솔리디티 스마트 컨트랙트의 스크립트 기반 시나리오 테스트

문현아 ○ 박수용
서강대학교 컴퓨터공학과
{hamoon,sypark}@sogang.ac.kr

최광훈
전남대학교 전자컴퓨터공학부
kwanghoon.choi@jnu.ac.kr

A Script-based Scenario Test for Smart Contracts in Solidity

HyeonAh Moon○ Sooyong Park
Dept. of Computer Science
Sogang University

KwangHoon Choi
Dept. of Elect. and Computer Science
Chonnam National University

요 약

본 연구에서는 솔리디티 언어로 작성한 스마트 컨트랙트의 안정성 테스트를 위해 프로그램의 동작 시나리오를 쉽게 기술하고 직접 실행 가능하도록 설계한 스크립트 언어 이더스크립트(EtherScript)를 제안한다. 예제 시나리오를 통해 분산 환경에서 동작하는 스마트 컨트랙트의 특성을 고려하고, 이더의 흐름을 나타내는 다양한 시나리오를 구현하기에 적절한 언어임을 확인하였다.

1. 서 론

블록체인은 2008년 비트코인이 공개된 이후, 중개기관에 의존하지 않고 거래 당사자들간의 직접 거래를 가능하게 하는 기술로 많은 관심을 받고 있다. 특히, 이더리움 블록체인에서 처음 상용화된 스마트 컨트랙트는 블록체인 산업의 새로운 가능성을 보여주는 특성이다. 다양한 응용프로그램을 블록체인 플랫폼에서 사용할 수 있게 하는데, 예를 들면 가장 활발히 사용되는 ICO(Initial Coin Offering)을 위한 스마트 컨트랙트나 고양이를 사고 팔 수 있는 게임을 제작할 수도 있다. 이처럼 스마트 컨트랙트는 다양한 비즈니스 프로세스에 대응할 수 있지만 기존의 응용 프로그램들과는 다르게 탈중앙화 분산 환경에서 작동하는 특성으로 인해 프로그래머가 의도치 않은 오류가 발생할 가능성이 있다. 게다가 화폐와 밀접한 특성 또한 가지고 있어서 DAO공격[1]이나 Parity Wallet 사건[2][3]에서와 같이 프로그램의 작은 오류가 큰 자산의 손실을 부르기도 한다. 스마트 컨트랙트의 또다른 특성은 한번 배포된 후에는 수정이 불가능하다는 점이다. 이런 모든 특징들을 고려할 때 스마트 컨트랙트 개발에서의 테스트의 중요성은 더욱 증대된다.

스마트 컨트랙트를 개발한 후 메인 넷에 배포해 실제 테스트한다는 것은 리스크가 크고, 테스트 넷 테스트는 불필요한 지연 시간 등이 필요하므로 이전에 테스트 환경을 구축해 실행하는 것이 좋다. 사설망을 생성해 테스트 하는 경우는 사설망 생성부터 계정 생성 그리고 이후 계정의 비밀번호 관리까지 테스트와 관련없는 불필요한 작업에 시간을 많이 소모한다.

현재 개발에 많이 활용하는 트러플(Truffle) 프레임워크를 이용한 테스트는 자바스크립트와 솔리더티(Solidity) 모두 테스트 작성에 사용할 수 있다. 이 방식은 단위 테스트에는 유용하나 시나리오를 테스트 하려면 자바스크립트 문장 작성이 복잡하고 어렵다.

본 논문에서는 화폐의 가치를 지닌 이더의 흐름을 잘 기술하고 스마트 컨트랙트 프로그램의 동작을 표현하는 시나리오를 작성하기 쉽게 하는 이더스크립트 언어를 설계하고 구현하였다. 이 언어를 실제 스마트 컨트랙트의 시나리오 테스트에 적용하여 유용함을 확인하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 살펴보고, 3장에서는 스마트 컨트랙트 시나리오 테스트를 위한 스크립트 언어인 이더스크립트의 설계 동기와 실제 적용 결과를 평가하고 이더스크립트 해석기의 구조를 제시한다. 마지막으로 4장에서는 결론과 향후 연구 방향에 대해 논의한다.

