스마트 계약 버그 탐지 연구 동향

김태욱[1], 조영필[2]
[1]한양대학교 산학협력단 연구원
[2]한양대학교 컴퓨터소프트웨어학과 조교수
qkenr7895@hanyang.ac.kr, ypcho@hanyang.ac.kr

A Survey of Smart Contract Bug Detection

Tae-Wook Kim[1], Yeongpil Cho[2]
[1]Industry-University Cooperation Foundation, Hanyang University
[2]Dept. of Computer and Software, Hanyang University

요 약

스마트 계약은 다양한 형태의 계약을 체결하고 이행할 수 있게 해주는 오늘날 가장 인기있는 기술 중 하나이다. 그러나 적절히 구현되지 않은 스마트 계약은 공격 대상이 될 수 있으므로 버그 탐지 기법의 개발과 적용이 중요하다. 본 논문에서는 스마트 계약에서 발생하는 주요 버그 유형과, 이를 탐지하기 위한 최신 연구 동향을 조사한다. 이를 통해 스마트 계약의 안전한 구현과 블록체인 네트워크의 지속 가능한 발전에 기여하고자 한다.

1. 서론

스마트 계약 (Smart Contract)은 블록체인 기술을 기반으로 자동화된 계약 중개를 가능하게 하는 프로그램이다. 블록체인이 제공하는 높은 신뢰성과 안전성 덕분에 스마트 계약은 단순한 기술적 가치뿐만 아니라 경제적 가치도 지니고 있다

그러나 스마트 계약의 낯선 프로그래밍 환경과 언어, 네트워크에 등록된 후 수정이 불가능한 특징은 스마트 계약을 공격에 취약하게 만들고 있다. 실제로 2016년에 발생한 DAO 공격[1]은 6천만 달러의 손실을 초래했다

따라서 본 연구에서는 스마트 계약에서 빈번하게 발생하는 주요 버그 유형과 최신 버그 탐지 연구를 조사하여, 스마트 계약의 보안 강화에 도움이 되고자 한다.

2. 스마트 계약

스마트 계약의 사용은 이더리움 플랫폼 전용 언어인 Solidity로 코드를 작성함으로써 시작된다. 작성된 코드는 컴파일 과정을 거쳐 EVM (Ethereum Virtual Machine) 바이트코드로 변환된다. 개발자는 이렇게 변환된 바이트코드를 일정한 비용 (가스)을 지불하여 이더리움 블록체인 네트워크에 등록할 수 있다. 참고로 한 번 등록된 스마트 계약은 변경이 불가능하기 때문에, 코드에 버그가 있어도 수정이 어렵다.

네트워크에 등록된 스마트 계약은 해당 주소를 알고 있는 누구나 가스 비용을 지불하여 실행할 수 있다. EVM 바이트코드마다 비용이 정해져있고, 실행한 바이트코드 비용의 총합이 실행 비용이 된다.

3. 스마트 계약 버그

Reentrancy 재진입 취약점은 함수가 외부 계약을 호출할 때, 그 외부 호출이 다시 원래 함수로 재진입하여 예상치 못한 동작을 유발하는 취약점이다. 가장 유명한 DAO 공격은 재진입 버그를 활용하여 자금 인출을 반복적으로 수행했다.

재진입 취약점의 예시를 그림 1에서 확인할 수 있다. 가장 먼저, 공격자가 attack()을 실행해 withdraw(1)를 호출한다 (오른쪽 2번재 라인). withdraw 함수는 잔액을 확인하고 (왼쪽 2번째 라인) 잔액이 충분하면 인출 금액을 전송한다 (왼쪽 3번째 라인). 전송한 인출 금액을 공격자도 처리할 수 있어야 하므로 외부 계약인 공격자의 fallback 함수를 호출하는데, 이 때 fallback 함수는 다시 withdraw(1)를 호출한다 (오른쪽 5번째 라인). 재진입된 withdraw 함수는 잔액을 다시 확인하는데 (왼쪽 2번째 라인), 이전의 호출에서 인출 금액을 차감 (왼쪽 4번째 라인) 하지 않았기 때문에 잔액이 충분

하다고 착각하여 공격자에게 인출 금액을 한번 더 전송한다.

```
1) function withdraw(uint256 amount) public {
2)     require(balances[msg.sender] >= amount);
3)     msg.sender.call(amount)
4)     balances[msg.sender] -= amount;
5) }
```

```
1) function attack() {
2)     withdraw(1);
3) }
4) fallback() {
5)     withdraw(1);
6) }
```

그림 1 재진입 취약점 예시 - 취약한 코드 (좌), 공격자 코드 (우)

Timestamp Dependency 타임스탬프 의존성 취약점은 랜덤함수 등에서 블록의 타임스탬프를 신뢰함으로써 발생하는 취약점이다. 블록체인 네트워크에서 블록의 타임스탬프 값은 제한적이지만 어느정도 자유롭게 설정할 수 있다. 따라서 타임스탬프 값에 기반하여 스마트 계약의 특정 조건이 실행될 경우, 공격자가 타임스탬프 값을 조작하여 예상치 못한 동작을 수행할 수 있다.

