이더리움 스마트 컨트랙트 보안 취약점 분석 및 대응방안에 대한 연구

김현준†, 오석환‡, 김구민‡, 김휘수‡, 송효준‡, 하태균†, 김경백‡
한국인터넷진흥원†, 전남대학교 정보보안협동과정‡
hyunjun@kisa.or.kr, tjr73010@naver.com, gumin6395@naver.com, bennett95@naver.com,
thdgywns12@naver.com, niceha@kisa.or.kr, kyungbaekkim@jnu.ac.kr

A Study on the Analysis and Response of Ethereum Smart Contract Security Vulnerabilities

Hyunjun Kim†, Seokhwan Oh‡, Gumin Kim‡, Hwi Soo Kim‡, Hyojun Song‡, Taegyun Ha†, Kyungbaek Kim‡
Korea Internet & Security Agency†, Chonnam National Univ.‡

요 약

스마트 컨트랙트는 계약 당사자 간 서로 신뢰할 필요가 없으며 코드에 의해 정의되어 실행조건이 충족될 시 자동으로 실행되는 전자 계약이다. 수동적인 개입을 최소화하는 이점이 있어 금융, 의료, 교육 등 여러 분야에 접목되어 활발하게 활용되고 있다. 그러나 한 번 블록체인에 배포되면 수정할 수 없는 스마트 컨트랙트의 특성 관련하여 보안 취약점으로 인한 사고가 이어지고 있다. 본 논문에서는 최근 이더리움 플랫폼에서 널리 사용되고 있는 스마트 컨트랙트에 대한 보안 취약점을 분석하고 대응방안을 정리하였다.

Ⅰ. 서 론

블록체인은 현재 많은 분야에서 관심을 불러일으키며 차세대 인터넷을 이끌 혁신적인 기술 중 하나로 꼽히고 있다. 블록체인은 다수의 온라인 거래 기록을 한곳에 묶어 하나의 데이터 블록을 구성하고 해시값을 이용해 블록들을 체인처럼 연결한 뒤 P2P 방식으로 여러 컴퓨터에 분산 저장·관리하는 기술이다.

스마트 컨트랙트는 블록체인에서 널리 사용되며 당사자 간의 계약 조건을 코드로 작성하여 해당 조건이 충족되면 계약을 자동으로 이행하는 디지털 계약이다. 스마트 컨트랙트에는 트랜잭션에 대한 모든 정보가 포함되어 있으며 기존 계약과는 다르게 제3자가 관여하지 않고 암호화된 거래를 참여자 간 공유하기 때문에 신뢰성과 투명성이 보장된다.

그러나 스마트 컨트랙트는 한 번 블록체인에 배포되면 수정할 수 없어 이를 악용한 악의적인 공격이 발생하고 있는데 대표적으로 2016년 발생한 더 다오(The DAO) 사건이 있다[1]. 이는 스마트 컨트랙트 코드상의 보안 취약점을 통한 공격이었으며 이로 인해 스마트 컨트랙트 작성 시 보안의 중요성이 강조되는 계기가 되었다. 본 논문에서는 최근 널리 사용되고 있는 스마트 컨트랙트인 이더리움에 대한 보안 취약점을 분석하고 대응 방안과 보안 고려사항을 정리한다.

Ⅱ. 스마트 컨트랙트 개요

2-1 스마트 컨트랙트

스마트 컨트랙트는 자체 실행 가능한 프로그램 코드로서 블록체인에 배포되어 신뢰할 수 없는 두 당사자 간 계약을 자동으로 이행하는데 사용한다. 이더리움은 가장 일반적으로 사용되는 튜링 완전(Turing Complete) 블록체인 플랫폼으로 개발자가 소유권, 트랜잭션 형식 및 상태 전환 기능에 대한 자체 임의 규칙을 사용하여 스마트 컨트랙트를 작성하는데 사용한다.

2-2 스마트 컨트랙트 동작방식

스마트 컨트랙트로 구현하고자 하는 내용을 솔리디티 등의 프로그래밍 언어로 구현한다. 해당 코드를 컴파일하여 네트워크에 배포할 수 있도록 바이트코드를 생성하는데 트랜잭션에는 바이트코드를 담아 채굴자가 해당 트랜잭션이 담긴 블록을 채굴함으로써 트랜잭션은 블록체인 네트워크에 기록된다.

