스마트 컨트랙트 동적 분석 도구의 버그 오라클 구현에 따른 오류 탐지 양상에 대한 고찰*

최호진, 박정원, 최재승

서강대학교 컴퓨터공학과

kaler1234@u.sogang.ac.kr, jungwon000711@gmail.com, jschoi22@sogang.ac.kr

The Impact of Bug Oracle Implementation on the Effectiveness of Smart Contract Analysis Tools

Hojin Choi, Jungwon Park, Jaeseung Choi

Department of Computer Science and Engineering, Sogang University

kaler1234@u.sogang.ac.kr, jungwon000711@gmail.com, jschoi22@sogang.ac.kr

요 약

스마트 컨트랙트의 오류 탐지를 위한 동적 분석 기술을 구현할 때, 알맞은 오라클(bug oracle)을 구현하는 것은 중요하고 어려운 문제이다. 본 연구에서는 스마트 컨트랙트의 이더 유출(leakage of ether) 오류를 탐지하기 위해 여러 동적 분석 도구가 어떤 오라클을 사용하는지 비교 분석한다. 이를 바탕으로, 각 도구가 오탐 또는 미탐을 일으키게 되는 실제 컨트랙트 예시를 제시하고 그 의의를 논의한다.

1. 서 론

프로그램 동적 분석(dynamic program analysis)은 주어진 프로그램을 구체적인 하나의 경로를 따라 실행하며 관찰하는 방법이다. 이 과정에서 프로그램에서 발생하는 오류(bug)를 탐지하는 것도 가능하며, 최근 널리 사용되는 퍼즈 테스팅(fuzz testing, a.k.a. fuzzing)[1], 동적 기호 실행(dynamic symbolic execution)[2] 등이 동적 분석을 사용하여 버그를 탐지하는 대표적인 기술이다.

동적 분석으로 오류를 탐지하기 위해서는, 어떤 실행이 관찰되었을 때 그 실행에서 오류가 일어나는지 여부를 판단하는 오라클(bug oracle)을 정의하는 것이 중요하다. C/C++로 작성된 전통적인 소프트웨어의 경우, 간단하게는 프로세스가 크래시를 일으키며 비정상 종료되는 경우 오류가 발생한 것으로 판단할 수 있다. 한편으로는 프로그램 계측(instrumentation) 기법을 활용하여 버퍼 오버플로우와 같은 오류가 발생했는지 여부를 보다 정확하게 판별하는 것도 가능하다[3].

반면 비교적 최근에 등장한 스마트 컨트랙트 프로그램의 경우, 오류 발생을 판단하는 기준이 모호하기 때문에 오라클을 구현하는 것이 어려운 문제가 된다. 스마트 컨트랙트의 경우 근본적인 설계 상 크래시와 같은 비정상 종료 현상이 발생하지 않으며, 컨트랙트의 어떤 행동이 문제를 일으키거나 사용자에게 손해를 입힐 수 있는지를 기준으로 삼아 오류의 발생 여부를 판단해야 한다. 이러한 기준을 오류 탐지 도구가 자동으로 판단할 수 있도록 구현하는 것은 어려운 일이며, 때에 따라 어느 범위까지를 오류로 판단해야 하는지 불명확하기도 하다.

본 연구에서는, 현존하는 스마트 컨트랙트 동적 분석 도구들을 조사하여 오라클이 어떻게 구현되어 있는지 비교 분석한다. 구체적으로, 이더 유출(ether leak)이라 불리우는 오류에 대해, 최신 도구인 SmarTest[4], RLF[5], Smartian[6]이 오라클을 어떻게 구현하고 있는지 조사하고, 각 도구가 어떤 경우에 오탐(false positive) 또는 미탐(false negative)을 일으키는지 실제 예시를 통해 확인한다. 이를 통해, 스마트 컨트랙트 오류 탐지에 있어 오라클의 구현이 성능에 크게 영향을 미칠 수 있는 요소임을 보인다.

