**EOSFuzzer: Fuzzing EOSIO Smart Contracts for Vulnerability Detection\*#**

**EOSFuzzer：用于EOSIO智能合约漏洞监测的模糊化测试**

**ABSTRACT**

**摘要**

EOSIO is one typical public blockchain platform. It is scalable in terms of transaction speeds and has a growing ecosystem supporting smart contracts and decentralized applications. However, the vulnerabilities within the EOSIO smart contracts have led to serious attacks, which caused serious financial loss to its end users. In this work, we systematically analyzed three typical EOSIO smart contract vulnerabilities and their related attacks. Then we presented EOSFuzzer, a general black-box fuzzing framework to detect vulnerabilities within EOSIO smart contracts. In particular, EOSFuzzer proposed effective attacking scenarios and test oracles for EOSIO smart contract fuzzing. Our fuzzing experiment on 3963 EOSIO smart contracts shows that EOSFuzzer is both effective and efficient to detect EOSIO smart contract vulnerabilities with high accuracy.

EOSIO是一个典型的公共区块链平台。它在事务速度方面是可伸缩的，并且有一个支持智能合约和分散应用程序的不断增长的生态系统。然而，EOSIO智能合约中的漏洞导致了严重的攻击，给终端用户造成了严重的经济损失。在这项工作中，我们系统地分析了三个典型的EOSIO智能合约漏洞及其相关攻击。然后，我们提出了EOSFuzzer，一个通用的黑盒模糊框架来检测EOSIO智能合约中的漏洞。特别地，EOSFuzzer为EOSIO智能合约模糊化提出了有效的攻击方案和测试预言。通过对3963个EOSIO智能合约的模糊化实验表明，EOSFuzzer能够高效、准确地检测出EOSIO智能合约的漏洞。

**CCS CONCEPTS**

**CCS 概念**

• → Software and application security;**Security and privacy**

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

•→软件和应用程序安全；**安全和隐私**

•→软件测试和调试；**软件及其工程**

**KEYWORDS**

**关键词**

Fuzzing, Smart contract, Vulnerability detection, Blockchain

模糊化、智能合约、漏洞检测、区块链

# 1 Introduction

# 1 简介

The blockchain technology is proposed as a value transfer network among peers with limited trust. Currently, the blockchain technology has evolved as a trust machine with which untrusted peers can cooperate with each other. Most of the current blockchain platforms supports smart contracts with which developers can build Decentralized Applications (DApps). Among them, the ecosystem of EOSIO platform is growing steadily. The types of DApps range from game, gambling, decentralized exchanges, tools, microblogging, etc.

区块链技术被认为是一种有限信任的节点间的价值传递网络。目前，区块链技术已经演变为一种信任机器，不受信任的对等方可以与对方合作。目前大多数区块链平台都支持智能合约，开发者可以通过智能合约构建分散应用程序（DAPP）。其中，EOSIO平台的生态系统正在稳步增长。dapp的类型包括游戏、赌博、分散交流、工具、微型博客等。

However, the vulnerabilities within EOSIO smart contracts have led to non-trivial financial loss to its end users. For example, the random number generation vulnerabilities within the EOSIO gambling games (including EOSBet, EOSCast, FFGame, EOSDice, EOSWin, etc.) have led to the loss of around 170K EOS tokens [28]. The Fake EOS Transfer vulnerability within the EOSCast smart contract has led to the loss of around 60K EOS tokens [26]. The Forged Transfer Notification vulnerability within EOSBet has led to the loss of 140K EOS tokens [27]. Based on the price of the EOS tokens (about $5 on average) at the time of the attack, the accumulated amount of loss by these three vulnerabilities alone was around 1.9 million worth of USD. Therefore, building effective vulnerability detection tools for EOSIO smart contracts is valuable. However, no existing fuzzing tool is available to detect EOSIO smart contract vulnerabilities.

然而，EOSIO智能合约中的漏洞给终端用户带来了不小的经济损失。例如，EOSIO赌博游戏（包括EOSBet、EOSCast、FFGame、EOSDice、EOSWin等）导致了大约170K个EOS代币的损失[28]。EOSCast智能合约中的虚假EOS传输漏洞导致了约6万个EOS代币的丢失[26]。EOSBet中的伪造传输通知漏洞导致了140K个EOS代币的丢失[27]。根据攻击发生时EOS代币的价格（平均约5美元），仅这三个漏洞造成的累计损失就约为190万美元。因此，为EOSIO智能合约构建有效的漏洞检测工具具有重要意义。但是，没有现存的模糊工具可用于检测EOSIO智能合约漏洞。

Different from the fuzzing of the Ethereum smart contracts, there are two main challenges with the fuzzing of EOSIO smart contracts. First, the EOSIO smart contracts adopt a unique EOS token transfer and notification mechanism through the eosio.token system smart contract, which can lead to distinctive EOSIO smart contracts vulnerabilities. To effectively trigger such vulnerabilities, we must design new attacker agents to trigger such vulnerabilities as well as new test oracles to precisely detect such vulnerabilities. Second, EOSIO smart contracts are executed by the WASM virtual machine, which supports different bytecode instruction sets from EVM. Furthermore, new types of vulnerabilities will continue to emerge with the development of EOSIO platform. Therefore, we must design the instrumentation scheme within the WASM VM in a general way to capture adequate information to support the detection of different types of EOSIO smart contract vulnerabilities.

与以太坊智能合约的模糊化不同，EOSIO智能合约的模糊化面临两大挑战。首先，EOSIO智能合约采用独特的机制，该机制通过eosio.token系统智能合约为EOS代币执行传输和通知，这可能导致独特的EOSIO智能合约漏洞。为了有效地触发此类漏洞，我们必须设计新的攻击者代理来触发此类漏洞，以及设计新的测试预言机来精确地检测此类漏洞。其次，EOSIO智能合约由WASM虚拟机执行，该虚拟机支持来自EVM的不同字节码指令集。此外，随着EOSIO平台的发展，新类型的漏洞将不断涌现。因此，我们必须以一种通用的方式在WASM-VM中设计检测方案，以捕获足够的信息来支持不同类型EOSIO智能合约漏洞的检测。

In this work, we proposed EOSFuzzer, a general black-box fuzzing framework to detect vulnerabilities within EOSIO smart contracts. EOSFuzzer includes a fuzzing input generator, a fuzzing executor, an instrumented WASM-VM, and a vulnerability detection engine. The fuzzing input generator can generate inputs based on the ABIs of the smart contracts and realize specific attacking scenarios with the agent smart contracts. The fuzzing executor can efficiently execute the generated inputs against the smart contract under fuzzing. The instrumented WASM-VM will collect opcode execution, API invocation and other execution information valuable for general vulnerability detection. Finally, the vulnerability detection engine implements the proposed test oracles to report vulnerabilities. Our experiment on 3963 smart contracts showed that EOSFuzzer is effective to detect vulnerabilities within EOSIO smart contracts with high accuracy. In particular, we have successfully mounted attacks on smart contracts without source code using EOSFuzzer to make a bet without spending any EOS in our case study.

在这项工作中，我们提出了EOSFuzzer，一个通用的黑盒模糊框架来检测EOSIO智能合约中的漏洞。EOSFuzzer包括一个模糊输入生成器、一个模糊执行器、一个插入指令的WASM-VM和一个漏洞检测引擎。模糊输入生成器可以根据智能合约的ABIs生成输入，并利用代理智能合约实现特定的攻击场景。模糊执行器能够有效地执行在模糊条件下针对智能合约生成的输入。插入指令的WASM-VM将收集操作码执行信息、API调用信息和其他对一般漏洞检测有价值的执行信息。最后，漏洞检测引擎实现所提出的测试预言来报告漏洞。我们在3963个智能合约上的实验表明，EOSFuzzer能够有效地检测出EOSIO智能合约中的漏洞，具有较高的准确率。特别是，在我们的案例研究中，我们已经成功地在没有源代码的情况下使用EOSFuzzer对智能合约进行攻击，从而能够在没有花费任何EOS币的情况下进行下注。

The contributions of this work are three-fold. First, to the best of our knowledge, this work proposes the first fuzzing framework for detecting security vulnerabilities within EOSIO smart contracts. Second, the work proposes effective attacking scenarios and the corresponding test oracles that can trigger and detect typical vulnerabilities within EOSIO smart contracts with high accuracy. Third, we have systematically evaluated EOSFuzzer with a fuzzing experiment on real world EOSIO smart contracts, and EOSFuzzer has effectively identified more than 450 vulnerabilities within EOSIO smart contracts.