2. 관련 연구

현재 스마트 컨트랙트의 안정성을 위해 트러플(Truffle)[4], 슬리터(Slither: 사용자 개입없이 몇 초 안에 실제 취약점을 발견 할 수 있는 오픈 소스 정적 분석 프레임 워크)[5], 에키드나(Echidna: 추상적 상태 머신 모델링 및 최소한의 자동 테스트 케이스

생성과 같은 강력한 기능을 갖추고 있는 스마트 퍼저)[6], 오인트(Oyente: 심볼릭 실행을 이용한 취약점 분석도구)[7], 스마트첵(SmartCheck: 알려진 취약성 및 나쁜 습관을 온라인에서 자동으로 검사하고 코드에서 해당 부분을 강조 표시하고 문제에 대한 자세한 설명을 제공)[8], 미스릴(Mythril: concolic 분석, 오염 분석 및 제어 흐름 검사를 사용하여 다양한 보안 취약점을 탐지)[9] 등의 여러 테스트 및 디버깅 도구와 서드파티 감사 서비스를 이용할 수 있다. 그러나 이 도구들은 시나리오 테스트와 이더 흐름 분석에는 적절하지 않다.

특히, CLI를 이용하거나 리믹스(Remix)에서 시나리오 테스트 하는 경우는 1) 시나리오에 필요한 만큼 계정 생성, 시나리오에 따라 2) 어떤 계정에서 스마트 컨트랙트를 배포하고 3) 어떤 계정으로 스마트 컨트랙트 내의 함수를 호출할 것인지 등을 매번 바꾸어 주는 것은 매우 번거롭고, 다시 테스트 해야 하는 경우 같은 작업을 반복해야 한다.

3. 이더스크립트: 스마트 컨트랙트 시나리오 테스트를 위한 스크립트 언어

이 절에서는, 이더리움 기반 스마트 컨트랙트 프로그램의 동작 시나리오를 쉽게 기술하고 직접 실행 가능하도록 설계한 스크립트 언어 이더스크립트(EtherScript)를 제안한다.

3.1 이더스크립트 언어를 설계한 동기

스마트 컨트랙트 프로그램은 여러 계정 또는 다른 스마트 컨트랙트와 자유롭게 이더(ether)를 주고받거나 함수를 호출 하면서 동작하도록 구성되어 있어 시나리오 테스트를 준비하는 과정이 더 복잡하다. 주어진 시나리오에 따라 이 프로그램을 테스트하려면 첫째, 시나리오에 관여하는 계정을 생성하고 적절한 이더를 보유하도록 초기화하고, 시나리오에서 연동할 다른 스마트 컨트랙트도 추가로 설치해야 한다. 둘째, 이렇게 준비된 계정들과 스마트 컨트랙트들이 정해진 시나리오에 따라 이더를 주고받고 함수를 호출하는 순서를 정하고 이 순서대로 실행한다.

솔리디티 기반 스마트컨트랙트 프로그램의 개발환경 리믹스에서 시나리오 테스트를 진행하려면 웹 기반 사용자 인터페이스를 통해 텍스트 창에 계정 주소, 함수 인자, 송금할 이더 등을 입력하고 버튼을 눌러 실행하는 과정을 반복해야 한다. 시나리오 기반 회귀 테스트를 위해 이러한 작업은 자동화되어야 한다.

대표적인 이더리움 클라이언트인 고이더리움(Geth)나 이더리움제이(Ethereumj)에서 계정 만들기, 스마트 컨트랙트 설치하기, 송금, 함수 호출을 범용 프로그래밍언어인 자바스크립트와 자바 프로그램을 실행하여 수행할 수 있다. 하지만 간단한 시나리오를 테스트하기 위해서 상당히 긴 프로그램을 작성해야한다. 게다가 솔리디티 프로그램을 작성하는 프로그래머가 자바스크립트나 자바 언어에 능숙 해야 하는 문제가 있다.

따라서 이러한 문제를 개선하고자 솔리디티 프로그램의 시나리오를 쉽게 기술하고 자동으로 반복 실행하기 적합하며 솔리디티 언어와 유사한 구문으로 작성할 수 있는 스크립트 언어를 제안한다.

