KIPS Trans. Softw. and Data Eng.
Vol.11, No.10 pp.399~408
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2022.11.10.399

A De Facto Standard for ERC-20 API Functional Specifications and Its Conformance Review Method for Ethereum Smart Contracts

Hyeon-Ah Moon† · Sooyong Park††

ABSTRACT

ERC-20, the standard API for Ethereum token smart contracts, was introduced to ensure compatibility among applications such as wallets and decentralized exchanges. However, many compatibility vulnerability problems have existed because there is no rigorous functional specifications for each API nor conformance review tools for the standard. In this paper, we proposed a new review procedure and a tool to perform the procedure to review if ERC-20 token smart contract programs for the Ethereum blockchain conform to the de facto standards. Based on the knowledge from an analysis on the ERC-20 API functional behavior of the top 100 token smart contract programs in the existing Ethereum blockchain, a new specification for the de facto standard for ERC-20 API was explicitly defined. The new specification enabled us to design a systematic review method for Ethereum smart contract programs. We developed a tool to support this review method and we evaluated a few benchmark programs with the tool.

Keywords : Ethereum, Smart Contracts, Conformance, Review

이더리움 스마트 계약 프로그램의 ERC-20 API 기능 명세의 관례상 표준과 적합성 리뷰 방법

문 현 아† · 박 수 용††

요 약

이더리움 토큰 스마트 계약의 표준 API인 ERC-20은 지갑이나 분산 거래소같은 응용 프로그램들에서 호환성을 보장하기 위해 도입되었다. 그러나 API의 동작에 대한 엄밀한 기능 명세와 표준 적합성 리뷰 도구는 지원되고 있지 않아 호환성 취약점 문제가 발생할 수 있다. 본 논문에서는 이더리움 블록체인의 ERC-20 토큰 스마트 계약 프로그램들의 관례상 표준에 부합하는지 검사하는 새로운 리뷰 절차와 이를 지원하는 도구를 제안하였다. 기존 이더리움 블록체인 시장 상위 100개의 토큰 스마트 계약 프로그램들을 ERC-20 API 기능 동작면에서 분석한 지식을 바탕으로 관례상 표준을 명시적으로 정의하였고, 이렇게 정의된 관례상 표준으로 새로운 ERC-20 스마트 계약 프로그램을 체계적으로 리뷰할 수 있는 방법을 설계할 수 있었다. 이 리뷰 방법을 지원하는 도구를 개발하고 벤치마크 프로그램에 대해 실험 평가하였다.

키워드 : 이더리움, 스마트 계약, 적합성, 리뷰

1. 서 론

이더리움(Ethereum)[1,2]은 스마트 계약(Smart contracts)을 위한 블록체인으로 널리 사용되고 있다. 스마트 계약[3]은 컴퓨터 프로그램으로 작성된 디지털 계약이다. 이더리움 블록체인의 스마트 계약 프로그램은 보통 솔리디티(Solidity) 프로그래밍 언어로 작성한다. 이 프로그램은 토큰(tokens)과 트랜잭션(transactions)을 위한 함수들로 구성되어 있다. 토큰은 재화를 대체할 수 있는 디지털 화폐이고, 블록체인에 참여하는 사람들 또는 스마트 계약 프로그램들이 서로 토큰을 주고받는 것을 트랜잭션이라 한다. 블록체인은 이 트랜잭션들을 영구적으로 보관하는 장부 역할을 한다.

ERC-20(Ethereum Request for Comment 20)은 토큰을 다루는 가장 중요한 API 표준으로 이더리움 블록체인을 만든 부테린(V. Buterin)이 처음으로 제안하였다[4]. ERC-20 API의 목적은 블록체인의 계정들을 통해 서로 자유롭게 토큰을 주고받을 수 있는 표준 인터페이스를 만드는 것이었다. 예를 들어, 암호 화폐 지갑(wallet)이나 분산형 거래(decentralized exchange)에 사용된다.

ERC-20 API는 Table 1에 나열된 바와 같이 총 6개의 필수 함수(totalSupply, balanceOf, transfer, transferFrom, approve, allowance)와 2개의 이벤트(Transfer, Approval)로 구성되어 있다. 이 함수들을 통해 전체 토큰 발행량, 개별

※ 이 논문은 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터 지원사업의 연구결과로 수행되었음(IITP-2022-2017-0-01628).
† 준 회 원 : 서강대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 서강대학교 컴퓨터공학과 교수
Manuscript Received : May 2, 2022
First Revision : June 2, 2022
Second Revision : June 23, 2022
Accepted : June 24, 2022
* Corresponding Author : Sooyong Park(sypark@sogang.ac.kr)

Table 1. ERC-20 API: Methods and Events

Methods	
totalSupply	function totalSupply() public view returns (uint256)
balanceOf	function balanceOf(address owner) public view returns (uint256 balance)
transfer	function transfer(address to, uint256 value) public returns (bool success)
transferFrom	function transferFrom (address from, address to, uint256 value) public returns (bool success)
approve	function approve (address spender, uint256 value) public returns (bool success)
allowance	function allowance (address owner, address spender) public view returns (uint256 remaining)
Events	
Transfer	event Transfer (address indexed from, address indexed to, uint256 value)
Approval	event Approval (address indexed owner, address indexed spender, uint256 value)

계정 잔액을 확인하고, 상대 계정에 토큰을 직접 보내거나 대리인을 지정해 할당한 양 이내에서 토큰을 간접적으로 보낼 수 있다. 이벤트들은 이러한 트랜잭션들이 발생했음을 블록체인 외부의 클라이언트에서 확인할 때 사용한다.

현재 이더리움 블록체인에 ERC-20 API를 사용하는 다양한 스마트 계약 프로그램들이 활발하게 사용되고 있다. 이더스캔(EtherScan)[5] 웹 사이트에 집계된 바에 의하면 2020년 6월 기준 대략 268,052개의 ERC-20 토큰들이 존재하고 상위 100개 토큰들은 그 가치가 92억 달러로 대부분의 시장을 점유하고 있다.

이 상위 100개 토큰 스마트 계약 프로그램들을 조사 분석한 결과 ERC-20 API 표준을 서로 호환되지 않는 여러 방식으로 사용하고 있음을 발견하였다[6]. 우선 100개 중 7개 토큰은 아예 API 인자 또는 리턴 타입을 지키지 않았다. API 타입을 준수한 경우에도 토큰들이 API 기능 동작(functional behavior)이 서로 다른 12개 그룹으로 분류됨을 보고하였다. 그 이유는 ERC-20 표준 [4]에서 ERC-20 API 타입은 명시하고 있으나, 그 API 기능 동작에 관해서는 명확하게 정의하지 않고 모호한 부분이 많기 때문이다.

