**EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing**

EVMFuzzer：通过模糊测试检测EVM漏洞

**Ying Fu[1], Meng Ren, Fuchen Ma, Heyuan Shi, Xin Yang, Yu Jian, Huizhong Li, Xiang Shi**

School of Software, Tsinghua University, China

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](mailto:{yangx16,shy15}@mails.tsinghua.edu.cn,jiangyu198964@126.com,{wheatli,jimmyshi}@webank.com)

# ABSTRACT

# 摘要

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.

以太坊虚拟机（EVM）是智能合约的运行时环境，其漏洞可能会给以太坊生态带来严重问题。随着智能合约验证技术的不断发展，由于特殊的测试输入格式和预言机的缺乏，EVM的测试仍然具有挑战性。在本文中，我们提出了EVMFuzzer，第一个使用差分模糊技术来检测EVM漏洞的工具。其核心思想是不断生成种子合约并将其反馈给目标EVM和基准EVM，从而尽可能多地发现执行结果之间的不一致，最终通过输出交叉引用发现漏洞。给定一个目标EVM及其api，EVMFuzzer通过一组预定义的突变子生成种子合约，然后采用动态优先级调度算法来指导种子合约的选择并最大化不一致性。最后，EVMFuzzer利用基准EVMs作为交叉引用预言机来避免手动检查。通过EVMFuzzer，我们在四个广泛使用的EVM中发现了几个以前未知的安全漏洞，其中5个已被列入美国国家漏洞数据库的通用漏洞和暴露（CVE）id中。

**CCS CONCEPTS**

**CCS概念**

• Software and its engineering → Software testing and debugging.

•软件及其工程→软件测试和调试。

**KEYWORDS**

**关键词**

Differential testing, fuzzing, domain-specific mutation, EVM

差异测试，模糊化，域特异性突变，EVM

# 1 INTRODUCTION

# 1简介

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.

以太坊可以看作是一个基于交易的状态机[33]。以太坊虚拟机（EVM）通常被称为以太坊技术的操作系统，负责智能合约的执行和维护。近几年来，交易的安全保障问题层出不穷，造成了巨大的财产损失。作为以太坊事务执行的可信平台和标准，如果EVM的内部实现存在漏洞，必将导致严重后果。目前，EVM至少有10种广泛使用的不同编程语言的官方实现[7]。一些开源项目也在使用修改过的EVMs。由于缺乏成熟的EVM测试工具，EVM的安全性难以保证。因此，寻找一种有效的方法来保证EVM的安全就显得尤为迫切。

In this paper, we present EVMFuzzer, 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.

在本文中，我们提出了EVMFuzzer，第一个自动化的差分模糊测试工具，以有效地挖掘EVMs实现中的漏洞。EVMFuzzer首先定义了EVM执行差分的指标，它将执行的操作码序列和gas作为评价EVM在每个测试合约上的性能的两个重要指标。EVMFuzzer将一些广泛使用的EVM集成为基准EVM，并为目标EVM和基准EVM创建统一的运行环境。通过这种方式，它利用了差分测试的固有优势，无需手动检查即可快速发现输出不一致。然后，种子合约变异和选择算法能够连续生成增大度量差的合约，使得EVMFuzzer能够有效地挖掘触发EVMs差分性能的案例，并尝试得到执行输出不一致的角落案例。

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].

为了进行评估，我们从Etherscan[11]收集了36295个真实世界的智能合约作为我们的初始种子。通过引导模糊化，1596个初始种子合约的变体成功地触发了不同EVM之间不一致的执行输出。通过手动分析根本原因，我们在四个广泛使用的EVM中发现了几个以前未知的安全漏洞，其中5个已包含在Common Vulnerabilities and Exposure（CVE）数据库中[24]。

# 2 RELATED WORK

# 2相关工作

**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 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.

**模糊化技术**。模糊化是一种自动化测试技术，它覆盖了许多边界情况，使用无效数据作为输入，以确保不存在可利用的漏洞[18]。一些流行的AFL[34]族工具[2–4、14、17、19、20、28、30、32、35]应用各种策略来增强模糊过程，包括符号执行、调度算法等。例如，EnFuzz[4]集成了多种模糊策略，以获得比单独使用任何组成模糊器更好的性能。还有一些工具专注于其他领域的模糊化，例如，QuanFuzz[31]是一个基于搜索的量子程序测试输入生成器。

**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-ofthe-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.

**差异测试**。差异测试[23]非常成功地揭示了具有类似预期功能的独立实现之间的差异。例如，Chen 等人使用MCMC采样生成输入来执行jvm的差异测试[5]。DeepXplore[27]被认为是深度学习系统最先进的白盒差异测试框架。DLFuzz[13]通过比较多个相似输入，扩展了DL系统的差异测试框架，不需要多个平台。

