스마트 컨트랙트 개발 보안 권장사항에 대한 연구

김현준[1], 박소현[1], 이강효[1], 하태균[1], 김경백[2]
[1]한국인터넷진흥원
[2]전남대학교 인공지능융합학과 교수
hyunjun@kisa.or.kr, sohyeon@kisa.or.kr, kanghyo.lee@kisa.or.kr, niceha@kisa.or.kr,
kyungbaekkim@jnu.ac.kr

A Study on Smart Contract Development Security Recommendations

Hyunjun Kim[1], Sohyeon Park[1], Kanghyo Lee[1], Taegyun Ha[1], Kyungbaek Kim[2]
[1]Korea Internet & Security Agency
[2]Dept. Artificial Intelligence Convergence, Chonnam National Univ.

요 약

스마트 컨트랙트는 작성된 코드에 의해 정의되어 실행조건이 충족될 시 자동으로 실행되는 전자 계약으로 다양한 분야에서 활용될 수 있다. 스마트 컨트랙트가 블록체인 네트워크에 배포되면 수정이 불가한데, 개발자들이 스마트 컨트랙트를 작성하는데 코드 패턴 재상용 하는 경우가 대다수이다. 이로 인한 스마트 컨트랙트 취약점 사고를 예방하기 위해 개발 단계에서 보안을 고려하여 스마트 컨트랙트 작성이 필요하다. 본 논문에서는 대표적인 스마트 컨트랙트에 대한 보안 취약점을 확인하고 보안 권장 사항을 정리한다.

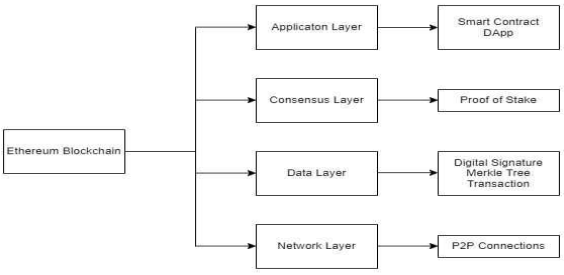
1. 서론

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 제삼자가 관여하지 않고 암호화된 거래를 참여자 간 공유하기 때문의 신뢰성과 투명성이 보장된다.

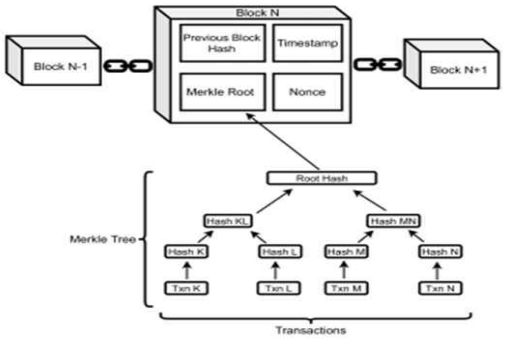
그러나 스마트 컨트랙트는 한 번 블록체인에 배포되면 수정할 수 없어 이를 악용한 악의적인 공격이 발생하고 있는데 대표적으로 2016년 발생한 더 다오(The DAO) 사건이 있다[1]. 이는 스마트 컨트랙트 코드상의 보안 취약점을 통한 공격이었으며 이에 따라 스마트 컨트랙트 작성 시 보안의 중요성이 강조되는 계기가 되었다. 본 논문에서는 최근 널리 사용되고 있는 스마트 컨트랙트 취약점을 예방하기 위한 보안 고려사항을 정리한다.

2. 스마트컨트랙트 개요

스마트 컨트랙트는 자체 실행할 수 있는 프로그램 코드로서 블록체인에 배포되어 신뢰할 수 없는 두 당사자 간 계약을 자동으로 이행하는 데 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 작성하는 데 사용한다.



(그림 1) 이더리움 블록체인 계층 구조



(그림 2) 이더리움 블록체인 노드 구조

그림 1과 그림 2는 각각 이더리움 블록체인의 계층 구조와 이더리움 블록체인의 노드 구조를 나타낸다.

체인의 각 노드는 이전 노드의 해시를 사용하여 다른 노드에 연결되고 트랜잭션은 블록체인의 머클 트리 형태로 해시 되며, 이는 트랜잭션 데이터 세트의 일관성과 무결성을 검증하는데 사용된다.

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.

3. 스마트 컨트랙트 취약점 및 개발 보안

스마트 컨트랙트를 작성하는 데 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[2].

ZEUS라는 논문에 따르면[3], 약 2만 2,400개의 스마트 컨트랙트 코드 중 약 95%에 달하는 스마트 컨트랙트가 하나 이상의 취약점을 가지고 있다. DASP TOP10과 SWC Registry에 명시된 대표 스마트 컨트랙트 취약점[4]-[5] 목록은 표 1과 같다.

<표 1> 대표 스마트 컨트랙트 취약점 목록

Re-entrancy
Access Control
Integer Over/Under Flow
Unchecked Low Level Calls
Denial of Service
Bad Randomness
Front Running
Time Manipulation
Delegate Call
Unexpected Ether

스마트 컨트랙트 배포 이전에 코드 검토를 통한 보안 취약점 존재 여부를 확인해야 한다. 이에 일반적으로 사용되는 함수를 작성하더라도 개발 보안 단계에서 검증된 스마트 컨트렉트 코드를 사용하거나 보안을 고려하여 작성해야 한다.

