Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.6, Dec. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.6.1103

TCST : 신뢰실행환경 내에서 스마트 컨트랙트의 제어 흐름 무결성 검증을 위한 기술*

박 성 환,[1†] 권 동 현[2‡]
[1,2]부산대학교 (대학원생, 교수)

TCST : A Technology for Verifying Control Flow Integrity for Smart Contracts within a Trusted Execution Environment*

Seonghwan Park,[1†] Donghyun Kwon[2‡]
[1,2]Pusan National University (Graduate student, Professor)

요 약

블록체인 기술은 일상생활을 비롯한 여러 산업 분야에서 활용도가 증가하고 있으며, 이는 분산원장 기술을 통하여 네트워크 참여자들 간의 거래 내역에 대한 정보의 무결성과 투명성을 보장한다. 무결성과 투명성을 보장하는데 가장 중요한 요소인 분산원장은 스마트 컨트랙트를 통하여 수정, 관리된다. 그러나, 스마트 컨트랙트 또한 블록체인 네트워크의 구성요소인 만큼 참여자들에게 투명하게 공개되고 있었으며 이로 인해 취약점이 쉽게 노출될 수 있었다. 이러한 단점을 보완하기 위하여 신뢰실행환경을 활용하여 기밀성을 보장하는 연구가 다양하게 진행되었으나, 실행된 스마트 컨트랙트의 무결성을 보장하기에는 어려움이 따른다. 해당 논문에서는 이러한 문제를 해결하기 위하여 신뢰실행환경내에서 실행되는 스마트 컨트랙트의 제어 흐름 무결성을 검증 함으로써 스마트 컨트랙트의 기밀성과 무결성을 동시에 제공하는 것을 목표로 한다.

ABSTRACT

Blockchain technology is widespread in everyday life and various industry fields. It guarantees integrity and transparency between blockchain network participants through a distributed ledger. The smart contract is modifying and managing the distributed ledger, which is the most important component of guaranteeing integrity and transparency of blockchain network. Still, smart contracts are also a component of blockchain networks, it is disclosed to network participants transparently. For this reason, the vulnerability of smart contracts could be revealed easily. To mitigate this, various studies are leveraging TEE to guarantee the confidentiality of smart contracts. In existing studies, TEE provides confidentiality of smart contracts but guaranteeing the integrity of smart contracts is out of their scope. In this study, we provide not only the confidentiality of smart contracts but also their integrity, by guaranteeing the CFI of smart contracts within TEE.

Keywords: Blockchain, CFI, IoT, TrustZone, Hyperledger fabric

Received(09. 16. 2022), Modified(10. 14. 2022), Accepted(11. 15. 2022)

* 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학 ICT연구센터육성지원사업의 연구결과로 수행되었음 (IITP-2022-2020-0-01797)

* 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(IITP-2022-0-01201)

† 주저자, starjara@pusan.ac.kr

‡ 교신저자, kwondh@pusan.ac.kr(Corresponding author)

I. 서 론

블록체인 네트워크는 스마트 컨트랙트를 통해 공유원장에 참여자들의 거래 내역을 기록함으로써 거래 내역에 대한 투명성과 무결성을 보장한다. 공유원장이란 참여자들이 각각의 원장에 동일한 내용을 기록함으로써 통일성의 유지를 보장하며 이러한 통일성으로 인하여 개인이 소유한 원장의 내용을 수정하여 악의적인 행위를 하는 경우 통일성 보장을 위한 합의 프로토콜을 통하여 탐지해낼 수 있다. 그러나 원장에 기록하는 행위는 원장을 수정하고 관리하는 프로그램인 스마트 컨트랙트를 통해 자동으로 이루어지며 이를 통해 참여자들이 동일한 내용의 공유원장을 가질 수 있도록 보장한다. 그러나 블록체인 네트워크의 투명성으로 인해 스마트 컨트랙트의 내용과 동작 상태 등도 참여자에게 투명하게 공개되며[1], 이는 스마트 컨트랙트의 기밀성 보장을 어렵게 한다는 단점도 가져왔다.

이러한 단점을 보완하기 위해 일반실행환경(Rich Execution Environment, REE)과 격리된 실행환경인 신뢰실행환경(Trusted Execution Environment, TEE)내에서 스마트 컨트랙트를 수행함으로써 기밀성을 보장하는 연구가 다수 진행되었으며, 기밀성을 보장함으로써 스마트 컨트랙트의 투명성에 대한 문제는 해결되었으나 수행 과정과 동작 상태에 대한 정보가 가려짐에 따라 수행의 무결성을 확인하는 것이 힘들어졌다. 이에 관해 Intel software guard extentions (Intel-SGX)[2]를 사용하는 경우에는 해당 보안기법이 제공하는 무결성 검증 메커니즘을 통해 일부 해결할 수 있으나, 이는 서버 등급의 프로세서에 제한된다는 단점이 존재한다. 이에 반해 임베디드기기에 많이 사용되는 ARM-TrustZone (ARM-TZ)[3]의 경우 이러한 기능을 전혀 지원하지 않기에 이러한 문제가 더욱 심각하다고 할 수 있다.