**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.

**智能合约验证**。智能合约已被证明存在严重的漏洞[1，15]，许多努力[16，21，26]都致力于确保其正确性。例如，Luu等人[21]设计了Oyente，它从字节码构建控制流图，然后执行符号执行并检查是否存在任何易受攻击的模式。Zeus[16]是一个sound分析器，它将智能合约转换为LLVM框架，并使用XACML作为编写属性的语言。

**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.

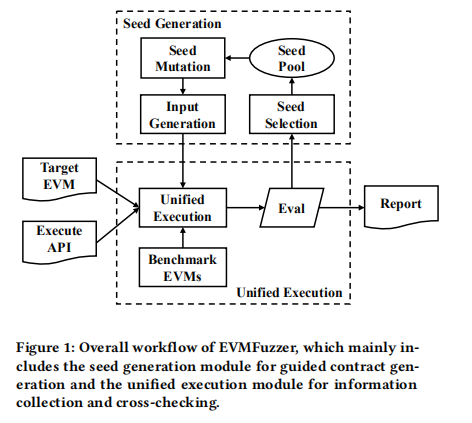
主要区别。与上述工作不同，EVMFuzzer主要致力于发现EVM中的漏洞。它率先关注EVM安全，其他人主要关注智能合约。特别地，EVMFuzzer结合了模糊化的基本思想，利用EVMs的多重实现来快速发现输出差异，减少人工检查。在EVMFuzzer中，我们还定义了特定领域的EVM测试指标，用以指导具有不同合约变异和选择策略的差分模糊过程。

# 3 EVMFUZZER DESIGN

# 3 EVMFUZZER设计

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的总体工作流程如图1所示，它由两个主要组件组成：种子合约生成模块和统一EVM执行模块。EVMFuzzer的目标是在EVMs上应用差分模糊测试。EVMFuzzer将持续向多个EVM平台（包括目标EVM和基准EVM）提供变异智能合约。然后监控这些EVM以捕捉某些输入的“不同行为”，如果是这样，我们可能会在一些EVM中发现一个bug。

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.

EVMFuzzer将目标EVM及其API作为输入，然后统一EVM执行模块将为目标EVM和基准EVM创建一个统一的执行环境。种子合约生成模块负责不断生成高质量的种子，扩大EVM之间的差异，并将种子送入统一的EVM执行模块。下面我们将简要介绍EVMFuzzer的两个主要组件，您可以参考我们的报告[12]了解更多细节。

## 3.1 Seed Contract Generation

## 3.1种子合约的产生

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 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.

种子合约生成模块可以看作是一个测试用例生成器。从图1可以看出，种子合约存储在种子池中。EVMFuzzer将根据动态优先级对候选合约进行排序，并选择第一位的合约进行下一次迭代。选择合约后突变，EVMFuzzer使用8个预先定义的突变子和组合策略来指导突变的产生。其目标是生成合约，以增加度量差异的程度，并触发目标EVM和基准EVM的不同执行输出。

**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].

**种子突变**。目前，我们根据智能合约的功能逻辑特点，设计了8个变异器。这些变体基于三种不同的粒度，包括单词级、字符级和语句级。每个变异器的性能都不同，因此我们还可以根据反馈度量的差异来维护优先级队列。对于种子合约，我们在多版本EVMs比较之后更新相应变异子的权重，这类似于度量差优先级的初始化。如果度量差增加，则按权重的降序将变异子ID推入队列；否则，队列将不会更新。除了权值更新外，我们还设计了五种变异子组合策略，以进一步增加每次迭代中变异的随机性和多样性，详见[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.

种子优先排序和选择。如上所述，所有合格的种子合约都存储在种子合约池中。根据种子的优先级，我们决定在一个新的迭代中变异哪个种子。一般来说，使EVM之间的度量差变大的合约应该是下一个变异迭代的候选。但同时，为了保证多样性，其他合约也应有一定的被选择概率。因此，我们采用动态优先级调度算法来维护候选队列。对于每个合约，我们给它一个初始优先级，然后它的值随着等待时间的增加而变化，以确保每个种子都能被选中。

## 3.2 Unified EVM Execution

## 3.2统一EVM执行

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.

