EVMFuzzer: detect EVM vulnerabilities via fuzz testing

Conference Paper · August 2019

DOI: 10.1145/3338906.3341175

Ying Fu
Tsinghua University

Fuchen Ma
Tsinghua University

Heyuan Shi
Tsinghua University

Yu Jiang
Tsinghua University

EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing

Ying Fu[1], Meng Ren[1], Fuchen Ma[1], Heyuan Shi[1], Xin Yang[1], Yu Jiang[1*], Huizhong Li[2], Xiang Shi[2]
[1]School of Software, Tsinghua University, China
[2]Software Development and Quality Assurance Group, WeBank, China
fy17@mails.tsinghua.edu.cn,renm8@mail2.sysu.edu.cn,mfc@bupt.edu.cn,
{yangx16,shy15}@mails.tsinghua.edu.cn,jiangyu198964@126.com,{wheatli,jimmyshi}@webank.com

ABSTRACT

Ethereum Virtual Machine (EVM) is the run-time environment for smart contracts and its vulnerabilities may lead to serious problems to the Ethereum ecology. With lots of techniques being continuously developed for the validation of smart contracts, the testing of EVM remains challenging because of the special test input format and the absence of oracles. In this paper, we propose EVMFuzzer, the first tool that uses differential fuzzing technique to detect vulnerabilities of EVM. The core idea is to continuously generate seed contracts and feed them to the target EVM and the benchmark EVMs, so as to find as many inconsistencies among execution results as possible, eventually discover vulnerabilities with output cross-referencing. Given a target EVM and its APIs, EVMFuzzer generates seed contracts via a set of predefined mutators, and then employs dynamic priority scheduling algorithm to guide seed contracts selection and maximize the inconsistency. Finally, EVMFuzzer leverages benchmark EVMs as cross-referencing oracles to avoid manual checking. With EVMFuzzer, we have found several previously unknown security bugs in four widely used EVMs, and 5 of which had been included in Common Vulnerabilities and Exposures (CVE) IDs in U.S. National Vulnerability Database.

The video is presented at https://youtu.be/9Lejgf2GSOk.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Differential testing, fuzzing, domain-specific mutation, EVM

ACM Reference Format:
Ying Fu[1], Meng Ren[1], Fuchen Ma[1], Heyuan Shi[1], Xin Yang[1], Yu Jiang[1*], Huizhong Li[2], Xiang Shi[2]. 2019. EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3338906.3341175

*Yu Jiang is the correspondence author.

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3341175

1 INTRODUCTION

Ethereum can be viewed as a transaction-based state machine [33]. Ethereum Virtual Machine (EVM) is often called the operating system of the Ethereum technology and is responsible for the execution and maintenance of smart contracts. Over the past few years, the safety and security problems of the transactions have emerged endlessly, causing huge property loss. As the authentic platform and standard for Ethereum transaction executing, if there are some vulnerabilities in EVM's internal implementation, it will definitely lead to serious consequences. At present, EVM has at least 10 widely used official implementations of different programming language [7]. Some open source projects are also using modified EVMs. Lack of mature testing tool for EVM, it is difficult to guarantee the security of EVM. So, it is of great urgency to find a efficient way to secure EVM.

In this paper, we present EVMFuzzer[1], the first automated differential fuzz testing tool to efficiently mine vulnerabilities of EVMs implementations. EVMFuzzer firstly defines the indicators of the EVM execution differences, it uses the opcode sequence executed and gas used as two important indicators to evaluate EVMs' performance on each test contract. EVMFuzzer integrates some of the widely used EVMs as benchmark EVMs and creates a unified running environment for the target EVM and benchmark EVMs. In this way, it takes the natural advantages of differential testing to quickly discover the output inconsistencies without manual checking. Then, the seed contract mutation and selection algorithms can continuously generate contracts that enlarge the metric difference, so that EVMFuzzer can efficiently mine cases that trigger differential performance of EVMs and try to get those corner cases with inconsistent execution output.