참여자는 ABI(Application Binary Interface)를 통해 배포된 스마트 컨트랙트 코드에 정의된 함수를 호출하는 바이트코드를 생성하고, 트랜잭션에 담아 블록체인 네트워크에 전달한다. 채굴자는 참여자의 바이트코드를 배포된 스마트 컨트랙트 코드에 따라 EVM(Ethereum Virtual Machine) 상에서 실행하며, 이때 수수료인 가스(gas)가 계산되면서 블록에 추가되고 실행 결과가 유효한 경우 실행 결과가 반영된다.

Ⅱ. 스마트 컨트랙트 취약점 및 대응방안 분석

3-1 코드 레벨 취약점 분석 필요성

스마트 컨트랙트는 스마트 컨트랙트를 작성하는데 있어서 코드 재사용을 하는 경우가 대다수이다. 일반적으로 사용되는 패턴을 사용하게 되면 어느 정도 검증된 코드를 사용할 수 있고, 유지보수하기에도 편하다는 장점이 있다. 하지만 코드를 재사용하게 되면 원본 코드에 취약점이 발생하는 경우 해당 코드를 사용한 모든 프로젝트에 같은 취약점이 존재하게 된다. 스마트 컨트랙트에 코드를 재사용한 경우 한번 배포가 되면 수정이 불가능한 점으로 인해 해당 컨트랙트는 더 이상 사용할 수 없는 상황이 발생하게 될 수 있다.

2018년 싱가포르 국립대학교와 런던대학교의 연구를 통해 이더리움 네트워크에서 약 34,000개 이상의 취약한 스마트 컨트랙트가 발견됐다. ZEUS라는 논문에 따르면[2], 약 2만2400개의 스마트 컨트랙트 코드 중 약 95%에 달하는 스마트 컨트랙트가 하나 이상의 취약점을 가지고 있다.

이와 같은 이유로 본 절에서는 DASP TOP10에 명시된 대표적인 스마트 컨트랙트 취약점을[3] 코드 레벨에서 분석하였다.

3-2 Re-entrancy

재진입 취약점은 컨트랙트의 어떤 함수가 호출된 후 호출이 완료되기 전 동일한 함수가 다시 호출되는 취약점으로 재귀호출과 동일하다. 재귀호출은 프로그램을 작성하는 여러 가지 방법 중 하나이며, 그 자체로는 취약점이 될 수 없다. 하지만 재진입 취약점은 의도하지 않은 재귀 호출이 발생함으로써 문제가 된다. 실제로 DAO Attack에 사용된 취약점으로 공격자는 이 취약점을 사용하여 컨트랙트에서 6천만 달러 상당의 이더 토큰을 탈취하였다. 그림 1은 재진입 취약점이 존재하는 코드로 두 번째 require 문 이하 코드에서 이더가 컨트랙트 외부로 전송된 후 상태 변수가 업데이트되어 취약점이 존재한다.

대응 방안으로는 솔리디티 공식 문서에 작성된 보안 권장 사항인 Checks-Effects-Interaction Pattern을 사용하는 방법이 있으며 내용은 다음과 같다[4]. 컨트랙트에 작성된 함수에 대해 함수를 호출한 사람, 범위 내 인수인지, 충분한 이더를 전송하였는지 등에 대한 검사를 수행한다. 다음으로 모든 검사를 통과하면 현재 컨트랙트의 상태 변수에 영향을 주고, 다른 컨트랙트와의 상호작용은 모든 기능의 마지막 단계에서 이루어져야 한다.

```
contract EtherStore {
  mapping(address => uint256) public balance;
  function depositFunds() public payable {
    balances[msg.sender] += msg.value;
  }
  function withdrawFunds(uint256 _weiTowithdraw) public {
  require(balances[msg.sender] >= _weiTowithdraw);
  require(msg.sender.call.value(_weiToWithdraw)());
  balances[msg.sender] -= _weiToWithdraw;
  }
}
```