2. 배경 및 관련 연구

2.1 스마트 컨트랙트의 실행 및 오류 발생 양상

스마트 컨트랙트는 바이트코드 형태로 컴파일 된 이후 이더리움 가상 머신(Ethereum Virtual Machine)이라 불리우는 실행기 위에서 실행된다. 이 때, 바이트코드를 최초에

* 본 연구는 2024년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업 지원을 받아 수행되었음 (2024-0-00043)

블록체인에 올려서 실행을 시작하는 사용자가 컨트랙트의 주인(owner)이 되며, 컨트랙트의 코드에는 이 사용자(주인)에게 높은 권한을 부여함으로써 컨트랙트를 관리할 수 있도록 하는 로직이 구현되어 있다. 예를 들어, 컨트랙트의 서비스를 중단하고 지금까지 컨트랙트에 누적된 이더(ether) 화폐를 특정 주소로 송금하는 기능은 일반적으로 컨트랙트의 주인만 실행 가능하다. 또다른 예로는, 현재 주인이 다른 임의의 유저에게 소유권을 이전(ownership transfer)할 수 있도록 하는 기능이 구현되어 있는 컨트랙트도 있다.

스마트 컨트랙트가 실행되는 도중, 컨트랙트의 주인이 의도하지 않았음에도 임의의 사용자가 이더를 탈취할 수 있게 되는 상황을 이더 유출(leakage of ether) 오류라 부른다[4, 6]. 아래의 간략화 된 예시 코드를 보면, 임의의 사용자가 **A()** 함수를 호출하여 컨트랙트의 주인을 자기 자신으로 변경할 수 있다. 따라서, 연이어 **sendEther()** 함수를 호출함으로써 해당 컨트랙트가 가지고 있던 이더 금액을 원하는 만큼 자신의 주소로 보내는 것이 가능하다. 이러한 실수는 개발자가 다른 컨트랙트의 코드를 복사하는 과정에서 특히 빈번하게 발생한다.

```
address owner; // State variable to record owner.
// Accidently, any user is allowed to call this.
function A( ) public {
  owner = msg.sender;
}
// Only 'owner' user can call this function.
function sendEther(address, amount) public {
  if (msg.sender == owner)
  { address.transfer(amount);  }
}
```

그 외에도 스마트 컨트랙트에는 정수 오버플로우 등 다양한 종류의 오류가 발생할 수 있지만, 본 논문에서는 이더 유출 오류에 집중하여 논의를 진행한다. 또한 논문에 따라서는 이더 유출 오류 중 특정한 유형을 "suicidal contract"라는 별도의 카테고리로 분류하기도 하지만[7], 본 논문에서는 이를 특별히 구분하지 않기로 한다.

2.2 스마트 컨트랙트의 동적 분석 도구

본 연구에서는 최근 우수한 성능을 보이는 것으로 발표된 세 개의 동적 분석 도구에 대해 조사를 시행한다. SmarTest[4]는 미리 학습한 모델을 바탕으로, 어떤 함수 호출 시퀀스를 생성해야 컨트랙트의 취약점을 효과적으로 발생시킬 수 있을지를 결정하는 기호 실행 도구이다. RLF[5]는 퍼즈 테스팅 과정을 Markov Decision Process 로 모델링하여, 함수 호출 시퀀스를 효과적으로 생성하기 위한 강화 학습 기술을 제안한다. Smartian[6]은 데이터 흐름에 대한 정적 분석을 통해 오류를 발생시킬 가능성이 높다고 판단되는 테스트 케이스들을 우선적으로 생성하는 퍼즈 테스팅 도구이다. 이들 각 논문은 연구의 핵심 기술을 충실하게 설명하고 있지만, 탐지하고자 하는 오류에 대한 오라클을 구체적으로 어떻게 구현하는지에 대한 설명은 많은 부분이 생략되어 있다.

3. 동적 분석 도구의 오라클 구현 조사

본 연구에서는 SmarTest의 커밋 `36d191e`, RLF의 커밋 `34b71b3`, Smartian의 커밋 `badd4ff`를 기준으로 각 구현체의 소스 코드를 분석하고 실험하였다.