For evaluation, we collected 36,295 real-world smart contracts from Etherscan [11] as our initial seeds. Through guided fuzzing, 1,596 variants of those initial seed contracts successfully triggered inconsistent execution output among different EVMs. With manual root cause analysis, we found several previously unknown security bugs in four widely used EVMs, and 5 of which had been included in Common Vulnerabilities and Exposures (CVE) database [24].

2 RELATED WORK

Fuzzing Technique. Fuzzing is an automatic testing technique that covers numerous boundary cases using invalid data as input to ensure the absence of exploitable vulnerabilities [18]. Some popular AFL [34] family tools [2–4, 14, 17, 19, 20, 28, 30, 32, 35] apply various strategies to boost fuzzing process, including symbolic execution, schedule algorithm and so on. For example, EnFuzz [4] integrates multiple fuzzing strategies to obtain better performance than that

[1]EVMFuzzer is available at https://github.com/EVMFuzzer/EVMFuzzer

of any constituent fuzzer alone. There are also some tools focus on fuzzing in other domains, for example, QuanFuzz [31] is a search-based test input generator for the quantum programs.

Differential Testing. Differential testing [23] has been very successful in uncovering differences between independent implementations with similar intended functionality. For example, Chen et. alperform differential testing of JVMs using MCMC sampling for input generation [5]. DeepXplore [27] was presented as the state-of-the-art white-box differential testing framework for deep learning systems. DLFuzz [13] extends differential testing framework for DL systems with the comparisons of multiple similar inputs, and does not need multiple platforms.

Smart Contract Validation. Smart contracts have been shown to be exposed to severe vulnerabilities [1, 15], and many efforts[16, 21, 26] have been devoted to ensure its' correctness. For example, Luu et. al [21] designed Oyente, which builds the control-flow graph from the bytecode and then performs symbolic execution and checks whether there exist any vulnerable patterns. Zeus [16] is a sound analyzer that translates smart contracts to the LLVM framework and uses XACML as a language to write properties.

Main Difference. Different from the above work, EVMFuzzer mainly focuses on discovering the vulnerabilities in EVM. It takes the lead in paying attention to EVM security while others mainly concerned about smart contracts. Particularly, EVMFuzzer combines the basic ideas of fuzzing and takes advantage of EVMs' multi-implementation to quickly find output discrepancies and reduce manual checks. Within EVMFuzzer, we also define the domain specific EVM test indicators to guide the differential fuzzing process with different contract mutation and selection strategies.

3 EVMFUZZER DESIGN

The overall workflow of EVMFuzzer is shown in Fig. 1, which consists of two major components, seed contract generation module and unified EVM execution module. The goal of EVMFuzzer is to apply differential fuzz testing on EVMs. EVMFuzzer will continuously provide mutated smart contract to several EVM platforms including target EVM and benchmark EVMs. These EVMs are then monitored for catching "different act" on some inputs, if so, we may find a bug in some of the EVMs.

EVMFuzzer takes the target EVM and its API as input, and then the unified EVM execution module will create a unified execution environment for the target EVM and the benchmark EVMs. The seed contract generation module is responsible for continuously generating high quality seeds which enlarge the difference between the EVMs and it will feed the seeds into the unified EVM execution module. We will briefly introduce the two major components of EVMFuzzer in the following part, and you can refer to our report [12] for more details.