4. 스마트 계약 버그 탐지

Symbolic Execution 기호 실행은 프로그램의 입력값을 구체적인 값이 아닌 기호로 대체하여 가능한 모든 실행 경로를 탐색하는 기법이다.

Oyente[2]는 가장 먼저 기호 실행을 사용하여 스마트 계약 버그 탐지를 시도한 연구이다. Oyente는 바이트코드와 스마트 계약의 상태 (state)를 입력으로 받아 문제가 되는 경로를 찾는다. 대표적으로 타임스탬프 의존성 취약점의 경우, 스마트 계약이 타임스탬프를 사용하는 경로를 추적하여 타임스탬프 조작에 취약한 경로를 탐지한다. 이밖에도 TOD (Transaction-ordering Dependent)와 예외처리 오류, 재진입 취약점을 탐지할 수 있다.

Manticore[3]는 전통적인 바이너리 환경과 이더리움 스마트 계약을 모두 지원하는 기호 실행 프레임워크이다. 기존의 기호 실행 프레임워크는 실제 환경에서, 특히 이더리움 플랫폼에서, 확장성이 부족해 널리 쓰이지 못했다. Manticore는 사용자 친화적인 인터페이스를 제공하여 플랫폼에 구애받지 않으면서도 높은 커버리지를 달성할 수 있다.

Fuzz Testing 퍼즈 테스팅은 랜덤한 입력을 생성하여 프로그램을 테스트하는 기법이다.

ContractFuzzer[4]는 퍼징을 기반으로 스마트 계약 취약점 탐지를 시도한 첫 번째 연구이다. 구체적으로 스마트 계약의 인터페이스를 분석하여 테스트 입력값을 생성하고, 이러한 입력값을 바탕으로 계약을 실행하여 취약점을 찾아내는 연구이다. 계약은 로그 값을 출력하도록 수정된 EVM위에서 실행되기 때문에, 로그를 수집하고 분석하여 취약점을 탐지할 수 있다.

SMARTIAN[5]은 정적 분석과 동적 분석을 결합하여 효과적인 퍼징을 수행하는 연구이다. 스마트 계약의 상태는 트랜잭션에 의해 변경되는데, 기존의 퍼저들은 의미있는 트랜잭션 순서를 찾지 못했다. SMARTIAN은 정적 분석을 통해 의미 있는 트랜잭션의 순서를 찾고 데이터 흐름 기반의 동적 분석 피드백을 통해 새로운 트랜잭션 순서를 찾는다.

Obfuscation 난독화는 코드를 수정하여 이해나 해석, 실행에 사용할 수 없도록 만드는 기법이다. 원래는 스마트 계약 배포 전 모든 버그를 찾아내는 것이 이상적이다. 하지만 이는 현실적으로 어렵기 때문에 스마트 계약 코드를 난독화 하여 배포하는 것도 한 가지 방법이 될 수 있어 함께 소개한다.

BiAn[6]은 스마트 계약 난독화를 수행하는 연구이다. BiAn은 조건문과 반복문 등의 제어 흐름을 복잡하게 만드는 제어 흐름 난독화와 변수 이름을 변경하고 데이터 구조를 바꾸는 데이터 흐름 난독화, 주석 제거와 같이 코드의 가독성을 떨어뜨리는 레이아웃 난독화를 통해 스마트 계약 난독화를 수행한다. BiAn은 82%의 추가적인 실행 비용으로 기존의 탐지 도구들의 성능을 절반으로 떨어뜨렸다.

5. 결론

본 연구에서는 재진입 취약점과 타임스탬프 의존성 취약점 버그 유형과 이를 탐지 하기 위한 기호 실행, 퍼즈 테스팅 등의 최신 연구 동향을 소개하였다. 이를 통해 스마트 계약의 안전성을 높일 수 있을 것으로 기대한다.

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정부통신기획평가원의 지원을 받아 수행된 연구 결과 임 (No. RS-2024-00337414, SW공급망 운영환경에서 역공학 한계를 넘어서는 자동화된 마이크로 보안 패치 기술 개발)

참고문헌

[1] P. Zhang, F. Xiao, and X. Luo, “A framework and dataset for bugs in Ethereum smart

contracts," in Proc. 36th IEEE Int. Conf. Softw. Maintenance Evolution (ICSME), 2020, pp. 139 - 150.

[2] Luu, L., Chu, D.H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 254 - 269). Association for Computing Machinery.

[3] Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G., Feist, J., Brunson, T., & Dinaburg, A. (2019). Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1186-1189).

[4] Jiang, B., Liu, Y., & Chan, W. (2018). ContractFuzzer: fuzzing smart contracts for vulnerability detection. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 259 - 269). Association for Computing Machinery.

[5] Choi, J., Kim, D., Kim, S., Grieco, G., Groce, A., & Cha, S. (2021). SMARTIAN: Enhancing Smart Contract Fuzzing with Static and Dynamic Data-Flow Analyses. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 227-239).

[6] Zhang, P., Yu, Q., Xiao, Y., Dong, H., Luo, X., Wang, X., & Zhang, M. (2023). BiAn: Smart Contract Source Code Obfuscation. IEEE Transactions on Software Engineering, 49(9), 4456-4476.