(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년03월24일
(11) 등록번호 10-2378377
(24) 등록일자 2022년03월21일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01)
(52) CPC특허분류
G06F 11/3696 (2013.01)
G06F 11/3684 (2013.01)
(21) 출원번호 10-2020-0151899
(22) 출원일자 2020년11월13일
심사청구일자 2020년11월13일
(56) 선행기술조사문헌
SUNBEOM SO 등, 'VERISMART: A Highly Precise Safety Verifier for Ethereum Smart Contracts', 2020 IEEE Symposium on Security and Privacy (SP), (2020.5.18.)
US20200128043 A1
KR1020200116011 A
JP2012099111 A

(73) 특허권자
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
오학주
서울특별시 강남구 압구정로 151, 122동 1002호
소순범
서울특별시 성북구 개운사길 75-30, 203호(안암동5가)
홍성준
서울특별시 종로구 난계로29가길 19, 1103호
(74) 대리인
김홍석

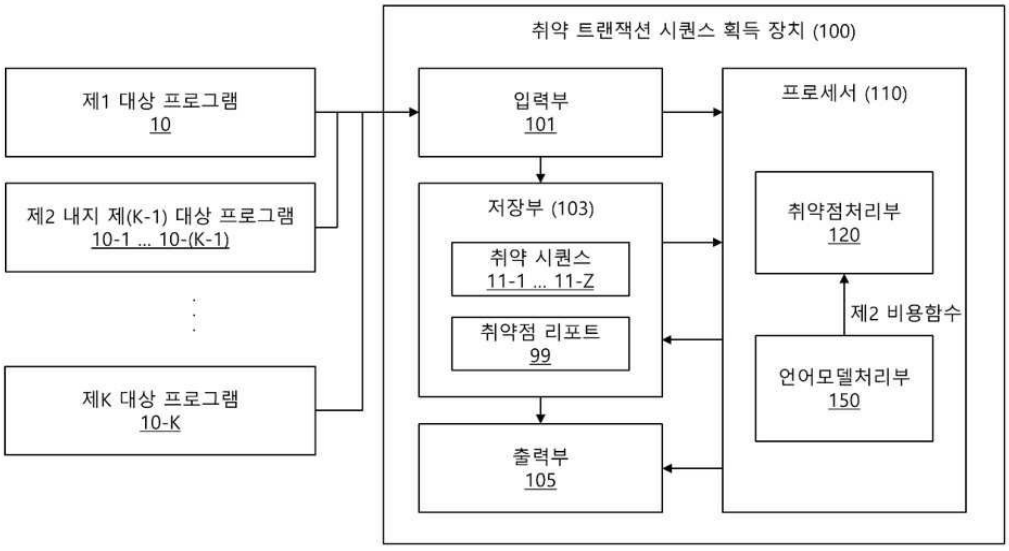
전체 청구항 수 : 총 14 항 심사관 : 장지혜

(54) 발명의 명칭 **스마트 컨트랙트 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법**

(57) 요 약

취약 트랜잭션 시퀀스 획득 장치 및 방법에 관한 것으로, 취약 트랜잭션 시퀀스 획득 장치는, 적어도 하나의 프로그램을 일시적 또는 비일시적으로 저장하는 저장부 및 상기 적어도 하나의 프로그램을 수신하고, 비용 함수를 이용하여, 적어도 하나의 프로그램 내에서의 취약 트랜잭션 시퀀스 후보를 선정하고, 상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하고, 상기 검증 조건에 대한 검증 결과, 취약 트랜잭션 시퀀스가 발견되지 않은 경우 검증 조건의 만족 여부를 확인하고, 검증 조건이 만족되면 상기 취약 트랜잭션 시퀀스 후보를 취약 트랜잭션 시퀀스로 결정하는 프로세서를 포함할 수 있다.

대 표 도 - 도1



(52) CPC특허분류
G06F 11/3688 (2013.01)
G06F 11/3692 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711116890
과제번호 2020-0-01337-001
부처명 과학기술정보통신부
과제관리(전문)기관명 정보통신기획평가원
연구사업명 SW컴퓨팅산업원천기술개발(R&D)
연구과제명 (SW 스타랩) 실용화 수준의 고성능 소프트웨어 오류 자동 수정 기술 개발
기 여 율 1/3
과제수행기관명 고려대학교산학협력단
연구기간 2020.04.01 ~ 2020.12.31
이 발명을 지원한 국가연구개발사업
과제고유번호 1711116238
과제번호 2019-0-01697-002
부처명 과학기술정보통신부
과제관리(전문)기관명 정보통신기획평가원
연구사업명 정보보호핵심원천기술개발(R&D)
연구과제명 블록체인 플랫폼 보안취약점 자동분석 기술 개발
기 여 율 1/3
과제수행기관명 고려대학교산학협력단
연구기간 2020.03.01 ~ 2020.12.31
이 발명을 지원한 국가연구개발사업
과제고유번호 1711102835
과제번호 2019-0-00099-002
부처명 과학기술정보통신부
과제관리(전문)기관명 정보통신기획평가원
연구사업명 블록체인융합기술개발(R&D)
연구과제명 스마트 컨트랙트 정형명세 블록체인 핵심 기술
기 여 율 1/3
과제수행기관명 고려대학교산학협력단
연구기간 2020.01.01 ~ 2020.12.31

명 세 서

청구범위

청구항 1

비용 함수를 이용하여, 적어도 하나의 프로그램 내에서의 취약 트랜잭션 시퀀스 후보를 선정하는 단계;

상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하는 단계; 및

상기 검증 조건에 대한 검증 결과, 취약 트랜잭션 시퀀스가 발견되지 않은 경우 검증 조건의 만족 여부를 확인하고, 검증 조건이 만족되면 상기 취약 트랜잭션 시퀀스 후보를 취약 트랜잭션 시퀀스로 결정하는 단계를 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 2

제1항에 있어서,

상기 비용함수를 획득하는 단계를 더 포함하되,

상기 비용함수를 획득하는 단계는,

취약 트랜잭션 시퀀스 집합을 획득하는 단계;

상기 취약 트랜잭션 시퀀스 집합 내의 취약 트랜잭션 시퀀스를 추상화하는 단계; 및

추상화된 취약 트랜잭션 시퀀스 및 언어 모델을 이용하여 상기 비용함수를 결정하는 단계;를 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 3

제1항에 있어서,

상기 검증 조건이 획득되면, 상기 검증 조건을 간략화하는 단계를 더 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 4

제3항에 있어서,

상기 검증 조건을 간략화하는 단계는,

검증과 무관한 논리식들을 검증 조건에서 제외하는 단계; 및

수량자 제거 최적화를 수행하는 단계 중 적어도 하나를 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 5

제1항에 있어서,

상기 적어도 하나의 프로그램 내의 루프를 언롤링하거나 또는 상기 적어도 하나의 프로그램 내의 함수의 호출 지점마다 호출될 함수를 인라인하는 단계를 더 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 6

제1항에 있어서,

상기 적어도 하나의 프로그램은, 스마트 컨트랙트를 포함하는 취약 트랜잭션 시퀀스 획득 방법.

청구항 7

제1항에 있어서,

상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하는 단계는,

상기 트랜잭션 시퀀스 후보에 대해 검증 조건 생성 함수를 이용하여 검증 조건을 획득하는 단계;를 포함하되,

상기 검증 조건 생성 함수는, 트랜잭션에 대한 기호 실행기를 기반으로 하는 트랜잭션 시퀀스에 대한 기호 실행기를 이용하는 것인 취약 트랜잭션 시퀀스 획득 방법.

청구항 8

적어도 하나의 프로그램을 일시적 또는 비일시적으로 저장하는 저장부; 및

상기 적어도 하나의 프로그램을 수신하고, 비용 함수를 이용하여, 적어도 하나의 프로그램 내에서의 취약 트랜잭션 시퀀스 후보를 선정하고, 상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하고, 상기 검증 조건에 대한 검증 결과, 취약 트랜잭션 시퀀스가 발견되지 않은 경우, 검증 조건의 만족 여부를 확인하고, 검증 조건이 만족되면 상기 취약 트랜잭션 시퀀스 후보를 취약 트랜잭션 시퀀스로 결정하는 프로세서;를 포함하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 9

제8항에 있어서,

상기 프로세서는, 취약 트랜잭션 시퀀스 집합을 획득하고, 상기 취약 트랜잭션 시퀀스 집합 내의 취약 트랜잭션 시퀀스를 추상화하고, 추상화된 취약 트랜잭션 시퀀스 및 언어 모델을 이용하여 상기 비용함수를 결정하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 10

제8항에 있어서,

상기 프로세서는, 상기 검증 조건이 획득되면, 상기 검증 조건의 간략화를 수행하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 11

제10항에 있어서,

상기 프로세서는, 검증과 무관한 논리식들의 검증 조건 제외 및 수량자 제거 최적화 중 적어도 하나를 수행함으로써, 상기 검증 조건을 간략화를 수행하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 12

제8항에 있어서,

상기 프로세서는, 상기 적어도 하나의 프로그램 내의 루프를 언롤링하거나 또는 상기 적어도 하나의 프로그램 내의 함수의 호출 지점마다 호출될 함수를 인라인하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 13

제8항에 있어서,

상기 적어도 하나의 프로그램은, 스마트 컨트랙트를 포함하는 취약 트랜잭션 시퀀스 획득 장치.

청구항 14

제8항에 있어서,

상기 프로세서는, 상기 트랜잭션 시퀀스 후보에 대해 검증 조건 생성 함수를 이용하여 검증 조건을 획득하되, 상기 검증 조건 생성 함수는, 트랜잭션에 대한 기호 실행기를 기반으로 하는 트랜잭션 시퀀스에 대한 기호 실행기를 포함하는 취약 트랜잭션 시퀀스 획득 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] 스마트 컨트랙트(Smart Contract, 스마트 계약, 지능형 계약 등으로 표현 가능함)는 소정의 프로그래밍 언어로 작성된 디지털 계약서로, 거래 등이 일정한 조건을 만족시키면 거래 당사자 간에 자동으로 거래가 체결되도록 마련된 소프트웨어 등을 의미한다. 스마트 컨트랙트를 이용하면, 조건의 만족 여부에 따라 자동으로 계약이 성립될 뿐만 아니라, 블록체인 등의 분산형 데이터 저장 기술을 이용하기 때문에 중개인이나 공증인 등과 같이 계약의 신뢰성을 위한 제3자를 필요로 하지 않아, 종래의 일반적인 계약에 비해 시간 및 비용을 획기적으로 절감할 수 있는 장점이 있다. 이에 따라 스마트 컨트랙트 기술은 금융이나 유통 분야 등 다양한 분야에서 각광을 받고 있다. 그러나, 이러한 스마트 컨트랙트는 계약의 체결에 따라 개인이나 회사 등의 권리나 재산과 밀접한 관계를 가지므로, 스마트 컨트랙트 내에 취약점이 존재하는지 여부는 매우 중요하다. 만약 스마트 컨트랙트 내에 확인되지 못한 취약점(예를 들어, 정수 오버플로우/언더플로우 등)이 존재하고, 제3자가 무단으로 이를 악용하는 경우, 개인 또는 회사에 막대한 재산 피해를 야기할 수 있다. 특히 스마트 컨트랙트는 배포가 개시되면, 그 구체적인 내용(코드)의 수정이 거의 불가능하므로, 취약점이나 알려지지 않은 버그 등으로부터 문제가 야기되는 것을 방지하기 위해서는, 배포 전에 반드시 스마트 컨트랙트 내의 취약점을 확인, 검증 및 수정할 필요성이 있다. 이런 이유로 스마트 컨트랙트에 대한 코드 감사는 기존의 다른 프로그램들에 대한 코드 감사 보다 더욱 엄밀하고 신중하게 수행된다. 그러나, 프로그램 코드의 복잡성에 기인하여 이러한 취약점을 탐지하는 것은 단순한 문제가 아니다. 특히 인간이 직접 취약점이나 버그를 확인하는 경우에는, 많은 시간과 비용이 소요됨에도 불구하고, 스마트 컨트랙트 내의 취약점이나 버그의 존재를 놓치는 실수가 빈번하게 발생되고 있다.

선행기술문헌

특허문헌

[0003] (특허문헌 0001) 미합중국 특허출원공개공보 제2018/0183600호 (2018.06.28. 공개)

(특허문헌 0002) 일본국 특허출원공개공보 특개2001-142937호 (2001.05.25. 공개)

(특허문헌 0003) 미합중국 등록특허 제7,146,352호 (2006.12.05. 공고)

발명의 내용

해결하려는 과제

[0004] 본 발명은 스마트 컨트랙트 내의 취약점을 자동으로 단시간 내에 탐지, 획득, 입증 또는 확인할 수 있는 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법을 제공하는 것을 해결하고자 하는 과제로 한다.

과제의 해결 수단

[0005] 상술한 과제를 해결하기 위하여 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법이 제공된다.