스마트 컨트랙트의 무결성 보장이 충분히 이루어지지 않을 경우, 공격자는 스마트 컨트랙트를 공격함으로써 트랜잭션의 결과를 임의의 값으로 변경하거나, 시스템에 치명적인 피해를 입힐 수 있게 된다[4]. 또한, TEE 내에서 공격이 성공할 경우 지문인증과 은행 업무와 같이 높은 보안성을 요구하는 응용 프로그램을 실행을 위하여 제공된다는 TEE의 특성으로 인하여 더욱 치명적일 수 있을 것이다.

이에 따라 해당 연구에서는 스마트 컨트랙트의 실행환경이 모바일 엣지 디바이스로 옮겨감에 따라 많은 임베디드 기기에서 제공하는 ARM-TZ를 대상으로 하여 스마트 컨트랙트의 제어 흐름 무결성을 검증하는 것을 목표로 한다. 제어 흐름 무결성을 검증함으로써 함수 재진입과 같이 제어 흐름을 변경하여 일어날 수 있는 공격[5],[6]을 탐지해내고 수행결과에 대한 무결성을 제공하는 것을 목표로 한다.

비록 제어 흐름 무결성을 활용하는 다양한 선행연구가 존재하나, 이를 임베디드 기기의 TEE에서 실행되는 스마트 컨트랙트를 대상으로 하기에는 다음과 같은 제약사항이 있다. 첫째로 제한된 메모리와 계산능력이다. 임베디드 기기는 본래 고성능으로 작동하는 기기가 아님에 따라 자원적인 한계가 존재한다. TEE는 제한적인 자원을 REE와 나누어 사용한다는 제약조건에 따라 실행 시 필요한 메모리양과 제어 흐름 무결성을 검증하는데 필요한 계산량을 최소화해야 할 필요가 있다. 또한, 계산량을 최소화함으로써 처리량을 늘리고 블록체인 네트워크의 가용성 저하를 최소화해야 할 것이다. 다음으로는 실행 중 기록되는 제어 흐름 정보에 대한 보안성을 보장해야 한다는 것이다.

이에 따라 본 연구에서는 Hyperledger fabric[7]을 대상으로 TEE에서 수행되는 스마트 컨트랙트의 기본 블록을 단위로 하여 해시값을 통해 제어 흐름을 기록하고, 화이트 리스트를 기반으로 효율적으로 수행 무결성을 검증하는 기술인 TCST를 제안한다.

II. 배경 지식

2.1 Hyperledger fabric

Hyperledger fabric이란 리눅스 재단에서 오픈소스로 제공하는 권한 기반 블록체인 플랫폼으로써 기업 간의 거래 내역 및 유통망에 대한 정보를 기록하기 위하여 많이 사용되고 있다. Hyperledger fabric의 특징은 권한 기반의 플랫폼이라는 점과 스마트 컨트랙트가 전용 언어가 아닌 일반 프로그래밍 언어로 작성된다는 점, 3단계로 구성된 트랜잭션 흐름이 있다.

Hyperledger fabric의 트랜잭션 흐름은 발의, 실행, 반영 3개의 과정으로 이루어진다. 첫 번째로 발의 단계에서는 외부의 클라이언트가 트랜잭션을 발의하며, 이는 인증 노드들로 전송된다. 다음으로 실

행 과정에서는 트랜잭션을 전송받은 인증 노드들이 트랜잭션의 파라미터값들을 인자로 하여 체인 코드를 호출하게 되며 수행결과가 적법하다면 서명을 반환한다. 마지막 반영 단계에서는 반환된 값들이 합의 프로토콜을 만족한다면 클라이언트는 발의한 트랜잭션을 모든 피어에게 전송하며 전송받은 피어는 해당 트랜잭션을 원장에 추가한다.

2.2 ARM-TrustZone

ARM-TZ는 ARM에서 제공하는 TEE를 위한 기능으로 실행환경을 Normal world(REE)와 Secure world(TEE)로 나누어 각각 격리된 실행환경을 제공한다. 격리된 실행환경은 독립된 메모리 및 주변기기 자원과 커널을 가지며, 시스템 상태 레지스터를 통해 실행환경을 확인하고 접근의 적법성 여부를 판단한다.

각 실행환경 사이에는 공유메모리를 통해 호출 인자와 필요 데이터를 전달하며, SMC(Secure Monitor Call)를 통해 각 실행환경을 전환한다. 또한, REE에서 TEE의 자원에 접근하는 것을 막기 위해 TEE는 REE보다 높은 권한 수준을 가지며, 이는 Secure boot 과정을 통해 보장된다.

OP-TEE[8]는 널리 사용되고 있는 오픈 소스 실행환경으로, ARM-TZ를 활용해 TEE를 제공한다. OP-TEE는 TEE 내에서 신뢰 애플리케이션(Trusted application, TA)만을 실행하며, TA를 격리된 실행환경에서 수행함으로써 중요한 정보 또는 프로그램에 대한 기밀성을 보장할 수 있다. 현재 TA 작성에는 현재 C언어만이 지원되고 있으며, 하나의 TEE 보안 커널 위에서 다수의 TA가 실행된다.