这项工作的贡献有三个方面。首先，据我们所知，这项工作提出了第一个用于检测EOSIO智能合约中安全漏洞的模糊框架。其次，提出了有效的攻击场景和相应的测试预言，能够高精度地触发和检测EOSIO智能合约中的典型漏洞。第三，通过对现实世界EOSIO智能合约的模糊化实验，系统地评估了EOSFuzzer，EOSFuzzer有效地识别了EOSIO智能合约中的450多个漏洞。

The organization of the remaining sections is as follows. In Section 2, we will present the basics of Web Assembly (WASM), EOSIO platform and EOSIO smart contracts. Then in Section 3, we will present 3 typical vulnerabilities of EOSIO smart contracts. In section 4, we will present the design of our EOSFuzzer framework in detail. After that, we will perform a comprehensive experimental study to evaluate the effectiveness and efficiency of EOSFuzzer in terms of vulnerability detection in Section 5. Then we present case studies about successful attacks on EOSIO smart contracts with EOSFuzzer in Section 6. Finally, we present related works, and conclusions in Section 7, and 8.

其余各节的组织安排如下。在第2节中，我们将介绍Web -Assembly（WASM）、EOSIO平台和EOSIO智能合约的基础知识。然后在第3节中，我们将介绍EOSIO智能合约的3个典型漏洞。在第4节中，我们将详细介绍EOSFuzzer框架的设计。之后，我们将在第5节中进行一项综合实验研究，以评估EOSFuzzer在漏洞检测方面的有效性和效率。然后，我们在第6节介绍了成功攻击EOSIO智能合约的案例研究。最后，我们将在第7、和8节中介绍相关的工作和结论。

# 2 Background

# 2 背景

In this section, we provide background knowledge on Web-Assembly, EOSIO platform, and the EOSIO smart contracts.

在本节中，我们将提供有关Web-Assembly、EOSIO平台和EOSIO智能合约的背景知识。

## 2.1 The WebAssembly

## 2.1简称Wasm

WebAssembly (Wasm for short) [32] is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications. Wasm is designed to be fast, safe, and ease of debugging. And it can be easily embedded in both web and non-web execution environment. The EOSIO platform also adopts Wasm VM for executing its smart contract.

WebAssembly（简称Wasm）[32]是基于堆栈的虚拟机的二进制指令格式。Wasm被设计为一种可移植的目标，用于编译高级语言（如C/C++ / Rust），从而在Web上部署客户端和服务器应用程序。Wasm被设计为快速、安全和易于调试。它可以很容易地嵌入到web和非web执行环境中。EOSIO平台还采用Wasm-VM来执行智能合约。

## 2.2 EOSIO Platform and EOSIO Smart Contracts

## 2.2 EOSIO平台和EOSIO智能合约

EOSIO platform is an open source public blockchain platform that focuses on the scalability of transaction speed. The EOS is not only the token of EOSIO platform, but it also represents the stake hold by its owners. The EOSIO platform adopts the Delegated Proof of Stake (DPoS) protocol for making consensus, which is much more scalable than the Proof of Work (PoW) consensus protocol used by other platforms such as Bitcoin. The computing resources is also distributed according to the EOS tokens owned by the users. Moreover, EOSIO platform has set an upper limit on the execution time of each transaction. Exceeding the resource limit will lead to the rollback of the transaction with exceptions.

EOSIO平台是一个开源的公共区块链平台，关注交易速度的可扩展性。EOS不仅是EOSIO平台的代币，而且代表着其所有者所持有的股份。EOSIO平台采用委托股权证明（DPoS）协议来达成共识，这比比特币等其他平台使用的工作证明（PoW）共识协议具有更大的可扩展性。计算资源也根据用户拥有的EOS代币进行分配。此外，EOSIO平台对每个事务的执行时间设置了上限。超过资源限制将导致事务回滚，但是有零星的例外。

The features and characteristics of the EOSIO blockchain platform are designed to be flexible such that they can be modified to suit specific business requirement. Core blockchain features such as consensus, fee schedules, account creation and modification, token economics, etc., are implemented inside system smart contracts [30], which are deployed on the EOSIO blockchain platform. For example. The eosio.token system smart contract [24] defines the structures and actions allowing users to issue and transfer tokens on EOSIO based blockchains. The management of the EOS token is also performed by eosio.token system smart contract.

EOSIO区块链平台的特性和特点设计得非常灵活，可以进行修改以适应特定的业务需求。核心区块链功能，如共识、费用计划、账户创建和修改、代币经济等，在部署在EOSIO区块链平台上的系统智能合约内实现[30]。例如。这个eosio.token系统智能合约[24]定义了允许用户在基于EOSIO的区块链上发行和转让代币的结构和操作。EOS代币也由eosio.token系统智能合约管理。

Smart contracts are programs running on the blockchain platform to help manage the assets of end users or to enforce the negotiation and execution of agreements among peers participating in the blockchain platform. The smart contracts [29] deployed on EOSIO platform contain two parts: the WebAssembly bytecode and the Application Binary Interface (ABI). The smart contract source code is compiled into Wasm bytecode for execution within the Wasm VM. And the ABIs describe the public interfaces of the smart contract to interact with.

智能合约是在区块链平台上运行的程序，用于帮助管理最终用户的资产或参与区块链平台的对等方之间的协议的协商和执行。部署在EOSIO平台上的智能合约[29]包含两部分：WebAssembly字节码和应用程序二进制接口（ABI）。智能合约源代码被编译成Wasm字节码，以便在WASM-VM中执行。ABI描述了智能合约的公共接口以与之交互。

Every EOSIO smart contract must provide an apply function as the entrance function to handle actions. The apply function will listen to all incoming actions and invoke the corresponding action handler functions accordingly. For example, the transfer function of a smart contract is usually used to handle transfer actions related to the contract [31].

每个EOSIO智能合约都必须提供一个apply函数作为处理动作的入口函数。apply函数将侦听所有传入的操作，并调用相应的操作处理程序函数。例如，智能合约的传递函数通常用于处理与合约相关的传递行为[31]。

The apply function uses the receiver, code, and action input parameters as filters to map the actions to the corresponding functions to handle [25]. The receiver is the account currently executing code. The code is the account that the action was originally sent to. And the action is the name of the action. It is important to understand the difference between code and receiver. To be specific, code is always the first receiver of the action, while receiver is the account currently executing the action. During the execution of a smart contract it may forward the action received to other account with the require\_recipient() function. The code and action parameters of the apply function are also forwarded to the new contract. As a result, the receiver has changed to the newly notified contract, but the code stays the same.

apply函数使用receiver、code和action输入参数作为过滤器，将操作映射到要处理的相应函数[25]。receiver是当前正在执行代码的帐户。code是操作最初所发送到的帐户。action就是行为的名字。理解code和receiver之间的区别是很重要的。具体来说，code总是动作的第一个接收者，而receiver是当前执行动作的帐户。在执行智能合约期间，它可以使用require\_ recipient（）函数将收到的操作转发给其他帐户。apply函数的code和action参数也被转发到新合约中。因此，receiver已更改为新通知的合约，但code保持不变。

# 3 Security Bugs in EOSIO Smart Contracts

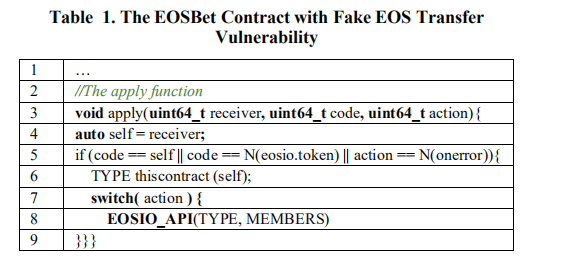
# 3 EOSIO智能合约中的安全漏洞

In this section, we will briefly review the security bugs in EOSIO smart contracts.

在本节中，我们将简要回顾EOSIO智能合约中的安全缺陷。

## 3.1 Fake EOS Transfer

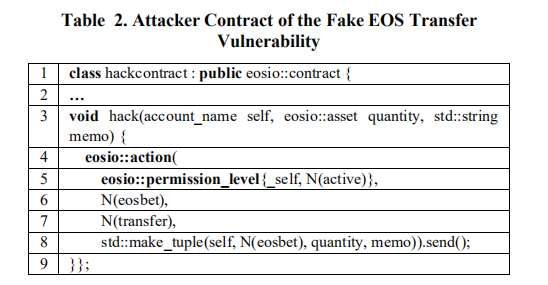
## 3.1虚假EOS传输

In the expected scenario, an EOSIO smart contract will only accept EOS transferred via the eosio.token system contract. If the contract under attack is vulnerable in that it does not check the code is eosio.token when the action is transfer within its apply function, an attacker may call the transfer function within the contract directly to fake an EOS transfer. As a result, the vulnerable contract may wrongly consider that the attacker has transferred EOS to it.

