Have I Been Exploited?
A Registry of Vulnerable Ethereum Smart Contracts

Daniel Connelly and Wu-chang Feng

Portland State University
Department of Computer Science

Abstract. Ethereum Smart Contracts, also known as Decentralized Applications (DApps), are small programs which orchestrate financial transactions. Though beneficial in many cases, such contracts can and have been exploited, leading to a history of financial losses in the millions of dollars for those who have invested in them. It is critical that users be able to trust the contract code they place their money into. One way for verifying a program's integrity is Symbolic Execution. Unfortunately, while the information derived from symbolic execution is beneficial, performing it is often financially and technically infeasible for users to do. To address this problem, this paper describes the design and implementation of a registry of vulnerable Ethereum contracts. The registry compiles the results of exhaustive application of symbolic analysis to deployed contracts and makes it available to users seeking to understand the risks associated with contracts they are intending to utilize.

1 Introduction

Blockchain technologies have reached proportions of popularity similar to other relatively newer fields such as Artificial Intelligence and Machine Learning. LinkedIn, for example, lists blockchain as the most in-demand hard skill in 2020 [1]. Other corroborating data suggests that it would behoove developers, companies, institutions, and governments to adopt blockchain as the technology claims to be a panacea for many problems. However, using blockchain for financial use cases, specifically with Smart Contracts, can be a dangerous gamble. For example, there are many instances where financial losses have been incurred due to insecure programming. These mistakes are often more serious than traditional software system bugs due to the financial consequences (which are uninsured) and the often-permanent bugs on an immutable blockchain.

Fortunately, techniques such as Symbolic Execution, such as the tool *Mythril* [2] that is used in this project, can be used to identify insecure programming. Such techniques can yield comprehensive results for up to 50% of programs and find 78% of the most important bugs [3]. While users would benefit from being able to evaluate the security of smart contracts, without access to significant resources to perform these techniques, they remain out of reach. To address this issue, this paper describes a registry for aggregating and publishing the results of security analyses of smart contracts where users of a blockchain such

as Ethereum can search to see what vulnerabilities a contract they may be wishing to use suffers from before sending it any currency.

2 Symbolic Execution

Symbolic Execution works like algebra for computer programs. The job of any engine is to discover if a particularly vulnerable state may be reached. For example, a state in Ethereum that carries business logic of "send all ETH to the caller of this program" is likely unwanted, but easily discoverable with this method. Fortunately, engines like Mythril work on EVM bytecode to detect this and other disastrous programming logic.

Mythril performs its analysis by taking the bytecode of a contract and decompiling it into EVM opcode instructions where all possible program states are explored over n transactions/calls (two by default). Mythril uses a Symbolic Virtual Machine (SVM) named LASER [4] and a number of analysis modules to determine if any of 16 vulnerabilities, a subset of the Smart Contract Weakness Classification Registry [5], exist. For states where Mythril is not able to traverse due to resource or logic constraints, this shortfall is reflected by the percentage of coverage it returns. In order to prove the vulnerabilities that it finds, it uses the Z3 automated theorem prover, also known as a constraint solver, released by Microsoft Research under the MIT license [6].

3 Methodology

A list of contract addresses was generated by syncing an Ethereum full node, specifically the implementation using the Go programming language, named a Go Ethereum (GETH) node [7]. Once synced, the node was queried for contract addresses. A cluster of machines were deployed using the Google Cloud Platform (GCP) for the purpose of symbolically executing on the addresses obtained from querying the node. Each machine ran a Python program which spawned 10-12 threads. Each thread launched a new bash shell where a Docker container ran an image of Mythril on a single contract address. All output was stored local to the main thread, then written to files/bins depending on the content (exceptions, errors, or output). To alert us when a machine had stopped analysis, an open source messaging system for long running processes was developed through collaboration with another researcher [8]. Figure 1 depicts a high level design of our data collection process.