[0006] 취약 트랜잭션 시퀀스 획득 방법은, 비용 함수를 이용하여 적어도 하나의 프로그램 내에서의 취약 트랜잭션 시퀀스 후보를 선정하는 단계, 상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하는 단계. 및 상기 검증 조건에 대한 검증 결과, 취약 트랜잭션 시퀀스가 발견되지 않은 경우 검증 조건의 만족 여부를 확인하고, 검증 조건이 만족되면 상기 취약 트랜잭션 시퀀스 후보를 취약 트랜잭션 시퀀스로 결정하는 단계를 포함할 수 있다.

[0007] 취약 트랜잭션 시퀀스 획득 장치는 적어도 하나의 프로그램을 일시적 또는 비일시적으로 저장하는 저장부, 및 상기 적어도 하나의 프로그램을 수신하고, 비용 함수를 이용하여 적어도 하나의 프로그램 내에서의 취약 트랜잭션 시퀀스 후보를 선정하고, 상기 트랜잭션 시퀀스 후보에 대해 기호 실행을 수행하여 검증 조건을 획득하고, 상기 검증 조건에 대한 검증 결과, 취약 트랜잭션 시퀀스가 발견되지 않은 경우 검증 조건의 만족 여부를 확인하고, 검증 조건이 만족되면 상기 취약 트랜잭션 시퀀스 후보를 취약 트랜잭션 시퀀스로 결정하는 프로세서를

포함할 수 있다.

발명의 효과

[0008] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 스마트 컨트랙트와 같은 프로그램으로부터 적어도 하나의 취약점, 일례로 트랜잭션 시퀀스와 같이 실행 경로에 따라 발생될 수 있는 취약점 등을 편리하고 정확하면서도 짧은 시간 내에 신속하게 탐지, 검출, 진단, 입증 또는 확인할 수 있게 된다.

[0009] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 스마트 컨트랙트 등과 같은 프로그램의 안전성을 보다 강화할 수 있게 되고, 이에 따라 이를 이용하는 산업 환경(예를 들어, 블록체인 생태계 등)의 안정성도 함께 개선할 수 있는 장점이 있다.

[0010] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 단순히 취약점의 존재 유무 자체를 추출하는 것을 넘어 어떤 트랜잭션(실행 경로)을 통해 취약점이 실제로 존재하는지 여부를 판단하거나 이와 같은 취약점을 존재하는 트랜잭션을 검출할 수 있게 되는 장점도 얻을 수 있다.

[0011] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 상대적으로 취약점을 유발할 가능성이 높은 실행 경로를 우선적으로 탐색하도록 탐색 공간을 효과적으로 편향시킴으로써, 프로그램으로부터 단 시간 내에 대량의 취약 트랜잭션 시퀀스를 검출할 수 있게 되는 효과도 얻을 수 있다.

[0012] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 스마트 컨트랙트 개발 및 배포에 있어서, 스마트 컨트랙트의 보안 감사에 대한 시간을 효율적으로 단축시킬 수 있으며, 이에 따른 개발 및 배포 비용을 적절히 절감할 수 있는 경제적 효과도 얻을 수 있다.

[0013] 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법에 의하면, 보안 검사 시에 발생 가능한 인적 실수에 대한 위험 부담을 절감할 수는 있는 장점도 존재한다.

도면의 간단한 설명

[0014] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 일 실시예에 따른 취약 트랜잭션 시퀀스 획득 장치에 대한 블록도이다.

도 2는 솔리디티 언어로 작성된 일 실시예에 따른 프로그램을 설명하기 위한 도면이다.

도 3은 일 실시예에 따른 취약점처리부의 동작을 설명하기 위한 블록도이다.

도 4는 일 실시예에 따른 취약점처리부의 동작을 설명하기 위한 알고리즘의 일례를 도시한 것이다.

도 5는 일 실시예에 따른 언어모델처리부의 동작을 설명하기 위한 블록도이다.

도 6은 일 실시예에 따른 취약 트랜잭션 시퀀스 획득 방법에 대한 흐름도이다.

도 7은 일 실시예에 따른 언어 모델을 기반으로 학습을 수행하는 과정에 대한 상세 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0015] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시 예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시 예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시 예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시 예들에 한정되지 않는다.

[0016] 본 발명의 개념에 따른 실시 예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시 예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시 예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.

[0017] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.

[0018] 어떤 구성 요소가 다른 구성 요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성 요소

에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성 요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성 요소가 다른 구성 요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는 중간에 다른 구성 요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성 요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0019] 본 명세서에서 사용한 용어는 단지 특정한 실시 예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0020] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0021] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시 예들을 상세히 설명한다. 그러나, 특허출원의 범위가 이러한 실시 예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[0023] 이하, 도 1 내지 도 5를 참조하여 취약 트랜잭션 시퀀스 획득 장치의 여러 실시예에 대해서 설명하도록 한다.

[0024] 도 1은 일 실시예에 따른 취약 트랜잭션 시퀀스 획득 장치에 대한 블록도이다.

[0025] 도 1에 도시된 바에 의하면, 취약 트랜잭션 시퀀스 획득 장치(100)는, 일 실시예에 있어서, 입력부(101)와, 저장부(103)와, 출력부(105)와 프로세서(110)를 포함할 수 있다.

[0026] 입력부(101)는, 취약 트랜잭션 시퀀스 획득 장치(100)의 설계자 또는 사용자(이하 사용자 등)로부터 또는 외부의 다른 장치로부터 취약 트랜잭션 시퀀스를 획득하기 위한 적어도 하나의 프로그램(10 내지 10-K, 이하 대상 프로그램)을 수신할 수 있다. 입력부(101)는, 사용자 등의 키보드 등에 대한 직접 조작에 따라 적어도 하나의 대상 프로그램(10 내지 10-K)을 입력 받을 수도 있고, 외부 메모리 장치(SD카드나, 범용 직렬 버스 메모리 장치나, 외장 하드디스크 장치 등)를 통해 적어도 하나의 대상 프로그램(10 내지 10-K)을 입력 받을 수도 있으며, 또는 유무선 통신네트워크를 통해 적어도 하나의 대상 프로그램(10 내지 10-K)을 입력 받을 수도 있다. 실시예에 따라서, 입력부(101)는, 키보드, 마우스, 태블릿, 터치 스크린, 터치 패드, 스캐너, 영상 촬영 모듈, 마이크로 폰, 트랙볼 및/또는 트랙패드 등을 포함할 수 있으며, 또한 실시예에 따라 외부의 장치(메모리 장치 등)로부터 데이터 등의 수신이 가능한 데이터 입출력 단자나, 외부의 장치와 유무선 통신 네트워크를 통해 연결되는 통신 모듈(일례로 랜카드, 근거리 통신 모듈 또는 이동통신 모듈 등) 등을 포함할 수도 있다.

[0027] 저장부(103)는, 취약 트랜잭션 시퀀스 획득 장치(100)의 동작에 필요한 정보를 일시적 또는 비일시적으로 저장할 수 있다. 예를 들어, 저장부(103)는, 입력부(101)를 통해 입력된 적어도 하나의 대상 프로그램(10, 10-1 내지 10-K)을 저장하거나, 및/또는 프로세서(110)의 처리에 의해 획득된 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z) 및 취약점 리포트(99) 중 적어도 하나를 저장하도록 마련된다. 또한, 저장부(103)는 언어모델처리부(150)에 의해 획득된 비용함수(이하 제2 비용함수) 등을 저장할 수도 있다. 이외에도 저장부(103)는 필요한 다양한 정보를 저장할 수 있다. 또한, 저장부(103)는 취약 트랜잭션 시퀀스 획득 장치(100), 일례로 프로세서(110)의 동작에 필요한 적어도 하나의 애플리케이션을 저장할 수도 있다. 여기서, 저장부(103)에 저장된 애플리케이션은, 프로세서(110)의 취약점처리부(120)의 각각의 동작을 구현하기 위한 적어도 하나의 변수, 명령어, 데이터, 라이브러리 및/또는 함수 등을 단독으로 또는 조합하여 구현된 것일 수 있으며, C, C++, C#, 자바, 파이썬, 솔리디티(Solidity) 언어, 닷넷(.NET) 및/또는 비주얼 베이직(Visual Basic) 등과 같이 종래 알려진 다양한 프로그래밍 언어 중 적어도 하나를 기반으로 작성된 것일 수 있다. 저장부(103)에 저장된 애플리케이션은, 실시예에 따라서 설계자에 의해 미리 직접 작성되어 저장부(103)에 저장된 것일 수도 있고, 및/또는 유선 또는 무선 통신 네트워크를 통해 접속 가능한 전자 소프트웨어 유통망을 통하여 획득 또는 갱신된 것일 수도 있다. 저장부(103)는, 프로세서(110)의 호출에 따라 필요한 정보나 명령 등을 프로세서(110)로 전달하고, 프로세서(110)는

이를 기반으로 취약 트랜잭션 시퀀스를 획득하기 위한 동작을 수행할 수 있다. 일 실시예에 의하면, 저장부(103)는, 주기억장치 및 보조기억장치 중 적어도 하나를 포함할 수 있다. 주기억장치는 롬(ROM) 및/또는 램(RAM)과 같은 반도체 저장 매체를 이용하여 구현된 것일 수 있으며, 보조기억장치는, 플래시 메모리 장치, SD(Secure Digital) 카드, 솔리드 스테이트 드라이브(SSD, Solid State Drive), 하드 디스크 드라이브(HDD, Hard Disc Drive), 자기 드럼, 컴팩트 디스크(CD), 디브이디(DVD) 또는 레이저 디스크 등과 같은 광 기록 매체(optical media), 자기 테이프, 광자기 디스크 및/또는 플로피 디스크 등과 같이 데이터를 영구적 또는 반영구적으로 저장 가능한 적어도 하나의 저장 매체를 이용하여 구현될 수 있다.

[0028] 출력부(105)는 적어도 하나의 데이터를 시각적 또는 청각적으로 외부에 출력할 수 있다. 예를 들어, 출력부(105)는, 프로세서(110)에 의해 획득되거나 및/또는 저장부(103)에 저장된 취약 트랜잭션 시퀀스(11-1 내지 11-Z)나 취약점 리포트(99)를 외부로 출력할 수 있다. 실시예에 따라서, 출력부(105)는, 취약 트랜잭션 시퀀스 획득 장치(100)와 일체형으로 마련된 것일 수도 있고, 또는 물리적으로 분리 가능하게 마련된 것일 수도 있다. 출력부(105)는, 예를 들어, 디스플레이, 프린터 장치, 스피커 장치, 영상 출력 단자, 데이터 입출력 단자 및/또는 통신 모듈 등을 이용하여 구현될 수 있으나, 이에 한정되는 것은 아니다.

[0029] 프로세서(110)는, 취약 트랜잭션 시퀀스 획득 장치(100)의 동작과 관련된 연산, 판단, 처리 및/또는 제어 동작 등을 동시에 또는 순차적으로 수행할 수 있다. 이와 같은 동작을 위해 프로세서(110)는, 저장부(103)에 저장된 애플리케이션을 구동시킬 수도 있다. 프로세서(110)는, 예를 들어, 중앙 처리 장치(CPU, Central Processing Unit), 마이크로 컨트롤러 유닛(MCU, Micro Controller Unit), 마이컴(Micom, Micro Processor), 애플리케이션 프로세서(AP, Application Processor), 전자 제어 유닛(ECU, Electronic Controlling Unit) 및/또는 각종 연산 처리 및 제어 신호의 생성이 가능한 다른 전자 장치 등을 포함할 수 있다. 이들 장치는, 예를 들어, 하나 또는 둘 이상의 반도체 칩 등을 이용하여 구현된 것일 수 있다.

[0030] 일 실시예에 의하면, 프로세서(110)는, 적어도 하나의 대상 프로그램(10, 10-1 내지 10-K)을 입력부(101)나 저장부(103) 등으로부터 획득하고, 획득한 적어도 하나의 대상 프로그램(10, 10-1 내지 10-K) 중 적어도 하나로부터 적어도 하나의 취약점을 갖는 적어도 하나의 트랜잭션 시퀀스(11-1 내지 11-Z, 이하 취약 트랜잭션 시퀀스)를 각각 획득할 수 있다. 여기서, 적어도 하나의 트랜잭션 시퀀스는, 제1 트랜잭션부터 제n 트랜잭션까지의 일련의 트랜잭션($t_0, t_1, \cdots, t_n \in T^*$)을 의미하고, 트랜잭션($t \in T$)은 트랜잭션을 식별하기 위한 식별자(id)가 함수 경로에 부가된 것으로, (id, f, x, a)의 4-튜플로 정의된 것일 수 있다. 함수 경로는, 함수의 입구에서 함수 출구까지의 명령문(일례로 기본 명령문)들의 시퀀스를 의미하며, 함수 $f(x) S \in F$에 대해 (f, x, a)의 형태로 주어질 수 있다(이하에서는 대상 프로그램(11)에 존재하는 모든 함수 경로들의 집합을 P로 표현한다). 대상 프로그램(10, 10-1 내지 10-K)으로부터 적어도 하나의 취약 트랜잭션 시퀀스를 획득하는 경우, 프로세서(110)는, 하나의 대상 프로그램(10, 10-1 내지 10-K)으로부터 하나 또는 복수의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 획득할 수도 있고, 다수의 대상 프로그램(10, 10-1 내지 10-K)으로부터 하나 또는 복수의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 각각 획득할 수도 있다. 일 실시예에 의하면, 프로세서(110)가 획득하는 취약 트랜잭션 시퀀스는 마지막 트랜잭션(t_n)에 의해 실행되는 명령문(일례로 기본 명령문) 중에서 안전 조건이 위반될 수 있는 단언문을 포함하고 있는 트랜잭션 시퀀스일 수도 있다.