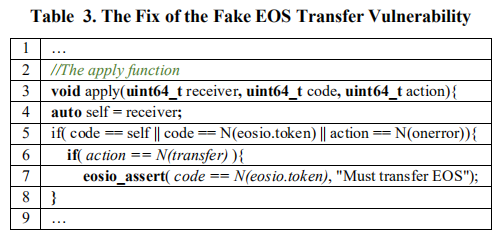
在预期的情况下，EOSIO智能合约将仅接受通过eosio.token系统合约传输的EOS。 如果受攻击的合约很容易被攻击成功，是因为在其apply函数中进行转移操作时，它不检查代码是否为eosio.token，则攻击者可以直接调用合约中的转移函数来伪造EOS转移。 结果，易受攻击的合约可能错误地认为攻击者已将EOS转移给了它。

As shown in Table 1, within the apply function of the EOSBet contract, it only checks whether the code is the contract itself or eosio.token, but it does not check whether the action transfer is originally sent to the eosio.token system smart contract. As a result, an attacker contract may directly call the transfer function of the vulnerable contract to make a bet without spending any EOS.

如表1所示，在EOSBet合约的apply函数中，它只检查代码是否是合约本身或者是否为eosio.token系统合约，但它不检查操作传输是否最初发送到eosio.token系统智能合约。因此，攻击者合约可以直接调用易受攻击合约的传递函数来下注，而无需花费任何EOS。



As shown in Table 2, a hacker can use the attacker contract to exploit the fake EOS transfer vulnerability. The attack is simple: it directly performs an inline call to the transfer function of EOSBet.

如表2所示，黑客可以使用攻击者合约来攻击假EOS传输漏洞。攻击很简单：它直接内联调用EOSBet的传递函数。

As shown in Table 3, The recommended way to fix the vulnerability is to add a check in the apply function to ensure eosio.token was the original receiver of the transfer action (line 6 and 7). In another word, when the action is transfer, the code must be eosio.token.

如表3所示，修复该漏洞的建议方法是在apply函数中添加一个检查，以确保eosio.token是转移行动的原始接收者（第6行和第7行）。换句话说，当动作是transfer时，代码必须是eosio.token.

## 3.2 Forged Transfer Notification

## 3.2伪造的传输通知

When the transfer function of a contract is called, a smart contract may never check the destination (i.e., to) of the transfer. However, this is vulnerable since the smart contract may just receive a notification of a transfer to another contract rather than to itself.

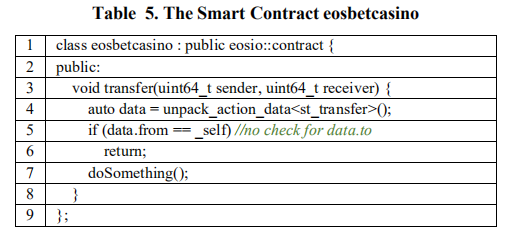
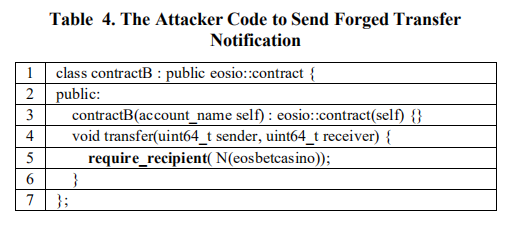
当调用合约的传递函数时，智能合约可能永远不会检查传递的目的地（即to）。然而，这是容易受到攻击的，因为智能合约可能只是收到一个转移到另一个合约的通知，而不是转移到自己的通知。

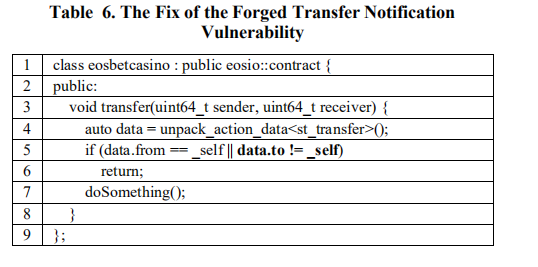
During a normal EOS transfer, a receiver contract can choose to forward the transfer notification to other accounts through require\_recipient (i.e. send a carbon copy of the transfer action). During a forged transfer notification attack [22], the attacker controls two accounts A and B. Then the attacker initializes the attack by transferring EOS from A to B through the system contract eosio.token. When the transfer is successful, both A and B will receive transfer notification. However, the contract deployed within account B can deliberately forwards the transfer notification to another contract (eosbetcasino in this example) with the require\_recipient function in order to mislead it as shown in Table 4.

在正常的EOS转账过程中，接收方合约可以选择通过require\_ recipient将转账通知转发给其他账户（即发送转账操作的副本）。在伪造的传输通知攻击[22]期间，攻击者控制两个帐户A和B。然后，攻击者通过系统合约将EOS从A传输到B来初始化攻击系统合约eosio.token. 当传输成功时，A和B都将收到传输通知。但是，部署在帐户B中的合约可以故意将转账通知转发给另一个带有require\_recipient()函数的合约（本例中为eosbecasino），以误导它，如表4所示。

As shown in Table 5, the eosbetcasino is vulnerable in that it does not check whether the destination (i.e., data.to) of EOS transfer is itself within its transfer function. Then, it may wrongly consider that it has received the EOS. As a result, it may wrongly credit the attacker account A, who in fact has sent nothing to eosbetcasino.

如表5所示，eosbetcasino易受攻击，因为它不检查在transfer函数中EOS的传递目的地（即data.to)是函数本身。然后，它可能会错误地认为它已经收到了EOS。因此，它可能会错误地记入攻击者A帐户，而攻击者实际上没有向eosbetcasino发送任何内容。

As shown in Table 6, the fix of the forged transfer notification vulnerability is to simply check the destination of the transfer (line 5). If the destination of the EOS transfer is not the current contract, then the contract should directly ignore the transfer notification.



如表6所示，修复伪传输通知漏洞的方法是简单地检查传输的目的地（第5行）。如果EOS传输的目的地不是当前合同，则合同应直接忽略传输通知。

## 3.3 Block Information Dependency

## 3.3块信息依赖

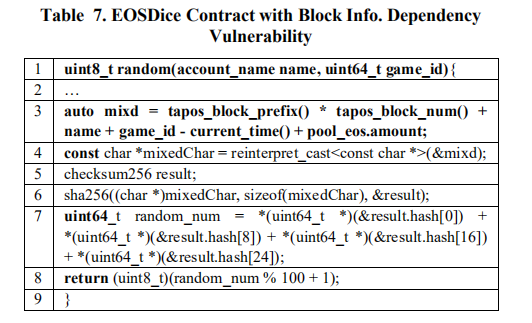
Due to the lack of source of randomness, an EOSIO smart contract may rely on the block information such as tapos\_block\_prefix and tapos\_block\_num to generate random numbers, which may in turn determine the transfer of EOS or the winner of a lottery. However, the tapos\_block\_prefix and tapos\_block\_num are not reliable source of randomness, because they can be directly calculated from ref\_block\_num, which is the id of the last irreversible block by default.

由于缺乏随机性来源，EOSIO智能合约可能依赖诸如tapos\_block\_ prefix和tapos\_block\_ num之类的块信息来生成随机数，而随机数又可能决定EOS的转移或彩票的中奖者。但是，tapos\_ block\_prefix 和tapos\_ block\_ num不是可靠的随机性来源，因为它们可以直接从ref\_ block\_ num计算，默认情况下，ref\_ block\_ num是最后一个不可逆块的id。

A gambling contract may use deferred action to determine the winner of a lottery. In such scenario, the reference block is the block just before the block making the bet. Therefore, when a smart contract uses tapos\_block\_prefix and tapos\_block\_num directly for random generation, the random number generated can be predicted.

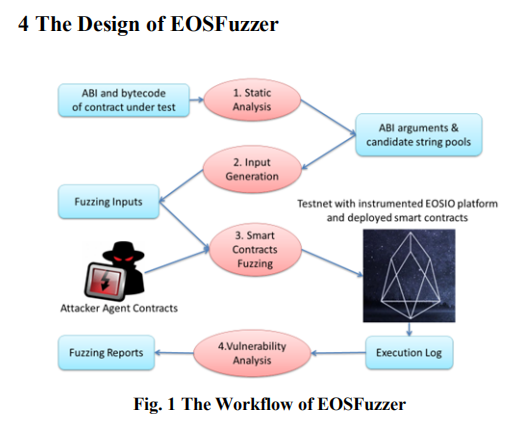
赌博合约可以使用延期行为来确定彩票的中奖者。在这种情况下，参考区块就是下注区块之前的区块。因此，当智能合约直接使用tapos\_ block\_prefix和tapos\_ block\_ num进行随机生成时，生成的随机数是可以预测的。

As shown in Table 7, the EOSDice smart contract [28] uses the tapos\_block\_prefix, tapos\_block\_num, current\_time, account\_name, game\_id, pool\_eos for random number generation. Unfortunately, it turns out all these variables can be determined before taking the bet. The account\_name (name of the contract), game\_id, and pool\_eos (balance of the current contract) are trivial to get. The current\_time refers to the timestamp when determine the winner of the lottery, which is the sum of the timestamp of making the bet and the delay time of the deferred action. As a result, all variables used for random number generation can be computed before making the bet. Finally, the attackers successfully calculated the random number and won 2,545 EOS through the lottery, which was about $13,500.