ERC-20 토큰들의 API 기능 동작에서 호환되지 않아 이더리움 트랜잭션에서 호환성 취약점을 발생시킬 수 있다. 예를 들어, 사용자 A가 사용자 B에게 토큰을 보냈을 때 사용자 B의 잔액과 받은 액수를 합하면 정수 범위를 초과할 수 있다. 100개의 토큰들 중 9개가 이러한 오버플로를 검사하지 않고 있다. 이러한 이유로 이더리움 스마트 계약이 ERC-20 규격에 적합한지(conformance) 검사할 필요가 있다. 공개적으로 알려진 컴퓨터 보안 결함 목록인 CVE(Common Vulnerabilities and Exposures)에도 ERC-20 토큰의 transfer 함수에서의 정수 오버플로 미처리가 보고되고 있다[7,8].

이 논문은 스마트 계약 프로그램에 대한 ERC-20 규격 적합성을 검사하는 방법을 제안한다. 이 검사 방법을 설계하기 위해 두 가지 기술적 문제를 해결하였다. 첫째, 상위 100개 토큰 프로그램들을 분석한 결과를 바탕으로 ERC-20 API 기능 동작에 대한 관례상 표준(de facto standard)을 명시적으로 정의하여 ERC-20 기능 동작에 대한 기준을 마련하였다. 둘째, 기존 상위 100개 토큰 스마트 계약 프로그램들의 다양한 패턴을 수집 분석한 결과들을 바탕으로 4가지 규격 적합성 위반 카테고리를 정의하고 세부적인 위반 가능성을 점검하기 위한 체크리스트를 설계하였다.

이 논문에서 제안한 ERC-20 스마트 계약 프로그램의 ERC-20 기능 동작 적합성 위반의 검사에 관한 연구는 기존에 수행된 바 없는 새로운 연구로 기여 사항을 3가지로 요약할 수 있다.

- 실제 ERC-20 토큰 스마트 계약에서 사용하는 관례상 표준 ERC-20 API 기능 명세를 명시적으로 정의하였다.
- ERC-20 토큰 스마트 계약 프로그램이 이 표준을 준수하는지 적합성을 확인하는 리뷰 절차를 설계하였다.
- 이 ERC-20 적합성 리뷰 절차를 도구로 구현하고 이 도구를 활용한 사례를 통해 평가하였다.

이 논문은 2장에서 ERC-20 스마트 계약 프로그램에 대한 배경을 설명하고, 3장에서 관련 연구에 대해 서술한다. 4장에서 ERC-20 API 기능 동작의 관례상 표준을 정의하고 이 표준에 부합하지 않는 규격 적합성 위반 카테고리와 이에 대한 체크리스트를 제시한다. 5장에서 ERC-20 스마트 계약 프로그램의 관례상 표준 적합성을 점검하고 체크리스트를 제시하는 도구에 대한 구현과 평가 결과를 제시하고, 6장에서 결론을 맺는다.

2. 배경 지식과 연구 배경

2.1 ERC-20 API와 스마트 계약 프로그램

Table 1은 이더리움 스마트 계약 프로그램에서 토큰을 주고받기 위한 표준 인터페이스이다. 코인이라고도 하는 토큰은 이더리움 블록체인에서 이더라고 하는 암호화폐 대신 사용되는 교환 매체로, 주로 ICO(Initial Coin Offering)를 통해 새로운 암호화폐로 출시되고 있다. totalSupply 함수는 총 토큰 수를 제공한다. 각 계정이 소유한 토큰 수는 계정 주소(주소 유형)를 인수로 하는 balanceOf 함수로 확인한다. 256 비트의 부호 없는 정수(uint256)는 이 두 함수를 포함해 모든 API에서 토큰의 수를 나타내는 데 사용된다. 토큰은 두 개의 인수, 즉 수신자 계정 주소(to)와 전송할 토큰 수(value)를 사용하는 transfer 함수에 의해 한 계정에서 다른 계정으로 전송할 수 있다. transferFrom 함수는 토큰을 대리 전송할 수 있고 보낸 사람 계정 주소(from)와 transfer 함수에서의 두 인수를 사용한다. 사용자에게 전송 성공 여부를 알리기 위해 두 함수는

bool 유형의 결과 값을 반환한다. 또한 웹 기반 클라이언트와 같은 외부 애플리케이션이 전송 정보에 접근할 수 있도록 이더리움 블록에 이벤트라는 데이터를 남기도록 설계되었다. transfer와 transferFrom은 모두 보낸 사람 계정 주소(from), 받는 사람 계정 주소(to), 전송 된 토큰 수(value)의 세 가지 값을 보유한 Transfer 이벤트를 발생시킬 수 있다.

토큰 소유자는 approve 함수를 통해 자신의 토큰을 인출할 수 있는 권한을 spender에게 위임할 수 있다. 첫 번째 인수인 spender는 대리인의 계정 주소이고 두 번째 인수인 value는 대리인에게 허용되는 총 금액이다. 이 함수는 성공 여부에 대한 결과로 부울 값을 반환한다. 또한 토큰 소유자 계정 주소(owner), 대리 인출자 계정 주소(spender) 및 승인한 토큰 수(value)와 같은 승인 정보를 제공하기 위해 Approval 이벤트를 남긴다. 함수 allowance는 spender가 owner로부터 대리 인출할 수 있는 토큰 수를 반환한다.

2.2 ERC-20 토큰 스마트 계약의 표준 적합성 문제

ERC-20 토큰 인터페이스는 사용자와 분산 애플리케이션(DApp)간에 일반적으로 일어나는 상호 작용의 관점에서 중요하며, 표준은 모든 지갑 및 거래소와 호환되도록 하여 각종 코인을 전송하고 잔액을 확인할 수 있게 한다.

토큰 스마트 계약 프로그램이 ERC-20 표준을 따르는지에 관한 문제는 일반적으로 두 가지이다. 첫째, 토큰 스마트 계약을 위한 ERC-20 인터페이스의 기존 설명이 모호하다. 즉, API의 의도하는 동작(또는 세만틱)이 아닌 함수 인터페이스만 설명한다. 둘째, ERC-20 인터페이스 설명은 주어진 스마트 계약이 표준 인터페이스를 준수하는지 확인하기 위해 사용 가능한 도구와 함께 제공되지 않는다.

1) ERC-20 API 기능 명세의 모호성

표준 인터페이스에 대한 정의는 여러 해석이 가능하도록 모호하게 되어 있다. ERC-20 인터페이스에 대한 서로 다른 해석은 이더리움 트랜잭션에서 규정 준수 취약성을 유발할 수 있다. 예를 들어, 사용자 A가 사용자 B에게 토큰을 전송할 때 B의 원래 토큰과 전송된 토큰의 합이 사용자의 토큰 수를 기록하는데 사용되는 정수의 상한을 초과할 수 있다. 표준의 transfer 함수는 이 상황의 처리에 대해 정의하지 않고 있다[4].

또 다른 일반적인 시나리오는 표준 ERC-20 인터페이스를 사용하는 에스크로라는 신뢰할 수 있는 제 3자를 통한 상호 신뢰할 수 없는 사용자 간의 거래를 나타낸다. 사용자 A가 상품으로 간주되는 토큰을 가지고 있고 사용자 B가 1이더로 구매하기를 원한다고 가정한다. 이러한 거래에서 사용자 A는 에스크로에 토큰을, 사용자 B는 에스크로에 1이더를 보낸 후 에스크로는 사용자 B에게 토큰을 보낸다. 그런 다음 토큰 전송이 성공한 후에만 에스크로는 사용자 A에게 1이더를 전송한다. 표준을 준수하지 않는 토큰의 경우 사용자 B에게 성공적으로 전송 한 후에도 에스크로에 보고하기 위한 표준에 따르는 부울 값을 에스크로에 반환하지 않을 수 있다. 따라서 에스크로는 이체 결과를 오해하고 사용자 B에게 1이더를 다시 보낼 수 있다.