[0031] 일 실시예에 있어서, 프로세서(110)는 적어도 하나의 대상 프로그램(10, 10-1 내지 10-Z) 각각에 대해 기호 실행(symbolic execution)을 수행함으로써, 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 획득할 수도 있다. 또한, 프로세서(110)는, 실시예에 따라서, 언어 모델을 더 이용하여, 적어도 하나의 대상 프로그램(10, 10-1 내지 10-K)으로부터 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 획득할 수도 있다. 언어 모델을 이용하는 경우, 프로세서(110)가 취약점을 유발할 가능성이 높은 취약 트랜잭션 시퀀스 후보(들)부터 우선적으로 기호 실행 및 취약 트랜잭션 시퀀스(11-1 내지 11-Z)의 탐색을 수행할 수 있게 되어, 기호 실행의 탐색 공간을 보다 효과적으로 편향시킬 수 있게 된다.

[0032] 도 1에 도시된 바와 같아, 일 실시예에 의하면, 프로세서(110)는 대상 프로그램(10, 10-1 내지 10-K)으로부터 하나 또는 복수의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 탐색하는 취약점처리부(120)와, 언어모델을 이용하여 적절한 제2 비용함수를 취약점처리부(120)에 제공하는 언어모델처리부(150)를 포함할 수 있다. 취약점처리부(120)와 언어모델처리부(150)는 각각 논리적으로 구분되는 것일 수도 있고 또는 물리적으로 구분되는 것일 수도 있다. 물리적으로 구분되는 경우, 취약점처리부(120)와 언어모델처리부(150) 각각은 서로 물리적으로 분리되되 전기적으로 연결된 적어도 하나의 반도체 칩 등을 이용하여 구현될 수 있다. 실시예에 따라서, 언어모델처리부(150)는 생략될 수도 있다. 언어모델처리부(150)가 부재한 경우, 프로세서(110)는, 실시예에 따라, 제2 비용함수를 이용하지 않고 다른 비용함수(이하 제1 비용함수)를 이용하여 적어도 하나의 대상 프로그램(10, 10-1 내지

10-K)으로부터 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 획득할 수도 있고, 또는 다른 정보처리장치(미도시) 등이 획득한 제2 비용함수를 이용하여 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 획득할 수도 있다. 여기서 다른 정보처리장치가 획득한 제2 비용함수는 입력부(101)나 저장부(130)를 통해 프로세서(110)로 전달될 수 있다. 프로세서(110)의 동작 및 기능에 대한 구체적은 설명은 후술한다.

[0033] 상술한 취약 트랜잭션 시퀀스 획득 장치(100)는, 정보의 획득 및 연산 처리가 가능한 적어도 하나의 정보처리장치를 이용하여 구현될 수 있다. 예를 들어, 취약 트랜잭션 시퀀스 획득 장치(100)는, 데스크톱 컴퓨터, 랩톱 컴퓨터, 서버 전용 컴퓨터 장치, 스마트 폰, 태블릿 피씨, 스마트 시계, 두부 장착형 디스플레이(HMD: Head Mounted Display) 장치, 내비게이션 장치, 휴대용 게임기, 개인용 디지털 보조기(PDA: Personal Digital Assistant), 디지털 텔레비전, 셋톱 박스, 가전 기기(냉장고나 로봇 청소기 등), 인공 지능 음향 재생 장치(인공 지능 스피커), 차량, 유인 또는 무인 비행체, 로봇, 산업용 기계 및/또는 이외 부호의 입력 및 수정이 가능한 정보 처리 장치를 포함할 수 있다. 그러나, 취약 트랜잭션 시퀀스 획득 장치(100)는, 이에 한정되는 것은 아니며, 설계자가 고려 가능한 다양한 전자 장치를 포함할 수 있다. 또한, 실시예에 따라서, 취약 트랜잭션 시퀀스 획득 장치(100)는 둘 이상의 정보처리장치를 이용하여 구현될 수도 있다.

[0034] 제1 내지 제K 대상 프로그램(11, 11-1 내지 11-K) 중 적어도 하나는, 입력부(101)를 통해 프로세서(110)로 전달되거나, 저장부(103)에 일시적 또는 비일시적으로 저장된 후 프로세서(110)로 전달된 후, 프로세서(110)에 의해 취약점의 존재 여부가 확인 및 검토될 수 있다. 이 경우, 제1 내지 제K 대상 프로그램(11, 11-1 내지 11-K) 중 적어도 하나는, 실시예에 따라, 동시에 또는 순차적으로 입력부(101)를 통해 입력되거나 저장부(103)에 저장될 수 있다.

[0035] 일 실시예에 의하면, 제1 내지 제K 대상 프로그램(11, 11-1 내지 11-K) 중 적어도 하나는 스마트 컨트랙트를 포함할 수 있다. 다시 말해서, 프로세서(110)는 스마트 컨트랙트를 분석하여, 스마트 컨트랙트로부터 이의 취약점, 일례로 취약 트랜잭션 시퀀스를 검출할 수 있다.

[0036] 제1 내지 제K 대상 프로그램(11, 11-1 내지 11-K)은 모두 동일한 종류의 프로그래밍 언어를 이용하여 코딩된 것일 수도 있고, 일부는 다른 일부와 상이한 프로그래밍 언어를 이용하여 코딩된 것일 수도 있으며, 또는 모두 상이한 종류의 프로그래밍 언어를 이용하여 코딩된 것일 수도 있다. 여기서, 프로그램 언어는 C, C++, C#, 자바, 파이썬, 솔리디티 언어 및/또는 비주얼 베이직 등과 같이 종래 알려진 다양한 프로그래밍 언어 중 적어도 하나를 포함할 수 있다. 예를 들어, 대상 프로그램(11, 11-1 내지 11-K)이 이더리움(Ethereum)과 같은 블랙체인에서 동작하는 스마트 컨트랙트인 경우, 대상 프로그램(11, 11-1 내지 11-K)은 솔리디티 언어로 작성된 것을 수 있다.

[0037] 도 2는 솔리디티 언어로 작성된 일 실시예에 따른 프로그램을 설명하기 위한 도면이다. 도 2에서 E는 통상적인 산술 표현식(일례로 a+b 등)을 의미하고, B는 부울 표현식(예를 들어, a≥b)을 의미한다.

[0038] 대상 프로그램(11, 11-1 내지 11-K)이 솔리디티 언어를 기반으로 작성된 스마트 컨트랙트의 경우, 대상 프로그램(11, 11-1 내지 11-K)은 도 2에 도시된 바와 같이 간략화된 솔리디티 언어로 표현될 수 있다. 즉, 솔리디티 언어를 기반으로 복잡하게 프로그램 코드가 작성되었다고 하더라도, 소정의 전처리 과정을 통해 대상 프로그램(11, 11-1 내지 11-K)은 도 2에 도시된 바와 같이 간략하게 표현될 수 있다. 이 경우, 도 2에 도시된 바를 참조하면, 스마트 컨트랙트(c)는 전역 상태 변수(global state variable) 선언들(G*)과 함수 정의들(F*)을 포함할 수 있다. 이 경우, 각 함수(F)는, 도 2의 첫번째 줄에 도시된 바와 같이, 함수 명칭(f), 적어도 하나의 형식 인자(x) 및/또는 적어도 하나의 명령문(S)을 포함하여 정의될 수 있다. 명령문(S)은, 도 2의 세번째 줄에 기재된 바와 같이, 기본 명령문(A), 조건문(if B S1 S2), while-루프문(while B S) 및 적어도 하나의 명령문들의 시퀀스(S1;S2) 중 적어도 하나를 포함할 수 있다. 여기서, 기본 명령문 a는, 두번째 줄에 기재된 바와 같이 변수 할당문(x:=E), 배열 원소 할당문(x[y]:=E), 가정문(assume(B)) 및/또는 단언문(assert^l(B))을 포함할 수 있다. 변수 할당문(x:=E)은 변수를 할당하고, 배열 원소 할당문(x[y]:=E)은, 배열 변수 원소들에 대한 값이나 솔리디티 언어에서 매핑 타입 변수 원소들에 대한 값을 할당할 수 있다. 가정문(assume(B))은 프로그램의 실행 경로를 결정하는 조건 표현식(일례로 if-조건문이나 while-루프 내에서의 B)을 단일 명령문 형태로서 구축할 수 있다. 단언문(assert^l(B))은, 소정의 대상(B)에 대한 판단을 수행하여 대상(B)의 참, 거짓 여부를 판단하기 위해 이용된다. 이에 따라, 단언문(assert^l(B))은 분석을 원하는 특징에 대한 조건(일례로 안전 조건)을 나타내거나 조건(안전 조건)의 부합 여부를 판단하기 위해 이용될 수 있다. 여기서 분석하려는 특징은 대상 프로그램(11, 11-1 내지 11-K), 즉 스마트 컨트랙트(c)의 취약점(예를 들어, 정수 오버플로우나 정수 언더플로우, 0 나누기 취약점 등)에 대한 안전 여부 등을 포함할 수 있다. 만약 단언문(assert^l(B)) 내의 조건이 위반되어, 조건이

거짓으로 계산되면, 이는 대상 프로그램(11, 11-1 내지 11-K) 내에 취약점이 존재함을 의미한다. 설계자는 대상 프로그램(11, 11-1 내지 11-K)에 단언문(assert1(B))을 삽입하여 이와 같은 안전 조건을 나타낼 수 있다. 실시예에 따라서, 단언문(assert1(B)) 등은 설계자의 조작 없이도 대상 프로그램(11, 11-1 내지 11-K) 내에 자동으로 삽입될 수도 있다. 필요에 따라서, 각각의 단언문(assert^1(B))은 단언문들을 구별해주는 고유한 레이블(1)이 부가되어 있을 수 있다.

[0039] 이하 프로세서(110)의 취약점처리부(120)의 동작의 일 실시예에 대해 설명하도록 하되, 제1 대상 프로그램(11)의 예를 들어, 취약점 검출 및 획득 동작을 설명하도록 한다. 그러나, 제2 내지 제K 대상 프로그램(11)에 대해서도 취약점처리부(120) 후술하는 바와 동일하게 또는 일부 변형된 방법으로 취약점을 검출 및 획득할 수 있다.

[0040] 도 3은 일 실시예에 따른 취약점처리부의 동작을 설명하기 위한 블록도이고, 도 4는 일 실시예에 따른 취약점처리부의 동작을 설명하기 위한 알고리즘의 일례를 도시한 것이다. 도 4의 알고리즘은 프로세서(110)에 의해 처리될 수 있다.

[0041] 도 3에 도시된 바와 같이, 취약점처리부(120)는 제1 대상 프로그램(11)을 수신하면, 제1 대상 프로그램(11)로부터 적어도 하나의 취약점(예를 들어, 취약 트랜잭션 시퀀스)을 검출하기 위해 사전 준비 작업 및 각종 데이터의 초기화 작업을 수행할 수 있다(121). 여기서, 사전 준비 작업은 제1 대상 프로그램(11)을 분해하여 처리에 적절한 형태로 변형하는 과정을 포함할 수 있다.