A hand-coded parser was written to run through the output files, which contained millions of lines of output, to extract contract address, type of vulnerability, severity of that vulnerability, and the coverage that was attained into comma-separated rows. A website was hosted on a Google Compute Engine at the address `www.haveibeenexploited.com`. The website was written in React and interacts with a Go server that accesses a local MySQL database which holds the obtained results. The website informs its audience of the results of vulnerable countracts as well as miscellaneous details about the project.

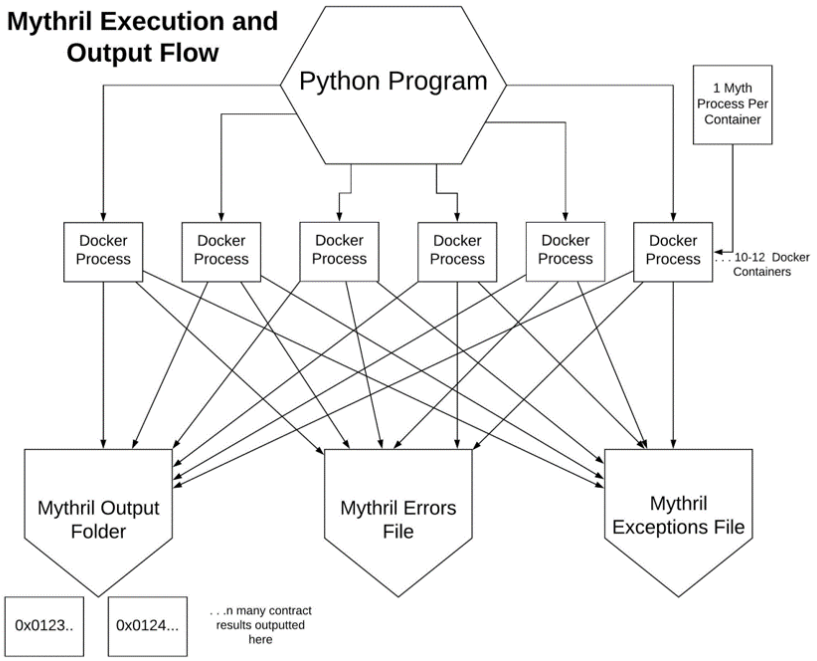

Fig. 1. High-Level Overview of Data Collection Program [9]

4 Results

After running the GETH node, Mythril was run over the addresses obtained through a portion of addresses ($\approx$ 1.6 million addresses) in the range 0-$\approx$ 8.4 millionth block.

4.1 Number of Contract Addresses

The total number of contract addresses created since the beginning of the Ethereum blockchain (up to the 9 millionth block) totaled $3,046,140$. The number of live contract addresses (i.e., destroyed contracts have zero length bytecode) on the Ethereum blockchain totaled $2,927,521$. Thus, approximately $119,000$ addresses have been destroyed over the lifetime of the blockchain. Figure 2 reveals the allocation of live contracts per millionth block range.

4.2 Number of Vulnerable Contracts

The number of contracts indicated by Mythril to contain vulnerabilities in this project, from the partially scanned 0-$\approx$8.4 millionth block, was $797,384$ contracts. This means that more than 27% of Ethereum smart contracts have one

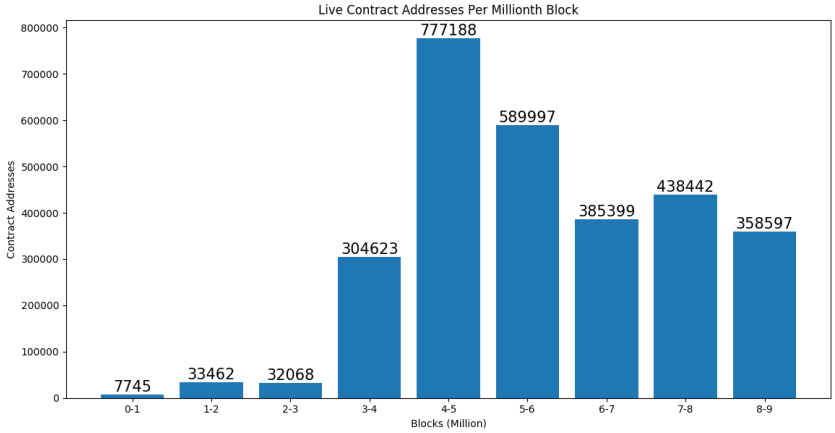

Fig. 2. Live Contract Addresses in 9 Million Blocks

or more vulnerabilities in them; more for a variety of reasons: we only analyzed a portion of contracts within 0-8.4 million blocks, bugs in Mythril, inability of achieving full code coverage in some cases, etc.