如表7所示，EOSDice智能合约[28]使用tapos\_ block\_ prefix、tapos\_ block\_ num、current\_ time、account\_ name、game\_ id、pool\_ eos生成随机数。不幸的是，所有这些变量都可以在下注前确定。帐户名（合同名称）、游戏id和奖池EOS（当前合同的余额）很容易得到。当前时间是指确定彩票中奖者的时间戳，即下注时间戳与延迟动作延迟时间之和。因此，可以在下注之前计算用于生成随机数的所有变量。最后，攻击者成功计算出随机数，并通过彩票中奖2545个EOS，奖金约为13500美元。

# 4 The Design of EOSFuzzer

# 4 EOS模糊器的设计



In this section, we will first present an overview of the EOSFuzzer. Then we will discuss each component of the EOSFuzzer in detail.

在本节中，我们将首先概述EOSFuzzer。然后我们将详细讨论EOSFuzzer的每个组件。

## 4.1 Overview of EOSFuzzer

## 4.1 EOS模糊器概述

As shown in Fig. 1, the general workflow of EOSFuzzer mainly consists four steps. First, EOSFuzzer performs static analysis on the ABI (application binary interface) and the bytecode of the smart contract under test. The static analysis step outputs the data types of the arguments for each ABI interface as well as a candidate string pool for string data type and memo parameter. Second, the EOSFuzzer performs input generation to generate fuzzing inputs for each ABI interfaces based on the static analysis result. Meanwhile, the EOSFuzzer will also use deployed attacker agent contracts to generate attacking scenarios for triggering specific vulnerabilities. Third, the EOSFuzzer performs fuzzing on the smart contract under analysis within the testnet through the cleos tool [23]. Note that the testnet is an instrumented EOSIO platform with smart contracts deployed on it. Through instrumentation, the execution information of the smart contract under analysis will be recorded into the execution log. Fourth, the EOSFuzzer will perform vulnerability analysis based on our proposed test oracles.

如图1所示，EOSFuzzer的一般工作流程主要包括四个步骤。首先，EOSFuzzer对被测智能合约的ABI（应用程序二进制接口）和字节码进行静态分析。静态分析步骤输出每个ABI接口的参数的数据类型以及字符串数据类型和备注参数的候选字符串池。其次，EOSFuzzer执行输入生成，根据静态分析结果为每个ABI接口生成模糊输入。同时，EOSFuzzer还将使用部署的攻击者代理合约来生成触发特定漏洞的攻击场景。第三，EOSFuzzer通过cleos工具对testnet中分析的智能合约执行模糊化[23]。注意，testnet是一个部署了智能合约的工具化EOSIO平台。通过插装，被分析的智能合约的执行信息将被记录到执行日志中。第四，EOSFuzzer将根据我们提出的测试预言进行漏洞分析。

## 4.2 The Input Generator

## 4.2输入生成器

The input generator is responsible for generating the inputs for the ABI interfaces of EOSIO smart contracts as well as generating the attacking scenarios with the attacker agents. The final inputs for fuzzing are the interleaved ABI invocations and the attack scenarios.

输入生成器负责为EOSIO智能合约的ABI接口生成输入，并与攻击者代理一起生成攻击场景。模糊化的最终输入是交错的ABI调用和攻击场景。

Generating Inputs for Different Data Types. The ABI interfaces for EOSIO smart contracts are in JSON format. Therefore, our tool parses the JSON file to extract the data type for each parameter. Then EOSFuzzer generates inputs for each data type and concatenates them to build the final input for invoking the ABI interfaces.

为不同的数据类型生成输入.。EOSIO智能合约的ABI接口是JSON格式的。因此，我们的工具解析JSON文件以提取每个参数的数据类型。然后，EOSFuzzer为每个数据类型生成输入，并将它们连接起来，以构建调用ABI接口的最终输入。

The primitive numeric data types include the 8-, 16-, 32-, and 64bit integer and unsigned integer types as well as the 32- and 64-bit float-point types. For those data types, EOSFuzzer can randomly generate the maximum, or the minimum, or a random value within the input domain.

基本数字数据类型包括8位、16位、32位和64位整数、无符号整数类型以及32位和64位浮点类型。对于这些数据类型，EOSFuzzer可以在输入域中随机生成最大值、最小值或随机值。

For the Boolean data type, the input generation algorithm will random return true (1) or false (0) values. For the string data type, our input generation algorithm will first build a candidate pool of strings with static analysis. More specifically, EOSFuzzer will iterate the data areas defining the constant data and collect all constant strings in the area as a pool of candidate strings. Then the EOSFuzzer will randomly return a string from the candidate pool for a string parameter during input generation.

对于布尔数据类型，输入生成算法将随机返回真（1）或假（0）值。对于字符串数据类型，我们的输入生成算法将首先通过静态分析构建一个候选字符串池。更具体地说，EOSFuzzer将迭代定义常量数据的数据区域，并将该区域中的所有常量字符串收集为候选字符串池。然后，EOSFuzzer将在输入生成期间从候选池中随机返回字符串参数。

For the asset data type, the algorithm will return random number of EOS tokens. For the symbol data type, the algorithm will only return the type EOS. In another word, we will mainly target at smart contract functions related to EOS token rather than other tokens.

对于资产数据类型，算法将返回随机数目的EOS代币。对于符号数据类型，算法只返回EOS类型。换言之，我们将主要针对与EOS代币相关的智能合约功能，而不是其他代币。

The name data type represents the name of smart contracts. Currently, our tool will return the name of the smart contract under test as input for the parameter of name data type. Similarly, for the public\_key data type, we will return the address of the current smart contract under test. This is because many of the ABIs are only authorized to work on the current smart contract under test. The use of the name and public\_key of other relevant smart contracts may further require the analysis of the relationships among smart contracts within a project, which we left as a future work.

name数据类型表示智能合约的名称。目前，我们的工具将返回被测智能合约的名称作为name数据类型参数的输入。类似地，对于public\_key数据类型，我们将返回当前被测智能合约的地址。这是因为许多abi只被授权处理当前正在测试的智能合约。使用其他相关智能合约的名称和公钥可能需要进一步分析项目中智能合约之间的关系，我们将此作为未来的工作。

For the array data type, our algorithm will iteratively generate inputs for its members of primitive data type. For struct data type, our algorithm will iteratively generate inputs for each of its members. If its member is of struct data type, the algorithm will also proceed recursively until primitive types are encountered.

对于数组数据类型，我们的算法将迭代地为其原始数据类型的成员生成输入。对于struct数据类型，我们的算法将迭代地为其每个成员生成输入。如果其成员是struct数据类型，则算法也将递归进行，直到遇到基本类型。

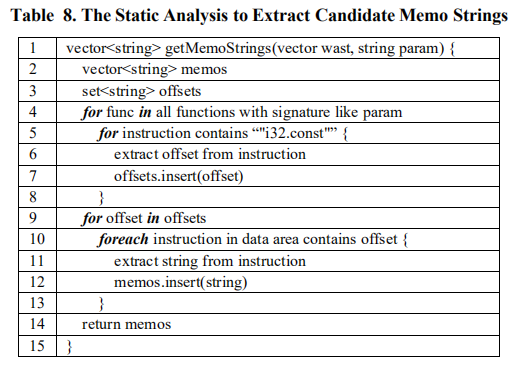
For each smart contract, the input generation algorithm will generate one test suite containing a set of test cases. Each test case is a combination of ABI function invocations and attacking interactions (from the agent contracts) to the smart contract under fuzzing. The number of function invocations and attacking interactions within a test case is of random length and can be configured within EOSFuzzer.

对于每个智能合约，输入生成算法将生成一个包含一组测试用例的测试套件。每个测试用例都是ABI函数调用和攻击交互（从代理合约）到模糊下的智能合约的组合。在一个测试用例中，函数调用和攻击交互的数量是随机长度的，可以在EOSFuzzer中配置。

Generating Seeds for the Memo Parameter. The memo argument is an argument used in the transfer function. Based on our analysis, we found that the sender and the receiver may use a memo argument of string type to confirm and guard a transfer action. To generate seeds for the memo parameter, our algorithm will perform static analysis on the smart contract under test to extract the strings used in the transfer function.

