Proposal: Ethereum Smart Contract Security

Kazi Alom, Lay Jain, Srijon Mukherjee

The Ethereum blockchain offers smart contracts as a way of handling financial promises. These are implemented as Turing-complete programs allowing a smart contract creator to execute arbitrary code in handling transactions. These smart contracts are available for public viewing and are susceptible to being exploited. We aim to do a security analysis on common smart contracts and explore what kinds of critical vulnerabilities may exist.

***Index Terms*—security analysis, ethereum, smart contract**

I. Foundations

Blockchain technology's greatest strength is allowing the decentralization of historically centralized services. Bitcoin [4] has a market cap of almost $200 billion and sees a daily trading volume of about $100 billion. Besides Bitcoin, there are countless other ”alt-coins” that each have their own platform and offer a type of service. Ethereum [1] is one platform that allows digital assets be directly controlled by a piece of code called a smart contract. Smart contracts essentially allow systems that automatically move digital assets according to pre-defined rules. Decentralized autonomous organizations (DAOs) are long-term smart contracts that are controlled by shareholders. The DAO [3], a crowdfunded venture capital fund that existed as a smart contract, had over $50 million stolen due to an exploited bug in the smart contract language Solidity. In fact, this incident is the reason why Ethereum was hard-forked and why we see another alt coin ”Ethereum Classic”. Other high profile smart contracts have also been targeted and lost their funds to these types of attacks. As the usage of smart contracts becomes more prevalent, it is critical to address bugs in smart contract implementations and in Solidity itself.

Smart contract vulnerabilities can be classified [6] as blockchain vulnerabilities, Solidity vulnerabilities, and software security vulnerabilities. The Transaction Ordering Dependency problem is one blockchain vulnerability which involves a new block on the chain containing multiple transactions invoking the same contract. There is no certainty in the state of the contract when either individual transaction invokes the contract because the order is not known until after the block is mined. The Timestamp dependency problem says that a smart contract using timestamps of blocks can potentially be manipulated by malicious miners who modify their local system's time.

Solidity compiler vulnerabilities are bugs within the smart contract's high level language and how it generates the Ethereum Virtual Machine (EVM) bytecode. There have not been many CVE's published on this which makes the EVM an excellent candidate for fuzzing.

Software security vulnerabilities are those due to erroneus smart contract code and improper Solidity practice. The DAO attack was victim to a re-entrancing attack where the contract uses an unsafe function call.value() that allows the attacker to take advantage of a callback to recursively withdraw funds from a victim contract. Other usual security vulnerabilities such as buffer overruns and integer over/underflows also fall under software bugs.

The parity multisig attack was one incident where $50 million of Ethereum was frozen due to an attacker calling an unprotected function and gaining ownership of a public smart contract library that was imported by other contracts. The attacker invoked the kill function that removed the contract from the blockchain. This could have been mitigated by using a private modifier on the function that allowed the attacker to escalate privileges.

Ethereum smart contracts rely on ”gas”, which is additional transaction fee paid to the miner for borrowing their computation ability on the blockchain. If this gas runs out before an operation is over, the callee contract throws an exception and these must be properly checked by the caller contract. This applys to all other exceptions that can be raised such as when exceeding stack capacity, or any other unknown system error occurs. These also fall under software security vulnerabilities that can be mitigated by safe code practice.

II. Approach

The current plan is to use Manticore [5], a symbolic execution engine that supports EVM bytecode (and other architectures) to maximize code coverage on smart contracts sampled from Etherscan [2]. Manticore can simulate the EVM as well as simulate an entire ”Ethereum world” where a contract may call on other contracts. One advantage is that this analysis can be done without any real Ethereum coin because Manticore can set arbitrary amounts for contracts and wallets you define in the API.

The objective is to identify devastating vulnerabilities we have seen before such as the re-entrancing attack on the DAO that allows an attacker to steal funds. Manticore will provide inputs that will hit various branches in smart contracts and the next step is to apply a fuzzer to user-controlled inputs in order to generate corner cases. In other words, we want to cause as many exceptions we can along every branch in the code and log them for further analysis. The collected smart contracts will also be analyzed for semantic bugs after reading up on the latest version of Solidity and recommended practice.

One difficulty in the approach may be dealing with all the versions of Solidity being used in the wild. Solidity is constantly being patched which causes smart contract writers to be slow in keeping up. Manticore fully supports Solidity versions pre-0.5.0 so we will have to scrape contracts using Solidity 0.4 and older.

III. Conclusion

Ultimately, we hope to shed some light on the current unfortunate state of Ethereum smart contract security and potentially report bugs, making cryptocurrency a little safer. Next steps from here may be directly fuzzing the Solidity compiler with randomly generated smart contract code and also fuzzing the EVM runtime as previously mentioned.

References

[1] V. Buterin. Ethereum white paper: A next generation smart contract & decentralized application platform. 2008. “[Online; accessed 5-April-2020]”.

[2] Etherscan. Etherscan. https://etherscan.io/aboutus, 2015. [Online; accessed 6-April-2020].

[3] A. Madeira. The dao, the hack, the soft fork and the hard fork. 2019. “[Online; accessed 5-April-2020]”.

[4] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. “[Online; accessed 5-April-2020]”.

[5] T. of Bits. Manticore. https://github.com/trailofbits/manticore, 2019. [Online; accessed 6-April-2020].

[6] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss. Security analysis methods on ethereum smart contract vulnerabilities — a survey. 2020. “[Online; accessed 5-April-2020]”.