실험을 통해 ERC-20 토큰 계약으로 선언된 실제 스마트 계약 중 일부가 이러한 표준을 준수하지 않는 취약성을 발견하였다. 예를 들어, 몇몇 토큰의 transfer 함수는 토큰 전송으로 인한 오버플로 가능성을 사전에 확인하지 않거나(GNO 토큰, NULS토큰), 토큰 전송 결과에 대한 부울 값을 반환하는 대신 아무것도 반환하지 않는다(Tether토큰, BNB토큰).

일반적으로 사용자는 토큰을 조작하기 위해 외부 도구를 사용한다. 예를 들어 지갑을 사용하여 토큰을 전송하고 교환 시장을 활용하여 토큰을 구매/판매하거나 블록체인 탐색기를 사용하여 트랜잭션을 확인한다. 이러한 도구는 토큰 표준에 정의된 표준 인터페이스 및 표준 이벤트를 통해 토큰과 상호 작용한다. 그러나 토큰 계약의 구현이 표준과 일치하지 않으면 외부 도구가 토큰과 제대로 상호 작용할 수 없으며 토큰을 인식하지도 못할 수 있다. 또한 지갑, 교환 시장, 블록체인 탐색기와 같은 도구들은 토큰 전송 시 표준 이벤트를 발생하지 않으면 통보를 받지 못하기 때문에 혼동할 수 있다. 따라서 토큰 계약 개발시 표준 적합성을 리뷰하는 것은 매우 중요하다.

실제 ERC-20 API 표준에 대한 해석의 차이로 이더델타(EtherDelta) 거래소[9]에서 토큰이 동결되고 도난된 사례가 보고된 바가 있다[10]. 또한 블록체인의 특성상 스마트 계약 프로그램은 배포 후 수정할 수 없는데, HXG 토큰과 LNC 토큰의 경우 ERC-20 표준 API의 취약점이 발견되어 CVE에 보고되었다[7,8].

2) ERC-20 API 적합성 기준 및 검사 도구의 부재

ERC-20 API 기능 동작에 대해 명확하게 정의된 바가 없을 뿐 아니라 이 표준을 준수하는지 여부를 확인하는 리뷰 방법이나 도구가 부족하다. 두 번째 문제를 해결하기 위해 개발된 도구로 Slither[11]와 KEVM기반 정형 검증 기법[12], 두 가지가 있으나 ERC-20 API 기능 동작을 확인하지 못하거나 많은 비용을 수반하고 더욱 중요한 것은 ERC-20 API 기능 동작에 대한 표준을 고려하고 있지 않은 문제가 있다.

Slither는 ERC-20 표준 API의 이름, 인수 유형 및 반환 유형이 주어진 토큰 스마트 계약에 존재하는지 확인하는 ERC-20 검증기를 제공한다[11]. 이 도구는 EVM 바이트 코드에서 스마트 계약에 대한 API 프로토타입을 자동으로 확인하므로 매우 간단하다.

다른 사례로 KEVM 기반 정형 검증 기법을 활용한 사례가 있다. KEVM을 사용하여 정형 명세를 설계하고 이더리움 가상 머신의 완전한 공식 의미 체계인 KEVM에서 HKG 토큰을 포함한 몇 가지 ERC-20 스마트 계약을 검증했다고 보고되었다[12]. 이 방법은 KEVM 프레임워크가 개별 명세에 대해 스마트 계약을 검증한 결과는 높은 정확도를 갖는 반면 스마트 계약에 대한 명세 설계에 많은 노력을 들여야 하는 단점이 있다. 특히 이 연구는 ERC-20 API의 기능 동작에 대한 표준을 고려

하지 않고 개별 스마트 계약 프로그램의 명세에 초점을 맞추어 검증을 진행하였기 때문에 이 연구의 목적과 다르다.

2.3 해결 방안

이러한 문제들을 해결하고자 이 연구에서는 먼저 ERC-20 API 기능 동작에 대한 명세로 기존 100개의 토큰 스마트 계약 프로그램들을 분석한 결과 실제 이더리움 블록체인 비즈니스에서 다수가 사용하고 있는 명세를 관례상 표준으로 정하고 이 ERC-20 API 기능 동작에 대한 명세를 명시적으로 정의한다. 그리고 이 표준을 준수하는지 적합성을 확인하는 항목들과 절차를 설계하고, 이 절차에 따라 적합성 여부를 검사하는 도구를 구현한다. 이 도구를 실제 토큰 스마트 계약 프로그램에 적용하여 검사한 결과를 기반으로 적합성 검사 리뷰 방법을 평가한다.

3. 관련 연구

이더리움 스마트 계약 프로그램을 분석하고 검증하는 기존 연구는 주로 취약점을 찾는 목적으로 진행되어 왔다[13]. TokenScope[10]는 이더리움 블록체인에 기록되어 있는 트랜잭션들을 분석하여 핵심 데이터 구조의 조작, 표준 인터페이스가 표시하는 작업 및 표준 이벤트 동작 등의 정보를 비교하여 일관성 없는 동작을 유발하는 ERC-20 토큰 계약을 발견하였다. 그 결과 결함이 있는 토큰, 표준 API 누락, 표준 이벤트 누락 등 불일치의 11가지 주요 원인을 밝혔다. 그러나 이 연구는 ERC-20의 모든 필수 API에 대한 기능 동작을 정의하고 분석하는 우리의 연구와 다르고 새로운 스마트 계약 프로그램을 리뷰 하는 방법을 제시하지 않았다.

그 이외에도 스마트 계약의 취약점을 찾는 연구로는 SmartCheck[14], ContractFuzzer[15] 그리고 VeriSmart[16]/SmarTest[17]가 있다. 이 연구들 역시 스마트 계약 프로그램의 취약점을 자동으로 찾는 문제를 해결하는 목적을 가지고 수행된 것으로 ERC-20 API 기능 명세를 정의하거나 분석하여 적합성을 검증한 것은 아니다.

SmartCheck[14]는 버그 패턴을 검색하여 Solidity 소스 코드에서 21가지 종류의 버그를 감지하는 정적 분석기이다. 앤틀러(ANTLR) 파서 도구로 솔리디티 문법을 구현하여 스마트 계약 프로그램 소스를 읽어 분석 가능한 트리 구조로 변환하고, 다양한 분석 패턴을 엑스패스 쿼리(XPath)로 작성하여 소스 프로그램 트리를 탐색하는 구조를 가지고 있다. 재진입 보안 이슈(Re-entrancy), 토큰을 주고 받을 수 없는 잠금 상태가 되는 이슈(Locked money), 가스를 많이 소모하는 반복문(Costly loop)를 포함하는 21가지 종류의 버그 패턴을 정의하여 해당 취약점을 찾을 수 있도록 확장 가능한 구조를 가지고 있다.