为Memo参数生成种子。 memo参数是传递函数中使用的参数。基于我们的分析，我们发现发送方和接收方可能使用字符串类型的memo参数来确认和保护传输操作。为了生成memo参数的种子，我们的算法将对测试中的智能合约进行静态分析，以提取传递函数中使用的字符串。

Table 8. The Static Analysis to Extract Candidate Memo Strings

表8。提取候选Memo字符串的静态分析

As shown in Table 8, the function getMemoStrings extracts the candidate strings that may be used to compare with the memo argument within the transfer function. The wast parameter is the WebAssembly text format of the smart contract under fuzzing. EOSFuzzer uses the wasm2wat tool [33] to translate wasm bytecode to wast representation. The param is a string representing the data types of the return value and parameters of the transfer function (i.e., the string “(param i32 i64 i64 i32 i32)”). At line 4, the algorithm iterates all functions with signatures like the transfer function (i.e., having the same type of parameters and return value). Within each function, it iterates each instruction containing the use of constant values in the code (i.e., containing i32.const), and it extracts the offset of the corresponding constant variable in such functions (line 4 to 7). Finally, the algorithm iterates the data areas defining the constant data, locates the string corresponding to the offset (if it is of type string), and saves it in the vector memos as candidates for the memo parameter (line 9 to 12).

如表8所示，函数getMemoStrings提取候选字符串，这些字符串可用于与传递函数中的memo参数进行比较。wast参数是模糊化下智能合约的WebAssembly文本格式。EOSFuzzer使用wasm2wat工具[33]将wasm字节码转换为wast表示。param是表示传递函数的返回值和参数的数据类型的字符串（即字符串“（param i32 i64 i64 i32 i32）”。在第4行，算法迭代所有具有传递函数（即具有相同类型的参数和返回值）等签名的函数。在每个函数中，它迭代包含代码中常量值用法的每个指令（即，包含i32.const），并提取此类函数中相应常量变量的偏移量（第4行到第7行）。最后，该算法迭代定义常量数据的数据区域，定位与偏移相对应的字符串（如果它是string类型），并将其保存在向量memos中，作为memo参数的候选（第9行到第12行）。

When there are functions with the same signature as the transfer function or when there are other constant strings used in the transfer function which are not related with memo parameter, the analysis may become imprecise. However, our analysis with the open source contracts showed that the memos vector only contains small number of irrelevant candidates, which is still effective for fuzzing.

当存在与传递函数具有相同签名的函数或传递函数中使用的其他常量字符串与memo参数无关时，分析可能变得不精确。然而，我们对开源合约的分析表明，memos向量只包含少量不相关的候选信息，这对于模糊化仍然有效。

The Design of Attacker Agents for Faked EOS Transfer. A safely written smart contract will check the code is eosio.token when the action is transfer within the apply function. Therefore, to trigger the fake EOS transfer vulnerability, we used an attacker contract called fakeTransferAgent to invoke the transfer function of the smart contract under fuzzing with code different from eosio.token. The invocation involves two scenarios and both are performed within the transfer function of the fakeTransferAgent. One is to invoke the transfer function with code equal to the receiver (i.e. the contract under fuzzing) while the other is to invoke the transfer function with code different from both the receiver and eosio.token. As a result, the agent contract will perform two possible attacks on the smart contract under fuzzing.

伪造EOS传输的攻击代理设计. 当action在apply函数中传递时，安全编写的智能合约将检查代码是否为eosio.token。因此，为了触发假EOS传输漏洞，我们使用了一个名为fakeTransferAgent的攻击者合约，在来自eosio.token的不同代码的模糊化下调用智能合约的传输函数. 调用涉及两种场景，都是在faketTransferAgent的传递函数中执行的。一种是用与接收者相同的代码（即模糊化下的合约）调用传递函数，另一种是用与接收者和eosio.token不同的代码调用传递函数. 因此，在模糊化条件下，代理合约将对智能合约执行两种可能的攻击。

One attack is to perform an inline function call to the transfer function of the smart contract under fuzzing. In another word, our tool deliberately avoids using the eosio.token contract to perform the transfer action. In this attack, the code is equal to receiver (i.e., the smart contract under fuzzing).

一种攻击是在模糊化下对智能合约的传递函数执行内联函数调用。换句话说，我们的工具故意避免使用eosio.token执行转移操作的合约。在这种攻击中，代码等于接收者（即模糊下的智能合约）。

The other kind of attack is to first use the cleos tool to call the transfer function of the fakeTransferAgent smart contract. Then, the fakeTransferAgent smart contract will forward the action to the smart contract under fuzzing through require\_recipient. In this way, the smart contract under fuzzing will receive an transfer action with code equal to fakeTransferAgent, which is different from both eosio.token and itself (the receiver).

另一种攻击是首先使用cleos工具调用fakettransferagent智能合约的传递函数。然后，fakettransferagent智能合约通过require\_ recipient将动作转发给模糊智能合约。这样，模糊化下的智能合约将收到一个代码等于fakettransferagent的转移动作，这与eosio.token以及它本身（接收器）不同。

Finally, if the transfer function of smart contract under fuzzing is successfully triggered by the attack, it will be considered vulnerable.

最后，如果攻击成功地触发了模糊智能合约的传递函数，则认为它是易受攻击的。

The Design of the Attacker Agents for Forged Transfer Notification. To effectively trigger the forged transfer notification vulnerability, we designed a notifier agent contract to mount the attack. To initiate the attack, EOSFuzzer will first send some EOS to the notifier contract through the eosio.token smart contract. The notifier contract in turn will forward the transfer action to the smart contract under fuzzing through the require\_recipient() function. Finally, the vulnerability detection module of EOSFuzzer will analyze the behavior of the smart contract under fuzzing on receiving the notification to determine whether it is vulnerable or not.

伪造传输通知攻击代理的设计. 为了有效地触发伪造的传输通知漏洞，我们设计了一个通知代理合约来装载攻击。为了发起攻击，EOSFuzzer将首先通过eosio.token智能合约。通知者合约将通过require\_recipient（）函数将转移操作转发到模糊的智能合约。最后，EOSFuzzer的漏洞检测模块将对智能合约在收到通知时的行为进行分析，判断其是否存在漏洞。

## 4.3 The Fuzzing Execution Engine

## 4.3模糊执行引擎

Having generated the inputs for fuzzing the smart contracts, the fuzzing execution engine is responsible for executing the inputs against the EOSIO smart contract. The fuzzing execution engine makes use of the cleos tool to interact with the smart contract. Within our design, the EOSFuzzer creates a pipe to communicate with the cleos process. It invokes the push action command of cleos tool to interact with the ABI interfaces of the smart contract under fuzzing or with the agent contracts.

在生成用于模糊智能合约的输入之后，模糊执行引擎负责针对EOSIO智能合约执行输入。模糊执行引擎利用cleos工具与智能合约进行交互。在我们的设计中，EOSFuzzer创建了一个管道来与cleos进程通信。它调用cleos工具的push-action命令与模糊智能合约的ABI接口或代理合约进行交互。

For each smart contract, the input generation algorithm will generate a set of test cases. Each test case is a combination of ABI function invocations and attacking interactions (from the agent contracts) to the smart contract under fuzzing. When a test case finishes execution, EOSFuzzer will reset the testnet. On the one hand, resetting the testnet will lead to extra time cost. On the other hand, resetting the testnet will make the debugging easier since the execution trace is shorter. In this way, we want to make a tradeoff between fuzzing efficiency and debuggability after fuzzing.

对于每个智能合约，输入生成算法将生成一组测试用例。每个测试用例都是ABI函数调用和攻击交互（从代理合约）到模糊化下的智能合约的组合。当一个测试用例完成执行时，EOSFuzzer将重置testnet。一方面，重置testnet将导致额外的时间开销。另一方面，重置testnet将使调试更容易，因为执行跟踪更短。通过这种方式，我们希望在模糊化效率和模糊化后的可调试性之间进行权衡。

When the execution of test cases ends, EOSFuzzer will start reading the instrumentation logs generated during execution and invoke the vulnerability detection module to analyze and report the vulnerability detection results.

当测试用例的执行结束时，EOSFuzzer将开始读取执行过程中生成的插装日志，并调用漏洞检测模块分析并报告漏洞检测结果。

## 4.4 Wasm VM Instrumentation

## 4.4 Wasm VM工具

To support the extensible analysis of vulnerabilities, we must carefully instrument the Wasm VM to collect information about opcodes and operands invoked during execution. This includes those control-flow related opcode, the unary opcode, the binary opcode, and their operands. Whenever an opcode is executed, the instrumentation will generate a line recording the opcode, the operands, and the results (if there is any) within the opcode log file. Although not all information within the opcode log file are used for detecting the vulnerabilities studied in this work, the instrumentation log is design to be general for the analysis of new vulnerabilities in the future.

为了支持对漏洞的可扩展分析，我们必须仔细地检测Wasm VM，以收集有关在执行期间调用的操作码和操作数的信息。这包括那些与控制流相关的操作码、一元操作码、二进制操作码及其操作数。每当执行一个操作码时，工具将生成一行，记录操作码、操作数和操作码日志文件中的结果（如果有）。虽然并非操作码日志文件中的所有信息都用于检测本工作中研究的漏洞，但检测日志的设计是通用的，用于将来分析新的漏洞。

In particular, the CallIndirect opcode is instrumented because each invocation of the execute\_action within the apply function will lead to the execution of CallIndirect opcode. Therefore, this opcode implies the invocation of an ABI function within the smart contract under test. Furthermore, we have also instrumented the Wasm API interface implementation code to collect information on whether the block information is queried during smart contract execution. To be specific, we have instrumented the tapos\_block\_num() and the tapos\_block\_prefix() functions.