3.1 Seed Contract Generation

The seed contract generation module can be viewed as a test case generator. From Fig. 1, we can see that the seed contracts are stored in the seed pool. EVMFuzzer will rank the candidate contracts accoring to dynamic priority, and the contract in the first place will be selected for the next iteration. After choosing the contract for

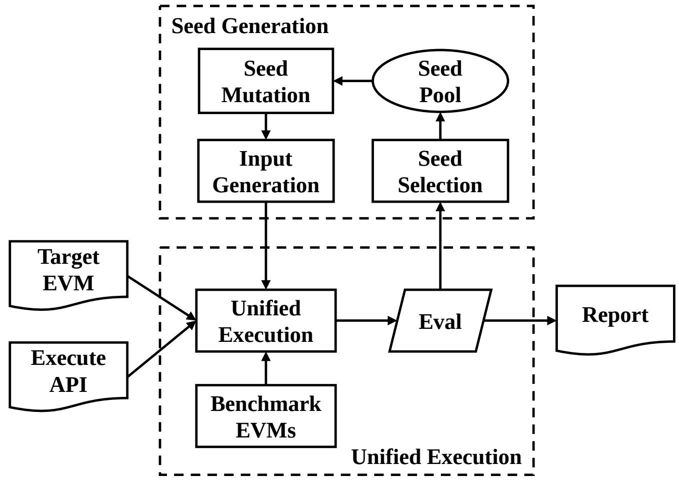

Figure 1: Overall workflow of EVMFuzzer, which mainly includes the seed generation module for guided contract generation and the unified execution module for information collection and cross-checking.

mutation, EVMFuzzer uses 8 predefined mutators and the combined strategy to guide mutant generation. The goal is to generate contracts that can increase the degree of metric difference and trigger different execution output of target EVM and benchmark EVMs.

Seed Mutation. Currently, we design 8 mutators according to the functional logic features of the smart contract. These variations are based on three different granularity, including the word-level, character-level and statement-level. Each mutator performs differently, so we can also maintain a priority queue based on the feedback metric difference. For a seed contract, we update the weight of corresponding mutators after the multi-version EVMs comparison, which is similar to the initialization of metric difference priority. If the metric difference increases, the mutator ID is pushed into the queue in an descending order of the weight; otherwise, the queue will not update. Except for the weight update, we design five mutator combined strategies to further increase the randomness and diversity of mutation in each iteration, as detailed in [12].

Seed Prioritization and Selection. As mentioned above, all the qualified seed contracts are stored in the seed contract pool. Based on the priority of the seed, we decide which seed to be mutated in a new iteration. In general, the contract that makes the metric difference among EVMs larger should be the candidates for the next mutation iteration. But at the same time, in order to ensure the diversity, other contracts should also have a certain probability of being selected. Therefore, we use the dynamic priority scheduling algorithm to maintain a candidate queue. For each contract, we give it an initial priority, and then its value changes with the increasing of waiting time to ensure that every seed would be selected.

3.2 Unified EVM Execution

EVM execution module provides a unified runtime environment for various EVMs. After receiving the contract file from the seed contract generation module, it compiles the seed into EVM bytecode. The input parameter is generated according to the data type of the called function, thus the uniform input for each EVM is obtained. Then EVMFuzzer automatically runs all EVMs, calculates the difference information according to the test metric, and compares the execution output results. Finally, according to the seed's ability to enhance the degree of metric difference, EVMFuzzer decides whether to put the seed contract into the seed pool where high-quality seeds preserved. Besides, when the execution output is inconsistent, this module will also record the potential exception for manual root cause analysis.

4 USING EVMFUZZER

4.1 Tool Implementation

We have implemented EVMFuzzer, as shown in Fig. 2. Specifically, EVMFuzzer provides a command line UI to interact with users as in Fig. 3. Currently, EVMFuzzer needs the target EVM's source code or executable file and corresponding APIs. In the backend, EVMFuzzer integrates four widely used EVMs as the benchmark EVMs. Those four benchmark EVM are ethereumjs-vm v2.4.0 [10], py-evm v0.2.0-alpha.31 [9], aleth v1.5.0-alpha.6 [6] and geth v1.8.13 [8]. EVMFuzzer uses the solc 0.4.24 compiler[29] to generate seed contract's EVM bytecode. The difference calculator is responsible for generating runtime difference among EVMs, which are used by Seed Selector to determine whether keep the seed contract or not and to update seed contract's priority.