ContractFuzzer[15]은 퍼징 기법을 활용하여 스마트 계약의 보안 취약점을 탐지하는 연구이다. 스마트 계약 프로그램의 ABI (Application binary interface)를 기반으로 퍼징 입력을 무작위로 생성하고, 보안 취약점의 조건을 판단하는 테스트 오라클을 정의하고, 이더리움 가상 기계(EVM)를 수정하여 스마트 계약 프로그램이 어떻게 실행되는지 그 과정을 로그로 남길 수 있도록 시스템을 구성하였다. 6991개의 스마트 계약 프로그램에 대한 로그를 분석하여 459개의 보안 취약점을 찾았다고 보고된 바가 있다.

정적 분석 또는 자동 프로그램 검증 기술을 기반으로 스마트 계약을 분석하는 연구들이 있다[16-22]. VeriSmart[16]와 SmarTest [17]는 기호 실행(Symbolic Execution) 기술을 사용하여 스마트 계약의 안전성을 검증하는 방법에 관한 연구이다. VeriSmart는 스마트 계약 프로그램 내에 프로그래머가 작성한 선행 조건(require 문)과 후행 조건(assert 문)이 항상 만족하는지를 기호 실행 방법으로 검증한다. 스마트 계약 프로그램에 포함된 반복문의 경우 불변 성질(invariants)가 필요한데 프로그래머가 직접 지정하지 않더라도 자동으로 불변 성질을 만들어내는 특징을 가지고 있다. 스마트 계약 프로그램의 특성상 전형적인 반복문이 주로 사용되고 있어 그러한 반복문에 대한 불변 성질의 대략적인 모양을 문맥 자유 문법(Context-free grammar)로 기술하고 그 문법으로 작성 가능한 불변 성질을 생성하고 맞는지 테스트하는 과정을 반복해서 적절한 불변 성질을 자동으로 합성한다.

SmartTest는 스마트 계약 프로그램의 모든 가능한 경로를 탐색하여 프로그램 내에 작성된 선행 조건(require 문)과 후행 조건(assert 문)을 위배하는 실행이 가능한지를 찾는다. 실제로 모든 가능한 경로를 탐색하기 위해 시간이 오래 걸리거나 무한히 많이 걸리는 문제가 있다. 이 문제를 해결하기 위해 통계적 자연어 처리 모델(Statistical language models)을 활용하여 이 탐색을 가이드, 효율적으로 탐색한다. CVE에 알려진 스마트 계약 취약점을 데이터셋으로 학습하여 취약점 경로 탐색을 위한 모델을 만든 다음 실제 스마트 계약 프로그램에 적용한 실험 결과를 리포트 하였다.

ETHBMC[18]는 memcopy 유형의 기호 오퍼레이션, 계약간 통신, 암호화 해시 함수를 보다 잘 처리하는 특징을 갖는 스마트 계약을 위한 제한된 모델 체커로 Tether 토큰 스마트 계약에 적용되어 계약을 탈취하는 데 사용될 수 있는 많은 취약 계정을 발견하였다. 오이엔트(Oyente)[19]는 기호 실행을 이용하여 스마트 계약의 취약점을 분석하는 도구로 모든 가능한 실행 경로를 따라가며 많이 알려진 4가지 보안 취약점 패턴(잘못 처리된 예외, 재진입, 트랜잭션 순서 종속성, 타임스탬프 의존성)을 찾는다. ZEUS[20]는 산술 버그를 감지하거나 존재하지 않음을 증명할 수 있는 정적 분석기이다. SMTChecker[21]는 Ehtereum Foundation에서 개발한 Solidity에 대한 "공식" 검증 도구로 SMT 기반 경계 검증을 수행하여 정수 오버/언더플로우와 0으로 나누기 같은 산술 버그가 없는지 확인한다.

2.2절에 기술한 Slither 정적 분석 프레임워크에서 사용되는 ERC-20 API 검증기는 계약이 유효한 ERC-20인지 여부를 확인한다[11]. ERC-20 API의 동작 측면에서 함수가 예상대로 작동하는지 검증하는 것이 아니라 API명과 반환 유형 등이 일치하는지를 확인하는 인터페이스만을 검증하는 방법이다.

프로그래밍 언어의 의미를 실행 가능한 모델로 만드는 K-프레임워크를 활용하여 EVM 바이트코드 언어에 대한 해석기 KEVM을 만들고, 이 모델을 사용해 토큰을 검증한 사례가 있다[12]. 하지만 이 연구에서 진행한 방법은 HKG 토큰 자체의 어플리케이션 로직에 관한 명세를 설계하고 검증한 것으로 다양한 토큰들에 대해 ERC-20 표준 준수 여부를 확인하는데 사용할 수 있는 일반적인 적합성 검증 방법은 아니다.

OpenZeppelin 테스트 스위트는 안전한 스마트 계약 개발을 위해 템플릿을 제공한다[23]. ERC-20 테스트 스위트는 API 기능 명세에 대해 정의한 문서를 제공하지 않는다.

적합성 테스트는 다양한 분야에서 활용되고 있다. Android 모바일 플랫폼 호환성 테스트 도구 모음(CTS)[24], W3C의 마크업 유효성 검사 서비스[25] 및 IEEE 표준 POSIX (Portable Operating System Interface)[26]와 같은 여러 영역에서 적합성 테스트를 위한 활동이 있어 왔다. 개방형 시스템 상호 연결(OSI, Open Systems Interconnection) 프로토콜에 대한 국제 표준 및 권장 사항의 적합성을 테스트하는 도구는 공급업체들 상호 운용성의 핵심이다[27]. 또한 JavaScript 구현의 ECMAScript 표준에 대한 적합성을 테스트하는 방법에 대한 연구도 있다[28,29].

4. 스마트 계약 프로그램의 ERC-20 API 표준 적합성에 관한 체계적인 리뷰 방법

블록체인의 특성으로 인해 스마트 계약의 소프트웨어 버그가 재정적 손실로 이어질 수 있기 때문에 스마트 계약을 철저히 검토하는 것이 중요하다[30]. 특히 스마트 계약은 일단 블록체인에 배포되면 변경할 수 없기 때문에 미리 검토하는 것이 더욱 중요하다. 실제 이더리움 토큰 스마트 계약 프로그램 사례에서 표준을 준수하지 않아 거래소에서의 재정 손실, 잔액 감소, 초과 대리 송금과 같은 문제가 발생한 사례가 있었다.

이와 같이 ERC-20 토큰 스마트 계약에 대한 적합성 검토 방법이 필요함을 확인할 수 있다. 이더리움 블록체인이 현재 시점을 기준으로 비트코인 블록체인 다음으로 널리 활용되고 있음에도 아직까지 개발자가 활용할 수 있는 ERC-20 토큰 스마트 계약에 대한 유용한 적합성 검토 방법이 없다. 이 장에서는 ERC-20 토큰 스마트 계약의 적합성 확인을 위한 체계적인 리뷰 방법을 제안한다.