2.3 제어 흐름 무결성

제어 흐름 무결성(Control flow integrity, CFI)[9]는 프로그램의 제어 흐름의 무결성을 보장함으로써, 프로그램의 제어 흐름을 변경하는 다양한 공격으로부터 보호한다. 제어 흐름의 무결성은 프로그램의 비정상적인 제어 흐름을 탐지함으로써 보장할 수 있으며, 탐지를 위한 방법으로는 라벨링[10], 섀도우 스택[11], 제어 흐름에 대한 해시값[12] 등 다양한 방법이 사용될 수 있다.

III. 위협 모델

해당 논문에서 다음과 같은 위협 모델을 가정한다. 공격자는 블록체인 네트워크의 채널을 확보해 인증 노드에 공격 페이로드를 전달할 수 있다고 가정한다. 공격자는 이를 통해 REE 또는 TEE에서 동작하는 소프트웨어 또는 스마트 컨트랙트 내에 존재하는 취약점을 악용하여 제어 흐름 변조와 같은 공격을 통하여 제어 흐름 무결성을 해칠 수 있다고 가정한다. 단, 이러한 공격으로부터 TEE내의 보안 커널은 TCB(Trusted Computing Base)로써 신뢰한다. 마지막으로 스마트 컨트랙트는 외부 트랜잭션과 상호작용이 없다고 가정하며 하드웨어 부 채널 공격 및 서비스 거부 공격과 같은 공격들은 고려하지 않는다.

IV. 디자인

4.1 TCST 개요

TCST는 TEE 내에서 실행되는 스마트 컨트랙트의 제어 흐름 무결성을 검증하기 위한 연구로 프로그램을 기본 블록 단위로 나누어 각 기본 블록을 벗어나기 전에 마지막으로 실행될 전송 제어 명령어의 주소를 누적해서 해시 함으로써 전송 제어 흐름에 대한 정보를 기록하게 된다. 이러한 기능을 제공하기 위하여 프로그램 실행 중 주소를 받아와 이전 해시값과 함께 누적하여 해시 하는 기능과 실행이 종료된 후 REE로 돌아가기 전에 실행 중 생성된 값을 사전에 알려진 화이트 리스트와 비교를 비롯해 비교 연산의 수행 여부를 결정하기 위한 해시값 확인과 사용된 해시값을 지우기 위한 연산을 포함한다.

Fig. 1.은 TCST의 구조를 도식화한 것으로, 그림의 TEE user space 영역은 TEE의 사용자 영역을 나타내며, REE user space 영역은 REE의 사용자 영역을 나타낸다. Secure kernel은 TEE의 보안 커널 영역을 나타내며 Embedded kernel은 REE의 커널 영역을 나타낸다. TEE의 사용자 영역에는 TCST 패스를 적용하여 생성한 스마트 컨트랙트 프로그램이 로드되며, 해당 프로그램은 각 기본 블록을 벗어나는 명령어 직전에 SVC 명령어가 추가되어, 기본 블록을 벗어나기 전에 기본 블록의 마지막 명령어 주소를 인자로 하여 TCST 모듈에 해시를 요청한다. TCST 모듈은 이전 해시값과 전달받은 주소를 함께 해시 함으로써 값을 누적해가며, 해당

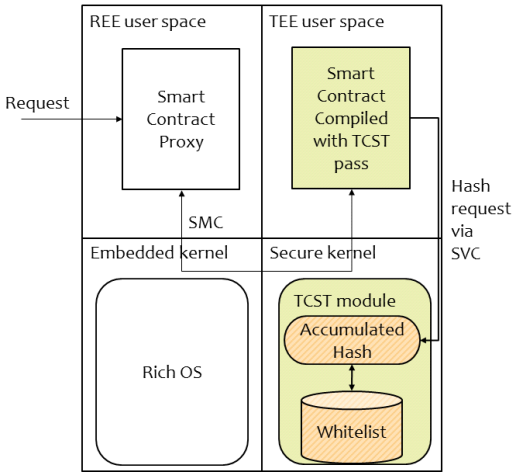

Fig. 1. TCST design overview

그림에서 확인할 수 있듯 생성되는 해시값은 TEE의 보안 커널 영역에 저장된다. 이로 인해 공격자는 TEE의 사용자 공간을 장악하더라도 제어 흐름 무결성 검증을 수행하는데 필요한 민감 정보를 확인할 수 없으며, 이는 공격표면을 최소화하고 실행 중 생성되는 민감 정보의 보안성을 높이는 효과를 가져온다. 또한, 프로그램의 수행이 종료된 후 비교에 사용하게 될 화이트 리스트 역시도 TCST 모듈 내에 기록되어 있어 외부에서는 그 값을 확인할 수 없다.

4.2 TCST의 동작

TCST는 실행 중 일어나는 전송 제어 명령어의 주소를 누적하여 해시 함으로써 제어 흐름 정보를 저장하고 이를 화이트 리스트와 비교해 제어 흐름의 무결성을 검증하는데 이는 해시 초기화, 누적 해시 생성, 제어 흐름 적법성 검사와 같은 동작으로 이루어진다.