特别是，检测CallIndirect操作码，因为apply函数中每次调用execute\_action都会导致执行CallIndirect操作码。因此，此操作码意味着在被测智能合约中调用ABI函数。此外，我们还检测了Wasm API接口实现代码，以收集有关在智能合约执行期间是否查询块信息的信息。具体来说，我们已经插入了tapos\_ block\_ num（）和tapos\_block\_prefix（）函数。

Finally, we have also instrumented the transfer function of eosio.token system smart contract, which is used in EOSIO platform for EOS transfer. Instrumenting this function will help us understand whether EOS transfer actually happened during smart contract execution.

最后，我们还检测了eosio.token系统智能合约，用于EOSIO平台的EOS传输。检测此函数将有助于我们了解EOS传输是否在智能合约执行期间实际发生。

## 4.5 Test Oracles for Vulnerability Detection

## 4.5漏洞检测预言机

In this section, we present the test oracles for vulnerability detection.

在本节中，我们将介绍用于漏洞检测的测试预言。

Fake EOS Transfer. To trigger the fake EOS transfer vulnerability, we used an attacker contract called fakeTransferAgent to perform two possible actions within its own transfer function. One is to directly perform an inline function call to the transfer function of the smart contract under fuzzing while the other is to forward a transfer action to the smart contract under fuzzing with require\_recipient.

假EOS传输。为了触发假EOS传输漏洞，我们使用了一个名为FaketTransferAgent的攻击者合约在其自身的传输函数中执行两个可能的操作。一种是直接对模糊化下的智能合约的传递函数进行内联函数调用，另一种是将传递动作转发给带有require\_recipient的模糊化下的智能合约。

The test oracle to detect the fake EOS transfer vulnerability under the designed attacking scenarios is:

在设计的攻击场景下检测假EOS传输漏洞的测试方法是：

CanReceiveEOS & TransferCalled

The CanReceiveEOS test oracle is to check whether the smart contract can receive EOS rather than other tokens. During our implementation, we send some EOS to the smart contract under fuzzing through the eosio.token contract, and then we check whether the transfer function of the smart contract is invoked. Because at least two member functions will be called during the attacking scenario (i.e., the transfer function of eosio.token and the transfer function of the smart contract under fuzzing), EOSFuzzer will check whether the CallIndirect opcode has been executed at least 2 times within its implementation accordingly.

CanReceiveEOS测试预言机检查智能合约是否可以接收EOS而不是其他代币。在我们的实现过程中，我们将一些状态方程通过eosio.token然后检查是否调用了智能合约的传递函数。因为在攻击场景中至少会调用两个成员函数（即eosio.token模糊化下的智能合约的传递函数），模糊器将相应地检查CallIndirect操作码在其实现中是否至少执行了2次。

The TransferCalled test oracle is to check whether the transfer function of the smart contract under fuzzing can be successfully triggered by the fakeTransferAgent. Similar to CanReceiveEOS, EOSFuzzer checks whether the CallIndirect opcode has been executed for at least 2 times (i.e., one for the transfer function of fakeTransferAgent and the other for the transfer function of the smart contract under fuzzing). Since the attacking scenario is well controlled by our agent contract, we can be sure that the transfer function is the only ABI function invoked in the smart contract under fuzzing. Therefore, the appearance of the second CallIndirect opcode corresponds to the invocation of the transfer function rather than other ABI functions within the smart contract under fuzzing.

transfertest-oracle是检查fakettransferagent是否能成功触发模糊智能合约的传递函数。与CanReceiveEOS类似，EOSFuzzer检查CallIndirect操作码是否至少执行了2次（即一次用于fakettransferagent的传递函数，另一次用于fuzzing下智能合约的传递函数）。由于我们的代理合约很好地控制了攻击场景，因此可以确定在模糊化条件下，传递函数是智能合约中唯一调用的ABI函数。因此，第二个CallIndirect操作码的出现对应于传递函数的调用，而不是在模糊化条件下智能合约中的其他ABI函数。

Forged Transfer Notification. As presented in previous sections, to detect forged transfer notification vulnerability, the EOSFuzzer makes use of a notifier agent to send a forged notification to the smart contract under fuzzing.

伪造转让通知书。如前几节所述，为了检测伪造的传输通知漏洞，EOSFuzzer在模糊化下使用通知代理向智能合约发送伪造的通知。

The test oracles to detect forged transfer notification is as follows:

检测伪造传输通知的测试预言器如下：

TransferCalled & !CheckRecipient

The TransferCalled sub-oracle is to check whether the transfer function of the smart contract under fuzzing is invoked during the attacking scenario. Within the call chain of our attack scenarios, if the transfer function of the smart contract is called, at least three ABI functions must be invoked. These corresponds to the transfer functions of eosio.token contract, the notifier contract, and the smart contract under fuzzing. Each invocation of a member function will lead to the execution of a CallIndirect opcode. Therefore, when realizing the oracle TransferCalled, EOSFuzzer checks whether there are at least 3 CallIndirect opcode executed during our attack. At the same time, EOSFuzzer also records the line number of the third CallIndirect opcode within the opcode log file for ease of further analysis, which corresponds to the starting position of the transfer function within the smart contract under fuzzing.

TransferCalled 子预言机检查在攻击场景中是否调用了模糊化下的智能合约的传递函数。在我们攻击场景的调用链中，如果调用智能合约的传递函数，则至少必须调用三个ABI函数。这些对应于eosio.token模糊合约、通知合约和智能合约。对成员函数的每次调用都将导致CallIndirect操作码的执行。因此，在实现TransferCalled时，EOSFuzzer会检查在我们的攻击过程中是否至少执行了3个CallIndirect操作码。同时，EOSFuzzer还将第三个CallIndirect操作码的行号记录在操作码日志文件中，便于进一步分析，对应于模糊化下智能合约中传递函数的起始位置。

The CheckRecipient condition is to check the smart contract under fuzzing has not checked whether it is the real recipient of the EOS transfer in all attacks by the notifier agent. To realize the suboracle !CheckRecipient, EOSFuzzer first checks whether there is any execution of comparison opcode (i.e., Eq or Ne) between the notifier contract (the actual receiver of EOS) and the smart contract under fuzzing in one round of attack by the notifier agent. The comparison starts from the starting position of the transfer function of the smart contract under fuzzing recorded in the opcode log file during the previous step. It should be noted that directly checking the instruction on comparing the to and \_self parameters of the transfer function may be imprecise because EOSIO does not enforce the consistency between the ABI interface and the actual implementation.

CheckRecipient条件是检查模糊化下的智能合约有没有检查它是否是EOS传输中所有攻击的真实接收者。去实现子预言！CheckRecipient，EOSFuzzer首先检查通知代理在一轮攻击中，在模糊化下，通知代理（EOS的实际接收者）与智能合约之间是否执行了比较操作码（Eq或Ne）。比较从上一步操作码日志文件中记录的模糊化下的智能合约传递函数的起始位置开始。需要注意的是，直接检查关于比较传递函数的to和\_self参数的指令可能是不精确的，因为EOSIO没有加强ABI接口和实际实现之间的一致性。

During our fuzzing process for a smart contract, EOSFuzzer will initiate the attack from the notifier agent many times, the !CheckRecipient condition is satisfied only when the check of the EOS transfer recipient is not performed in all attacks from the notifier agent. Since a Fuzzer may not traverse each path to the code for recipient checking, EOSFuzzer may produce false positives when reporting Forged Transfer Notification vulnerability. However, when the scale of the fuzzing is large, the probability of generating false positives will become very low, which is also confirmed by our experiment evaluation.

在我们的智能合约模糊化过程中，EOSFuzzer会多次从通知代理发起攻击！只有在notifier agent的所有攻击中未执行对EOS传输接收者的检查时，才满足CheckRecipient条件。由于Fuzzer可能不会遍历到用于接收者检查的代码的每条路径，因此在报告伪造的传输通知漏洞时，Fuzzer可能会产生误报。然而，当模糊化的规模较大时，产生误报的概率会很低，这也得到了实验评估的证实。

Block Information Dependency. The test oracle to detect block information dependency is as follows:

块信息相关性. 检测块信息相关性的测试步骤如下：

BlockInfoRead & EOSTransferCalled

The BlockInfoRead test oracle checks whether there is any invocation of APIs reading block information. During the implementation, this is realized through instrumenting the tapos\_block\_num and tapos\_block\_prefix API. The EOSTransferCalled oracle checks whether the transfer function of the eosio.token smart contract is called. The EOSFuzzer realizes this by simply instrumenting the transfer function of eosio.token system smart contract and re-deploying it within EOSIO.