4. 스마트 컨트랙트 개발 보안 권장 사항

4-1. 외부 호출 시 주의

신뢰할 수 없는 계약을 호출하면 여러 가지 예상치 못한 위험이나 오류가 발생할 수 있다. 외부 호출은 해당 계약 또는 의존하는 다른 계약에서 악성 코드를 실행할 수 있다. 따라서 모든 외부 호출은 잠재적인 보안 위험으로 취급되어야 한다. 외부 호출을 제거하는 것이 불가능하거나 바람직하지 않으면 아래의 두 가지 권장 사항을 사용하여 위험을 최소화해야 한다.

외부 계약과 상호 작용할 때 변수, 메서드 및 계약 인터페이스와 상호 작용하는 것이 잠재적으로 안전하지 않다는 것을 그림 3과 같이 명확히 하는 방식(예:주석)으로 이름을 지정한다. 이것은 외부 계약을 호출하는 당신 자신의 기능에 적용된다.

```
// 나쁜 예
Bank.withdraw(100); // 신뢰 가능 여부가 불명확함
function makeWithdrawal(uint amount) {
// 이 함수가 잠재적으로 안전하지 않은지 확실하지 않음
    Bank.withdraw(amount);
}

// 좋은 예
UntrustedBank.withdraw(100); // 신뢰할 수 없는 외부 호출
TrustedBank.withdraw(100);
// XYZ Corp에 의해 유지되는 외부적이지만 신뢰할 수 있는 bank 컨트랙트

function makeUntrustedWithdrawal(uint amount) {
    UntrustedBank.withdraw(amount);
}
```

(그림 3) 신뢰할 수 없는 계약 표시 예시.

4-2. transfer() 또는 send() 함수 사용 주의

.transfer와 .sends는 정확히 2,300개의 가스를 수신자에게 전달한다. 이 하드코드 가스 제공의 목적은 재진입 취약성을 방지하는 것이었지만, 이는 가스 비용이 일정하다는 가정하에서만 타당하다. .transfer() 및 .send() 사용을 중지하고 대신 .call()을 사용하는 것이 좋다. 그림 4는 좋은 예와 나쁜 예의 코드이다.

```
// 나쁜 예
contract Vulnerable {
    function withdraw(uint256 amount) external {
        msg.sender.transfer(amount);
    }
}

// 좋은 예
contract Fixed {
    function withdraw(uint256 amount) external {
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success, "Transfer failed.");
    }
}
```

(그림 4) call() 사용 예시.

다만, .call()은 재진입 공격을 완화하는 데 아무런 도움이 되지 않으므로 다른 예방 조치를 취해야 한다. 재진입 공격을 방지하려면 check-effect-interaction 패턴을 사용하는 것이 좋다.

4-3. 대체 함수(Fallback Function) 단순 유지

대체 함수는 계약이 인수 없이 전송될 때(또는 일치하는 함수가 없을 때) 호출되며, .send() 또는 .transfer()에서 호출될 때만 2,300개의 가스에 접근할 수 있다. .send() 또는 .transfer()로부터 Ether를 수신할 수 있는 경우, 대체 함수에서 할 수 있는 가장 큰 일은 이벤트 로그이다. 가스 연산이 더 필요할 경우 적절한 함수를 사용해야 한다. 그림 5는 대체 함수 사용의 예시이다.

```
// 나쁜 예
function() payable { balances[msg.sender] += msg.value; }

// 좋은 예
function deposit() payable external {
balances[msg.sender] +=
\ msg.value; }

function() payable { require(msg.data.length == 0); emit
\ LogDepositReceived(msg.sender); }
```

(그림 5) 대체 함수 사용 예시.

4-3. 특정 컴파일러 버전으로 pragma 잠그기

계약(Contract)는 가장 많이 테스트 된 것과 같은 컴파일러 버전 및 플래그로 배포되어야 한다. pragma를 잠그면 발견되지 않은 버그의 위험이 더 클 수 있는 최신 컴파일러를 사용하여 계약이 실수로 배포되지 않도록 하는 데 도움을 준다. 계약은 타인이 배포할 수 있으며 pragma는 원래 작성자가 의도한 컴파일러 버전을 나타낸다. 그림 6은 pragma를 잠그는 것을 보여주는 예시이다.

```
// 나쁜 예
pragma solidity ^0.4.4;

// 좋은 예
pragma solidity 0.4.4
```

(그림 6) pragma 잠금 예시.

5. 결론

본 논문에서는 스마트 컨트랙트 취약점을 소개하고 작성 개발 권장 사항을 정리하였다. 권장 사항에 맞게 스마트 컨트랙트를 작성하여 검증하는 테스트넷 환경을 만드는 것이 필요하다. 또한 스마트 컨트랙트 취약점 진단 도구들을 활용한 비교 탐지를 통해 보안 위협 사전 예방이 필요하다.

최근 블록체인 기반의 DID, NFT 등 기술들을 활용한 서비스들이 상용화되고 있다. 이러한 블록체인 킬러 서비스에는 스마트 컨트랙트가 필수적으로 작성되기 때문에, 스마트 컨트랙트 취약점 분석에 따른 최신화된 진단 도구와 가이드라인을 지속적으로 연구할 필요가 있다.

참고문헌

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

[2] H. Kim, S. Oh and G. Kim et al., “Study on the Analysis and Response of Ethereum Smart Contract Security Vulnerabilities,” in Proc. KICS Winter Conf. 2023., YongPyung Resort, 2023, pp. 585-586.

[3] 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, 2018, pp. 1-12.

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

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