된 스마트 컨트랙트에 대한 AST(Abstract Syntax Tree)를 생성할 수 있다.
- [0240] 취약점이 존재하는 문장을 검출하는 단계(S170)에서는 검출부(170)가, AST에 기초하여 소스 코드를 구성하는 각 문장을 검증하여 취약점이 존재하는 문장을 검출할 수 있다.

- [0241] 취약점이 존재하는 문장을 검출하는 단계(S170)에서는 검출부(170)가, 사전에 마련되는 밸류 범위 검증(Value Range Verification) 함수를 이용하여 각 문장 내에 정수형 변수가 포함되는지를 검증하고, 검증이 완료된 문장에 대한 상태(state)를 업데이트할 수 있다.
- [0242] 여기서 상태는, 변수에 대한 범위를 의미하는 매핑변수로, 솔리디티 언어의 정수 값 범위 내의 정수 또는 사전에 설정된 특정 값일 수 있다.
- [0243] 그리고 사전에 마련되는 밸류 범위 검증 함수는, 정수형 변수가 포함되는 문장에 대해서는 주석이 존재하는지 추가 확인하며, 정수형 변수가 포함되는 문장에 대한 상기 상태를 업데이트하되, 업데이트된 상태와 주석을 동시에 만족하는 입력이 미존재하는 경우, 오류를 반환하도록 마련될 수 있다.
- [0244] 또한 본 실시예에 따른 취약점 분석 방법은, 출력부(190)가 미완성된 스마트 컨트랙트를 디스플레이하되, 오류를 반환한 문장을 취약점이 존재하는 문장으로 식별 가능하도록 표시하는 미완성된 스마트 컨트랙트를 디스플레이하는 단계를 더 포함할 수 있다.
- [0245] 이와 같은 본 발명의 취약점 분석 방법은 다양한 컴퓨터 구성요소를 통하여 수행될 수 있는 프로그램 명령어의 형태로 구현되어 컴퓨터 판독 가능한 기록 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능한 기록 매체는 프로그램 명령어, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다.
- [0246] 상기 컴퓨터 판독 가능한 기록 매체에 기록되는 프로그램 명령어는 본 발명을 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 분야의 당업자에게 공지되어 사용 가능한 것일 수도 있다.
- [0247] 컴퓨터 판독 가능한 기록 매체의 예에는, 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체, CD-ROM, DVD 와 같은 광기록 매체, 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 ROM, RAM, 플래시 메모리 등과 같은 프로그램 명령어를 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다.
- [0248] 프로그램 명령어의 예에는, 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드도 포함된다. 상기 하드웨어 장치는 본 발명에 따른 처리를 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.
- [0249] 이상에서는 본 발명의 다양한 실시예에 대하여 도시하고 설명하였지만, 본 발명은 상술한 특정의 실시예에 한정되지 아니하며, 청구범위에서 청구하는 본 발명의 요지를 벗어남이 없이 당해 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 의해 다양한 변형실시가 가능한 것은 물론이고, 이러한 변형실시들은 본 발명의 기술적 사상이나 전망으로부터 개별적으로 이해되어져서는 안될 것이다.

부호의 설명

- [0250] 100 : 취약점 분석 장치 110 : 입력부
130 : 추정부 150 : 파서부
170 : 검출부 190 : 출력부

도면

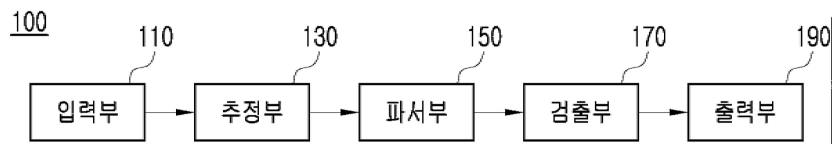
도면1

```

26 function buyItem(
27     address _nftAddress,
28     uint256 _tokenId,
29     address _owner,
30     uint256 _quantity,
31     uint256 _pricePerItem
32 ) external {
33     (, uint256 pricePerItem,) = marketplace.listings(_nftAddress, _tokenId, _owner);
34
35     require(pricePerItem == _pricePerItem, "pricePerItem changed!");
36
37     uint256 totalPrice = _pricePerItem * _quantity;
38     IERC20(marketplace.paymentToken()).safeTransferFrom(msg.sender, address(this), totalPrice);
39     IERC20(marketplace.paymentToken()).safeApprove(address(marketplace), totalPrice);
40
41     marketplace.buyItem(_nftAddress, _tokenId, _owner, _quantity);
42
43     if (IERC165(_nftAddress).supportsInterface(INTERFACE_ID_ERC721)) {
44         IERC721(_nftAddress).safeTransferFrom(address(this), msg.sender, _tokenId);
45     } else {
46         IERC1155(_nftAddress).safeTransferFrom(address(this), msg.sender, _tokenId, _quantity, bytes(""));
47     }
48 }

```

도면2



도면3

```
sourceUnit
: (
    pragmaDirective
  | importDirective
  | usingDirective
  | contractDefinition
  | interfaceDefinition
  | libraryDefinition
  | functionDefinition
  | constantVariableDeclaration
  | structDefinition
  | enumDefinition
  | userDefinedValueTypeDefinition
  | errorDefinition
)* EOF ;
```