EVM执行模块为各种EVM提供了一个统一的运行时环境。在从种子合约生成模块接收合约文件之后，它将种子编译成EVM字节码。根据被调用函数的数据类型生成输入参数，从而得到每个EVM的统一输入。然后EVMFuzzer自动运行所有EVM，根据测试度量计算差异信息，并比较执行输出结果。最后，根据种子增强度量差异度的能力，EVMFuzzer决定是否将种子合约放入保存优质种子的种子库中。此外，当执行输出不一致时，本模块还将记录潜在异常，以便手动进行根本原因分析。

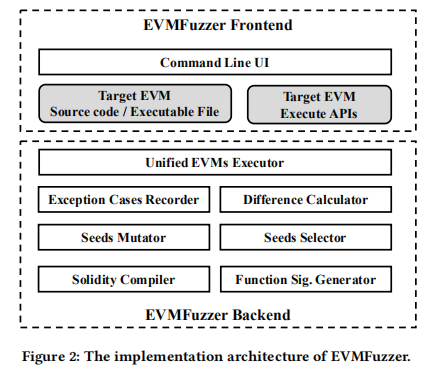
# 4 USING EVMFUZZER

# 4使用EVMFUZZER

## 4.1 Tool Implementation

## 4.1工具实现

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.

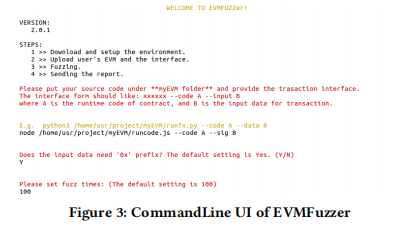
我们已经实现了EVMFuzzer，如图2所示。具体来说，EVMFuzzer提供了一个命令行UI来与用户交互，如图3所示。目前，EVMFuzzer需要目标EVM的源代码或可执行文件以及相应的api。在后端，EVMFuzzer集成了四个广泛使用的EVMs作为基准EVMs。这四个基准EVM是ethereumjs-vm v2.4.0[10]、py-evm v0.2.0-alpha.31[9]、aleth v1.5.0-alpha.6[6]和geth v1.8.13[8]。EVMFuzzer使用solc0.4.24编译器[29]生成种子合约的EVM字节码。差异计算器负责生成EVM之间的运行时差异，种子选择器使用这些差异来确定是否保留种子合约以及更新种子合约的优先级。

## 4.2 Running Example

## 4.2运行示例

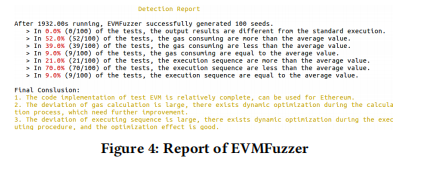
Fig. 3 shows a screenshot of EVMFuzzer. We can see that EVMFuzzer first shows the version information and then lists the fuzz steps. It also displays the help information to let users know what to input. Since EVMFuzzer is an automated testing tool, users only need to put target EVM in the specified directory("myEVM" folder), enter the API, and set the fuzz times, the EVMFuzzer will start. Here we construct a reinforced js-EVM which is able to stop dangerous transaction for the test.[[2]](https://fanyi.baidu.com/" \l "_ftn2" \o ")

图3显示EVMFuzzer的屏幕截图。我们可以看到EVMFuzzer首先显示版本信息，然后列出模糊步骤。它还显示帮助信息，让用户知道要输入什么。由于EVMFuzzer是一个自动化测试工具，用户只需将目标EVM放在指定的目录（“myEVM”文件夹）中，输入API，设置模糊时间，EVMFuzzer就可以启动了。在这里，我们构造了一个增强的js-evm，它能够为测试停止危险的交易。[[2]](https://fanyi.baidu.com/" \l "_ftn2" \o ")



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.

当模糊结束时，EVMFuzzer将为目标EVM生成测试报告。图4示出了EVMFuzzer的检测报告。自动生成的报表从代码实现的完整性、gas计算的准确性和执行路径规划的合理性三个维度对目标EVM进行评估，使用户可以初步了解目标EVM，并设计定制的优化方案。用户可以在“TestOut”目录中找到所有生成的测试输入，并且result.json“文件记录了模糊测试期间的所有不一致。



# 5 PRELIMINARY EVALUATION

# 5初步评价

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.