해시값의 초기화는 스마트 컨트랙트의 실행 전, 실행 후 각각 1회씩 이루어지며 이를 통해 해시값이 계속해서 누적되어 올바르지 못한 결과가 생성되거나 값이 유출되는 일을 방지한다.

누적 해시 생성의 경우 스마트 컨트랙트 실행 중 기본 블록을 벗어나는 전송 제어 명령의 수행 직전에 동작한다. 생성되는 값을 보안 커널의 메모리 영역에 저장하기 위하여 전송 제어 명령의 주소값을 인자로 하여 SVC(Super Visor Call)를 호출하여 이루어지며, 이를 위하여 새로운 SVC 번호인 71을 부여하고 이에 대응하는 SVC 서비스를 작성하였다. 또

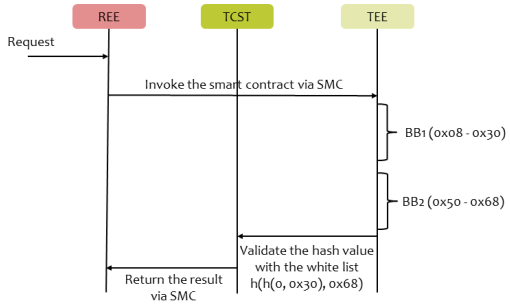

Fig. 2. TCST behavior model

한, SVC 호출의 원자성을 보장하기 위하여 기존 레지스터에 SVC 번호를 저장한 뒤 라이브러리를 통해 호출하는 방식이 아닌 명령어의 Imm필드를 활용해 번호를 디코딩하여 호출하는 방식으로 동작한다. 해시를 수행하기 위한 SVC 명령어는 컴파일을 하는 과정에서 기본 블록을 벗어나는 전송 제어 명령어 직전에 자동으로 삽입된다.

제어 흐름 적법성 검사는 TEE에서 요청받은 실행을 완료한 후 REE로의 컨텍스트 전환이 일어나는 과정에서 자동으로 이루어지며 무결성 검증에 실패할 경우 보안 커널이 패닉 모드에 들어가 수행을 종료시킨다.

Fig. 2.은 TCST의 동작을 간략하게 나타낸 것으로 먼저 REE에서 외부로부터 스마트 컨트랙트의 호출을 요청받아 TEE의 스마트 컨트랙트를 호출한다. TEE내에서 스마트 컨트랙트를 실행하는 동안 전송 제어 명령어가 실행된 주소를 누적하여 해시한다. 해당 그림에서는 이러한 과정이 BB1과 BB2에서 이루어져 각 기본 블록의 마지막 주소인 0x30과 0x68이 누적하여 해시된 결괏값이 생성된 것을 확인할 수 있다. 마지막으로 TEE내에서의 실행이 완료되고 결괏값을 REE로 돌려주기 위한 컨텍스트 전환 과정에서 사전에 지급된 화이트 리스트를 기반으로 생성된 결괏값에 대한 검증이 이루어진다.

V. 구 현

TCST의 보안 기능을 위한 구현사항은 누적 해시를 수행하고 생성된 해시값과 화이트 리스트를 비교해 무결성을 검증하기 위한 보안 커널과 스마트 컨트랙트의 컴파일 타임에 SVC 명령어의 삽입을 자동화하기 위한 컴파일러 구현으로 나뉜다.

5.1 TCST 커널 모듈

보안 커널은 OP-TEE의 보안 커널을 활용하였으며, 누적 해시와 무결성 검증을 수행하기 위한 부분들이 새로이 구현되었다. 누적 해시의 경우 스마트 컨트랙트로부터 SVC 호출을 통해 수행되며, 이를 위하여 71번 SVC가 추가되었다. 해당 SVC는 다음 전송 제어 명령어의 주소를 인자로 하여 호출된다. 기존의 SVC 호출은 레지스터에 SVC 번호를 저장하여 라이브러리를 통해 호출하였으나, 이는 레지스터의 값이 변경될 경우 임의의 SVC를 호출할 위험이 있으며 이를 방지하기 위해 새로 추가된 SVC의 호출 번호를 명령어의 Imm 필드에 직접 삽입하여 라이브러리를 거치지 않고 커널을 직접 호출하여 수행한다.

이러한 동작을 수행하기 위해 보안 커널 영역에 해시 기록을 위한 자료구조를 추가하였으며 이는 32바이트로 이루어진 해시 기록부와 4바이트 크기의 주소부로 이루어져 있다. 해시 연산은 입력값의 길이와 관계없이 일정한 길이의 출력값을 생성한다. 이러한 점으로 인해 위와 같은 자료구조에 대해 해시 연산을 수행함으로써 전송 제어 명령어의 주소를 누적 해시를 할 수 있으며 이는 언제나 32바이트의 일정한 크기로 저장되어 TEE의 메모리 사용량을 줄일 수 있다.