[0042] 보다 구체적으로 예를 들어, 취약점처리부(120)는 도 4의 프로그램 코드의 제1 내지 제4 줄에 기재된 바와 같이 사전 준비 및 초기화 작업을 수행할 수 있다. 예를 들어, 취약점처리부(120)는 먼저 대상 프로그램(11) 내의 각각의 루프(일례로 while-루프 등)를 언롤링(unrolling)하여 풀어주고, 및/또는 함수의 호출 지점 각각마다 호출될 함수들을 인라인(inline)하여 부가하여 대상 프로그램(11)을 변형할 수 있다(제1 줄). 여기서, 루프의 언롤링은 미리 정의된 회수(m, m은 자연수)를 최대값으로 하여 수행될 수 있으며, 함수의 인라인도 미리 정의된 회수(중첩 함수의 호출 횟수, n, n은 자연수, 여기서, n은 m과 동일할 수도 있고, 또는 상이할 수도 있음)를 최대로 하여 수행될 수 있다. 언롤링 및/또는 인라인의 수행에 의해 취약점처리부(120)가 무한으로 함수 경로를 나열하여 분석하는 것을 회피할 수 있게 된다. 순차적으로 취약점처리부(120)는 언롤링 및/또는 인라인에 의해 변형된 대상 프로그램(11)에 존재하는 각 함수에 대응하는 함수 경로를 수집할 수 있다(제2 줄). 이때, 취약점처리부(120)는 미리 정의된 개수(o)만큼 또는 미리 정의된 개수(o)보다 적게 각각의 함수에 대한 함수 경로를 수집하여 함수 경로에 대한 집합(P)을 획득할 수 있다. 상황에 따라서, 하나의 함수로부터 하나 또는 둘 이상의 함수 경로가 획득될 수 있다. 한편, 대상 프로그램(11)의 변형 및 함수 경로 획득 과정과 더불어, 선행하여 또는 후행하여, 취약점처리부(120)는 취약 트랜잭션 시퀀스 후보군에 대한 집합(W, 이하 워크셋)을 초기화할 수 있다(제3 줄). 보다 구체적으로 취약점처리부(120)는 생성자 함수(대상 프로그램(11)의 배포 시 1회만 호출되고 이후에는 실행되지 않는 함수)로부터 유래한 적어도 하나의 초기 트랜잭션(t0)을 이용하여 워크셋(W)을 초기화할 수 있다. 이와 더불어, 취약점처리부(120)는 취약 트랜잭션 시퀀스가 기록될 취약점 리포트(99, R:L→T*xModel)를 초기화할 수도 있다. 예를 들어, 취약점처리부(120)는 $\lambda l.\perp$를 취약점 리포트(99)에 입력함으로써 취약점 리포트(99)를 $\lambda l.\perp$로 초기화할 수 있다(제4 줄). 여기서, $\lambda l.\perp$는 모든 단언문 레이블 l이 ⊥로 매핑되어 있음을 람다 표기법으로 나타낸 것으로, 구체적으로 단언문 내의 안전 조건이 위반될 수 있는 트랜잭션 시퀀스가 미발견 상태임을 의미한다.

[0043] 상술한 바와 같이 사전 준비 및 초기화 작업이 종료되면, 취약점처리부(120)는 취약 트랜잭션 시퀀스 후보를 선정할 수 있다(123). 이 경우, 취약점처리부(120)는 소정의 비용함수(cost(ω), 이하 제1 비용함수)를 기반으로 비용을 최소화하는(즉, argmin_ω∈W cost(ω)) 취약 트랜잭션 시퀀스 후보(s=t_0, t_1, …, t_n)를 워크셋(W)으로부터 선택할 수 있다(제6 줄). 선택된 취약 트랜잭션 시퀀스 후보(s)는 이와 동시에 또는 순차적으로 워크셋(W)에서 제거될 수 있다(제7 줄). 일 실시예에 의하면, 제1 비용 함수는, 길이가 상대적으로 짧은 후보들을 우선적으로 탐색할 수 있도록 cost(ω)=n로 정의될 수 있다. 실시예에 따라서, 취약점처리부(120)는, 제1 비용함수 대신에 제2 비용함수를 이용하여, 비용을 최소화하는 취약 트랜잭션 시퀀스 후보(s)를 워크셋(W)으로부터 선택할 수도 있다. 제2 비용함수는, 언어모델처리부(150)로부터 취약점처리부(120)로 전달된 것일 수 있다. 제2 비용함수에 대해선 후술한다.

[0044] 이어서 취약점처리부(120)는 선정된 트랜잭션 후보(s)에 대해 기호 실행을 수행하고(125), 기호 실행의 결과에 대응하여 검증 조건(VC, verification condition)을 획득할 수 있다(127, 제8줄). 검증 조건은 취약 트랜잭션 시퀀스 후보(s)가 단언문들에 대해 실제로 취약 트랜잭션 시퀀스인지 여부를 확인하기 위해 검증되어야 할 1차 술어 논리(first-order logic)로 표현된 논리식을 포함할 수 있다. 기호 실행의 수행 결과, 단언문 레이블(1)

및 이에 대응하는 검증 조건(VC)을 함께 포함하는 별도의 집합(∏)이 획득될 수 있다.

[0045] 일 실시예에 의하면, 취약점처리부(120)는 미리 정의된 소정의 함수(이하 검증 조건 생성 함수, 도 4의 GenerateVC(s))를 이용하여 기호 실행을 수행할 수도 있다. 검증 조건 생성 함수(Generate(t_0, t_1, …, t_n))는, 트랜잭션 시퀀스(t_0, t_1, …, t_n)에 대해 단언문 레이블(l) 및 검증 조건의 쌍들을 기반으로 소정의 집합(∏)을 출력하도록 설계된 것일 수 있다. 여기서, 일 실시예에 의하면, 소정의 집합(∏)은 트랜잭션에 대한 기호 실행기(T)를 변형하여 획득된 트랜잭션 시퀀스에 대한 기호 실행기(T')를 이용하여 획득될 수 있다. 예를 들어, 소정의 집합(∏)을 획득하기 위해 하기의 수학식 1이 이용될 수 있다.

[0046] [수학식 1]

[0047] $$(-, \Pi) = \mathrm{T}'(a_n) \circ \cdots \circ \mathrm{T}'(t_0)(\wedge_{g \in G} \in init(g),\ \varnothing)$$

[0048] 수학식 1에서 init(g)은 전역 상태 변수 g∈G에 대해 각 변수의 타입에 적절한 기본값을 할당하는 함수를 의미한다. 트랜잭션 시퀀스에 대한 기호 실행기(T')는 하기의 수학식 2에 기재된 바와 같이 같이 정의될 수 있다.

[0049] [수학식 2]

[0050] $$\mathrm{T}'(t_i)(\phi'', \Pi'') = \begin{cases} (\phi', \Pi') & \text{if } i = n \\ (\phi', \emptyset) & \text{otherwise} \end{cases}$$

[0051] 수학식 2에 기재된 바와 같이 논리식 ϕ' 및 집합 ∏'는 트랜잭션에 대한 기호 실행으로부터 획득할 수 있다. 즉, (ϕ', ∏')=T(t_i)(ϕ'', ∏''))가 성립하게 된다. 이 경우, 인덱스 값이 n인 경우(즉, 트랜잭션 시퀀스 중 마지막 트랜잭션인 경우)에만 레이블(l) 및 검증 조건의 쌍의 집합(∏')이 획득될 수도 있다. 다시 말해서, 마지막 트랜잭션(t_n)으로부터만 검증 조건이 획득될 수도 있다. 보다 상세히 설명하면, i번째 트랜잭션(t_i) 이후의 트랜잭션(t_j1, j1 > i)들은 i번째 트랜잭션(t_i) 이전의 트랜잭션(t_j2, j2 < i)의 안전 조건 위반 여부에 거의 영향을 미치지 않는다. 그러므로, 마지막 트랜잭션(t_n)으로부터만 검증 조건을 획득한다고 하더라도, 대상 프로그램(11)으로부터 취약 트랜잭션 시퀀스를 검출하는 것에는 문제가 발생하지 않는다.

[0052] 상술한 트랜잭션에 대한 기호 실행기(T: T→FOL x 2^L x FOL→ FOL x 2^L x FOL(여기서, L은 모든 단언문 레이블(l)의 집합이고, FOL은 1차 술어 논리(first-order logic)로 표현될 수 있는 논리식의 집합이고, ':' 오른쪽의 T는 트랜잭션의 집합임)는, 하기의 수학식 3과 같이 주어질 수 있다.

[0053] [수학식 3]

[0054] $$\mathrm{T}(i, f, x, (a_1, \cdots, a_n))(\phi, \Pi) = (\mathrm{RENAMEL}(\phi', i), \{(l, \mathrm{RENAMEL}(F, i)) \mid (l, F) \in \Pi'\})$$

[0055] 수학식 3에서, ϕ' 및 ∏'는 하기의 수학식 4에 나타난 바와 같이 기본 명령문에 대한 기호 실행기, 즉 후조건 변환기(sp)를 이용하여 획득할 수 있다.

[0056] [수학식 4]

[0057] $$(\phi', \Pi') = \mathrm{sp}^{(a_n)} \circ \cdots \circ (a_2) \circ (a_1))(\phi \wedge x^e = x \wedge \varphi,\ \Pi)$$

[0058] 수학식 4에서, x^e=x ∧ φ의 x^e는 형식 인자의 함수 입구(function entry)에서의 상태(state)를 나타내는 변수이고, φ는 적절한 트랜잭션 호출 인자 값을 생성하기 위한 부가적 제약 조건으로, 논리곱 형태(conjunctive formula)로 주어질 수 있다. 실시예에 따라, φ는 솔리디티에 특화된 부가적인 제약 조건을 포함할 수 있다. 예를 들어, φ는 msg.sender≠0과 같은 가능한 원자식(atomic formula)을 포함할 수 있다. 수학식 4의 후조건변환기(sp: a→FOL x 2^L x FOL→ FOL x 2^L x FOL)는 기본 명령문 a∈A에 대한 실행 의미를 1차 술어 논리 상의 논리식으로 변환할 수 있는 변환기를 의미한다. 후조건변환기(sp)는, 예를 들어, 하기의 수학식 5와 같이 정의될 수도 있으나, 이에 한정되는 것은 아니다. 설계자의 선택에 따라 다양한 방식으로 후조건변환기(sp)는 정의될 수 있다.

[0059] [수학식 4]

$$\mathsf{sp}(x := e)(\phi, \Pi) = (\phi[x'/x] \wedge (x = e[x'/x])^{\circ}, \Pi)$$
$$\mathsf{sp}(x[y] := e)(\phi, \Pi) = (\phi[x'/x] \wedge (x = x'\langle y \lhd e[x'/x]\rangle)^{\circ}, \Pi)$$
$$\mathsf{sp}(assume(e))(\phi, \Pi) = (\phi \wedge e^{\bullet}, \Pi)$$
$$\mathsf{sp}(assert^{l}(e))(\phi, \Pi) = (\phi, \{(l, \phi \wedge \neg e)\} \cup \Pi)$$

[0060]

[0061] 수학식 4에서 ϕ[x'/x]은 ϕ에 존재하는 변수 x가 새로운 변수 x'으로 교체된 논리식을 의미하고, e[x'/x]은 e에 존재하는 변수 x가 새로운 변수 x'으로 교체된 표현식을 의미한다. 수학식 4의 두번째 줄의 x' < y ◁ e[x'/x]>은 배열 타입이거나 매핑 타입인 변수 x'에 대해, 인덱스 y에 위치한 값이 e로 교체된 배열을 의미한다. 수학식 4에 기재된 바를 참조하면, 후조건변환기(sp)는, 실시예에 따라, 전제 조건(precondition)과 현재까지 누적된 레이블-검증 조건의 쌍의 집합(∏)이 주어졌을 때, 단언문을 제외한 각 기본 명령문(a, 일례로 변수 할당문, 배열 원소 할당문 및/또는 가정문)의 실행 의미에 따라 전제조건을 후조건(postcondition)으로 변형하도록 할 수 있다(수학식 4의 첫번째 내지 세번째 줄). 또한, 후조건변환기(sp)는, 단언문의 경우, 새로운 단언문 레이블-검증조건 쌍을, 단언문 레이블-검증조건 쌍의 집합(∏)에 누적시킬 수도 있다(수학식 4의 네번째 줄). 보다 구체적으로 단언문에 대한 후조건변환기(sp)는, 프로그램(11)의 현재 상태를 나타내는 논리식 ϕ와 단언문 안의 안전 조건을 부정한 것(¬e)을 ∧ 연산자를 이용하여 결합하고, 이를 대응하는 단언문 레이블(l)에 매칭한 후, 레이블(l) 및 검증조건(VC) 쌍의 집합(∏)에 부가하도록 정의될 수 있다. 일 실시예에 의하면, 수학식 4에 도시된 바와 같이, 후조건변환기(sp)에 있어서 변수 할당문에 부가되는 논리식에는 • 기호가 더 추가되고, 가정문에 부가되는 논리식에는 ° 기호가 더 추가될 수도 있다. 이는 변수 할당문 및 가정문 각각에서 유래한 동등 논리식(equality constraint)을 구별함으로써 논리식을 간략화하기 위함이다. 이에 대해선 후술한다.

[0062] 한편, 상술한 수학식 3으로 표현된 트랜잭션에 대한 기호 실행기(T)에 있어서, 재명명함수(RenameL())는 여러 트랜잭션에 걸쳐 존재하는, 동일 명칭의 지역변수들을 구별하기 위한 함수로, 구체적으로 주어진 논리식에 존재하는 소정의 변수(들)을 트랜잭션 식별자를 이용하여 재명명함으로써, 이들 지역변수들이 구별될 수 있도록 한다. 이 경우, 재명명함수는, 전역 변수(지역 변수와 상이하게 특정 트랜잭션 실행 과정에서 장시간 존재하는 변수) 및/또는 프라임 전역 변수(들)에 대해선 재명명을 수행하지 않도록 마련될 수도 있다. 보다 구체적으로 예를 들어, 트랜잭션 식별자 j에 대해서 논리식 a'=0 ∧ b=1 ∧ c'=2가 주어지고 a가 유일한 전역 상태 변수인 경우, 재명명함수는 상술한 논리식이 입력되면, 이에 대응하여 a'=0 ∧ b_j=1 ∧ c'_j =2의 결과를 출력할 수 있다. 즉, 지역 변수 b 및 c는 각각 트랜잭션 식별자 j가 부가된 다른 명칭의 변수 b_j 및 c_j로 변경되되, a'는 전역 상태 변수의 프라임 변수이므로, 그 명칭이 유지된다.

