Clairvoyance: Cross-contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts

Jiaming Ye
sa517462@mail.ustc.edu.cn
University of Science and Technology of China

Mingliang Ma
sa517245@mail.ustc.edu.cn
University of Science and Technology of China

Yun Lin
llmhyy@gmail.com
National University of Singapore

Yulei Sui
yulei.sui@uts.edu.au
University of Technology Sydney

Yinxing Xue
yxxue@ustc.edu.cn
University of Science and Technology of China

ABSTRACT

Reentrancy bugs in smart contracts caused a devastating financial loss in 2016, considered as one of the most severe vulnerabilities in smart contracts. Most of the existing general-purpose security tools for smart contracts have claimed to be able to detect reentrancy bugs. In this paper, we present Clairvoyance, a cross-function and cross-contract static analysis by identifying infeasible paths to detect reentrancy vulnerabilities in smart contracts. To reduce FPs, we have summarized five major path protective techniques (PPTs) to support fast yet precise path feasibility checking. We have implemented our approach and compared Clairvoyance with three state-of-the-art tools on 17770 real-worlds contracts. The results show that Clairvoyance yields the best detection accuracy among all the tools.

KEYWORDS

reentrancy detection, path feasibility analysis, cross contract analysis, smart contract security

ACM Reference Format:
Jiaming Ye, Mingliang Ma, Yun Lin, Yulei Sui, and Yinxing Xue. 2020. Clairvoyance: Cross-contract Static Analysis for Detecting Practical Reentrancy Vulnerabilities in Smart Contracts. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377812.3390908

1 INTRODUCTION

Smart contract applications, built upon the block-chain platform Ethereum [1], belong to the most security-critical and data-sensitive application category. These contracts are mainly written by a turing-complete language Solidity. The majority of concerns on smart contracts security begins in 2016 since a devastating financial loss because of the DAO attack [3].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICSE '20 Companion, October 5–11, 2020, Seoul, Republic of Korea
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3390908

The principal reason of this attack is due to the fallback mechanism. Users often misconceive that smart contracts provide secure transactions and easy-to-use interfaces, by assuming that the other side in the transaction is trustworthy. Unfortunately, this assumption does not always hold. We find that attackers can write a function call in fallback function and form an end-to-end call chain, and execute functions in the chain without paying to the seller. We further find that, this attack often utilizes cross-contract calls, which are not considered by the state-of-arts. To understand the functionality of state-of-the-art tools (e.g., Oyente [4], Securify [6] and Slither [5]), we revisit the state-of-art available tools and find that they have limited capability in identifying the cross-contract reentrancy vulnerabilities. Besides, they usually focus on generic categories of vulnerabilities, not specific to the reentrancy. Even when this fatal vulnerability has been reported for years, it still appears unknown why the weakness in these tools is not patched. Overall, the existing work cannot represent the leading comprehension of this vulnerability, and the state-of-arts are ineffective in identifying cross-contract reentrancy bug.

To study the essential property and various evolutions of reentrancy and help to ensure the smart contract security, in this paper, we propose a more-sound cross-contract and cross-function static analyzer. We further propose a light-weighted symbolic analysis to help to precisely detect bugs. In summary, this paper makes the following contributions:

- We present a large-scale empirical study to evaluate the effectiveness of three recent general-purpose static tools using 11714 real-world contracts from Etherscan[2].
- We present a new static reentrancy detection approach to enable (1) more sound analysis by modeling cross-function cross-contract behaviors, and (2) more precise analysis by applying a light-weight symbolic analysis based on PPTs.
- Our tool, named Clairvoyance has *significantly better* accuracy than all the other tools.

2 APPROACH

To identify cross-contract reentrancy effectively, we construct the cross-contract call graph and CFG (named XCFG) among the input smart contracts. Next, we identify the suspicious objects or addresses, and track how they are used and propagated along the call chain from XCFG. In this step, a path matches our criterions will be added to the candidate pool. Meanwhile, in order to mitigate

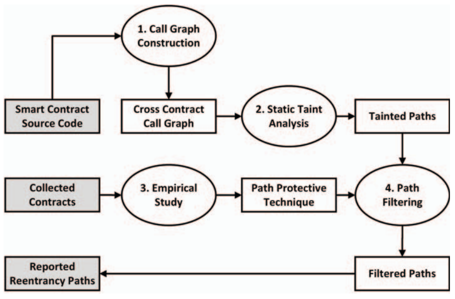
1. Call Graph Construction
Smart Contract Source Code
Cross Contract Call Graph
2. Static Taint Analysis
Tainted Paths
Collected Contracts
3. Empirical Study
Path Protective Technique
4. Path Filtering
Reported Reentrancy Paths
Filtered Paths

Figure 1: System diagram

the FPs of reported paths, we summarize 5 PPTs for reentrancy attack by reviewing the state-of-arts detection results on 11714 contracts. For these PPTs, we define the corresponding filtering patterns via program analysis. Finally, we filter out infeasible ways and the remaining reachable paths are produced as results.