그림 1. Re-entrancy 취약점

3-3 Access Control

Solidity에서 함수는 함수를 호출할 수 있는 방법을 지정하는 접근자(public, private, internal, external)가 있다. 해당 지정하는 함수를 사용자가 외부적으로 호출할 수 있는지, 다른 컨트랙트에 의해 내부적으로만 또는 외부적으로만 호출할 수 있는지에 대한 여부를 결정한다. 함수는 기본적으로 사용자가 외부에서 호출할 수 있는 public으로 지정되어있다. 접근제어 취약점은, 접근해서는 안되는 함수와 코드에 함수의 접근자를 무시함으로써, 허용되지 않은 사용자가 접근함으로써 발생한다. 그림 2에서 함수 A, B와 같이 접근자를 설정하지 않으면 기본적으로 public으로 설정된다. public으로 설정된 함수는 공격자가 배포된 컨트랙트에서 해당 함수를 호출하여 소유자를 공격자의 주소로 재설정할 수 있다.

대응 방안으로는 컨트랙트를 작성할 때 모든 함수에 대해 접근자를 항상 명시하는 간단한 방법으로도 취약점에 대한 예방이 가능하다. Solidity 최신버전에서는 지정자가 설정되지 않은 함수에 대해 컴파일 중 경고를 표시한다.

```
contract Access Control {
  function A() {
    // code line
  }
  function b() {
    // code line
  }
}
```

그림 2. Access Control 취약점

3-4 Denial of Service

이더리움의 블록은 담을 수 있는 Gas 양의 제한이 있다. Gas Limit에 도달했거나 예상치 못한 throw를 실행시켰을 때 사용할 수 없는 계약이 되어버리는 취약점이다. Gas Limit를 초과하게 되는 경우 트랜잭션 수행 이전 상황으로 컨트랙트가 초기화가 되고, 컨트랙트를 영원히 사용할 수 없을 가능성이 있다. 그림 3의 코드는 공격자가 의도적으로 배열에 인자를 무수히 늘릴 수 있어 블록의 Gas Limit을 초과시킬 수 있는 취약점이 존재한다.

대응 방안으로는 컨트랙트에 시간이 지남에 따라 무한히 커질 수 있을 것으로 예상되는 배열이 있는 경우, 전체 데이터 구조에 걸쳐 반복하는 형태의 작업을 지양해야 한다. 알 수 없는 크기의 배열을 반복하여 사용하는 경우 단일 트랜잭션에서 이더를 전송하지 않도록 하여 각 트랜잭션에 필요한 가스를 줄여서 블록 가스 한도를 초과할 위험을 줄이는 것이 중요하다.

```
contract DoS {
  address public owner;
  address[] investors;
  uint[] investorTokens;
  function invest() public payable {
    investors.push(msg.sender);
    investorTokens.push(msg.value * 5);
  }
  function invest() public payable {
    require(msg.sender == owner);
    for(uint i=0; i<investors.length; i++){
      transferToken(investors[i], investorTokens[i]);
    }
  }
}
```

그림 3. Denial of Service 취약점

Ⅳ. 결론

본 논문에서는 이더리움 스마트 컨트랙트 보안 취약점과 취약점별 대응 방안을 분석하였다. 스마트 컨트랙트 취약점에 대한 분석과 대응 방안에 맞춰 스마트 컨트랙트를 작성하는 것이 보안상 권장된다.

최근에 블록체인 기반의 DID, NFT 등 기술들이 개발되어 서비스화 되어가고 있고, 특히 NFT의 경우 이더리움 표준에 따라 스마트 컨트랙트 코드가 작성되고 있다. 이에, 스마트 컨트랙트 보안취약점 대응방안에 대한 최신화된 진단 도구와 가이드라인을 지속적으로 연구할 필요가 있다.

참 고 문 헌

[1] GeeksforGeeks, What was the DAO Hack?, Available: https://www.geeksforgeeks.org/what-was-the-dao-hack/.

[2] S. Kalra, S. Geol, M. Dhawan, and S. Sharma, “ZEUS: Analyzing Safety of Smart Contracts,” Network and Distributed Systems Security(NDSS) Symposium, San Diego: CA, pp. 1-12, 2018., http://dx.doi.org/10.14722/ndss.2018.23082.

[3] NCC Group, DASP Top 10, Available: https://www.dasp.co/.

[4] Solidity documentation, Security Considerations, Available: https://docs.soliditylang.org/en/v0.8.15/security-considerations.html.