4.3 Number of Vulnerabilities

The number of vulnerabilities discovered during this project totaled $1,224,486$. This number does not reflect duplication, for example, when a contract may have repeated vulnerabilities of the same type within a contract. Thus, the number of ways a user may potentially exploit a vulnerable contract is one or more. Figure 3 shows the number of non-repeating vulnerabilities discovered during the analysis, by vulnerability type.

As the figure shows, the most common vulnerabilities are Exception State (ES; $646,957$) and State Change After External Call (SCAEC; $327,122$). An ES vulnerability has a severity rating of low according to Mythril. A SCAEC vulnerability has a severity rating of low or medium depending on if the external address is a user-supplied address (medium severity) or a hardcoded one by the developer (low severity). The results here indicate that the majority of vulnerabilities present in the contracts are not likely to cause much damage to users as the majority of vulnerabilities are of a mostly low severity rating.

However, even a few contracts with higher severity vulnerabilities could contain a large quantity of ETH and a single potential vulnerability could lead the way to an exploit that could cause damage in the millions of dollars across multiple users of a contract as prior exploits have shown.

Another informative way to examine the data is to analyze what blocks have the most vulnerabilities inside of them. As shown in Figure 4, among the results obtained, most of the contracts that contain insecure code were created within the 4-5 millionth block. Though these results are partial, vulnerabilities seem to have generally decreased. This could mean that there are factors at work that are

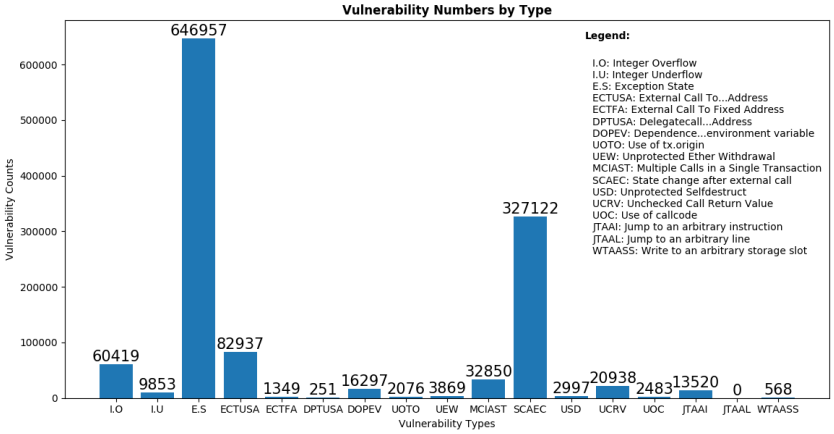

Fig. 3. Number of Vulnerabilities by Type

making Ethereum contracts more secure over time such as the use of non-Turing complete languages such as Vyper [10].

4.4 Coverage

The coverage that Mythril was able to obtain from each contract was also collected. As the symbolic execution was constrained to 1-hour, this was necessary for performance and cost reasons. Overall, the average code coverage achieved for this proof of concept was 76%.

5 Discussion

5.1 Risk analysis

Perez and Livshits have shown from pooling the results of 5 different academic research articles studying vulnerable smart contracts that "at most 504 out of 21,270 contracts [examined were]...subjected to exploits", representing "only 9,066 ETH (1.8 million USD)...0.29% of the 3 million ETH (600 million USD)" [11]. In the larger context of Ethereum's supply, this amount is small. However, users and the general public do not likely think of amounts in the millinos as small, though other parties may. In fact, if the results from Perez and Livshits are generalized to the larger subset discussed in this research encompassing several orders of magnitude more contracts, if the same percentage of vulnerable contracts were exploited (2%), then approximately 15,948 contracts would be exploitable. If 504 contracts housed 1 million USD then that means, given the same ratio of contracts likely to be exploited (2%), 31 million USD ($\approx$15948/504) may likely be at risk in this dataset. Assuming the same USD:ETH (the value of ETH has risen since Perez and Livshits' research so would in truth