[0063] 상술한 바와 같이 소정의 검증 조건 생성 함수를 이용한 기호 실행의 수행 결과에 따라, 적어도 일 검증 조건이 획득되면, 취약점처리부(120)는 도 3 및 도 4에 도시된 바와 같이 적어도 하나의 단언문 레이블(l) 및 단언문 레이블(l)에 대응하는 검증 조건(VC) 각각에 대한 검증을 수행할 수 있다(131, 도 4의 제9 내지 13줄). 단언문 레이블(l) 및 검증 조건(VC)에 대한 검증은 순차적으로 수행될 수도 있고, 또는 임의적인 순서로 수행될 수도 있다.

[0064] 만약 레이블 l의 단언문에 대해 안전 조건을 위반할 수 있는 취약 트랜잭션 시퀀스가 미발견 상태라면(즉, 취약점 리포트(99, R)의 단언문 레이블 l이 ⊥로 매핑되어 있음, 도 4의 제10 줄), 주어진 검증 조건에 대한 만족(satisfiability) 여부를 확인할 수 있다(도 4의 제11줄). 일 실시예에 의하면, 주어진 검증 조건에 대한 만족 여부의 확인을 위해 타당성 여부의 판단을 수행할 수 있는 적어도 하나의 해석 프로그램이 이용될 수 있다. 여기서, 적어도 하나의 해석 프로그램은, 예를 들어, Z3 등과 같은 SMT 솔버(Satisfiability Modulo Theories Solver)를 포함할 수 있다. SMT 솔버는, 실시예에 따라서, 프로세서(110)에 의해 구동되는 애플리케이션에 포함되어 있을 수도 있고 또는 애플리케이션에 불포함되어 애플리케이션의 구동에 따라 외부로부터 호출된 것일 수도 있다. 또한, SMT 솔버는, 취약 트랜잭션 시퀀스 획득 장치(100)에 마련된 저장부(103)에 저장되고, 취약점처리부(120)의 호출에 따라 취약점처리부(120)에 제공되도록 마련된 것일 수도 있고 및/또는 취약점처리부(120)의 호출에 따라 통신 모듈 등과 같은 입력부(101)를 통해 외부의 다른 장치(미도시)로부터 수신된 후 취약점처리부(120)에 제공되도록 마련된 것일 수도 있다.

[0065] 검증 조건이 만족된다는 것은 취약 트랜잭션 시퀀스 후보(s) 내에 취약점을 유발하게 하는 경우가 존재한다는 것을 의미한다. 다시 말해서, 현재의 분석 대상이 되는 취약 트랜잭션 시퀀스 후보(s)가 취약 트랜잭션 시퀀스

로 판단됨을 의미한다. 검증 조건의 만족에 응하여, 취약점 리포트(99, R)는 갱신될 수 있다(133, 도 4의 제11줄). 취약점 리포트(99, R)의 갱신은, 예를 들어, 검증 조건이 만족되면, 단언문 레이블 l에, 취약점을 갖는 것으로 판명된 취약 트랜잭션 시퀀스 후보(s, 즉 취약 트랜잭션 시퀀스(11-1 내지 11-Z 중 적어도 하나)와 검증 조건에 대한 만족 모델(model(VC)을 매핑함으로써 수행될 수도 있다. 여기서, 만족 모델(model(VC))은 검증 조건을 만족하는 해로, 각각의 트랜잭션의 형식 인자에 대한 실제 인자 값 정보를 포함하고 있을 수 있다. 그러므로, 필요에 따라 이를 기반으로 각 트랜잭션 호출에 필요한 인자 값을 획득할 수도 있다. 또한, 필요에 따라 취약한 것으로 판단된 취약 트랜잭션 시퀀스 후보(s)는 취약 트랜잭션 시퀀스(11-1 내지 11-Z)로 저장부(103)에 저장되거나 또는 출력부(105)를 통해 출력될 수 있다. 취약 트랜잭션 시퀀스(11-1 내지 11-Z)의 저장이나 출력은 취약점 리포트(99, R)의 갱신과 더불어 또는 순차적으로 수행될 수도 있다.

[0066] 한편, 상술한 바와 반대로 레이블 l의 단언문에 대해 안전 조건을 위반할 수 있는 취약 트랜잭션 시퀀스가 이미 발견된 상태라면(즉, $R(l) \neq \perp$), 상술한 검증 조건의 만족 여부 판단은 수행되지 않는다(도 4의 제10줄 및 제12줄).

[0067] 순차적으로 새로운 취약 트랜잭션 시퀀스 후보군이 생성되어 워크셋(W)에 추가될 수 있다(135, 도 4의 제13줄). 상세하게 예를 들어, 현재 분석 대상이 되었던 적어도 하나의 취약 트랜잭션 시퀀스 후보군(s)에 대해 트랜잭션을 추가적으로 부가하여 새로운 취약 트랜잭션 시퀀스 후보군이 생성되고, 생성된 새로운 취약 트랜잭션 시퀀스 후보군은 워크셋(W)에 추가될 수 있다.

[0068] 상술한 비용 함수 기반 취약 트랜잭션 시퀀스 후보 선정 과정 내지 새로운 취약 트랜잭션 시퀀스 후보군의 생성 과정(123 내지 135, 도 4의 제5 내지 14줄)은 적어도 일 회 반복되되, 미리 정의된 설정이 만족될 때까지 계속해서 반복될 수 있다. 예를 들어, 상술한 과정(123 내지 135, 도 4의 제5 내지 14줄)은, 워크셋(W)에 어떠한 취약 트랜잭션 시퀀스 후보군(s)도 존재하지 않거나($W = \varnothing$), 모든 단언문들에 대한 취약 트랜잭션 시퀀스가 발견되거나($\forall l.R(l) \neq \perp$), 및/또는 미리 정의된 시간이 초과된 경우에 종료되도록 설계된 것일 수도 있다(137, 도 4의 제14줄). 상술한 과정이 종료되면, 취약점처리부(120)는 획득한 취약점 리포트(99, R)를 출력하여, 저장부(103) 및 출력부(105) 중 적어도 하나로 전달할 수 있다.

[0069] 일 실시예에 의하면, 단언문 레이블 및 검증 조건 각각에 대한 검증의 수행 과정(131)에 선행하여, 취약점처리부(120)는 검증 조건의 간략화를 더 수행할 수도 있다(129). 구체적으로 검증 조건의 만족 여부를 판단할 때, 검증 조건식이 복잡한 경우 만족 여부 판단에 대량의 리소스나 긴 처리 시간을 요구할 수도 있다. 따라서, 대량의 리소스 소비나 장시간의 처리를 방지하기 위해 검증 이전에 검증 조건을 간략화 할 필요가 존재한다.

[0070] 예를 들어, 검증 조건의 간략화는, 소정의 간략화 규칙을 적용하여 수행될 수도 있다. 예를 들어, $a \leq a$와 같이 참, 거짓 여부가 자명한 논리식을 참 값 또는 거짓 값으로 치환하는 등의 방법으로 검증 조건을 간략화할 수도 있다.

[0071] 다른 예를 들어, 검증 조건의 간략화는, 검증과 무관한 논리식들을 검증 조건에서 제외함으로써 수행될 수도 있다. 예를 들어, 직접적인 검증 대상인 안전 조건이나 가정문에서 유래한 조건(경로 조건) 등과 관련이 없는 논리식들을 검증 조건에서 제외함으로써 검증 조건을 간략화할 수도 있다. 예를 들어, 검증 조건이 $(x=y)^{\circ} \wedge (z=10)^{\bullet} \wedge \neg(y+1 \geq y)$ 로, $(y+1 \geq y)$ 이 안전 조건으로 주어진 경우, 통산 단일(atomic) 논리식 $(x=y)^{\circ}$ 은 할당문(즉, $x := y$)으로부터 생성되고, 논리식 $(z=10)^{\bullet}$ 은 가정문($assume(z=10)$)으로부터 생성된다. 논리식 $(x=y)^{\circ}$ 의 $^{\circ}$ 기호를 고려하면, 논리식 $(x=y)^{\circ}$ 에서 값은 y에서 x로 전달된다. 그러므로, x에 대한 정보는 안전 조건의 위반 여부를 판단하거나 가정문을 통과하는데 불필요한 정보이다. 그러므로, 이를 반영한 후 $^{\bullet}$ 및 $^{\circ}$ 기호를 삭제하면, 위의 검증 조건 $(x=y)^{\circ} \wedge (z=10)^{\bullet} \wedge \neg(y+1 \geq y)$ 은 $(z=10) \wedge \neg(y+1 \geq y)$ 의 형태로 간략화될 수 있게 된다.

[0072] 또 다른 예를 들어, 검증 조건의 간략화는, 수량자 제거(quantifier elimination) 최적화를 통해 수행될 수도 있다. 구체적으로, 만약 검증 조건 $\forall i.x'[i]=0 \wedge x=x' < y \triangleleft 10 > \wedge \neg(x[y] < 10)$ 이 주어지고 변수 x 및 x'가 매핑 타입의 변수인 경우, 주어진 검증 조건 내에서는 y가 x 및 x'의 원소에 접근하는 유일한 인덱스임을 알 수 있다. 그러므로, x 및 x'의 만족 여부를 판단에 불필요한 원소에 대한 정보(들)을 검증 조건에서 제거하면, 주어진 검증 조건 $\forall i.x'[i]=0 \wedge x=x' < y \triangleleft 10 > \wedge \neg(x[y] < 10)$ 은 간략화되어,

$x'[y]=0 \ \wedge \ x=x' < y \lhd 10 > \wedge \ \neg(x[y] < 10)$ 의 형태로 치환되게 된다.

[0073] 검증 조건이 간략화되면, 취약점처리부(120)는 간략화된 검증 조건에 대해 검증을 수행할 수 있으며, 예를 들어, 상술한 바와 같이 SMT 솔버 등을 호출하여 간략화된 검증 조건에 대해 만족 여부를 검증 및 확인하고, 이를 기반으로 취약점 리포트(99)를 생성 또는 갱신할 수 있다.

[0074] 도 5는 일 실시예에 따른 언어모델처리부의 동작을 설명하기 위한 블록도이다.

[0075] 언어모델처리부(150)는, 입력부(101)를 통해 입력되었거나 및/또는 취약점처리부(120)의 동작 결과에 따라 획득한 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)와, 적어도 하나의 언어모델을 기반으로 제2 비용함수를 획득하고, 획득한 제2 비용함수를 취약점처리부(120)로 전달할 수 있다. 취약점처리부(120)는 언어모델처리부(150)가 전달한 제2 비용함수를 이용하여 취약 트랜잭션 시퀀스 후보를 선정할 수도 있다(도 3의 123). 이에 따라 취약점처리부(120)는 취약점을 유발할 가능성이 높은 취약 트랜잭션 시퀀스 후보(들)을 우선적으로 선정 및 탐색할 수 있게 되고, 기호 실행의 탐색 공간이 효과적으로 편향될 수 있게 된다.

[0076] 일 실시예에 의하면, 언어모델처리부(150)는, 도 5에 도시된 바와 같이, 먼저 입력부(101)를 통해 입력되었거나 및/또는 취약점처리부(120)에 의해 결정된 적어도 하나의 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 포함하는 적어도 하나의 취약 트랜잭션 시퀀스 집합({T_1, …, T_n}, (T_i∈T*))을 획득한다(151). 적어도 하나의 취약 트랜잭션 시퀀스(11-1 내지 11-Z)는 복수의 대상 프로그램, 일례로 제1 대상 프로그램 내지 제(K-1) 대상 프로그램(10, 10-1 내지 10-(K-1))에 대한 취약점처리부(120)의 상술한 동작 결과에 따라 획득된 것일 수 있다.

[0077] 이어서 언어모델처리부(150)는, 취약 트랜잭션 시퀀스(11-1 내지 11-Z)를 적절한 형태로 추상화할 수 있다(153). 취약 트랜잭션 시퀀스의 추상화는 학습된 언어모델이 다양한 종류의 새로운 프로그램, 트랜잭션 또는 트랜잭션 시퀀스 등에 대해서도 적절하게 적용될 수 있도록 하기 위해 수행된다. 일 실시예에 의하면, 언어모델처리부(150)는 하기의 수학식 5를 이용하여 취약 트랜잭션 시퀀스(11-1 내지 11-Z, $T_i = t_i^0 \cdots t_i^n$)를 대응하는 추상화된 트랜잭션 시퀀스로 변환함으로써, 추상화된 트랜잭션 시퀀스의 집합(Y⊆T*)을 획득할 수 있다.