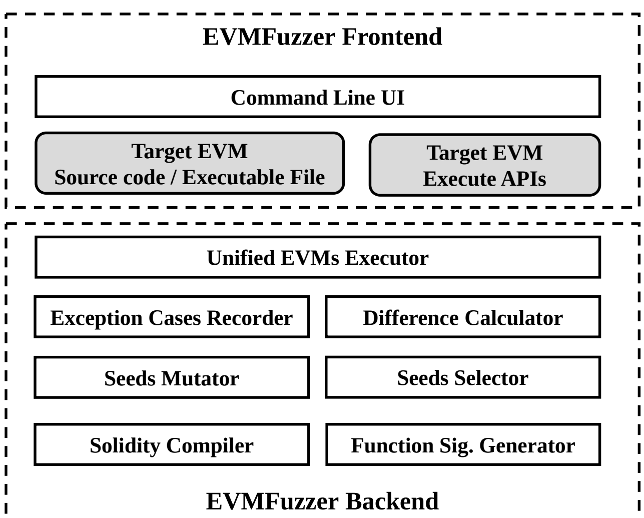

Figure 2: The implementation architecture of EVMFuzzer.

4.2 Running Example

Fig. 3 shows a screenshot of EVMFuzzer. We can see that EVMFuzzer first shows the version information and then lists the fuzz steps. It also displays the help information to let users know what to input. Since EVMFuzzer is an automated testing tool, users only need to put target EVM in the specified directory("myEVM" folder), enter the API, and set the fuzz times, the EVMFuzzer will start. Here we construct a reinforced js-EVM[2] which is able to stop dangerous transaction for the test.

```
WELCOME TO EVMFUZZer!

VERSION:
   2.0.1

STEPS:
   1 >> Download and setup the environment.
   2 >> Upload user's EVM and the interface.
   3 >> Fuzzing.
   4 >> Sending the report.

Please put your source code under **myEVM folder** and provide the trasaction interface.
The interface form should like: xxxxxx --code A --input B
where A is the runtime code of contract, and B is the input data for transaction.


E.g.  python3 /home/usr/project/myEVM/runTx.py --code A --data B
node /home/usr/project/myEVM/runcode.js --code A --sig B


Does the input data need '0x' prefix? The default setting is Yes. (Y/N)
Y


Please set fuzz times: (The default setting is 100)
100
```

Figure 3: CommandLine UI of EVMFuzzer

When the fuzz ends, EVMFuzzer will generate test report for the target EVM. Fig. 4 shows the detection report of the EVMFuzzer. The auto-generated report evaluates the target EVM from three dimensions: code implementation completeness, accuracy of gas calculation and rationality of execution path planning, so that users can have a preliminary understanding of the target EVM and design a customized optimization. Users can find all generated test inputs in the "TestOut" directory, and the "result.json" file records all the inconsistencies during the fuzz test.

```
Detection Report

After 1932.00s running, EVMFuzzer successfully generated 100 seeds.
   > In 0.0% (0/100) of the tests, the output results are different from the standard execution.
   > In 52.0% (52/100) of the tests, the gas consuming are more than the average value.
   > In 39.0% (39/100) of the tests, the gas consuming are less than the average value.
   > In 9.0% (9/100) of the tests, the gas consuming are equal to the average value.
   > In 21.0% (21/100) of the tests, the execution sequence are more than the average value.
   > In 70.0% (70/100) of the tests, the execution sequence are less than the average value.
   > In 9.0% (9/100) of the tests, the execution sequence are equal to the average value.

Final Conslusion:
1. The code implementation of test EVM is relatively complete, can be used for Ethereum.
2. The deviation of gas calculation is large, there exists dynamic optimization during the calcula
tion process, which need further improvement.
3. The deviation of executing sequence is large, there exists dynamic optimization during the exec
uting procedure, and the optimization effect is good.
```

Figure 4: Report of EVMFuzzer

5 PRELIMINARY EVALUATION

We use those four benchmark EVMs described in tool implementation for cross-validation, this section shows some preliminary evaluation results. Our initial seed contracts are the 36,295 real-world contracts which were crawled from the Etherscan [11]. All experiments were performed atop a machine with 8 cores (Intel i7-7700HQ @3.6GHz), 16GB of memory, and Ubuntu 16.04.4 as the host operating system.