도면4

```
1 contract test {
2   |   uint public var1 = 0;
3 }
```

도면5

```
contractDefinition
: 'abstract'? 'contract' identifier
( 'is' inheritanceSpecifier (',' inheritanceSpecifier)* )?
'{' contractBodyElement* '}' ;
```

도면6

```
identifier
: Identifier ;
```

도면7

```
contractBodyElement
: constructorDefinition
| functionDefinition
| modifierDefinition
| fallbackFunctionDefinition
| receiveFunctionDefinition
| structDefinition
| enumDefinition
| userDefinedValueTypeDefinition
| stateVariableDeclaration
| eventDefinition
| errorDefinition
| usingDirective ;
```

도면8

```
stateVariableDeclaration
: typeName
( PublicKeyword | InternalKeyword | PrivateKeyword | ConstantKeyword | ImmutableKeyword | overrideSpecifier )*
identifier ('=' expression)? ';' ;
```

도면9

```
1 contract test {
2
3 }
```

도면10

```
2 uint public var1=0;
```

도면11

```
pragma solidity ^0.8.0;

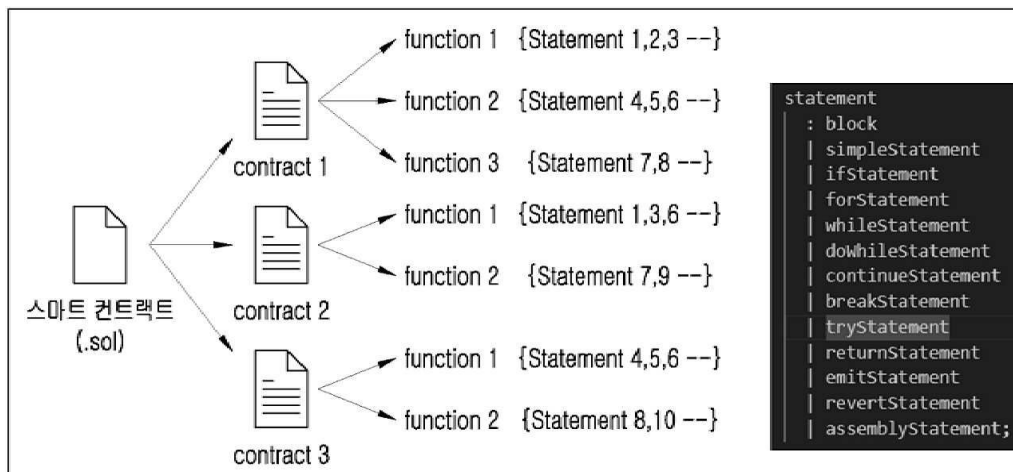
contract ExampleContract {
    // 상태 변수 선언
    int public stateVariable;

    function exampleFunction() public {
        // 로컬 변수 선언
        // 이 변수는 함수 호출 시 생성되었다가 함수 호출이 끝나면 사라집니다.
        int localVariable = 20;
    }
}
```

도면12

Contract Block Line Number	Function Block Line Number	Statement Line Number	Code Line number	Context
1-100	0-10	---		--
				--
				--
	10-30	11-14	11	--
			14	--
		16		--
---	---			--
---	---			--

도면13



도면14

```

contract StatementExample {
    // 1. 변수 선언 문 (variableDeclarationStatement)
    uint public count = 0;

    // 2. 표현식 문 (expressionStatement)
    function anotherFunction() public {
        updateData(5); // 표현식 문 (함수 호출)
    }

    function updateData(uint256 newValue) internal {
        count = newValue; // 표현식 문 (변수 할당)
    }

    // 3. if 조건문 (ifStatement)
    function increaseCount(uint amount) public {
        if (amount > 0) {
            count += amount; // 표현식 문
        }
    }

    // 4. for 문 (forStatement)
    function resetCount() public {
        for (uint i = 0; i < count; i++) {
            count -= 1;
        }
    }

    // 5. while 문
    function decreaseCount(uint amount) public {
        while (amount > 0) {
            count -= 1;
            amount -= 1;
        }
    }

    // 6. do-while 문
    function doWhileExample() public {
        uint i = 0;
        do {
            i++;
        } while (i < 5);
    }
}

```

도면15

```
// 7. continue 문
function skipEvenNumbers() public {
    for (uint i = 0; i < 10; i++) {
        if (i % 2 == 0) {
            continue;
        }
        count += i;
    }
}

// 8. break 문
function findFirstEvenNumber() public {
    for (uint i = 0; i < 10; i++) {
        if (i % 2 == 0) {
            count = i;
            break;
        }
    }
}

// 9. try 문
function tryExample() public {
    try this.resetCount() {
        // 성공적인 경우
    } catch {
        // 오류 발생시
    }
}

// 10. return 문
function getCount() public view returns(uint) {
    return count;
}

// 11. emit 문 (이벤트 발생)
event CountChanged(uint newCount);
function emitEvent() public {
    emit CountChanged(count);
}

// 12. revert 문
function revertExample() public {
    if (count == 0) {
        revert("Count is zero");
    }
}
```