[0078] [수학식 5]

$$Y = \{\langle s\rangle\langle s\rangle\alpha_\tau(t_i^0)\cdots\alpha_\tau(t_i^n)\langle e\rangle\langle e\rangle \mid T_i = t_i^0\cdots t_i^n, i \in [1,m]\}$$

[0079]

[0080] 여기서, τ:Type→N(N은 정수 집합)은 타임 빈도 테이블로, 각각의 타입마다 해당 타입을 구비하면서 추상화 이전의 각 트랜잭션에서 할당문을 통해 정의되거나 또는 가정문에서 사용된 변수들의 등장 횟수에 대한 정보를 갖고 있다. τ를 기반으로 각 트랜잭션을 추상화된 형태의 트랜잭션(일례로 단어(ω∈^T))로 변환하는 변환 함수(α_τ : T→^T)는, 예를 들어, 하기의 수학식 6과 같이 정의될 수 있다.

[0081] [수학식 6]

$$\alpha_\tau(t) = \begin{cases} \textbf{if } t = (-, f_0, -, -) \textbf{ then } \langle i\rangle \\ \textbf{else} \langle D_\tau^1(t), \cdots, D_\tau^k(t), U_\tau^1(t), \cdots, U_\tau^k(t), P(t), E(t), X(t)\rangle \end{cases}$$

[0082]

[0083] 이 경우, 변환함수(α_τ)의 변환 결과 출력되는 각 단어(ω)는 실시예에 따라, 가상 시작 단어(<s>, pseudo-start word), 가상 종료 단어(<e>, pseudo-end word), 생성자 함수 경로를 추상화한 생성자 단어(<i>) 및/또는 혹은 상술한 τ에서 상위 k개의 타입들만 고려하여 추상화한 2^k+3 차원을 지닌 부울 벡터(boolean vector) 등을 포함할 수 있다. 상술한 수학식 6에서 $D_\tau^i (1 \le i \le k)$ 는 τ에서 i번째 타입의 전역 상태 변수가 할당문을 통해 정의되고 있는지 여부를 확인하는 술어(predicate)이다. 또한, $U_\tau^i (1 \le i \le k)$ 는 τ에서 i번째 타입의 전역 상태 변수가 가정문을 통해 사용되고 있는지를 여부를 확인하는 술어이다. P(t)는 트랜잭션이 유래된 함수가 지불 가능(payable)에 대한 키워드를 지니고 있는지 확인하는 술어이다. 솔리디티 언어 내에서 지불 가능에 대한 키워드를 갖는 함수는 블록체인 내에서의 가상 화폐(예를 들어, 이더(ether) 등)를 수신할 수 있는 기능을 가지고 있으므로, 스마트 컨트랙트인 프로그램(10, 10-1 내지 10-(K-1))을 분석하는 경우에는 지불 가능에 대한 키워드를 갖는 함수의 존부 여부는 확인될 필요가 있다. B(t)는 트랜잭션이 유래된 함수 경로에 가상 화폐를 송금

하는 함수(일례로 transfer 함수)가 존재하는지를 여부를 확인하는 술어이다. X(t)는 트랜잭션이 유래된 함수 경로에 컨트랙트를 비활성화시키는 함수(예: selfdestruct 함수)가 존재하는지를 확인하는 술어이다. 수학식 6에는 상술한 여러 술어를 모두 포함하는 것으로 기재되어 있으나, 설계자의 선택에 따라서 이들 술어 중 적어도 하나는 생략될 수도 있고 및/또는 이들 외에 다른 술어가 더 추가될 수도 있다.

[0084] 상술한 바와 같이 추상화된 트랜잭션 시퀀스의 집합(Y)이 획득되면, 추상화된 트랜잭션 시퀀스의 집합(Y)을 기반으로 소정의 언어 모델을 이용하여 취약점처리부(120)의 취약 트랜잭션 시퀀스 후보에 대한 탐색 방법이 결정될 수 있다(155). 구체적으로 언어모델처리부(150)는 추상화된 트랜잭션 시퀀스의 집합(Y)를 기반으로 비용함수(즉, 제2 비용함수)를 결정할 수 있으며, 취약점처리부(120)는 제2 비용함수를 이용하여 취약 트랜잭션 시퀀스 후보를 선정함으로써, 모든 취약 트랜잭션 시퀀스 후보에 대한 탐색 순서가 적절하게 변경될 수 있다. 이 경우, 언어 모델은 예를 들어, 통계적 언어 모델(SLM: Statistical Language Model)을 포함할 수 있으며, 보다 구체적으로는 일부의 단어(n개의 단어)를 이용하는 n-gram 언어 모델을 포함할 수 있다. 그러나, 언어 모델은 이에 한정되는 것은 아니며, 설계자는 취약 시퀀스 후보 탐색 순서 결정을 위해 다양한 종류의 언어 모델을 고려 및 적용할 수 있다.

[0085] 언어모델(일례로 n-gram 언어 모델)을 기반으로 제2 비용함수를 결정하기 위해, 먼저 적어도 하나의 트랜잭션 시퀀스(t_0, t_1, …, t_n)는, 적어도 하나의 단어 시퀀스 <s><s>ω_0 … ω_n으로 변환될 수 있다. 이 경우, 변환은 하기의 수학식 7를 통해 표현된 함수 α'_τ:T→T를 이용하여 수행될 수 있다. 이는 알려지지 않은 단어를 기 알려진 단어(ω∈V)로 적절하기 변환하기 위해 마련된 함수이다. 여기서, V(V⊆^T)는 추상화된 트랜잭션 시퀀스의 집합(Y)에 존재하는 모든 추상화된 형태의 트랜잭션(예를 들어, 모든 단어)들의 집합(V={$w_i \mid w_1 \cdots w_n \in Y,\ i \in [1,m]$}이다.

[0086] [수학식 7]

$$\alpha'_\tau(t) = \begin{cases} \alpha_\tau(t) & \text{if } \alpha_\tau(t) \in V \\ \text{argmax}_{w \in V}\, \text{similarity}(\alpha_\tau(t), w) & \text{if } \alpha_\tau(t) \notin V \end{cases}$$

[0087]

[0088] 여기서, 유사도 함수(similarity(ω_1, ω_2))는, 두 단어(ω_1, ω_2) 간의 유사도를 연산하는 함수로, 예를 들어, 하기의 수학식 8과 같이 정해질 수 있다.

[0089] [수학식 8]

$$\text{similarity}(\langle v_1, \cdots, v_{2k+3} \rangle, \langle v'_1, \cdots, v'_{2k+3} \rangle) = \sum_{i=1}^{2k+3} N_i \times v_i \times v'_i$$

[0090]

[0091] 수학식 8에서 N_1, …, N_2k+3은 각각의 특징 벡터(feature vector)에 대한 가중치를 의미한다. 가중치는 설계자 또는 사용자에 의해 결정되되, 함수를 판단함에 있어서 상대적으로 중요한 부분을 강조하기 위해 이용될 수 있다. 예를 들어, 만약 확률을 계산하는데 있어 함수에 지불 가능에 대한 키워드의 존재 유무를 판단하는 것이 중요하다고 판단되는 경우에는, N_1,…N_2k < N2k+1,N_2k+2,N_2k+3가 성립하도록 가중치가 결정될 수 있다. 한편, 주어진 트랜잭션 시퀀스를 단어 시퀀스로 변환할 시 가상 시작 단어 <s>는, 단어 시퀀스 전단에 삽입된다. 한편, 가상 종료 단어 <e>는 삽입되지 않을 수도 있다. 이는 트랜잭션 시퀀스가 주어졌을 때, 그 시퀀스 자체가 취약한 지의 여부를 판단하는 것이 아니라, 해당 시퀀스에 대한 탐색을 지속하는 것이 취약 시퀀스를 찾는데 도움이 되는지 여부를 판단하기 위해서이다. 이 경우, 일 실시예에 의하면, 제2 비용 함수는 하기의 수학식 9와 같이 정의될 수 있다.

[0092] [수학식 9]

$$\text{cost}(t_0, \cdots, t_n) = -\prod_{i=0}^{n} P(w_i \mid w_{i-2} w_{i-1})$$

[0093]

[0094] 수학식 9에서, ω_i-1는 ω_i-2 및 <s>와 동일하고, ω_i는 $w_i = \alpha'_\tau(t_i)$ 로 주어질 수 있다(j∈[0,n]). 상술한 비용 함수 기반 취약 트랜잭션 시퀀스 후보 선정 과정(123)에서 탐색의 효율성을 위해 비용이 적은 후보를 우선적으로 탐색하므로, 수학식 9에서는 음의 확률(negative probability)을 계산한다.

[0095] 일 실시예에 의하면, 알려지지 않은 문맥(unknown context)의 일반화를 위해, 제2 비용 함수를 결정함에 있어서 스무딩(smoothing) 기법을 더 이용하는 것도 가능하다. 예를 들어, 스무딩 기법 중에서 단순 선형 보간법(simple linear interpolation) 등이 이용될 수 있다. 단순 선형 보간법은 n-gram 언어모델에서 n이 1인 경우(1-gram), n이 2인 경우(2-gram) 및 n이 3인 경우(3-gram) 각각에 대한 확률 모델을 적절히 혼합하는 방법이다. 만약 단순 선형 보간법을 이용하는 경우, 수학식 9의 확률 P는 하기의 수학식 10과 같이 주어질 수 있다.

[0096] [수학식 10]

$$P(w_i \mid w_{i-2}w_{i-1}) = \lambda_1 P_{\text{add-k}}(w_i \mid w_{i-2}w_{i-1}) + \lambda_2 P_{\text{add-k}}(w_i \mid w_{i-1}) + \lambda_3 P(w_i).$$

[0097]

[0098] 여기서, 확률 P에 대한 모든 결과값의 합은 1이어야 하므로(즉, $\Sigma_{w_j \in V} P(w_j \mid w_{i-2}w_{i-1}) = 1$), 각 언어 모델의 가중치(λ_1, λ_2, λ_3)들의 합도 마찬가지로 1이다. 한편, n이 2이거나 3인 경우(즉, 2-gram 모델 또는 3-gram 모델)에서의 확률을 계산할 때는 분모가 0이 되는 것을 방지하기 위하여, add-k 스무딩 기법을 이용할 수 있다. add-k 스무딩 기법은, 전체 데이터에 0보다 크고 1보다 작은 k를 더해 제수가 0이되는 것을 방지하는 기법이다. 이 경우, 2-gram 모델 및 3-gram 모델 각각에 대한 확률 P는 하기의 수학식 11 및 수학식 12로 주어질 수 있다.

[0099] [수학식 11]

$$P_{\text{add-k}}(w_i \mid w_{i-2}w_{i-1}) = \frac{Count(w_{i-2}w_{i-1}w_i) + k}{Count(w_{i-1}w_i) + k\,V}$$

[0100]

[0101] [수학식 12]

$$P_{\text{add-k}}(w_i \mid w_{i-2}w_{i-1}) = \frac{Count(w_{i-1}w_i) + k}{Count(w_{i-1}) + k\,V}$$

[0102]

[0103] 한편, n이 1인 경우, 즉, 1-gram 모델을 이용하는 경우, 피제수 및 제수가 모두 0이 되는 경우는 발생하지 않으므로, 스무딩 기법을 이용하지 않을 수도 있다. 이 경우, 1-gram 모델 에서의 확률 P는 하기의 수학식 13과 같이 주어질 수 있다.

[0104] [수학식 13]

$$P(w_i) = \frac{Count(w_i)}{\Sigma_{w_j \in V} Count(w_j)}$$

[0105]

[0106] 이하, 도 6 및 도 7을 참조하여 취약 트랜잭션 시퀀스 획득 방법의 여러 실시예에 대해서 설명하도록 한다.

[0107] 도 6은 일 실시예에 따른 취약 트랜잭션 시퀀스 획득 방법에 대한 흐름도이다.

[0108] 도 6에 도시된 일 실시예에 의하면, 먼저 취약 트랜잭션 시퀀스를 획득하기 위한 적어도 하나의 대상 프로그램에 대해 사전 준비 과정 및 데이터 초기화 과정이 병행하여 또는 순차적으로 수행된다(201). 사전 준비는, 대상 프로그램 내의 적어도 하나의 루프를 언롤링하거나 및/또는 호출될 함수들을 인라인하는 등의 과정을 포함할 수 있다. 또한, 사전 준비는 대상 프로그램에 존재하는 각 함수에 대응하는 함수 경로를 수집하는 과정을 포함할 수도 있으며, 여기서 대상 프로그램은 언롤링 및/또는 인라인 처리된 것일 수 있다. 데이터 초기화 과정은, 취

약 트랜잭션 시퀀스 처리 과정에서 필요한 데이터(셋)를 초기화하는 과정으로, 예를 들어, 취약 트랜잭션 시퀀스 후보군에 대한 집합(워크셋)이나 취약점 리포트 등이 초기화될 수 있다. 실시예에 따라 사전 준비 과정 및 초기화 과정 중 적어도 하나는 생략 가능하다.