3.2 스크립트 언어 및 적용 사례

솔리디티 프로그램에 대한 시나리오를 실행하도록 작성한 스크립트 프로그램 사례를 들어 이더스크립트 언어를 소개한다.

```
// simple_storage.sol
contract SimpleStorage {
 uint public storedData;
 function set(uint x) public {   storedData = x; }
 Function get() public view returns (uint)
   {    return storedData;     }

// simple_storage.es
Line1: account{balance:10ether} owner;
Line2: account{balance:50ether} user;
Line3: account{contract:“simple_storage.sol”,
          by:owner}  simplestorage{“SimpleStorage”};
Line4: simplestorage.set(123) {by:user};
Line5: uint x;
Line6: x = simplestorage.get() {by:user};
Line7: assert x==123;
```

그림 1. 솔리디티 프로그램과 이더스크립트 프로그램

[그림 1]의 simple_storage.sol은 컨트랙트 SimpleStorage를 선언한 솔리디티 프로그램이다. 이 컨트랙트는 uint 타입의 상태 변수 storedData와 이 변수 값을 설정하고 읽는 두 개의 함수 set과 get으로 구성되어 있다.

[그림 1]의 다음 예제 simple_storage.es는 본 논문에서 제안한 이더스크립트로 작성한 프로그램이다. 앞의 컨트랙트에서 선언한 두 함수의 동작을 테스트하기 위해 간단한 시나리오를 작성하였다. 첫 번째와 두 번째 줄에서 컨트랙트를 설치할 계정과 이 컨트랙트를 사용할 계정으로 owner와 user를 선언한다. 각 계정을 선언할 때 balance 속성으로 초기 보유할 이더를 지정한다. 세 번째 줄에서 [그림 1]의 솔리디티 프로그램 이름, 컨트랙트를 생성한 계정, 컨트랙트 이름을 지정하여 컨트랙트 계정 simplestorage를 만든다. 이 계정을 통해 컨트랙트에서 제공하는 함수를

호출할 수 있다. 네번째 줄에서 set함수로 123을 상태 변수에 설정하고, 여섯 번째 줄에서 get함수를 통해 다시 읽어온 다음, 일곱 번째 줄에서 assert문으로 확인한다.

본 연구에서 이더스크립트 해석기를 구현하였다. 스크립트 프로그램을 실행하면 먼저 이더리움 블록체인을 새로 만들고, 스크립트 프로그램에 선언된 계정을 생성한 다음 프로그램 문장들을 차례대로 실행하도록 해석기를 구성하였다.

이더스크립트 활용 사례로 이더리움의 대표적인 취약점으로 알려진 다오(DAO) 공격 시나리오와 블록체인과 스마트 컨트랙트를 활용하는 예탁 판매(Escrow) 시나리오를 작성하였다.

```
// dao.sol
contract SimpleDAO {
  mapping (address => uint) public credit;
  function donate(address to) payable {
    credit[to] += msg.value;
  }
  function queryCredit(address to) constant returns (uint)
  { return credit[to]; }
  function withdraw(uint amount) {
    if (credit[msg.sender] >= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender] -= amount;
}}}

contract Mallory {
  SimpleDAO public dao;
  address owner;
  function Mallory(SimpleDAO _dao) {
    dao = _dao; owner = msg.sender;
  }
  function() payable
    { dao.withdraw(dao.queryCredit(this));  }
  function getJackpot()
    { owner.send(this.balance);  }
}
```

그림 2. 다오 컨트랙트와 맬러리 컨트랙트

[그림 2]에서 컨트랙트 SimpleDAO는 여러 사용자가 기부한 재원을 모으는 프로그램이고, 컨트랙트 Mallory는 다오 취약점 공격에서 해커가 사용할 프로그램이다. 이 컨트랙트들과 다오 취약점에 대한 자세한 설명은 [10]을 참고한다.