BlockInfoRead测试预言机检查是否有任何api调用读取块信息。在实现过程中，这是通过检测tapos\_block\_num和tapos\_block\_prefix API来实现的。EOSTransferCalled预言机检查是否来自eosio.token智能合约的传输方法被调用，EOSFuzzer通过简单地检测eosio.token系统智能合约并在EOSIO中重新部署。

## 4.5 Discussions

## 4.5讨论

In this section, we discuss the possible extensions to EOSFuzzer in two aspects.

在本节中，我们将从两个方面讨论EOSFuzzer的可能扩展。

First, EOSFuzzer is designed as a blackbox fuzzer, but it can be also extended to be a grey-box fuzzer. The key to the extension is to collect code coverage information during fuzzing process. Currently, EOSFuzzer instruments all kinds of bytecode instructions within WASM VM. We can extend EOSFuzzer by collecting branch coverage information within VM instrumentation: whenever a branch instruction is encountered, we can log the location of the two branches as well as the coverage of new branch. The incremental branch coverage information can be used to guide the mutation-based grey-box fuzzing process. New mutation and selection strategies can be proposed to further improve the effectiveness of the fuzzer.

首先，EOSFuzzer被设计成黑盒模糊器，但它也可以扩展成灰盒模糊器。扩展的关键是在模糊化过程中收集代码覆盖信息。目前，EOSFuzzer在WASM VM中插入了各种字节码指令。我们可以通过在VM工具中收集分支覆盖信息来扩展EOSFuzzer：每当遇到分支指令时，我们都可以记录两个分支的位置以及新分支的覆盖范围。增量的分支覆盖信息可以用来指导基于变异的灰盒模糊化过程。可以提出新的变异和选择策略来进一步提高模糊器的有效性。

# 5 Experiment and Results Analysis

# 5实验及结果分析

In this section, we present our fuzzing experiment to evaluate the vulnerability detection effectiveness of the EOSFuzzer.

在本节中，我们将介绍我们的模糊化实验，以评估EOSFuzzer的漏洞检测效果。

## 5.1 Subject Programs

## 5.1实验研究

The subject programs used in our experimental study include 82 open source EOSIO smart contracts and 3881 smart contracts without source code. The 82 open source smart contracts are mainly used to manually verify the effectiveness of our tool. However, collecting large number of EOSIO smart contracts with source code is difficult because most EOSIO DApps are not open-source. The 3881 smart contracts without source code contain Wasm bytecode and ABI file only, which are used to evaluate our tool at scale. For the smart contracts with source code, we first compiled them into bytecode and then deployed them in our testnet for fuzzing. For the smart contracts with Wasm bytecode only, we directly deployed them within our testnet.

实验研究中使用的主题程序包括82个开源EOSIO智能合约和3881个无源代码智能合约。82个开源智能合约主要用于手动验证我们工具的有效性。然而，收集大量带有源代码的EOSIO智能合约是困难的，因为大多数eosiodapp都不是开源的。3881个没有源代码的智能合约只包含Wasm字节码和ABI文件，它们用于在规模上评估我们的工具。对于带有源代码的智能合约，我们首先将它们编译成字节码，然后将它们部署在我们的测试网中进行模糊化。对于只使用Wasm字节码的智能合约，我们直接将它们部署在我们的测试网中。

## 5.2 Experiment Setup

## 5.2实验装置

The experiment was performed within a docker. The host for the docker was a desktop equipped with 8 cores of Intel® Core™ i76700 CPU @ 3.4GHz and 16GB of memory. The host was running Ubuntu 16.04.1 and Docker version 18.06.1. Furthermore, the docker was running Ubuntu 18.04.4. We ran the instrumented EOSIO client (node) and the EOSFuzzer within the Ubuntu OS in the docker. The EOSIO client used is the nodeos version 1.5.2. And the EOSFuzzer talks to the EOSIO node through the cleos tool.

这个实验是在一个docker容器内进行的。docker的主机是一台配备8核Intel®Core的台式机™ i76700 [CPU@3.4GHz和16GB内存。主机运行的是ubuntu16.04.1和Docker版本18.06.1。此外，docker运行的是Ubuntu18.04.4。我们在docker中运行了Ubuntu操作系统中插入指令的EOSIO客户机（node）和EOSFuzzer。使用的EOSIO客户端是nodeos版本1.5.2。EOSFuzzer通过cleos工具与EOSIO节点通信。](mailto:CPU@3.4GHz和16GB内存。主机运行的是ubuntu16.04.1和Docker版本18.06.1。此外，docker运行的是Ubuntu18.04.4。我们在docker中运行了Ubuntu操作系统中插入指令的EOSIO客户机（node）和EOSFuzzer。使用的EOSIO客户端是nodeos版本1.5.2。EOSFuzzer通过cleos工具与EOSIO节点通信。)

Before performing the fuzzing experiment, we must properly configure the EOSIO testnet. First, we started the keosd wallet to provide key manager service daemon for storing private keys and signing digital messages. Second, we ran the instrumented nodeos daemon to start the single node testnet. Third, we deployed the instrumented eosio.token system smart contract and the attacker agent contracts. Fourth, we created 2 EOSIO accounts to cooperate with the attacker agent contracts for performing EOS transfer during attack. Finally, we deploy the smart contracts under fuzzing on the EOSIO testnet. The deployed smart contracts include 3881 smart contracts without source code and 82 smart contracts with source code.

在执行模糊化实验之前，我们必须正确配置EOSIO测试网。首先，我们启动keosd wallet来提供密钥管理器服务守护进程，用于存储私钥和签名数字消息。第二，我们运行检测nodeos守护进程来启动单节点测试网。第三，我们部署了eosio.token系统智能合约和攻击者代理合约。第四，我们创建了2个EOSIO帐户，与攻击者代理合约合作，在攻击期间执行EOS传输。最后，我们在EOSIO测试网上部署了模糊化下的智能合约。部署的智能合约包括3881个没有源代码的智能合约和82个有源代码的智能合约。

When the EOSIO testnet had been configured, we further configured EOSFuzzer to perform around 1000 interleaved ABI function invocations and attacks from agent contracts in total for each smart contract. Finally, we started to perform fuzzing with the EOSFuzzer against the deployed smart contracts.

在配置了EOSIO 测试网之后，我们进一步配置了EOSFuzzer，以便为每个智能合约执行大约1000次来自代理合约的交错ABI函数调用和攻击。最后，我们开始用EOSFuzzer对部署的智能合约执行模糊化。

## 5.3 Experiments and Results Analysis

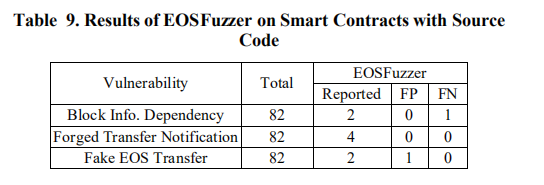
## 5.3实验及结果分析

In this section, we first present the results of EOSFuzzer on contracts with and without source code. Then we further compare the EOSFuzzer and EVulHunter tool [17] in terms of vulnerability detection effectiveness. The EVulHunter is an EOSIO smart contract vulnerability detection tool based on static analysis. Finally, we present the efficiency of EOSFuzzer in terms of vulnerability detection.

在这一节中，我们首先介绍了有源代码和无源代码的合约上的EOSFuzzer的结果。然后我们进一步比较了EOSFuzzer和EVulHunter工具[17]在漏洞检测方面的有效性。EVulHunter是一个基于静态分析的EOSIO智能合约漏洞检测工具。最后，从漏洞检测的角度给出了EOSFuzzer的有效性。

EOSFuzzer Results on Contracts with Source Code. The vulnerability detection result of EOSFuzzer is shown in Table 9. The column Total represents the total number of smart contracts under analysis. The column Reported, FP, and FN represents the number of vulnerabilities, false positive cases and false negative cases reported for the corresponding vulnerability type, respectively. We manually checked the source code of the 82 smart contracts to identify the false positive or false negative cases.

使用源代码的合约的结果. EOSFuzzer的漏洞检测结果如表9所示。“总计”列表示正在分析的智能合约总数。Reported、FP和FN列分别表示针对相应漏洞类型报告的漏洞数量、假阳性案例和假阴性案例。我们手动检查了82个智能合约的源代码，以识别假阳性或假阴性案例。

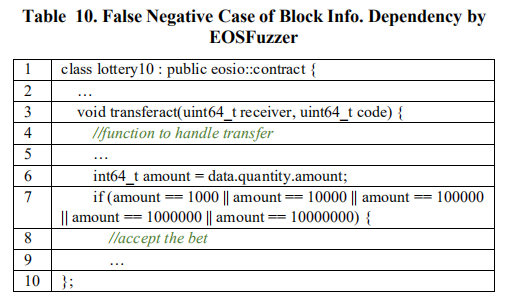


Among the 82 smart contracts, the EOSFuzzer has detected 2 block dependency vulnerabilities. Both of them are confirmed with our manual check. There is one false negative case by EOSFuzzer for the block dependency vulnerability.