[0109] 순차적으로 취약 트랜잭션 시퀀스 후보가 선정될 수 있다(203). 취약 트랜잭션 시퀀스 후보의 선정은 적어도 하나의 비용함수를 이용하여 수행될 수 있으며, 비용함수는, 예를 들어, 선정된 취약 트랜잭션 시퀀스 후보가 비용을 최소화할 수 있도록 정의된 함수일 수 있다. 적어도 하나의 비용함수는 제1 비용함수 및 제2 비용함수 중 적어도 하나를 포함할 수 있다. 제1 비용함수는 길이가 상대적으로 짧은 취약 트랜잭션 시퀀스를 후보로 선정하도록 마련된 것일 수 있다. 제2 비용함수는 언어 모델을 기반으로 획득된 것일 수 있다. 제2 비용함수는, 도 7에 도시된 바와 같은 학습 수행 과정에 의해 획득된 것일 수 있다. 이에 대해선 후술한다.

[0110] 취약 트랜잭션 시퀀스 후보가 선정되면, 취약 트랜잭션 시퀀스 후보에 대한 기호 실행이 수행되고, 이에 따라서 적어도 하나의 검증 조건이 획득될 수 있다(205). 실시예에 따라, 검증 조건이 복수개인 경우 검증 조건 집합이 획득될 수 있다. 검증 조건의 획득을 위해서, 기호 실행의 일례로 소정의 검증 조건 생성 함수(도 4에 도시된 GenerateVC(s)) 등)가 이용될 수 있다. 검증 조건 생성 함수는, 트랜잭션 시퀀스에 대해 단언문 레이블 및 검증 조건의 쌍을 입력받고, 이에 따라 소정의 집합을 출력할 수 있도록 마련된다. 검증 조건 생성 함수는, 예를 들어, 트랜잭션 시퀀스에 대한 기호 실행기를 이용하여 구현될 수 있으며, 트랜잭션 시퀀스에 대한 기호 실행기는 트랜잭션에 대한 기호 실행기를 변형하여 획득된 것일 수도 있다.

[0111] 검증 조건의 획득에 응하여 검증 조건에 대한 검증이 수행될 수 있다. 검증 조건 집합의 각 원소(검증 조건)에 대한 검증이 수행되는데, 이를 위해 검증 조건 집합 순회 종료 여부를 결정할 수 있다(207). 즉, 모든 검증 조건에 대한 검증이 수행된 경우 207 단계가 수행되고, 그렇지 않은 경우 각 검증 조건에 대한 검증이 수행될 수 있다. 이 경우, 검증의 수행은 적어도 하나의 단언문 레이블 및 이에 대응하는 검증 조건의 쌍에 대해 수행될 수도 있다. 실시예에 따라서, 검증 수행 이전에 검증 조건의 간략화가 더 수행되는 것도 가능하다. 검증 조건의 간략화는, 소정의 간략화 규칙을 이용하여 수행될 수 있으며, 예를 들어, 검증과 무관한 논리식을 검증 조건에서 제외하거나 및/또는 수량자 제거 최적화를 통해 수행될 수도 있다.

[0112] 검증 수행에 따라 취약 트랜잭션 시퀀스의 발견 여부가 먼저 판별될 수 있다(209). 만약 취약 트랜잭션 시퀀스가 발견되지 않은 상태라면(209의 예), 검증 조건에 대한 만족 여부가 확인될 수 있다(211). 검증 조건에 대한 만족 여부의 확인은, 적어도 하나의 해석 프로그램을 이용하여 수행될 수 있다. 여기서, 적어도 하나의 해석 프로그램은, 예를 들어, SMT 솔버 등을 포함할 수 있다. 반대로 취약 트랜잭션 시퀀스가 발견된 상태라면(209의 아니오), 검증 조건의 만족 여부 판단 내지 취약점 리포트 갱신과 같은 후술하는 과정(211 내지 215)은 수행되지 않는다. 이 경우, 검증되지 않은 검증 조건에 대한 검증을 위해 207 단계가 재차 수행될 수 있다.

[0113] 만약 SMT 솔버 등을 이용한 검증 조건에 대한 만족 여부의 확인 결과, 만약 검증 조건이 만족된다면(213의 예, 다시 말해서, 분석 중인 취약 트랜잭션 시퀀스 후보에 취약점을 유발하게 하는 경우가 존재한다면), 이에 따라 초기화 되었거나 또는 갱신된 취약점 리포트가 갱신될 수 있다(215). 취약점 리포트의 갱신은, 단언문 레이블에, 취약한 것으로 판명된 트랜잭션 시퀀스와 검증 조건에 대한 만족 모델을 매핑함으로써 수행될 수도 있다. 반대로 검증 조건이 만족되지 않으면(213의 아니오), 취약점 리포트의 갱신(215) 등은 수행되지 않으며 207 단계가 재차 수행될 수 있다. 또한, 취약점 리포트가 갱신된 후에는 207 단계가 재차 수행될 수 있다.

[0114] 207 단계에서 모든 검증 조건에 대한 검증이 수행된 경우, 새로운 취약 트랜잭션 시퀀스 후보군이 생성되고, 생성된 취약 트랜잭션 시퀀스 후보군이 워크셋에 추가될 수 있다(217). 이 경우, 새로운 취약 트랜잭션 시퀀스 후보군의 생성은, 현재의 취약 트랜잭션 시퀀스 후보군에 대해 새로운 트랜잭션을 추가함으로써 수행될 수도 있다.

[0115] 상술한 과정(203 내지 217)은 적어도 일회 이상 반복될 수 있다. 상술한 과정(203 내지 217)의 반복은 미리 정의된 조건이 만족될 때까지 수행될 수 있다. 예를 들어, 상술한 과정(203 내지 217)의 반복은 워크셋 내에 어떠한 취약 트랜잭션 시퀀스 후보군이 존재하지 않거나, 모든 단언문(들)에 대한 취약 트랜잭션 시퀀스가 발견되거나, 및/또는 미리 정의된 시간이 경과된 경우에 종료될 수 있다.

[0116] 이하 상술한 제2 비용함수를 획득하는 과정을 설명하도록 한다.

[0117] 도 7은 일 실시예에 따른 언어 모델을 기반으로 학습을 수행하는 과정에 대한 상세 흐름도이다.

[0118] 도 7에 도시된 언어 모델을 기반으로 학습을 수행하여 제2 비용함수를 획득하는 과정의 일 실시예에 의하면, 먼저 취약 트랜잭션 시퀀스의 집합이 획득된다(220). 집합 내의 취약 트랜잭션 시퀀스는 도 6을 통해 설명한 과정

을 통해 취약한 것으로 판명된 취약 트랜잭션 시퀀스 후보를 포함할 수 있다. 집합 내의 취약 트랜잭션 시퀀스는, 언어 모델을 기반으로 학습을 수행하는 장치에 의해 획득된 것일 수도 있고, 및/또는 외부의 다른 장치에 의해 획득된 것일 수도 있다.

[0119] 집합 내의 취약 트랜잭션 시퀀스는 적절한 형태로 추상화 되고, 이에 따라 추상화된 취약 트랜잭션 시퀀스가 획득될 수 있다(222). 추상화된 취약 트랜잭션 시퀀스의 획득을 위해 상술한 수학식 5 및 수학식 6이 이용될 수도 있다.

[0120] 추상화된 트랜잭션 시퀀스의 집합이 획득되면, 추상화된 트랜잭션 시퀀스의 집합 및 언어 모델을 이용하여 제2 비용함수가 결정될 수 있다(224). 여기서, 언어 모델은 통계적 언어 모델을 포함할 수 있으며, 보다 구체적으로는 n-gram 언어 모델을 포함할 수도 있다. 이 경우, 제2 비용함수는 상술한 수학식 9와, 수학식 10 내지 수학식 13 중 어느 하나를 이용함으로써 결정될 수도 있다.

[0121] 제2 비용함수는 현재 취약 트랜잭션 시퀀스를 검출하는 논리적 부분이나 물리적 장치로 전달되거나, 추후 취약 트랜잭션 시퀀스 획득 장치의 프로세서에 의해 이용되기 위해 저장부에 저장될 수 있다(226).

[0122] 상술한 실시예에 따른 스마트 컨트랙트 내의 취약 트랜잭션 시퀀스 획득 방법은, 컴퓨터 장치에 의해 구동될 수 있는 프로그램의 형태로 구현될 수 있다. 여기서 프로그램은, 프로그램 명령, 데이터 파일 및 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 프로그램은 기계어 코드나 고급 언어 코드를 이용하여 설계 및 제작된 것일 수 있다. 프로그램은 상술한 스마트 컨트랙트 내의 취약 트랜잭션 시퀀스 획득 방법을 구현하기 위하여 특별히 설계된 것일 수도 있고, 컴퓨터 소프트웨어 분야에서 통상의 기술자에게 기 공지되어 사용 가능한 각종 함수나 정의를 이용하여 구현된 것일 수도 있다. 또한, 여기서, 컴퓨터 장치는, 프로그램의 기능을 실현 가능하게 하는 프로세서나 메모리 등을 포함하여 구현된 것일 수 있으며, 필요에 따라 통신 장치를 더 포함할 수도 있다.

[0123] 상술한 스마트 컨트랙트 내의 취약 트랜잭션 시퀀스 획득 방법을 구현하기 위한 프로그램은, 컴퓨터에 의해 판독 가능한 기록 매체에 기록될 수 있다. 컴퓨터에 의해 판독 가능한 기록 매체는, 예를 들어, 솔리드 스테이트 드라이브(SSD), 롬, 램 또는 플래시 메모리 등과 같은 반도체 저장 장치, 하드 디스크나 플로피 디스크 등과 같은 자기 디스크 저장 매체, 콤팩트 디스크나 디브이디 등과 같은 광 기록 매체, 플롭티컬 디스크 등과 같은 자기-광 기록 매체 및 자기 테이프 등 컴퓨터 등의 호출에 따라 실행되는 특정 프로그램을 저장 가능한 적어도 한 종류의 물리적 장치를 포함할 수 있다.

[0124] 이상 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법의 여러 실시예에 대해 설명하였으나, 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법은 오직 상술한 실시예에 한정되는 것은 아니다. 해당 기술 분야에서 통상의 지식을 가진 자가 상술한 실시예를 기초로 수정 및 변형하여 구현 가능한 다양한 장치나 방법 역시 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법의 일례가 될 수 있다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성 요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나 다른 구성 요소 또는 균등물에 의하여 대치되거나 또는 치환되더라도 상술한 프로그램 내의 취약 트랜잭션 시퀀스 획득 장치 및 방법의 일 실시예가 될 수 있다.

[0126] 이상에서 설명된 장치는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨어 구성 요소의 집합으로 구현될 수 있다. 예를 들어, 실시 예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 콘트롤러, ALU(Arithmetic Logic Unit), 디지털 신호 프로세서(Digital Signal Processor), 마이크로컴퓨터, FPA(Field Programmable 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(Operation System, OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 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 기술 분야에서 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.

[0127] 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 처

리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성 요소(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.