[그림 3]은 이 다오 취약점을 공격하는 시나리오를 이더스크립트로 작성한 프로그램이다. 1번째 줄에서 5번째 줄까지 이 시나리오에 관여하는 계정을 만들고, 6번째 줄에서 다오 컨트랙트 SimpleDao의 계정 dao를 생성한다. 이 계정의 donate함수를 통해서 사용자 계정 user1, user2, user3에서 1이더씩을 이 dao 계정에 보낸다. 10번째 줄에서 해커는 컨트랙트 Mallory 계정 mallory를 만들고 11번째 줄에서 100피니를 먼저 다오 계정에 기부한 다음 12번째 줄에서 Mallory 컨트랙트의 폴백(fallback) 함수를 호출하여 다오의 재진입 취약점(reentrancy vulnerability)을 공격하여 다오 계정에 세가지 사용자 계정에서 기부한 3이더를 mallroy 계정으로 뺏고, 13번째 줄에서 이 돈을 해커 계정으로 옮겨온다. 14번째 줄에서 해커 계정의 금액을 assert문을 통해 출력하여 확인한다.

```
Line01: account{balance:50ether} owner;
Line02: account{balance:50ether} user1;
Line03: account{balance:50ether} user2;
Line04: account{balance:50ether} user3;
Line05: account{balance:50ether} hacker;
Line06: account{contract:"dao.sol", by:owner}
          dao("SimpleDAO");
Line07: dao.donate(user1) {by:user1, value:1ether};
Line08: dao.donate(user2) {by:user2, value:1ether};
Line09: dao.donate(user3) {by:user3, value:1ether};
Line10: account{contract:"dao.sol", by:hacker}
          mallory("Mallory", dao);
Line11: dao.donate(mallory) {by:hacker, value:100finney};
Line12: mallory.() {by:hacker};
Line13: mallory.getJackpot() {by:hacker};
Line14: assert hacker.balance;
```

그림 3. 다오 취약점 공격 시나리오를 구현한 스크립트

아래 GitHub 프로젝트에서 지금까지 소개한 두 가지 예제를 포함해서 예탁 판매와 경매 예제 (King of the Throne) 관련 솔리디티 프로그램과 이더스크립트 프로그램을 내려 받을 수 있다.

- https://github.com/kwanghoon/ethereumj/tree/working

3.3 평가

이 논문에서 제안한 이더스크립트를 제대로 평가하기에 아직 초기 개발 단계이지만 현재까지 연구를 수행한 경험을 바탕으로 정량적 평가와 정성적 평가를 수행하였다.

정량적 평가는 3가지 동일한 시나리오에 대해 이더스크립트 언어로 작성한 스크립트 프로그램과 이더리움제이 플랫폼 상에서 자바로 작성한 프로그램을 LOC(Line Of Count) 기준을 측정하여 시나리오 작성의 용이한 정도를 비교하였다. 이 자바 프로그램도 위 GitHub 사이트에서 찾아볼 수 있다.

표 1. LOC 비교

LOC	EtherScript	Java (EthereumJ)
Simple Storage	8	47
Dao	18	82
Escrow	13	62

동일한 시나리오를 이더스크립트 언어를 사용하면 3가지 사례를 기준으로 평균 20%에 해당하는 LOC 정도로 작성할 수 있었다. 자바 프로그램의 LOC를 셀 때 주석이나 공백뿐만 아니라 이더스크립트 프로그램과 직접적인 비교 대상이 아닌 줄도 모두 제외하였기 때문에 이 프로그램의 실제 LOC는 더 늘어난다. 이러한 점을 비교해보면 이더스크립트로 시나리오를 더 간단하게 작성할 수 있음을 확인하였다.

정성적 측면에서 이더스크립트는 두 가지 장점을 가지고 있다. 첫째, 이더리움제이 플랫폼에서 자바로 시나리오를 작성한다면 두 가지 언어를 알아야 하는 부담이 있다. 솔리디티 프로그래머가 테스트하고 싶은 시나리오를 직접 작성하도록 하기 위해서 이더스크립트는 솔리디티 언어와 유사한 구문을 사용하도록 설계하였다. 특히 화폐의 가치가 있는 이더를 주고받는 이더리움 도메인에 특정적인 새로운 스크립트 언어로 솔리디티 프로그램에서 가장 중요한 특징인 이더의 흐름을 시나리오로 구현하기 매우 간단하다. 즉, 솔리디티 사용자에게 익숙한 계정 생성, 함수 호출을 통한 계정간의 이더 전송, 각 계좌의 잔고 확인을 위한 이더스크립트 구문을 이용하여 쉽게 구현할 수 있다.