Inconsistency among EVMs. After the experiment, based on the two internal test indicators: *gasUsed* and opcode sequence, we found large number of EVM discrepancies.

There are 33,424 contracts executed normally with the same opcode sequence. They are used for *gasUsed* comparison, which excludes the inconsistency of gas consumption caused by different execution opcode sequences. Table 2 shows the number of contracts with *gasUsed* inconsistencies among different EVMs. We can see that almost every platform has over 50% average inconsistency rate of *gasUsed* with others, and aleth even produces a different gas consumption over 90% of contracts.

[2]We implemented this test EVM according to the idea of EVM*[22]

Table 1: Description of 5 high-risk vulnerabilities detected by EVMFuzzer

CVE-ID	platform	version	language	description	created date
CVE-2018-18920	Py-EVM	v0.2.0-alpha.33	python	Py-EVM v0.2.0-alpha.33 allows attackers to make a vm.execute_bytecode call that triggers illegal values shown in stack.	20181103
CVE-2018-19183	js-evm	v2.4.0	JavaScript	ethereumjs-vm 2.4.0 allows attackers to cause a denial of service (vm.runCode failure and REVERT) via a "code: Buffer.from(my_code, 'hex')" attribute.	20181111
CVE-2018-19184	geth	v1.8.17	golang	cmd/evm/runner.go in Go Ethereum (aka geth) 1.8.17 allows attackers to cause a denial of service (SEGV) via crafted bytecode.	20181111
CVE-2018-19330	aleth	v1.5.0-alpha.6	cpp	** RESERVED ** Details would be public after the vulnerability has been repaired to avoid potential attack.	20181117
CVE-2019-7710	aleth	v1.5.0-alpha.7	cpp	** RESERVED ** Details would be public after the vulnerability has been repaired to avoid potential attack.	20190210

Table 2: gasUsed inconsistency

	js-evm	Py-EVM	aleth	geth	total
js-evm	0	18115	31166	17486	66767
Py-EVM	18115	0	31176	1358	50649
aleth	31166	31176	0	31163	93505
geth	17486	1358	31163	0	50007

In total, 1,275 seed contracts were successfully executed on four EVM platforms and return the same output, but the sequence lengths were different. From Table 3, we can see that the sequence length of geth and aleth on these 1,275 contracts were always the same so we take it as baseline. And after calculation, the length of the opcode sequence of js-evm is small and always below the baseline; but the length of the execution sequence of Py-EVM is large and always above the baseline.

Table 3: opcode inconsistency

	js-evm	Py-EVM	aleth	geth
js-evm	0	1275	52	52
Py-EVM	1275	0	1240	1240
aleth	52	1240	0	0
geth	52	1240	0	0

From the above statistics, it is reasonable to conclude that there are inconsistencies among the implementation and execution of different EVMs, and it is possible to leverage the metric difference of $gasUsed$ and opcode sequence indicator to guide the generation of contracts resulting in potential inconsistent execution output.

Vulnerabilities detected by EVMFuzzer. After discovering thousands of output inconsistencies, we conducted the manual analysis and tried to explore the root causes. We ensured its reproducibility and then carefully reviewed the source code of EVMs. Finally, we found defects in the EVM platforms, of which, 5 previously unknown vulnerabilities were registered as Common Vulnerabilities and Exposures, numbered as CVE-2018-18920, CVE-2018-19183, CVE-2018-19184, CVE-2018-19330 and CVE-2019-7710, shown in Table 1.