3.1 각 도구의 이더 유출 오라클 구현

SmarTest는 다음의 두 가지 조건 중 하나가 만족될 경우, 이더 유출 오류가 발생한 것으로 판단한다. 첫째로, 컨트랙트의 주인이 아닌 일반 사용자가 컨트랙트의 어떤 함수를 호출함으로써, 컨트랙트에 지금까지 보낸 양보다 많은 이더를 받아올 수 있을 경우 컨트랙트에서 이더 유출이 발생한 것으로 판단한다. 둘째로, 주인이 아닌 사용자가 호출한 함수 내부에서 `selfdestruct` 명령이 실행되어, 해당 사용자에게 이더가 전송될 경우 이더 유출이 발생한 것으로 판단한다. 이는 `selfdestruct` 명령이 컨트랙트의 실행을 중단시킴과 동시에, 지정된 사용자에게 컨트랙트의 남은 이더 금액을 모두 전송하는 명령이기 때문에 가능하다.

다음으로, RLF는 SmarTest와 거의 동일한 오라클을 구현하되, 컨트랙트의 주인이 직접 `selfdestruct` 명령을 실행해서 컨트랙트 서비스를 중단시키는 경우도 이더 유출 오류가 발생한 것으로 판단한다. 이것이 의도된 구현인지 아니면 단순한 실수인지는 불명확하지만, 본 연구에서는 이러한 차이점으로 인해 RLF 도구가 오탐으로 판단되는 오류를 보고하는 실제 케이스를 발견하였다.

마지막으로, Smartian은 SmarTest와 유사한 오라클을 구현하되, 임의의 사용자가 이더 유출을 일으키는 함수 호출을 시행하기 전에, 한 번이라도 컨트랙트의 주인이 어떤 함수를 호출한 적이 있었다면 오류를 보고하지 않는다. 이는 만약 컨트랙트가 다른 사용자를 주인으로 임명하는 기능을 구현하고 있을 경우, 컨트랙트의 의도된 로직에 따라 정상적으로 행동하는 것을 이더 유출로 오탐할 가능성이 있기 때문이다. 다시 말해, Smartian은 잠재적인 오탐을 줄이기 위해, 다른 도구들에 비해 엄격한 조건을 체크하여 오류 발생을 판단한다. 이로 인해, Smartian은 다른 도구에서 발생하는 오탐을 줄일 수도 있지만, 동시에 다른 도구가 탐지하는 오류를 보고하지 못하는 미탐이 발생할 수도 있다.

3.2 오라클 간 비교 및 케이스 스터디

본 절에서는 각 오류 탐지 도구마다 상이하게 구현된 오라클 로직에 때문에 서로 다른 탐지 결과가 나타나는 실제 컨트랙트 예시들을 소개한다.

먼저, 아래의 첫번째 예시 코드는 컨트랙트의 주인에 해당하는 사용자만 컨트랙트 실행을 멈추고 남은 이더 금액을 회수할 수 있도록 작성된 컨트랙트의 일부이다. 여기서 **owner** 변수는 컨트랙트를 블록체인에 올리는 사용자로 초기화되며, 실행이 종료될 때까지 변화 없이 유지된다고 가정한다. 컨트랙트의 주인이 서비스를 중지하고 컨트랙트를 종료하는 것은 개발자가 의도한 정상적인 작동임에도 불구하고, RLF는 3.1절에서 설명한 오라클의 로직 때문에 아래 컨트랙트에 대해 이더 유출 오류를 보고하는 오탐이 발생한다.

```
address owner; // State variable to record owner.
// Only 'owner' user can call this function.
function stopService( ) {
  if (msg.sender == owner)
  { selfdestruct(msg.sender);  }
}
```