둘째, 리믹스와 같은 솔리디티 프로그래밍 환경에서 테스트하는 방법과 비교하면 이더스크립트 프로그램은 자동으로 반복 실행이 쉬운 장점을 가지고 있다. 따라서 회귀 테스트에 더 적합하다.

3.4 이더스크립트 해석기 구현

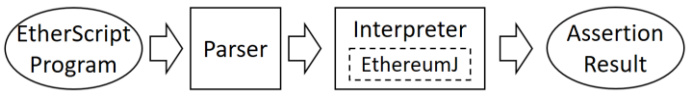

그림 4. 이더스크립트 해석기 구성

이더스크립트 해석기는 크게 파서와 해석 모듈로 구성되어 있다. 솔리디티 언어의 구문 정의를 최대한 참조하되 이더스크립트에만 있는 계정 선언 구문과 컨트랙트 계정 선언 구문을 추가하였다.

해석 모듈은, 파서 모듈에서 출력한 이더스크립트 추상구문트리를 따라 순서대로 한 문장씩 실행하는 구조이다. 스크립트 프로그램 실행 전에 이더리움 블록체인을 생성하고, 일반 계정과 컨트랙트 계정 생성, 송금이나 컨트랙트 함수 호출을 구현하기 위해서 이더리움제이에서 제공하는 라이브러리를 활용하였다.

4. 결론 및 향후 연구

본 논문에서 이더리움 솔리디티 프로그램의 동작에 관한 시나리오를 쉽게 표현할 수 있는 이더스크립트 언어를 설계하고 구현하였다. 이 언어를 사용하여 대표적인 스마트 컨트랙트 활용 시나리오를 작성한 경험에 대해 평가하였다. 화폐 가치가 있는 이더를 주고받는 이더리움 도메인을 고려하여 설계한 언어이기 때문에 이더 흐름에 대한 시나리오를 구현하는데 적절한 언어임을 확인하였다.

향후 연구로, 첫째, 더 많은 사례 연구를 통해 이더스크립트 언어의 활용도를 평가하고자 한다. 둘째, 솔리디티 프로그램과 시나리오에 대한 모델을 바탕으로 테스트 시나리오를 자동 생성하고 이 시나리오를 이더스크립트로 작성하여 테스트하는 방법에 대한 연구를 수행하고자 한다.

참고문헌

[1] Analysis of the DAO exploit. http://hackingdistributed.com/2016/06/18/analysis-of-thedao-exploit/. Last access, 2018.

[2] The Parity Wallet Hack Explained. https://blog.zeppelin.solutions/on-the-parity-wallet-multisighack- 405a8c12e8f7. Last access, 2018

[3] The Wallet Smart Contract Frozen by the Parity Bug. https://github.com/paritytech/parity/blob/4d08e7b0aec46443bf26547b17d10cb302672835/js/src/contracts/snippets/enhanced-wallet.sol. Last access, 2018.

[4] TRUFFLE SUITE. https://truffleframework.com/. Last access, 2018.

[5] Slither.https://blog.trailofbits.com/2018/10/19/slither-a-solidity-static-analysis-framework/. Last access, 2018.

[6] Echidna.https://blog.trailofbits.com/2018/03/09/echidna-a-smart-fuzzer-for-ethereum/. Last access, 2018.

[7] L. Luu, D.H. Chu, H. Olickel, et al. Making Smart Contracts Smarter. In Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security, pp:254-269, Vienna, Austria, 2016.

[8] SmartCheck. https://tool.smartdec.net/. Last access, 2018.

[9] Mythril. https://mythril.ai/. Last access, 2018.

[10] N. Atzei, M. Bartoletti, T. Cimoli. A Survey of Attacks on Ethereum Smart Contracts. In proceedings of the 6th International Conference on Principles of Security and Trust, pp: 164-186, Berlin, Heidelberg, 2017.