We choose one of the CVEs for detailed elaboration. CVE-2018-19184[25] is an execution segmentation violation that occurred on EVM of Go Ethereum (geth)[8]. The code associated with this vulnerability was in the cmd/evm folder, where the exception handling mechanism of EVM before geth v1.8.14 did not cover enough corner cases. Although the problematic code snippet is not the API that directly exposed to the end users, this problem can be exploited by malicious attackers to cause the denial of service.

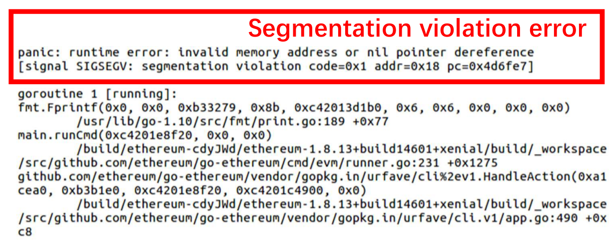

Figure 5: Segmentation violation error on geth-evm v1.8.13.

6 CONCLUSION

In this paper, We propose EVMFuzzer, the first differential fuzz testing tool, to efficiently detect vulnerabilities of EVM implementations. EVMFuzzer introduces the definition of EVM fuzz testing metrics–$gasUsed$ and opcode sequence, which measure the internal difference in execution information between EVMs. Besides, EVMFuzzer designs 8 mutators for smart contracts, so that it can generate plenty of seed contracts without syntax error in a short time. Under the guided seed generation and selection algorithm, EVMFuzzer shows strong defects mining capabilities.

We evaluated EVMFuzzer based on four widely used EVM implementations and conducted numerous mutation on 36,295 real-world smart contracts. Among the generated 253,153 smart contracts, more than half successfully showed the differential performance, including 1,596 variant contracts triggered inconsistent output results among the four EVM platforms. With manual root cause analysis, 5 vulnerabilities have been assigned with unique CVE IDs. Our future work mainly includes developing more general seeds mutators and conducting more extensive evaluations on more EVMs.

REFERENCES

[1] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2016. A survey of attacks on Ethereum smart contracts. *IACR Cryptology ePrint Archive* 2016 (2016), 1007.

[2] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed Greybox Fuzzing. In *ACM Conference on Computer and Communications Security*.

[3] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based Greybox Fuzzing as Markov Chain. In *ACM Conference on Computer and Communications Security*.

[4] Yuanliang Chen, Yu Jiang, Jie Liang, Mingzhe Wang, and Xun Jiao. 2018. EnFuzz: From Ensemble Learning to Ensemble Fuzzing. *CoRR* abs/1807.00182 (2018).

[5] Yuting Chen, Ting Su, Chengnian Sun, Zhendong Su, and Jianjun Zhao. 2016. Coverage-directed differential testing of JVM implementations. In *PLDI*.

[6] Ethereum. 2018. Ethereum C++ client, tools and libraries. https://github.com/ethereum/aleth. (2018).

[7] Ethereum. 2018. Ethereum clients. https://github.com/ethereum/wiki/wiki/Clients,-tools,-dapp-browsers,-wallets-and-other-projects. (2018).

[8] Ethereum. 2018. Go Ethereum. https://github.com/ethereum/go-ethereum. (2018).

[9] Ethereum. 2018. Python Implementation of the EVM. https://github.com/ethereum/py-evm. (2018).

[10] EthereumJS. 2018. The Ethereum VM implemented in Javascript. https://github.com/ethereumjs/ethereumjs-vm. (2018).

[11] Etherscan. 2018. Ethereum (eth) blockchain explorer. https://etherscan.io/. (2018).