4.1 ERC-20 API 기능 동작에 대한 관례상 표준 명세

체계적인 리뷰 방법을 제안하기 위해서 먼저 ERC-20 API의 기능 동작에 대한 기준이 필요하다. 앞서 배경에서 설명한 바와 같이 현재 ERC-20 API 표준은 인터페이스는 정의하고 있으나 기능 동작에 대한 표준은 명확히 정의하고 있지 않다.

이 연구에서는 이더리움 블록체인에서 관례적으로 API 기능 동작을 어떻게 해석하고 사용해왔는지에 따라 기준을 정하고자 한다. 지난 연구[6]에서 시장 점유율 기준 상위 100개 ERC-20 토큰 스마트 계약 프로그램을 분석한 결과 65개 프로그램들이 공통적으로 같은 기능 동작을 사용하고 있음을 밝힌 바가 있다.

본 연구에서 실제 이더리움 블록체인 시장에서 관례적으로 해석해온 6개 API 기능 동작에 대한 명세를 명시적으로 작성한다. 이 명세는 이후 ERC-20 API 관례상 표준에 따르는지 적합성 여부를 판단하는 절차를 만들 때 기준으로 사용한다. 이제까지 관례상 표준을 조사하거나 명시적으로 기술한 사례가 없었다. 심지어 지난 연구[6]에서도 65개를 한 그룹으로 분류했을 뿐 그 그룹에 대한 명세를 명시적으로 정의하지 않았다.

Table 2는 65개 상위 토큰 스마트 계약 프로그램들에서 묵시적으로 사용하고 있는 ERC-20 API 기능 명세를 명시적으로 작성한 내용이다. 이 명세의 첫 번째 컬럼은 각 세부 명세를 식별하기 위한 이름이다. 두 번째 컬럼 ERC-20 API에서는 Table 1에서 선언한 6개 함수 인터페이스와 이벤트를 따른다. 지면이 부족한 관계로 Table 1을 참조하고 해당 컬럼에는 함수 이름만 나열하였다. 세 번째와 네 번째 컬럼들에 각 세부 명세에서 다루는 함수의 가능한 모든 입력을 빠짐없이 고려하여 입력과 출력(이벤트 포함)을 정의하였다.

Table 2의 각 세부 명세 중, 예를 들어 transfer 함수에 대한 5가지 세부 명세 dFS-TR1~dFS-TR5는 보내는 계정의 토큰 잔액(sender.balance)과 받는 계정의 잔액(to.balance)의 전송 전후 관계를 모두 나열함으로 작성한 것이다. 이전 연구[6]에서 정의한 기능 동작 명세와 비교하여 dFS-TR3와 dFS-TF4의 명세는 출력으로 false대신 throw하는 점이 다르다. 이는 65개의 ERC-20 스마트 계약에서 따르는 관례를 반영한 것이다. 65개 토큰이 19개 세부 명세에서 같은 동작 결과를 공유했으나 fail의 원인이 달라 본 연구에서 정한 관례상 표준을 준수하는 토큰은 62개이다. 나머지 3개의 토큰은 dFS-TR3와 dFS-TF4의 출력으로 throw하지 않고 오버플로를 미처리하고 있다.

이 기능 명세는 각 함수의 동작에 관여하는 전역 상태인 이더리움 블록체인의 상태와 지역 상태인 함수 지역 상태를 모두 고려하여 작성하였다. 이더리움 블록체인의 상태로 (1) 토큰 총량(totalSupply), (2) 계정 별 토큰 잔액({..., $account_i$ -> $balance_i$, ...}), (3) 계정 별 대리자와 위임 총액(allowance) 목록({..., accounti -> {..., $delegateAccount_j$ -> $allowance_j$, ...}, ...}), (4) Transfer와 Approval 이벤트 기록이 있다. 함수 지역 상태는 (5)송신 계정, (6) 수신 계정, (7)함수 인자와 (8) 리턴 값 (또는 예외 발생 여부)이 있다. 19개 기능 명세는 (1)부터 (8)의 상태에 대하여 해당 함수를 호출하기 전 선행 조건을 입력으로, 호출한 직후의 후행 조건을 출력으로 정의한 것이다.

예를 들어, transfer 함수의 dFS-TR1 세부 기능 명세의 입력(선행 조건)으로 다음의 상황을 가정한다. 송신 계정(sender)의 유효성(valid)은 전역 상태(계정 별 토큰 잔액)에 이 송신 계정의 잔액이 기록되어 있어야 한다. 또한 송신 계정 잔액이 이 함수로 보내려는 금액(value)보다 크거나 같고 (sender.balance 〉= value), 받는 쪽에서 이 금액을 받아도 오버플로가 발생하지 않는 (to.balance+value 〈= MAX) 금액이되, 0이 아닌 금액(value != 0)이다.

Table 2. A De Facto Standard for ERC-20 API Functional Specification

Name	ERC-20 API	Input	Output	
dFS-TS1	totalSupply		token's total supply	
dFS-BA1	balanceOf	valid owner	owner's balance	
dFS-BA2	balanceOf	invalid owner	0	
dFS-TR1	transfer	valid sender, sender.balance≥value, to.balance+value≤MAX, value≠0	true	sender.balance -= value to.balance += value Transfer(sender,to,value)
dFS-TR2	transfer	valid sender, sender.balance≥value, to.balance+value≤MAX, value=0	true	sender.balance (no change) to.balance (no change) Transfer(sender,to,value)
dFS-TR3	transfer	valid sender, sender.balance≥value, to.balance+value〉MAX	throw	sender.balance (no change) to.balance (no change) (no event)
dFS-TR4	transfer	valid sender, sender.balance〈value	throw	sender.balance (no change) to.balance (no change) (no event)
dFS-TR5	transfer	invalid sender	throw	sender.balance (no change) to.balance (no change) (no event)
dFS-TF1	transferFrom	valid from, from.balance≥value, to.balance+value≤MAX, (caller's allowance by from)≥value, value≠0	true	from.balance -= value to.balance += value (caller's allowance by from) -= value Transfer(from, to, value)
dFS-TF2	transferFrom	valid from, from.balance≥value, to.balance+value≤MAX, (caller's allowance by from)≥value, value=0	true	from.balance (no change) to.balance (no change) (caller's allowance by from) (no change) Transfer(from, to, value)
dFS-TF3	transferFrom	valid from, from.balance≥value, to.balance+value≤MAX, (caller's allowance by from)〈value,	throw	from.balance (no change) to.balance (no change) (caller's allowance by from) (no change) (no event)
dFS-TF4	transferFrom	valid from, from.balance≥value, to.balance+value〉MAX	throw	from.balance (no change) to.balance (no change) (caller's allowance by from) (no change) (no event)
dFS-TF5	transferFrom	valid from, from.balance〈value	throw	from.balance (no change) to.balance (no change) (caller's allowance by from) (no change) (no event)
dFS-TF6	transferFrom	invalid from	throw	from.balance (no change) to.balance (no change) (caller's allowance by from) (no change) (no event)
dFS-AP1	approve	valid caller, caller's balance≥value	true	(spender's allowance by caller:= value) Approval(caller, spender, value)
dFS-AP2	approve	valid caller, caller's balance〈value	true	(spender's allowance by caller:= value) Approval(caller, spender, value)
dFS-AP3	approve	invalid caller	true	(spender's allowance by caller:= value) Approval(caller, spender, value)
dFS-AL1	allowance	valid owner	spender's allowance by owner	
dFS-AL2	allowance	invalid owner	0	

이러한 입력이 주어졌을 때 dFS-TR1 세부 기능 명세의 출력(후행 조건)은 다음과 같이 정했다. 우선 송신 계정 잔액과 수신 금액 잔액이 각각 보낸 금액만큼 줄어들고 늘어나야 한다(sender.balance' == sender.balance-value ∧ to.balance' == to.balance+value). 그리고 정상적으로 송금이 이루어 졌음을 클라이언트에서 확인할 수 있도록 Transfer(sender,to,value) 이벤트를 이더리움 블록체인에 기록으로 남겨야 한다.