Fig. 1 shows an overview of our approach. In Fig. 1, each rectangle represents an artifact, each ellipse represents a step (or process), each edge represents the workflow of how a step takes some artifacts and outputs some new artifact. As illustrated in the diagram, our system takes Solidity code as input, then outputs reentrancy paths after path collecting and path filtering steps.

3 EMPIRICAL STUDY

In our empirical study, Slither [5], Oyente [4], Securify [6] run on 11714 frequently-used real-world contracts from the well-known third-party website Etherscan [2] for contract indexing and browser. For the detection result, we recruit 4 researchers to spend 2 months in reviewing the results and summarizing the patterns of FPs for rules in these tools.

In our findings, these four tools are not sufficiently effective in handling the reentrancy bug. The reason is that most of their reports are false negatives. So as to avoid these false positives happening to our tool, we audit incorrect reports manually and summarize the following PPTs. The filtering pattern for PPT1 is to check whether `msg.sender` is within a list of authorized contracts or addresses, or has the permission to do this (e.g., `msg.sender==owner`) The filtering pattern for PPT2 is to check whether the tainted address or object has been initialized in declaration or modified before the external call. The filtering pattern for PPT3 is actually applying the above two patterns in self-defined modifiers of the function. The pattern for PPT4 is to check the existence of the execution lock. The pattern for PPT5 is to check the existence of checks-effects-interactions pattern.

To make the PPT-based filtering more accurate, a light-weight symbolic analysis is employed across PPT1-PPT4, assisting the reentrancy detection. Our light-weight symbolic analysis leverages the intra-procedural symbolic execution for synthesizing a symbolic path from tainted source to the fallback call. Then we feed the path into Z3 solver to check its feasibility. In this work, our approach is designed to favor soundness over completeness.

4 EVALUATION

To evaluate the effectiveness of Clairvoyance, we conduct extensive experiments. Specifically, we attempt to answer the research questions: How effective are the summarized PPTs? Compared with the three available static tools, how is the precision of Clairvoyance?

The experiments are performed on 17770 new contracts obtained from Google Big Query open dataset. Tools employed in this experiment are all in their latest version. Further, Clairvoyance is implemented in Python using 5000 lines of code, processing and accepting SlithIR.During the evaluation, all the experiments and tools are conducted on a machine running on Ubuntu 18.04.

For the numbers of detection results of the four static tools, Slither reports 162 vulnerabilities in total, of which 3 reports are true positives (TPs), while the other 159 reports are false positives (FPs). Oyente has least 28 reports, of which 4 results are TPs, while the rest 24 reports are FPs. Securify reports 3 TPs, but it has 605 FPs. Comparatively, Clairvoyance has outstanding precision in this experiment, for 124 true positives in total 168 reports.

Securify, Oyente, Slither reports more bugs, but most of their result are false positives. Comparatively, Clairvoyance has the highest true positives. The reason behind the FPs reported by three tools is due to the inconsideration of PPTs. In our observation:

1 Securify fails to consider permission controls, hard-coded addresses and self-defined modifiers. It also falsely reports the write operations after calling built-in functions `send()` and `transfer()` as vulnerable, causing FPs since it does not consider PPT5.
2 Oyente basically ignores the protections in self-defined modifiers and has many FPs mostly because it ignores PPT3.
3 Among all the FPs of Slither, it generally has a good support for PPT3 by considering the code of security check in modifiers. Considering a relatively small number of cases in using execution lock (PPT4), Slither totally ignores the protection by execution lock(s).
4 The 44 of FPs of Clairvoyance are due to the complicated path conditions, which cannot be easily handled by our light-weighted symbolic analyzer.

5 CONCLUSION

In this paper, we present a reentrancy detection approach based on two steps, first applying the cross-contract static taint analysis to find reentrancy candidates, then integrating the PPTs to refine the results. On the publicly collected 17770 contracts, Clairvoyance significantly outperforms the three static tools in terms of precision. In future, we will extend our approach for other types of bugs and also combine with dynamic approaches.

REFERENCES

[1] 2015. Ethereum: Blockchain App Platform. https://www.ethereum.org/. Online; accessed 29 January 2019.
[2] 2019. A Block Explorer and Analytics Platform for Ethereum. https://etherscan.io/. Online; accessed 29 January 2019.
[3] David Siegel. [n.d.]. Understanding the DAO Attack. Website. https://www.coindesk.com/understanding-dao-hack-journalists.
[4] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *CCS 2016*. 254–269.
[5] trailofbits. 2019. Slither. github. https://github.com/trailofbits/slither.
[6] Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin T. Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *CCS 2018*. 67–82.