[12] Ying Fu, Meng Ren, Fuchen Ma, Yu Jiang, Heyuan Shi, and J. Tangting Sun. 2019. EVMFuzz: Differential Fuzz Testing of Ethereum Virtual Machine. *CoRR* abs/1903.08483 (2019).

[13] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jia-Guang Sun. 2018. DLFuzz: differential fuzzing testing of deep learning systems. In *ESEC/SIGSOFT FSE*.

[14] Hertz and Newsham. 2015. TriforceAFL - AFL/QEMU fuzzing with full-system emulation. https://github.com/nccgroup/TriforceAFL/. (2015). Accessed February 18, 2019.

[15] Yoichi Hirai. 2016. Formal verification of Deed contract in Ethereum name service. *November-2016.[Online]. Available: https://yoichihirai. com/deed. pdf* (2016).

[16] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts. In *NDSS*.

[17] Caroline Lemieux and Koushik Sen. 2018. FairFuzz: a targeted mutation strategy for increasing greybox fuzz testing coverage. In *ASE*.

[18] Hongliang Liang, Xiaoxiao Pei, X. Jia, Wuwei Shen, and Jian Guang Zhang. 2018. Fuzzing: State of the Art. *IEEE Transactions on Reliability* 67 (2018), 1199–1218.

[19] Jie Liang, Yu Jiang, Yuanliang Chen, Mingzhe Wang, OldÅŹrich Motyka, and Jia-Guang Sun. 2018. PAFL: extend fuzzing optimizations of single mode to industrial parallel mode. In *ESEC/SIGSOFT FSE*.

[20] Jie Liang, Mingzhe Wang, Yuanliang Chen, Yu Jiang, and Renwei Zhang. 2018. Fuzz testing in practice: Obstacles and solutions. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2018), 562–566.

[21] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. *IACR Cryptology ePrint Archive* 2016 (2016), 633.

[22] Fuchen Ma, Ying Fu, Meng Ren, Mingzhe Wang, Yu Jiang, Kaixiang Zhang, Huizhong Li, and Xiang Shi. 2019. EVM*: From Offline Detection to Online Reinforcement for Ethereum Virtual Machine. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019), 554–558.

[23] William M. McKeeman. 1998. Differential Testing for Software. *Digital Technical Journal* 10 (1998), 100–107.

[24] MITRE. 2018. Common vulnerabilities and exposures. https://cve.mitre.org/. (2018).

[25] MITRE. 2018. CVE-2018-19184. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-19184. (2018).

[26] Ivica Nikolic, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. 2018. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. *CoRR* abs/1802.06038 (2018).

[27] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *SOSP*.

[28] Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, and Thorsten Holz. 2017. kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels. In *USENIX Security Symposium*.

[29] Solidity. 2018. Solidity Programming Language. https://git.io/vFA47/. (2018).

[30] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Krügel, and Giovanni Vigna. 2016. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. In *NDSS*.

[31] Jiyuan Wang, Ming Gao, Yu Jiang, Jian-Guang Lou, Yue Gao, Dongmei Zhang, and Jia-Guang Sun. 2018. QuanFuzz: Fuzz Testing of Quantum Program. *CoRR* abs/1810.10310 (2018).

[32] Mingzhe Wang, Jie Liang, Yuanliang Chen, Yu Jiang, Xun Jiao, Hao Liu, Xibin Zhao, and Jia-Guang Sun. 2018. SAFL: Increasing and Accelerating Testing Coverage with Symbolic Execution and Guided Fuzzing. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)* (2018), 61–64.

[33] Dr Gavin Wood. 2014. Ethereum: a Secure Decentralised Generalised Transaction Ledger.

[34] Michal Zalewski. 2015. american fuzzy lop. http://lcamtuf.coredump.cx/afl/. (2015).

[35] Google Project Zero. 2015. WinAfl - A fork of AFL for fuzzing Windows binaries. https://github.com/googleprojectzero/winafl. (2015). Accessed February 18, 2019.