이와 같은 방식으로 전역 상태와 지역 상태를 모두 고려하여 나머지 18개 세부 기능 명세를 작성하였다.

4.2 ERC-20 적합성에 관한 리뷰 방법 및 검사 도구의 구현

앞 절에서 정의한 ERC-20 API 기능 동작의 관례상 표준을 따르는지 검사하는 리뷰 방법은 API 프로토타입 호환성 확인, API 동작 호환성에 대한 일련의 체크리스트 제공, 비교 및 참조를 위해 유사한 동작 특성을 가진 배포된 ERC-20 토큰의 이름 목록 제안의 세 가지 검토 범주로 구성된다.

ERC-20 토큰 스마트 계약 프로그램에 설계한 리뷰 방법을 기계적으로 적용할 수 있도록 도구를 개발하였다. 이 도구는 크게 두 가지에 의존하고 있다. 우선 4.1절에서 정의한 관례상 API 기능 동작 표준을 테스팅 방법을 통해 검사하고, 둘째, 이더리움 블록체인에서 사용하고 있는 시장 점유율 상위 100개의 이더리움 토큰을 분석하여 얻은 ERC-20 API 기능 동작에 관한 지식을 기반으로 리뷰 결과를 리포트하도록 구성되어 있다. 특히 기존의 잘 알려진 스마트 계약들과 본인이 작성한 스마트 계약을 ERC-20 API 기능 동작 측면에서 체계적으로 비교할 수 있는 장점이 있다.

Fig. 1은 ERC-20 토큰 스마트 계약의 구현을 입력으로 받아 적합성 검토 결과를 생성하는 검토 절차를 보여준다. 3단계로 구성되어 있다. 절차의 첫 번째 단계에서는 API 호환성을 검사한다. Table 1에서 정의한 표준 인터페이스를 따르지 않거나 아예 API 함수가 없는 위반 사항이 발견되면 위반 내용을 결과로 제공하고 절차가 중지된다.

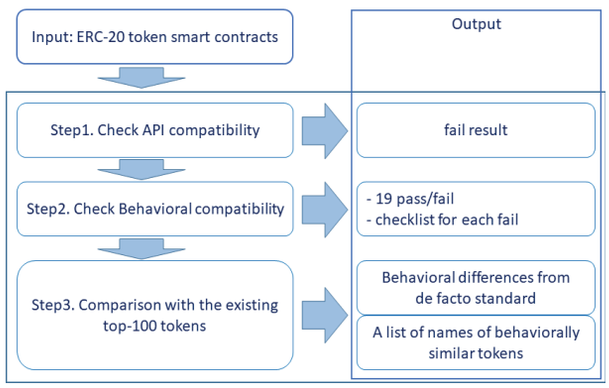

Fig. 1. A Systematic Review Process on ERC-20 API Functional Specification de Facto Standard

인터페이스 검사 단계를 통과한 후 두 번째 단계에서 대상 스마트 계약 프로그램의 API 구현에서 4.1절에서 정의한 관례상 표준인 ERC-20 기능 동작 명세를 따르는지 동작 호환성을 확인한다. Table 2의 19개 사례에 대해 입력 조건에 맞는 입력 데이터를 주어 테스트하고 출력 조건에 맞는 출력을 하는지를 검사한다. 실패한 각 결과에 대해서 Table 3, Table 4, Table 5에 기술되어 있는 해당 항목의 체크리스트

Table 3. A Checklist on the Failure of transfer in the API Behavioral Conformance

Name	Category	Checklist	Failed Rules
FB-ZH-1	0 value	Check if it reverts without processing 0	dFS-TR2
FB-ZH-2		Check if it returns false without processing 0 value	
FB-ZH-3		Check if a Transfer event is fired	
FB-OH-1	an overflow occurs in the recipient's balance	Check if overflow is not handled	dFS-TR3
FB-OH-2		Check for code errors even though overflow is handled	
FB-OH-3		Check if it returns False (Throw is recommended instead of returning false)	
FB-DH-1	Insufficient tokens to send in the sender's account balance	Check if it returns false (It should throw)	dFS-TR4
FB-IS-1	Invalid sender	Check if it returns false (It should throw)	dFS-TR5

Table 4. A Checklist on the Failure of transferFrom in the API Behavioral Conformance

Name	Category	Checklist	Failed Rules
FB-ZH-4	0 value	Check if it reverts without processing 0	dFS-TF2
FB-ZH-5		Check if it returns false without processing 0 value	
FB-ZH-6		Check if a Transfer event is fired	
FB-DH-2	Insufficient allowance for caller approved by from	Check if it returns false (It should throw)	dFS-TF3
FB-OH-4	an overflow occurs in the recipient's balance	Check if overflow is not handled	dFS-TF4
FB-OH-5		Check for code errors even though overflow is handled	
FB-OH-6		Check if it returns False (Throw is recommended instead of returning false)	
FB-DH-3	Insufficient tokens to send in the sender's account balance	Check if it returns false (It should throw)	dFS-TF5
FB-IS-2	Invalid sender	Check if it returns false (It should throw)	dFS-TF6

Table 5. A Checklist on the Failure of approve in the API Behavioral Conformance

Name	Category	Checklist	Failed Rules
FB-EV-1	caller's balance≥value	Check if an Approval event is fired	dFS-AP1
FB-DH-4	caller's balance<value	Check if it reverts	dFS-AP2
FB-DH-5		Check if an Approval event is fired	
FB-IS-2	Invalid caller	Check if it reverts	dFS-AP3
FB-IS-3		Check if an Approval event is fired	

를 출력하여 스마트 계약의 소스 프로그램에서 어느 부분을 살펴보아야 할지 제시하여 개발자가 스스로 실패 이유를 점검할 수 있도록 한다.