무결성 검증을 위한 동작은 스마트 컨트랙트를 수행한 후 REE 영역으로 결괏값을 반환하는 과정에서 자동으로 수행되며, 기존의 TA들은 해시값을 생성하지 않기 때문에, 해시값의 존재 여부를 판단해 무결성 검증 과정을 수행 여부를 판단한다. 이를 통하여 기존 TA들과의 호환성을 제공하며 새롭게 수정된 스마트 컨트랙트의 제어 흐름 무결성을 검증할 수 있게 하였다.

5.2 TCST 컴파일러 패스

LLVM 컴파일러를 기반으로 하여 TCST를 적용하기 위해 SVC를 자동으로 삽입하는 기능을 추가하였으며. 해당 컴파일러를 통해 스마트 컨트랙트의 컴파일을 수행함으로써 컴파일 과정 중 자동으로 프로그램의 각 기본 블록의 전송 제어 명령어 직전에 새로 추가된 SVC를 삽입함으로써 별도의 수고를 들이지 않고 TCST의 보안 기능을 활용할 수 있다. 또한, 최적화 과정 중 추가된 명령어가 지워지는 일을 방지하기 위하여 컴파일러의 ARM-64bit 아키텍처를 위한 프런트 엔드 중 마지막 패스에 추가하였다.

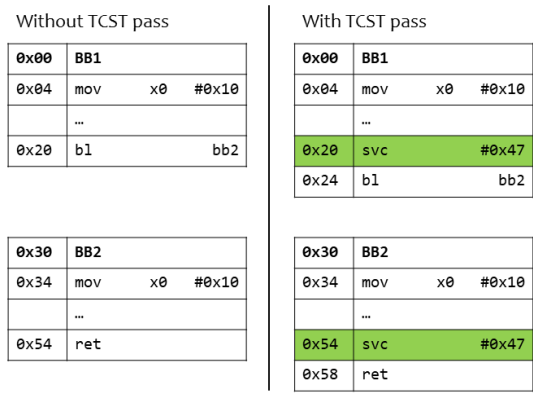

Fig. 3. Left side column presents a TA compiled without TCST pass, right column presents TCST presents compiled with TCST pass. It shows added instruction (svc #0x47) at address 0x20 and 0x54.

Fig. 3.은 TCST 패스가 적용된 컴파일러로 인해 생기는 변경점을 간략하게 나타나는 그림으로, 좌측의 그림은 TCST 패스가 적용되지 않은 기존 코드이며, 우측은 TCST 패스가 적용된 컴파일러로 생성한 코드이다. TCST 패스를 적용함으로써 프로그램 기본 블록의 수행을 끝내는 명령어와 함수 실행의 마지막 명령어 직전에 SVC #0x47 명령어가 삽입됐으며, 이를 녹색 배경(0x20, 0x54)으로 표현하였다.

VI. 실험 결과

6.1 실험 환경

TCST의 성능을 평가하기 위하여 기존의 기밀성만 제공하던 스마트 컨트랙트와 TCST를 통해 기밀성과 무결성 검증을 모두 제공할 때의 성능 및 생성되는 코드의 크기를 비교한다. 실험에는 라즈베리 파이 3B+ 모델이 사용되었으며, 해당 모델은 1.4GHz로 동작하는 ARM Cortex-A53 코어 4개와 1GB의 램으로 구성된다. 라즈베리 파이와 클라이언트 간에는 100Mbit/s 이더넷으로 연결된다. TEE의 보안 커널로는 3.12 버전의 OP-TEE가 사용되었으며, 스마트 컨트랙트의 컴파일에는 12.0.1 버전의 LLVM 컴파일러와 TCST 패스가 추가된 동일 컴파일러를 사용했다.

6.2 코드 크기 평가

코드 크기 평가를 수행하기 위하여 TCST 패스를 적용하여 생성된 바이너리와 기존 바이너리의 코드 크기를 비교하였다. llvm-size 툴을 사용해 각 바이너리의 섹션 별 크기 및 전체 크기를 구했으며, 이를 비교를 하여 코드 크기에 대한 오버헤드를 평가하였다. 또한, 구현과 실험의 편의를 위하여 컴파일 과정에서의 최적화 옵션을 모두 제거하였다.

Table. 1.은 산출된 코드 크기를 표로 정리하여 나타낸 것으로 Legacy 기존의 바이너리, TCST는 TCST 패스를 적용하여 생성한 바이너리에 대한 결괏값을 나타낸다. TCST 컴파일 패스를 적용함으로써 추가적인 명령어가 삽입되어 text 섹션의 크기가 2.63%가량 증가한 것을 확인할 수 있으며 이를 제외한 다른 섹션의 크기는 모두 동일하다는 것을 확인할 수 있다. 또한, 해당 증가량은 바이너리 전체 코드 크기로 비교할 경우 1.45%의 적은 코드 크기 오버헤드만으로 TCST가 적용된 바이너리 파일을 생성할 수 있음을 알 수 있다.