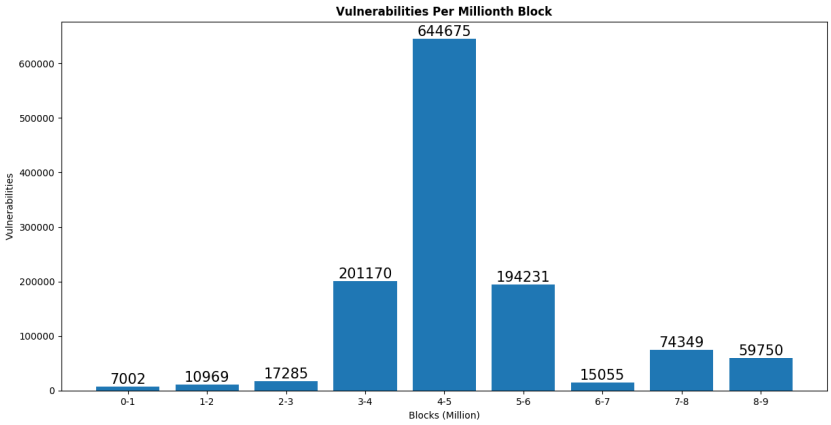

Fig. 4. Exploits Per Millionth Block

be a higher dollar amount) this would mean that, at minimum, 8% of the total Ethereum supply chain is at risk.

5.2 Moral Concerns

Users cannot be alerted directly of vulnerabilities in their contract code. Therefore, a chief ethical concern comes to mind: can this dataset allow malevolent users to exploit vulnerable contracts? Though malevolent parties may use this data as a steppingstone to take advantage of a vulnerability, this project is an attempt at being proactive with exposing and indirectly alerting users of vulnerabilities. Since contract owners and users cannot be alerted directly, at least they can be knowledgeable if they make use of this dataset and, with that knowledge, may attempt to pull money out and/or close a contract(s). For the newest contracts that contain vulnerabilities, if developers and users utilize this service, they will not fill a vulnerable contract full of money and setup a malevolent user for future success.

5.3 Veracity of Results

A small subset of addresses never achieved output or only partial output due to bugs or inefficiencies in Mythril, all of which were documented for reanalysis at a later date. It should be noted, therefore, that contracts deemed to have no vulnerability may very well indeed have one or more uncaught vulnerabilities. Additionally, unless technology improves in some dramatic way, there is no way to deem a contract truly safe and the results in this research conclusively true for all output data. A user should not assume a query to this registry which returns no result is an indication a contract is free of vulnerabilities. Finally, it should be noted that the results here are the opinion of one engine and it is always best practice to get a second opinion before deciding whether to use a contract. Multiple perspectives from engines is a future goal of this project.

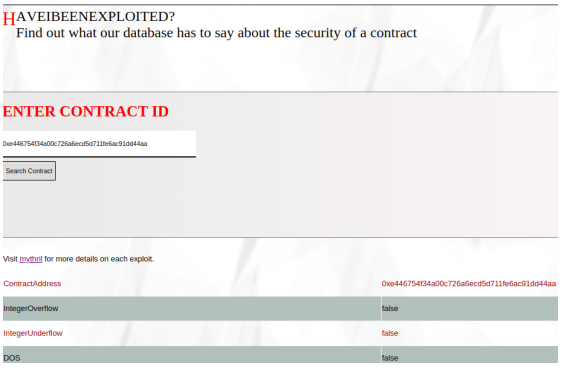

Fig. 5. www.haveibeenexploited.com

6 Registry

`www.haveibeenpwned.com` is a website that allows users to see if their information has been leaked in a security breach by entering any email address into a simple search bar. Our registry is a similar service, but for Ethereum Smart Contracts, and was named in honor of the inspiration taken. The website, located at the address `www.haveibeenexploited.com`, acts as a proof-of-concept registry where users can type in a contract's address into a search bar and determine if a contract is safe before use, as shown in Figure 5. It contains all the results gained from this research.