예를 들어, 대상 스마트 계약의 transfer 함수에 대한 특정 세부 기능 사양(dFS-TR1 ~ dFS-TR5)에 대해 실패하는 경우 Table 3의 해당하는 경우를 찾아 연관된 체크리스트를 결과로 함께 출력하여 스마트 계약의 개발자로 하여금 검토할 수 있도록 제안한다. Table 4는 transferFrom 함수에 대한 특정 세부 기능 사양(dFS-TF1 ~ dFS-TF6)을 테스트하는 도중 실패하면 역시 같은 방법으로 해당하는 체크 리스트를 찾아 개발자가 검토하도록 한다. Table 5는 approve 함수에 대한 테스트가 실패하였을 때 살펴보도록 제안하는 체크리스트이다. 이 체크리스트들은 기존 100개의 토큰 스마트 계약을 관례상 표준 ERC-20 API 기능 동작과 비교 분석하여 도출한 것이다. 기존의 토큰들이 테스트에 실패한 원인을 코드를 검토하여 분석하였다. 예를 들어 Table 3의 FB-OH-1,2,3은 transfer 함수의 오버플로 처리에 관한 것이다. FB-OH-1은 오버플로 처리를 지원하지 않는 경우로 9개 토큰에서 발생하였다. FB-OH-2의 오류는 ANT 토큰의 코드 Fig. 2에서 확인할 수 있다. balanceOfAt 함수는 128 비트 정수를 변환해 256 비트 부호없는 정수를 반환하도록 작성되었다. 잔액의 저장소는 128 비트이므로 실제로는 오버플로가 발생하는 경우에도 previousBalanceTo과 _amount를 더하면 오버플로가 발생하지 않아 코드의 if 조건에 걸리지 않으므로 throw하지 않고 true 값을 반환하여 오버플로가 없음을 나타내게 된다. FB-OH-3는 오버플로시 throw하지 않고 false값을 리턴하는 경우이다[6].

마지막으로 세 번째 단계에서, 관례상 표준 ERC-20 API (상위 100개 토큰 중 65개 토큰이 공유하는 관례상 표준 클래스의 API 동작)의 기능 동작을 준수하는지 적합성 여부를 최종 결과로 출력한다. 만일 최종 적합성 여부가 실패로 나오면 테스트 대상의 스마트 계약과 19개 테스트(dFS-TS1 ~ dFS-AL2) 결과가 유사한 스마트 계약 프로그램들을 제시하여 비교할 수 있도록 가이드 한다. 100개 스마트 계약 프로그램들 중 관례상 표준을 조금이라도 따르지 않는 기존의 35개 스마트 계약 프로그램들 중에서 유사한 API 기능 동작을 보이는 것을 선택한다.

```
var previousBalanceTo = balanceOfAt(_to, block.number);

// Check for overflow
if (previousBalanceTo+_amount<previousBalanceTo) throw;
```

Fig. 2. Code snippet of ANT Token

기존 100개 ERC-20 토큰 스마트 계약 프로그램들에 대한 분석 결과가 없었다면 Table 3 ~ Table 5의 체크 리스트를 도출하기가 어려웠을 것이다. 아직까지 ERC-20 표준의 기능 동작에 대한 엄밀한 정의를 제시하거나 관례상 표준을 정의한 연구가 없었을 뿐만 아니라 그러한 이유로 이 논문에서 제시한 ERC-20 스마트 계약 프로그램의 표준 적합성을 검토하기 위한 체크리스트에 대한 연구가 진행되지 못한 것으로 판단된다.

5. 실험 및 평가

이 논문에서 제안한 ERC-20 API 기능 동작에 관한 관례상 표준에 적합한지 확인하는 리뷰 절차를 구현하고 새로운 토큰 스마트 계약 프로그램 WBTC와 INS 토큰에 적용한 실험을 수행하고 제안한 리뷰 방법에 대해 평가한다.

Fig. 3은 이더리움 네트워크의 통계 사이트인 이더스캔 ERC-20 토큰 목록에서 이 실험의 벤치마크 프로그램 목적으로 가져온 WBTC 토큰[31]에 대해 리뷰 절차를 수행한 결과를 보여준다. 리뷰 결과 1단계를 통과하여, Table 1의 ERC-20 API 인터페이스를 모두 따르고 있음을 확인하였다. 2단계에서는 전체 19개 테스트 모두를 통과하였다. 3단계에서는 이러한 테스트 결과가 기존 관례상 표준을 따르는 스마트 계약의 테스트 결과와 동일함을 리포트하고 있다. 기존 관례상 표준을 따르는 동일한 행동의 토큰 스마트 계약 프로그램의 예시로 LINK, LEO, USD, MKR, HEDG, CRO를 마지막으로 제시하여 비교할 수 있도록 하였다.

INS 토큰의 리뷰 결과는 Fig. 4이다. 1단계를 통과하여 모든 표준 API 인터페이스를 따르고 있음을 확인하였고 2단계에서는 TR4, TR5, TF3, TF5, TF6에서 실패하는데 이는 코드 리뷰한 결과 체크리스트에 나온바와 같이 throw해야함에

```
[Result of API prototype compatibility of ERC-20]
  : Pass

[Result of Behavioral compatibility of ERC-20]
  • Pass: 19
  • Fail: 0

[Behavioral differences from de facto standard]
  : None

[A list of names of behaviorally similar tokens]
  • Tokens of Same behavior: LINK,LEO,USD,MKR,HEDG,CRO
```

Fig. 3. An Example of Running the ERC-20 API Conformance Review Procedure with WBTC

```
[Result of API prototype compatibility of ERC-20]
  : Pass

[Result of Behavioral compatibility of ERC-20]
  • Pass: 14
  • Fail: 5
   - TR4 Functional Specification fail.
     TR4: transfer(): sender.balance < value
       1) Check if it returns false (It should throw)
   - TR5 Functional Specification fail.
     TR5: transfer(): invalid sender
       1) Check if it returns false (It should throw)
   - TF3 Functional Specification fail.
     TF3: transferFrom(): sender.balance >= value & to.balance+value <= MAX &
                          caller's allowance by from < value
       1) Check if it returns false (It should throw)
   - TF5 Functional Specification fail.
     TF5: transferFrom(): from.balance < value
       1) Check if it returns false (It should throw)
   - TF6 Functional Specification fail.
     TF6: transferFrom(): invalid from
       1) Check if it returns false (It should throw)

[Behavioral differences from de facto standard]
  TR4, TR5, TF3, TF5, TF6

[A list of names of behaviorally similar tokens]
  • Tokens of Same behavior: AION,CEL,FUN,RCN,EURS
```

Fig. 4. An Example of Running the ERC-20 API Conformance Review Procedure with INS

도 불구하고 false값을 반환하고 있어서이다. 3단계에서는 관례 표준과 다른 기능 명세 목록을 제시하고 또한 유사한 동작을 하는 토큰들로 AION, CEL, FUN, RCN, EURS을 제시하고 개발자가 비교해볼 수 있게 한다.

이 논문에서 리뷰 절차는 테스팅 기반으로 적용하기가 매우 쉬워 개발자들이 개발한 토큰 스마트 계약 프로그램의 ERC-20 API 기능 동작을 관례상 표준과 비교하기 좋은 장점을 가지고 있다. 다만 테스팅 방법의 한계상 테스트 실패는 분명히 표준을 따르지 않는 반례를 찾았지만 테스트 성공으로 표준을 반드시 따른다고 확인한 것은 아니다. 이러한 단점을 극복하기 위하여 리뷰 3단계에서 테스트 결과가 유사한 기존 스마트 계약 프로그램을 제시하여 리뷰 이후에 개발자 스스로 비교하게 하여 이러한 단점을 보완하도록 하였다. 향후 기호 실행[16-18]에 기반하여 이론적으로 모든 경로를 테스트하는 더욱 정교한 검사 방법을 도입하여 이 연구에서 제안한 리뷰 절차의 단점을 보완할 수 있을 것이다.