도면16

```
// 13. 어셈블리 문 (인라인 어셈블리)
function assemblyExample() public {
    assembly {
        // 인라인 어셈블리 코드
    }
}
```

도면17

```
# 중첩된 딕셔너리를 사용한 계층적 자료구조
smart_contract_structure = {
    "Contract": {
        "range": (1, 100),
        "context": "ContractLevel",
        "Functions": [
            {
                "name": "Function1",
                "range": (10, 20),
                "context": "FunctionLevel",
                "Statements": [
                    {"range": (12, 12), "context": "Assignment", "code": "uint localk"},
                    {"range": (14, 16), "context": "IfStatement", "code": "if (input"}
                ]
            },
            {
                "name": "Function2",
                "range": (21, 30),
                "context": "FunctionLevel",
                "Statements": [
                    {"range": (22, 22), "context": "Loop", "code": "while (localResul"}
                ]
            }
        ]
    }
}
```

도면18

```
// ensure that we do not overshoot the min/max tick, as the tick bitmap is not aware of these bounds
if (step.tickNext < TickMath.MIN_TICK) {
    step.tickNext = TickMath.MIN_TICK;
} else if (step.tickNext > TickMath.MAX_TICK) {
    step.tickNext = TickMath.MAX_TICK;
}
```

도면19

```
function getResult() public view returns(uint){
    // This is a comment. It is similar to comments in C++

    /*
        * This is a multi-line comment in solidity
        * It is very similar to comments in C Programming
    */
    uint a = 1;
    uint b = 2;
    uint result = a + b;
    return result;
}
```

도면20

```
interactiveExpressionStatement
: expression ';' lineComment? ;
```

도면21

```
lineComment
: '//' (expression ('<' | '>' | '<=' | '>=') expression
| expression ('==' | '!=') expression)*;
```

도면22

```
expressionStatement
: expression ';' ;
```

도면23

```
19  function vi_test() {
20      a:[1,1]
21      int a = 1;
22      a:[1,1], b:[3,3]
23      int b = 3;
24
25      if(a>=1) {
26          a:[1,1], b:[3,3], c:[4,4]
27          int c = a + b; // c > 0
28      }
```

도면24

$$\begin{aligned} VRV &= (Command, State, AnConstraint) \rightarrow State \\ Command &= nonint \mid assignint \mid consint \mid funcint \\ State &= var \rightarrow Range \\ Range &= \{any\} \cup [[m, M] \mid m, M \in \mathbb{Z} \cup \{-2^{256}, 2^{256} - 1\} \wedge m \leq M \\ AnConstraint &\in P(Constraint) \\ Constraint &= x \geq a \mid x \leq a \mid x > a \mid x < a \mid x == a \end{aligned}$$

도면25

```

1. def value_ran_verif(Command, State, AnCons) :
2.     global solver # SMT solver
3.
4.     if Command is nonint :
5.         return State
6.     else :
7.         if AnCons is not None :
8.             State = update_state(Command, State)
9.             result = solver(State, AnCons)
10.            if result is unsat :
11.                return StateError
12.            else :
13.                State = update_state(Command, State)
14.
15.    return State

```

도면26

```

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.0;
3.
4. contract FixedIntervalAnalysisExample {
5.     uint public result = 0;
6.
7.     // 함수: 입력에 관계없이 localResult의 최종 값이 조건에 따라 결정됨
8.     function fixedResultFunction(uint input) public {
9.         uint localResult = 0; // 초기 값은 0
10.
11.        // 조건문: input의 값에 따라 localResult를 조정
12.        if (input > 100) {
13.            localResult = 5; // input이 100보다 크면 localResult를 5로
14.        } else {
15.            localResult = 3; // 그렇지 않으면 localResult를 3으로 설정
16.        }
17.
18.        // 반복문: localResult가 4보다 작은 동안 localResult를 1씩 증가
19.        while (localResult < 4) {
20.            localResult += 1; // localResult를 1씩 증가
21.        }
22.
23.        result = localResult; // 최종 결과 할당
24.    }
25. }

```

도면27

```

1  contract test {
2
3      function buyItem(
4          address _nftAddress,
5          uint256 _tokenId,
6          address _owner,
7          uint256 _quantity
8          uint256 _pricePerItem) {
9          pricePerItem : any
10         (, uint256 pricePerItem,) = marketplace.listings(_nftAddress, _tokenId, _owner);
11         pricePerItem : any, _pricePerItem : any
12         require(pricePerItem == _pricePerItem,
13             "pricePerItem changed!");
14         pricePerItem : any, _quantity : any, totalPrice : any
15         uint256 totalPrice = _pricePerItem * _quantity; // quantity > 0
16     }
17
18     function vi_test() {
19         a : [1,1]
20         int a = 1;
21         a : [1,1], b : [3,3]
22         int b = 3;
23
24         if(a>=1) {
25             a : [1,1], b : [3,3], c : [4,4]
26             int c = a + b; // c > 0
27         }
28     }
29 }
30

```

도면28