Table. 1. Code size overhead table. Size presented as byte unit. Legacy row shows code size of the smart contract compiled without TCST pass. TCST row shows code size of the smart contract with TCST pass.

	text	data	bss	total
Legacy	54487	700	45200	100387
TCST	55959	700	45200	101859
Overhead	2.63%	0%	0%	1.45%

6.3 TCST 내부 연산 성능 평가

TCST 내부 연산의 성능을 평가하기 위하여 보안 커널 내에 제어 흐름 무결성 검증을 위해 추가된 코드들의 실행시간을 측정하였다.

Table. 2.는 추가된 연산을 수행하는데 필요한 시간을 나타낸 표로, 첫 번째 열은 연산의 종류 두 번째 열은 해당 연산을 수행하는데 걸린 시간을 us 단위로 보여주고 있다. 각 연산에 필요한 시간을 측정하기 위하여 각 연산을 100만 회 반복 수행한 뒤 평균값을 생성하였다. 연산 수행에 걸린 시간은 커널의 Physical count register의 값과 Physical counter의 주기 19.2MHz를 기준으로 소모시간을 계산하였다. Context switching의 경우 SVC 호출을 통하여 커널 컨텍스트로 전환한 후 다시 사용자 컨텍스트로 돌아오기까지 소모된 시간을 측정한 값으로 1.874us의 시간이 소모됐다. Hashing의 경우 누적 해시를 수행하여 해시값을 생성하는 연산으로 18.854us의 시간을 소모하였다. Validating은 TEE에서의 연산이 끝난 후 REE로 결과를 반환할 때 생성된 해시값과 화이트 리스트의 값을 비교하여 무결성을 검증하는 연산으로 1.562us의 소요시간을 보였으며, Clean과 Empty는 각각 해시값을 비워내고 해시값이 비워졌는지 확인하는 연산으로 0.156us의 시간을 소모하였다.

Table. 2. Micro evaluation result table. Column Name shows operation type. Column Time shows took time of the operation by microsecond unit.

Name	Time(us)
Context Switching	1.874
Hashing	18.854
Validating	1.562
Clean	0.156
Empty	0.156

6.4 스마트 컨트랙트 연산 성능 평가

응답시간에 대한 오버헤드를 평가하기 위하여 기존 기밀성을 제공하기 위한 연구[13]에 사용된 스마트 컨트랙트의 소스 코드를 사용하였으며, 1천회의 트랜잭션을 발의하고 이를 처리하는데 걸린 시간을 측정하여 얻은 값의 평균값을 사용했다.

기밀성만을 제공하는 스마트 컨트랙트의 경우 기존의 OP-TEE 상에서 기존의 스마트 컨트랙트를 실행했으며, Read와 Write에 각각 130.1ms와 137.7ms의 응답시간을 보였다. TCST를 통해 기밀성과 무결성 검증을 모두 제공하는 경우는 TCST 모듈이 추가된 OP-TEE 상에서 TCST 패스가 적용된 컴파일러로 생성된 스마트 컨트랙트를 사용하였다. Read와 Write에 각각 133.7ms와 141.6ms의 응답시간을 보였다. Read의 경우 원장으로부터 값을 읽어오고 읽어온 결과를 전송하기 위해 2회의 컨텍스트 전환이 이루어진다. 그러나, Write의 경우 원장으로부터 값을 읽어와 수정하고 결과를 전송하기 위하여 3번의 컨텍스트 전환을 필요로한다. TCST

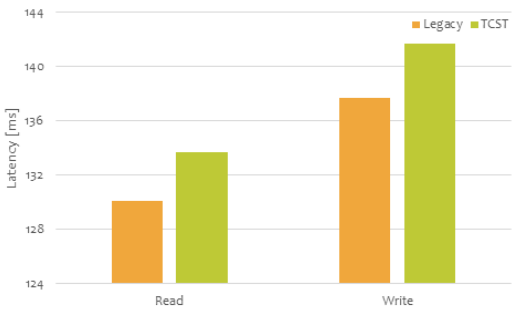

Fig. 4. TCST performance evaluation graph

TEE에서의 연산이 종료된 후 REE로의 컨텍스트 전환이 일어나는 과정에서 무결성 검증을 수행하게 되므로 Write 과정에서 무결성 검증을 1회 더 수행하게 되며, 그럼에도 추가적으로 발생하는 응답시간 오버헤드가 매우 작은 것을 확인할 수 있다.

Fig. 3.은 TCST(TCST)와 기존 스마트 컨트랙트(Legacy)의 응답에 걸린 시간을 그래프로 나타내고 있으며, 이에 대한 오버헤드를 백분율로 나타내면 Read와 Write에서 각각 2.78%와 2.9%의 오버헤드가 발생하는 것을 확인할 수 있다.

VII. 관련 연구

TCST는 기밀성이 제공되는 스마트 컨트랙트에 대하여 무결성 검증을 수행함으로써 스마트 컨트랙트의 보안성을 높이는 것을 목표로 한다. 해당 장에서는 이처럼 스마트 컨트랙트의 보안성을 높이기 위한 연구들과 제어 흐름 무결성 검증을 위한 연구들을 간략하게 소개한다.