6. 결 론

본 논문에서 이더리움 블록체인의 ERC-20 토큰 스마트 계약 프로그램들의 관례상 표준에 부합하는지 확인하는 리뷰 절차와 이를 지원하는 도구를 만들어 실험하고 평가하였다. 기존 이더리움 블록체인 시장 상위 100개의 토큰 스마트 계약 프로그램들을 ERC-20 API 기능 동작 면에서 분석한 지식을 바탕으로 관례상 표준을 명시적으로 정의하였고, 이렇게 정의된 관례상 표준으로 새로운 ERC-20 스마트 계약 프로그램을 리뷰할 수 있는 방법을 설계할 수 있었다.

향후 연구로 기호 실행을 통해 관례상 기능 명세 표준 준수 여부를 판단하는 방법을 추진하고자 한다. 현재 제안한 방법에서 기능 명세를 테스트케이스를 통해서 구현하여 리뷰하는 방법은 테스팅 방법의 속성상 테스트 케이스를 실패한 기능 명세는 준수하지 않았음을 확인할 수 있지만 테스트 케이스를 통과한 기능 명세를 반드시 준수한다고 증명한 것은 아니다. 기호 실행 방법은 스마트 계약 프로그램의 가능한 모든 경로를 탐색하기 때문에 이러한 점을 보완할 수 있으리라 기대한다.

References

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Yellow Paper, [Internet], https://ethereum.github.io/yellowpaper/paper.pdf, 2018, Accessed May 2022.

[2] V. Buterin, "A next-generation smart contract and decentralized application platform," Ethereum White Paper, [Internet], https://ethereum.org/en/whitepaper/, Accessed May 2022.

[3] N. Szabo, "Smart contracts: Formalizing and securing relationships on public networks," First Monday, Vol.2, No.9, 1997.

[4] F. Vogelsteller, and V. Buterin, "EIP-20: ERC-20 Token Standard," [Internet], https://eips.ethereum.org/EIPS/eip-20, 2015, Accessed May 2022.

[5] Etherscan [Internet], https://etherscan.io, Accessed May 2022.

[6] H. Moon, and S. Park, "Conformance evaluation of the top-100 Ethereum relationships on public token smart contracts with Ethereum Request for Comment-20 functional specifications," *IET Software*, Vol.16, No.2, pp.233-249, 2022.

[7] CVE-2021-33403, Integer overflow in LNC token [Internet], https://github.com/MRdoulestar/SC-RCVD/blob/main/Vulnerabilities/LNCToken.md, Accessed May 2022.

[8] CVE-2018-11239, burnOverflow in Hexagon token [Internet], https://peckshield.medium.com/new-burnoverflow-bug-identified-in-multiple-erc20-smart-contracts-cve-2018-11239-52cc4f821694, Accessed May 2022.

[9] EtherDelta. 2018. [Internet], https://etherdelta.com/, Accessed May 2022.

[10] T. Chen et al., "TokenScope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, New York, pp.1503-1520, 2019.

[11] J. Feist, G. Greico, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB '19)*, IEEE Press, pp.8-15, 2019.

[12] E. Hildenbrandt et al., "KEVM: A complete formal semantics of the ethereum virtual machine," in *Proceedings of 2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pp.204-217, 2018.
[13] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, Vol.7, pp.77894-77904, 2019.
[14] S. Tikhomirov et al., "SmartCheck: Static analysis of ethereum smart contracts," in *Proceedings of 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp.9-16, 2018.
[15] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE'18, pp.259-269, 2018.
[16] S. So, M. Lee, J. Park, H. Lee, and H. Oh, "VERISMART: A highly precise safety verifier for ethereum smart contracts," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, IEEE, pp.1678-1694, May 2020.
[17] S. So, S. Hong, and H. Oh, "SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution," in *Proceedings of 30th USENIX Security Symposium*, pp.1361-1378, 2021.
[18] J. Frank, C. Aschermann, and T. Holz, "ETHBMC: A bounded model checker for smart contracts," in *Proceedings of the 29th USENIX Security Symposium*, pp.1-18, 2020.
[19] L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. pp.254-269, 2016.
[20] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proceedings of 25th Annual Network and Distributed System Security Symposium*, pp.1-15, 2018.
[21] L. Alt, and C. Reitwießner, "SMT-Based verification of solidity smart contracts," in *Proceedings of Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice: 8th International Symposium*, pp.376-388, 2018.
[22] P. Tsankov, A. Dan, D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp.67-82.
[23] OpenZeppelin, "An ERC-20 test suite," [Internet], https://github.com/OpenZeppelin/openzeppelin-contracts, Accessed May 2022.
[24] Google, "Compatibility Test Suite", 2020 [Internet], https://source.android.com/compatibility/cts, Accessed May 2022.
[25] W3C, "Markup Validation Service," [Internet], https://validator.w3.org/, Accessed May 2022.
[26] IEEE and The Open Group, "PosixTM Certification", [Internet], http://get.posixcertified.ieee.org/, 2020, Accessed May 2022.
[27] J. Tretmans, "An Overview of OSI Conformance Testing", 2001.
[28] G. Ye et al., "Automated conformance testing for JavaScript engines via deep compiler fuzzing," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*, pp.435-450, 2021.
[29] B. Loring and J. Kinder, "Systematic generation of conformance tests for JavaScript", 2021 [Internet], https://doi.org/10.48550/arXiv.2108.07075, Accessed May 2022.
[30] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," in *Proceedings of the 6th International Conference on Principles of Security and Trust*, Vol.10204, pp.164-186, 2017.
[31] WBTC token smart contract [Internet], https://etherscan.io/token/0x2260fac5e5542a773aa44fbcfedf7c193bc2c599, Accessed May 2022.

문 현 아

https://orcid.org/0000-0001-7359-3298
e-mail : hamoon@sogang.ac.kr
1994년 서강대학교 컴퓨터공학과(학사)
1996년 서강대학교 컴퓨터공학과(석사)
2016년 ~ 현 재 서강대학교 컴퓨터공학과 박사과정
관심분야 : 소프트웨어개발 교육, 소프트웨어 안전성, 블록체인

박 수 용

https://orcid.org/0000-0002-3979-0586
e-mail : sypark@sogang.ac.kr
1986년 서강대학교 컴퓨터공학과(학사)
1988년 Florida State University, Computer and Information Science(석사)
1995년 George Mason University, Information Technology(박사)
1998년 ~ 현 재 서강대학교 컴퓨터공학과 교수
2017년 ~ 현 재 지능형 블록체인연구센터 센터장
2018년 ~ 현 재 한국블록체인학회 학회장
관심분야 : 요구공학, 동적 아키텍처, 블록체인