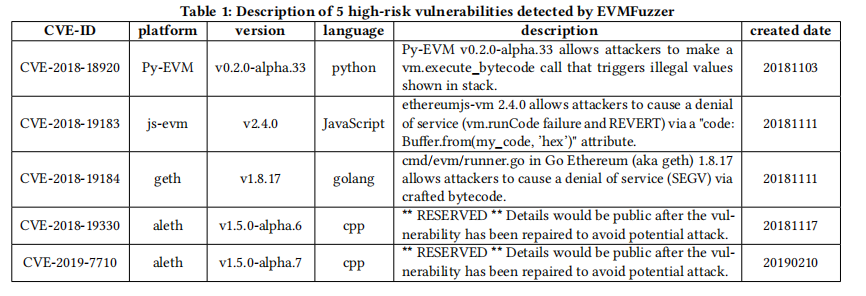
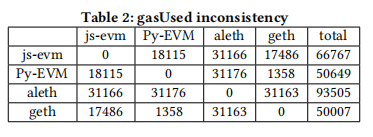
我们使用工具实现中描述的四个基准EVM进行交叉验证，本节展示了一些初步的评估结果。我们最初的种子合约是36295份现实世界的合约，这些合约是从Etherscan[11]抓取的。所有实验都是在一台拥有8核（Intel i7-7700HQ@3.6GHz）、16GB内存和ubuntu16.04.4主机操作系统的机器上进行的。

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

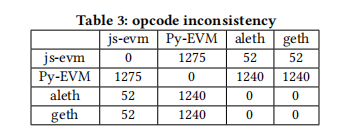
**EVM之间不一致性**。经过实验，基于两个内部测试指标：gas使用量和操作码序列，我们发现了大量的EVM差异。

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

共有33424个合约以相同的操作码序列正常执行。它们用于比较，排除了不同执行操作码序列引起的gas消耗的不一致性。表2显示了不同EVM之间不一致的合约数量。我们可以看到，几乎每个平台与其他平台的平均不一致率都在50%以上，而aleth甚至在90%以上的合约中产生了不同的gas量。

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.

总共有1275个种子合约在四个EVM平台上成功执行，并且返回相同的输出，但是序列长度不同。从表3可以看出，这1275份合同中geth和aleth的序列长度始终相同，因此我们将其作为基线。并且经过计算，js-evm的操作码序列长度较小，始终低于基线；而Py-evm的执行序列长度较大，始终高于基线。

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 gas-used and opcode sequence indicator to guide the generation of contracts resulting in potential inconsistent execution output.

从以上统计数据可以合理地得出结论，不同EVM的实现和执行之间存在不一致性，并且可以利用度量差和操作码序列指示符来指导合约的生成，从而导致潜在的不一致执行输出。

**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.

**EVMFuzzer检测到的漏洞**。在发现数千个输出不一致之后，我们进行了手动分析，并试图探索根本原因。我们确保了它的再现性，然后仔细审查了EVMs的源代码。最后，我们发现EVM平台存在缺陷，其中5个以前未知的漏洞被注册为常见漏洞并暴露，编号为CVE-2018-18920、CVE-2018-19183、CVE-2018-19184、CVE-2018-19330和CVE-2019-7710，如表1所示。

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.

我们选择其中一个CVE进行详细阐述。CVE-2018-19184[25]是发生在Go Ethereum（geth）[8]的EVM上的执行分段冲突。与此漏洞相关的代码位于cmd/evm文件夹中，gethv1.8.14之前的evm异常处理机制没有覆盖足够的角落案例。尽管有问题的代码段不是直接暴露给最终用户的API，但恶意攻击者可以利用此问题造成拒绝服务。

# 6 CONCLUSION

# 6结论

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–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.

本文提出了第一个差分模糊测试工具EVMFuzzer来有效地检测EVM实现的漏洞。EVMFuzzer引入了EVM模糊测试度量和操作码序列的定义，它们度量EVM之间执行信息的内部差异。此外，EVMFuzzer还为智能合约设计了8个变异子，使得它能够在短时间内生成大量的种子合约而不会出现语法错误。在引导种子生成和选择算法下，EVMFuzzer具有很强的漏洞挖掘能力。

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.

我们评估了基于四个广泛使用的EVM实现的EVMFuzzer，并对36295个真实世界的智能合约进行了多次变异。在生成的253153份智能合约中，超过一半的合约成功地表现出了不同的表现，其中1596份变体合约触发了四个EVM平台之间不一致的输出结果。通过手动根本原因分析，已为5个漏洞分配了唯一的CVE ID。我们未来的工作主要包括开发更通用的种子突变子和对更多EVMs进行更广泛的评估。

# REFERENCES

# 参考文献

[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. Coveragebased 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. abs/1807.00182 (2018).CoRR

[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. abs/1903.08483 (2019).CoRR

[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. 10 (1998), 100–107.Digital TechnicalJournal

[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. abs/1802.06038 (2018).CoRR

[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. abs/1810.10310 (2018).CoRR

[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), 61–64.2018 IEEE/ACM 40thInternational Conference on Software Engineering: Companion (ICSE-Companion)

[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.