다음으로, 아래의 두 번째 예제는 컨트랙트의 현재 주인이 다른 사용자를 주인으로 임명하는 기능을 갖고 있는 프로그램이다. 이 경우 컨트랙트의 원래 주인이 **changeOwner()**를 호출하여 다른 사용자를 새로운 주인으로 만들고, 새로운 주인이 **stopSerivce()**를 호출해서 컨트랙트의 이더를 가져가는 것은 개발자가 의도한 정상적인 행동이 된다. 그러나 SmarTest나 RLF는 이러한 컨트랙트 소유권의 이전(ownership transfer)을 감지하지 못하고, 제일 처음 컨트랙트를 블록체인에 올린 사용자를 항상 주인으로 판단하기 때문에, 아래의 컨트랙트에 대해 이더 유출 오류를 보고하는 오탐이 발생한다.

```
address owner; // State variable to record owner.
// Current 'owner' transfer the ownership.
function changeOwner(newOwner) {
    if (msg.sender == owner)
    { owner = newOwner; }
}
// Only 'owner' user can call this function.
function stopService( ) {
  if (msg.sender == owner)
  { selfdestruct(msg.sender);  }
}
```

반면, Smartian의 경우 컨트랙트의 원래 주인이 한번이라도 함수를 호출하는 순간, 이더 유출이 아니라고 판단하기 때문에 위 예제에서는 오탐을 일으키지 않는다. 그러나 이와 같은 오라클의 휴리스틱 때문에, 다른 컨트랙트에서는 Smartian이 미탐을 빈번하게 일으키기도 한다. 예를 들면, 원래의 주인이 호출하는 함수가 소유권 이전과 관계없는 함수인 경우, 이더 유출이 일어남에도 불구하고 오류를 보고하지 못하게 된다.

4. 결론 및 향후 연구

본 논문에서는 스마트 컨트랙트 동적 분석 도구에서 버그 오라클의 구현 차이가 각 도구의 이더 유출 오류 탐지 성능에 미치는 영향을 분석하였다. SmarTest와 RLF는 상대적으로 유연한 오라클을 적용하였기 때문에 오탐이 발생할 가능성이 높았다. 반면, Smartian은 보다 엄격한 오라클을 사용하여 오탐은 줄었으나, 이로 인해 다른 도구에서 발생하지 않은 미탐이 발생하였다. 이러한 결과는 오라클의 구현 방식이 오류 탐지 성능에 큰 영향을 끼칠 수 있음을 보여준다.

추후 연구에서는 이러한 오라클 구현 방식의 차이가 대규모 벤치마크에서 어떠한 성능 차이를 나타내는지 실험할 계획이다. 대표적으로, RLF 논문에 보고된 실험 결과를 재현해 보고, 오라클 구현의 차이가 실험 결과에 얼마나 큰 영향을 미치고 있는지 검증해 볼 계획이다. 나아가, 각 도구가 구현하는 오라클 로직의 장단점을 결합하여, 보다 안전하면서도 정확한(sound and precise) 오라클을 설계하는 것도 추후의 과제이다.

참 고 문 헌

[1] Valentin J.M. Manès et al., "The art, science, and engineering of fuzzing: A survey." IEEE Transactions on Software Engineering, vol. 47, issue 11, 2019

[2] Sooyoung Cha et al., "Enhancing dynamic symbolic execution by automatically learning search heuristics," in IEEE Transactions on Software Engineering, vol. 48, issue 9, 2021

[3] Konstantin Serebryany et al., "AddressSanitizer: A fast address sanity checker," in Proceedings of USENIX Annual Technical Conference (USENIX ATC), 2012

[4] Sunbeom So, Seongjoon Hong, and Hakjoo Oh, "SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution," in Proceedings of USENIX Security Symposium (USENIX Security), 2021

[5] Jianzhong Su et al., "Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing," in Proceedings of IEEE/ACM International Conference on Automated Software Engineering (ASE), 2022

[6] Jaeseung Choi et al., "Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses," in Proceedings of IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021

[7] Ivica Nikolić et al., "Finding the greedy, prodigal, and suicidal contracts at scale," in Proceedings of the Annual Computer Security Applications Conference. (ACSAC), 2018