[0128] 실시 예에 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 실시 예를 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), 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 실시 예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

부호의 설명

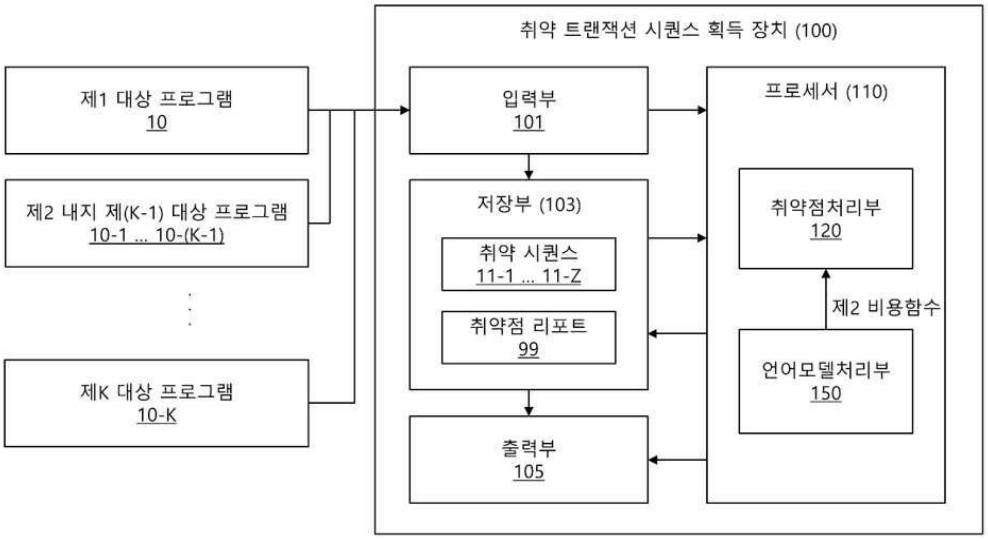
[0129] 100: 취약 트랜잭션 획득 장치 101: 입력부

103: 저장부 105: 출력부

110: 프로세서

도면

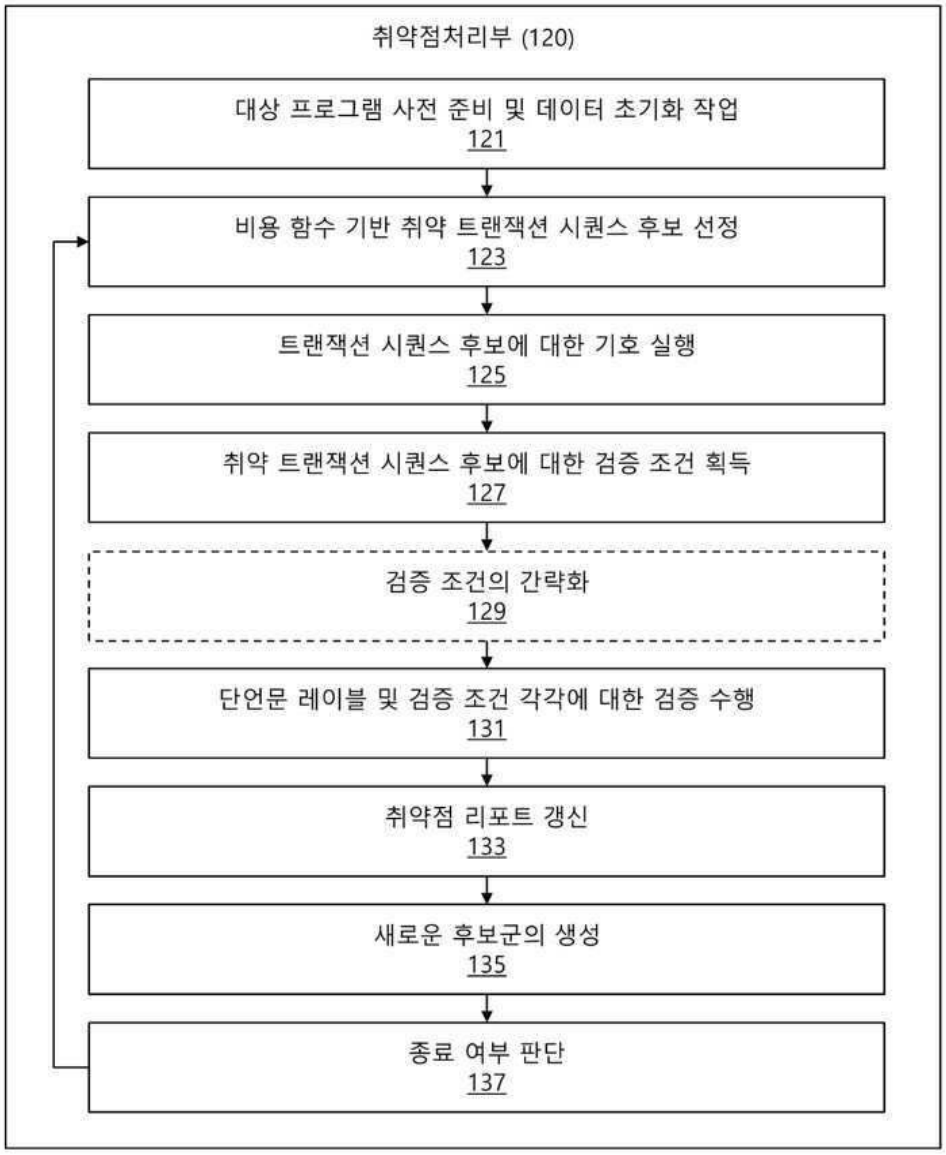
도면1



도면2

$$c \in C ::= G^* F^*, \qquad F ::= f(x)\{S\}$$
$$a \in A ::= x := E \mid x[y] := E \mid assume(B) \mid assert^l(B)$$
$$s \in S ::= A \mid if\ B\ S_1\ S_2 \mid while\ B\ S \mid S_1; S_2$$

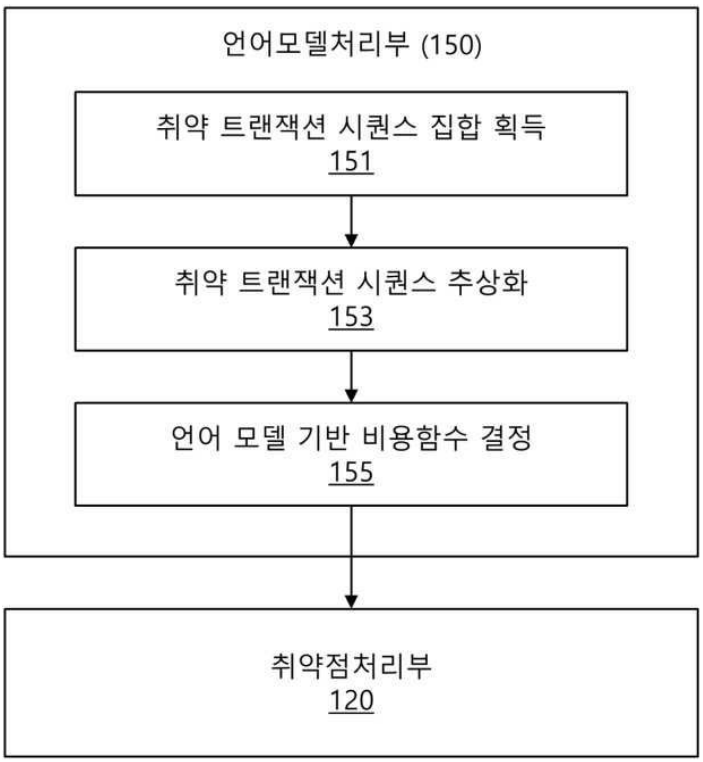
도면3


취약점처리부 (120)
대상 프로그램 사전 준비 및 데이터 초기화 작업
121
비용 함수 기반 취약 트랜잭션 시퀀스 후보 선정
123
트랜잭션 시퀀스 후보에 대한 기호 실행
125
취약 트랜잭션 시퀀스 후보에 대한 검증 조건 획득
127
검증 조건의 간략화
129
단언문 레이블 및 검증 조건 각각에 대한 검증 수행
131
취약점 리포트 갱신
133
새로운 후보군의 생성
135
종료 여부 판단
137

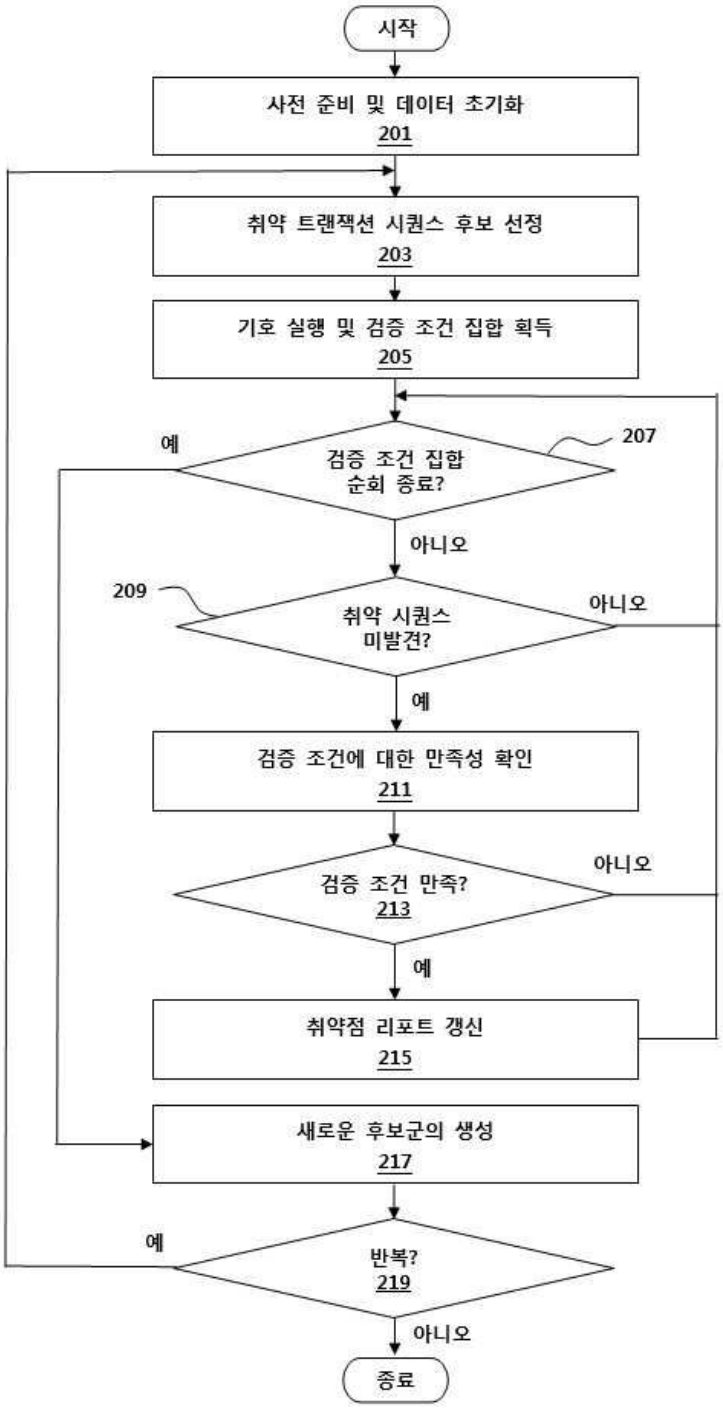
도면4

1: Unroll loops and inline function calls in c
2: $P \leftarrow$ The set of function paths in c
3: $W \leftarrow \{(id,\ f_0,\ x,\ a) \mid (f_0,\ x,\ a) \in P,\ \text{new } id\}$
4: $R \leftarrow \lambda l.\bot$
5: **repeat**
6: $s \leftarrow \text{argmin}_{w \in W}\ \text{cost}(w)$
7: $W \leftarrow W \setminus \{s\}$
8: $\Pi \leftarrow \text{GenerateVC}(s)$
9: **for each** $(l, VC) \in \Pi$ **do**
10: **if** $R(l) = \bot$ **then**
11: **if** $SAT(VC)$ **then** $R \leftarrow [l \mapsto (s,\ \text{model}(VC))]$
12: **end for**
13: $W \leftarrow \{s \cdot (i,f,x,a) \mid (f,x,a) \in P,\ f \neq f_0,\ \text{new } i\} \cup W$
14: **until** $W = \varnothing$ or $\forall l.R(l) \neq \bot$ or timeout
15: **return** R

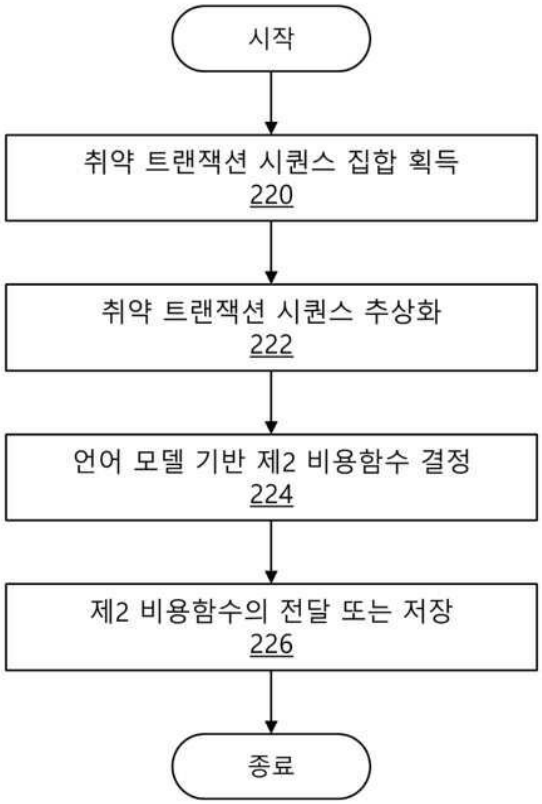
도면5



도면6


시작
사전 준비 및 데이터 초기화
201
취약 트랜잭션 시퀀스 후보 선정
203
기호 실행 및 검증 조건 집합 획득
205
207
예
검증 조건 집합
순회 종료?
아니오
209
취약 시퀀스
미발견?
아니오
예
검증 조건에 대한 만족성 확인
211
검증 조건 만족?
213
아니오
예
취약점 리포트 갱신
215
새로운 후보군의 생성
217
예
반복?
219
아니오
종료

도면7


시작
취약 트랜잭션 시퀀스 집합 획득
220
취약 트랜잭션 시퀀스 추상화
222
언어 모델 기반 제2 비용함수 결정
224
제2 비용함수의 전달 또는 저장
226
종료