7.1 스마트 컨트랙트의 기밀성 보장

스마트 컨트랙트의 기밀성을 보장하기 위한 대표적인 기법은 스마트 컨트랙트를 TEE내에서 실행함으로써 외부에서 데이터 및 실행상태에 대해 접근을 차단하여 기밀성을 보장하는 방식이 있으며 이와 관련된 연구들을 [1]를 통하여 확인할 수 있다.

Hyperledger fabric을 대상으로 하여 스마트 컨트랙트의 기밀성을 보장하기 위해 [13], [14]와 같은 연구들이 진행되었다. [13]의 경우 ARM-TZ을 기반으로 구현되었으며, 반대로, [14]는 Intel-SGX를 활용하여 스마트 컨트랙트의 기밀성을 보장하였다. [14]의 경우 Intel-SGX에서 제공하는 원격 증명 기술[15]을 통해 수행 환경에 대한 검증을 수행할 수 있으나, ARM-TZ의 경우 원격 증명을 위한 기능이 존재하지 않아 수행 환경에 대한 검증을 수행하기 어렵다. TCST는 제어 흐름 무결성 검증을 통해 로컬에서 수행 환경에 대한 검증을 가능하게 한다.

7.2 수행 중 이상 탐지

스마트 컨트랙트의 수행 중 이상 행위를 탐지하기 위한 연구들로는 [16], [17], [18], [19]와 같은 연구들이 존재한다. 그러나 해당 연구들은 Ethereum을 활용하여 구현이 이루어졌다. Ethereum의 경우 Solidity와 같은 전용 언어로 작성된 스마트 컨트랙트를 바이트 코드로 변환하여 EVM(Ethereum Virtual Machine)상에서 실행하는 방식으로 스마트 컨트랙트를 실행한다[20]. 그러나 Hyperledger fabric 일반 언어로 작성되어 호스트 머신에서 실행되기 때문에, 유사한 탐지 방식을 사용하더라도 디자인과 구현에서 차이를 보일 것으로 생각된다.

7.3 스마트 컨트랙트의 제어 흐름 변조와 임의 실행

스마트 컨트랙트는 블록체인 네트워크에서 분산원장을 관리하는 주요 컴포넌트로 이를 공격받을 경우 막대한 재산상의 피해를 입을 수 있을 것이다.

이러한 스마트 컨트랙트를 대상으로 하는 가장 대표적인 공격인 재진입[21] 공격의 경우 실행 중인 함수가 종료되기 전에 함수를 다시 호출하여 특정 함수를 반복적으로 실행시키는 공격으로 제어 흐름 변조를 통해 이루어지는 공격으로 보여진다.

[4]에서는 Hyperledger fabric의 스마트 컨트랙트인 체인코드의 취약점을 악용하여 악성 코드를 삽입하고 임의의 연산을 수행할 수 있으며, 이를 통해 원격 실행, 데이터 유출등의 공격을 수행하였다. 이러한 결과는 스마트 컨트랙트를 통해 지능형 지속공격(Advanced persistent threat, APT)이 이루어질 수 있음을 보였다.

VIII. 고 찰

해당 연구를 통하여 TEE에서 실행되는 스마트 컨트랙트에 대하여 제어 흐름 무결성 검증을 수행함

으로써 보안성을 높일 수 있음을 보였다. 이러한 무결성 검증은 실행된 제어 흐름 명령어의 주소를 누적하여 해시한 값을 화이트 리스트를 기반으로 하여 비교 함으로써 수행된 흐름의 적법성을 판별하는 방식으로 이루어진다.

이를 통해 적은 오버헤드로 높은 보안성을 제공할 수 있었으나, 비결정적인 수행 흐름 또는 루프와 같은 수행 흐름의 경우 정확한 화이트 리스트를 생성하는데 어려움이 있다. 이를 해결 하기 위하여 외부 컨트랙트 호출과 같이 비결정적인 수행 흐름을 통해 도달할 수 있는 대상을 정확하게 판별하고 정교한 제어 흐름 그래프를 생성, [12]와 같은 논문을 참고로 하여 루프와 같은 경우에 대하여 최적화된 화이트 리스트 생성하기 위한 추가적인 연구가 진행되어야 할 것이다.

또한, REE와 TEE사이의 호출 인자를 전송함에 있어서 보안 채널을 설립하여 전송되는 데이터에 대한 무결성을 보장하고[22], REE영역에서 수행된 코드에 대한 무결성을 검증을 수행함으로써 더욱 보안성 높은 스마트 컨트랙트의 실행환경을 보장할 수 있을 것이다.

IX. 결 론

본 연구에서는 TEE에서 실행되는 스마트 컨트랙트에 대하여 무결성을 검증하는 기술인 TCST를 제안하였다. 이는 1.26%의 코드 크기 오버헤드와 2.46% 및 2.54%의 응답시간 오버헤드만으로 무결성 검증을 제공한다. 이를 위해 TEE의 보안 커널에 TCST 모듈을 추가하고, 컴파일러 패스를 추가해 개발자의 노력을 적게 들이고도 TCST를 적용할 수 있도록 하였다.