As it is now, a user may query a contract at the website for vulnerable contracts. This limits the ability for individuals to access the entirety of the database and see all contracts which have vulnerabilities. It was thought that this way honest users who wish to check on the safety of contracts they are concerned with may receive this information. On the other hand, those who wish to obtain a large listing of insecure contract addresses for exploitation purposes may not do so, short of brute-force querying of all of the contract addresses or hacking into the database holding the symbolic execution results.

7 Future Work

Future goals include finishing the Mythril execution and adding new engines to the project to gain multiple opinions on the safety of Smart Contracts [12]. There exist similar work, namely `www.contract-library.com`, to base off of, but this existing work is limited and mostly involves the use of a single engine. A rating system similar to Consumer Reports [13] and what VirusTotal [14] provides with malware analysis and detection is eagerly planned. Analysis, parsing, and updates to a database may also be automated to provide close to real-time feedback on contracts by a variety of tools. Finally, this type of research idea is not germane to a single blockchain or language and can be used wherever DApps exist.

8 Conclusion

Ethereum is an example of increasing attention to blockchain and decentralized applications seeking to replace and/or improve upon current financial infrastructure. Unfortunately, however, it is a truism that there always exists a group of individuals who wish to exploit vulnerable loopholes in new technologies.

These vulnerabilities pose a different risk than do traditional software systems bugs. Mainly, that these bugs can carry financial consequences and, once deployed to the network, are immutable programs which live on the network, unless the bytecode is destroyed. Without a system to declare contracts reasonably secure, there will always be doubt as to whether a contract or, more generally, the Ethereum blockchain is truly fit for financial use cases.

This research is an attempt to bring attention to the vulnerabilities of this technology and offer such a system – a digital registry of vulnerable contracts – for users, developers, and Ethereum enthusiasts. If users feel that new technology is safe and trustworthy, only then will they give up older technology in favor of the new. By analyzing the state of Ethereum and providing a system that creates transparency in specific contracts which contain vulnerabilities, we hypothesize that the number of users of the network will likely increase and adoption of Ethereum for financial use cases will grow.

References

1. B. Anderson: The Most In-Demand Hard and Soft Skills of 2020 (January 2020) https://business.linkedin.com/talent-solutions/blog/trends-and-research/2020/most-in-demand-hard-and-soft-skills.
2. B. Mueller: Mythril (2017) https://github.com/ConsenSys/mythril.
3. A. Groce: 246 Findings From Our Smart Contract Audits: An Executive Summary (September 2019) https://blog.trailofbits.com/2019/08/08/246-findings-from-our-smart-contract-audits-an-executive-summary/.
4. B. Mueller: laser-ethereum (October 2017) https://github.com/b-mueller/laser-ethereum.
5. SmartContractSecurity: SWCRegistry (January 2020) https:/swcregistry.io.
6. Microsoft: Z3 (September 2012) https://github.com/Z3Prover/z3.
7. Ethereum: go-ethereum (December 2013) https://github.com/ethereum/go-ethereum.
8. T. Dulcet: Send-Msg-CLI (December 2019) https://github.com/tdulcet/Send-Msg-CLI.
9. LucidChart: High-Level Program Overview (January 2020) https://www.lucidchart.com.
10. Vyper Team: Pythonic Smart Contract Language for the EVM (2017) https://github.com/vyperlang/vyper.
11. Perez, D., Livshits, B.: Smart contract vulnerabilities: Does anyone care? (May 2019) http://arxiv.org/abs/1902.06710.
12. TrailOfBits: Manticore (February 2017) https://github.com/trailofbits/manticore.
13. Consumer Reports Team: Consumer Reports (February 2020) https://www.consumerreports.org/cro/index.htm.
14. Virus Total Team: VirusTotal (February 2020) https://www.virustotal.com//.