References

[1] R. Li, Q. Wang, Q. Wang, D. Galindo and M. Ryan, "SoK: TEE-assisted confidential smart contract", Proceedings on Privacy Enhancing Technologies, vol. 2022, no. 3, pp. 711-731, Aug. 2022.

[2] V. Costan and S. Devadas, "Intel SGX Explained", IACR Cryptol, EPrint Arch., pp. 1-118, Jan. 2016.

[3] ARM Developer "ARM Security Technology Building a Secure System using TrustZone Technology" https://developer.arm.com/documentation/PRD29-GENC-009492/c, Accessed .09.15.2022.

[4] Z. Li, Y. Wang, S. Wen and Y. Ding, "Evil chaincode: Apt attacks based on smart contract.", Communications in Computer and Information Science, vol. 1286., pp. 178-196, Nov. 2020.

[5] X. Wang, J. He, Z. Xie, G. Zhao and S. C. Cheung, "ContractGuard: Defend ethereum smart contracts with embedded intrusion detection.", IEEE Transactions on Services Computing, vol. 13, no. 2, pp. 314-328, Oct. 2022.

[6] T. Min and W. Cai, "A Security Case Study for Blockchain Games," 2019 IEEE Games, Entertainment, Media Conference (GEM), pp. 1-8, Jun. 2019.

[7] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. A. De Caro, ... and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains.", In Proceedings of the thirteenth EuroSys conference, pp. 1-15, Apr. 2018.

[8] OP-TEE.org "Open Portable Trusted Execution Environment" https://www.op-tee.org, Accessed .09.15.2022.

[9] M. Abadi, M. Budiu, U. Erlingsson and J. Ligatti, "Control-flow integrity principles, implementations, and applications.", ACM Transactions on Information and System Security (TISSEC), vol. 13, no. 1, pp. 1-40, Oct. 2009.

[10] J. Li, L. Chen, G. Shi, K. Chen and D. Meng, "ABCFI: Fast and

lightweight fine-grained hardware-assisted control-flow integrity.", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3165-3176, Oct. 2020.

[11] N. Burow, X. Zhang and M. Payer, "SoK: Shining light on shadow stacks.", In 2019 IEEE Symposium on Security and Privacy (SP), pp. 985-999, May. 2019.

[12] T. Abera, N. Asokan, L. Davi, J. E. Ekberg, T. Nyman, A. Paverd, ... and G. Tsudik, "C-FLAT: control-flow attestation for embedded systems software.", In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 743-754, Oct. 2016.

[13] C. Müller, M. Brandenburger, C. Cachin, P. Felber, C. Göttel and V. Schiavoni, "TZ4Fabric: Executing Smart Contracts with ARM TrustZone:(Practical Experience Report).", 2020 International Symposium on Reliable Distributed Systems (SRDS), pp. 31-40, Sep. 2020.

[14] M. Brandenburger, C. Cachin, R. Kapitza and A. Sorniotti, "Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric.", arXiv preprint arXiv:1805.08541., May. 2018.

[15] Intel "Attestation Services for Intel® Software Guard Extensions" https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html, Accessed .09.01.2022.

[16] X. Wang, J. He, Z. Xie, G. Zhao and S. C. Cheung, "ContractGuard: Defend ethereum smart contracts with embedded intrusion detection.", IEEE Transactions on Services Computing, vol. 13, no. 2, pp. 314-328, Oct. 2019.

[17] M. Rodler, W. Li, G. O. Karame and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks.", arXiv preprint arXiv:1812.05934, Dec. 2018.

[18] C. Ferreira Torres, M. Baden, R. Norvill and H. Jonker, "ÆGIS: Smart shielding of smart contracts.", In Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp. 2589-2591, Nov. 2019.

[19] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, ... and X. Zhang, "SODA: A Generic Online Detection Framework for Smart Contracts.", NDSS, Feb. 2020.

[20] Ethereum "Ethereum Whitepaper" https://ethereum.org/ko/whitepaper/, Accessed .09.01.2022.

[21] H. Hasanova, U. J. Baek, M. J. Shin, K. Cho and M. S. Kim, "A survey on blockchain cybersecurity vulnerabilities and possible countermeasures.", International Journal of Network Management, vol. 29, no. 2, pp. 1-36, Jan. 2019.

[22] J. S. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "Secret: Secure channel between rich execution environment and trusted execution environment.", NDSS, pp. 1-15, Feb, 2015.

〈저자소개〉

박 성 환 (Seonghwan Park) 학생회원
2021년 2월: 동서대학교 컴퓨터공학 학사 졸업
2021년 3월~현재: 부산대학교 정보융합공학과 석사과정
〈관심분야〉 시스템 보안, 정보보안

권 동 현 (Donghyun Kwon) 정회원
2012년 2월: 서울대학교 전기컴퓨터공학과 학사 졸업
2019년 2월: 서울대학교 전기컴퓨터공학과 석박통합과정 졸업
2019년 3월~2020년 2월: 한국전자통신연구원 연구원
2020년 3월~현재: 부산대학교 정보컴퓨터공학부 조교수
〈관심분야〉 시스템 보안, 소프트웨어보안