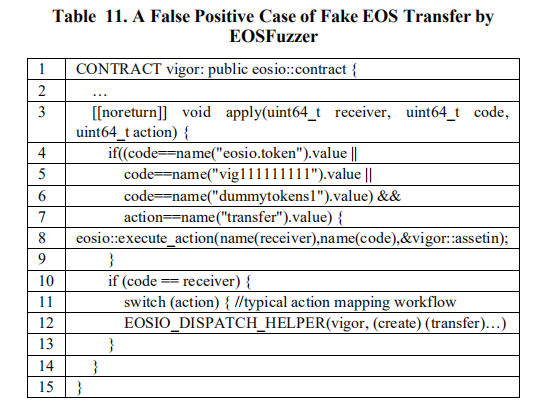
在82个智能合约中，EOSFuzzer检测到2个块依赖漏洞。这两个都是通过我们的人工检查确认的。EOSFuzzer对块依赖漏洞有一个误报。

The smart contract lottery10 is the false negative case of EOSFuzzer. As shown in Table 10, the transferact function is designated to handle the transfer of EOS for making a bet. However, the contract only accepts EOS transfer with some pre-determined amount (i.e., 1000, 10000, 100000…). Moreover, the contract further requires the 10th invocation of the transfer function with the predetermined amount can actually make the bet, which makes the EOSFuzzer hard to trigger without a large-scale fuzzing effort. In general, performing a very large-scale fuzzing campaign against a smart contract or using feedback mechanisms for fuzzing may help expose the vulnerability, which is left as a future work.

智能合约lottery10是EOSFuzzer的假阴性案例。如表10所示，transferact函数用于处理EOS的转移以进行下注。但是，合约仅接受预先确定金额（即1000、10000、100000……）的EOS转让。此外，该合约还要求第10次调用传递函数时，预先确定的数量实际上可以下注，这使得没有大规模的模糊化努力就很难触发EOSFuzzer。一般来说，针对智能合约执行非常大规模的模糊化活动或使用反馈机制进行模糊化可能有助于暴露漏洞，这将留待以后的工作。

For the Forged Transfer Notification vulnerability, the EOSFuzzer has identified 4 smart contracts. After manual checking all the 82 smart contracts, we confirmed that EOSFuzzer reports neither false positives nor false negatives.

对于伪造的传输通知漏洞，EOSFuzzer已识别出4个智能合约。在手动检查所有82个智能合约后，我们确认EOSFuzzer报告既不是假阳性也不是假阴性。

For the Fake EOS Transfer Vulnerability, EOSFuzzer identified 2 vulnerabilities out of the 82 smart contracts. After manual checking, we confirmed that EOSFuzzer reports no false negatives. But one of the identified vulnerable contracts (called vigor) is a false positive case.

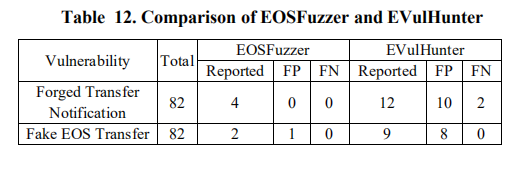
对于假EOS传输漏洞，EOSFuzzer在82个智能合约中发现了2个漏洞。经过人工检查，我们确认EOSFuzzer没有漏报。但其中一个已确认的易受攻击的合同（称为vigor）是一个假阳性案例。

As show in Table 11, the smart contract vigor is a false positive case of Fake EOS Transfer reported by EOSFuzzer. In typical workflow, the apply function will use the transfer function to handle the transfer of EOS token, which is also the assumption our test oracle implementation based on. However, within the vigor contract, the assetin function (rather than the transfer function) is used to handle the transfer of EOS (line 4 to 8). Therefore, the sub-oracle can CanReceiveEOS can be satisfied (i.e., CallIndirect opcode has been executed for at least 2 times). Furthermore, when the attacker agent performs an inline invocation of the transfer function on vigor, the transfer function can still be called through the default dispatcher macro (line 10 to 12). So, the sub-oracle TransferCalled can also be satisfied (since the CallIndirect opcode has been executed for at least 2 times). Since both sub-oracles are satisfied, our EOSFuzzer wrongly reports the contract as a Fake EOS Transfer vulnerability. The reason for the false positive is due to the fact that vigor uses a different function called assetin rather than the function transfer to handle EOS transfer, which are hard to know without analyzing the source code. However, we consider the use of functions other than transfer to handle EOS transfer as unusual for EOSIO smart contracts.

如表11所示，智能合约是EOSFuzzer报告的假EOS转移的假阳性案例。在典型的工作流中，apply函数将使用传递函数来处理EOS代币的传递，这也是我们测试oracle实现所基于的假设。然而，在vigor合同中，assetin函数（而不是传递函数）用于处理EOS的传递（第4行到第8行）。因此，子oracle can receiveeos可以满足（即CallIndirect操作码至少执行了2次）。此外，当攻击者代理在服务器上执行传递函数的内联调用时，仍然可以通过默认的dispatcher宏（第10行到第12行）调用传递函数。因此，也可以满足sub-oracle TransferCalled（因为CallIndirect操作码至少执行了2次）。由于两个子预言器都满意，我们的EOSFuzzer错误地将合同报告为一个假的EOS传输漏洞。之所以会出现误报，是因为vigor使用了另一个名为assetin的函数而不是函数传递来处理EOS传递，如果不分析源代码就很难知道这一点。然而，我们认为使用非传递函数来处理EOS传递对于EOSIO智能合约来说是不寻常的。

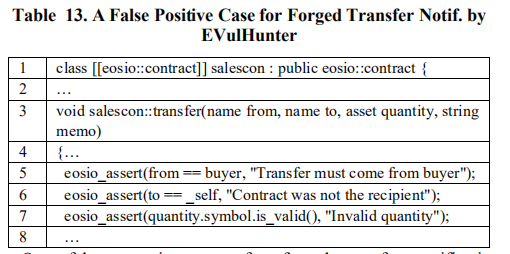
Comparison of EOSFuzzer and EVulHunter. In this section, we further compare EOSFuzzer with the EVulHunter tool for vulnerability detection as shown in Table 12. Since the EVulHunter tool does not the support the detection of Block Information Dependency vulnerability, we compare EOSFuzzer and EVulHunter in terms of the Forged Transfer Notification and Fake EOS Transfer vulnerability only.

EOSFuzzer和EVulHunter的比较. 在本节中，我们将进一步比较EOSFuzzer和EVulHunter工具的漏洞检测，如表12所示。由于EVulHunter工具不支持块信息依赖漏洞的检测，我们仅从伪造的传输通知和伪造的EOS传输漏洞两个方面对EOSFuzzer和EVulHunter进行了比较。

Among the 82 smart contracts, the EVulHunter successfully analyzed 74 smart contracts. It fails to generate output for the other 8 contracts. Within the 74 smart contracts, The EVulHunter has reported 12 smart contracts with Forged Transfer Notification vulnerability. However, after manual check, we found 10 of them are false positives. Furthermore, EVulHunter missed 2 vulnerable smart contracts. Therefore, the EOSFuzzer tool outperforms the EVulHunter when detecting Forged Transfer Notification vulnerability.

在82个智能合约中，EVulHunter成功分析了74个智能合约。它无法为其他8个合同生成输出。在74个智能合约中，EVulHunter报告了12个具有伪造转移通知漏洞的智能合约。然而，经过人工检查，我们发现其中10个是假阳性。此外，EVulHunter遗漏了2个易受攻击的智能合约。因此，在检测伪造的传输通知漏洞时，EOSFuzzer工具的性能优于EVulHunter。

As shown in Table 13, the smart contract salescon is a false positive case for forged transfer notification vulnerability by EVulHunter. It uses the assertion statements (i.e., eosio\_assert) to perform conditional checks. However, the EVulHunter tool seems only works well with conditional checks using if statement, leading to false positives when assertion is used. During our manual confirmation, when we change the assertion statement to if statement in the contract, the EVulHunter tool will not generate the false positive any more.

如表13所示，智能合约salescon是EVulHunter伪造传输通知漏洞的假阳性案例。它使用断言语句（即eosio\_ assert）执行条件检查。然而，EVulHunter工具似乎只适用于使用if语句的条件检查，在使用assert时会导致误报。在手动确认期间，当我们在合约中将断言语句更改为if语句时，EVulHunter工具将不再生成假阳性。

One false negative case for forged transfer notification vulnerability by the EVulHunter tool is the smart contract vigor. During its analysis, the EVulHunter tool fails to find any path leading to the indirect call. However, the EOSFuzzer tool did find a path leading to the indirect call during the fuzzing process. Therefore, the EVulHunter seems not precise enough during its static analysis process.

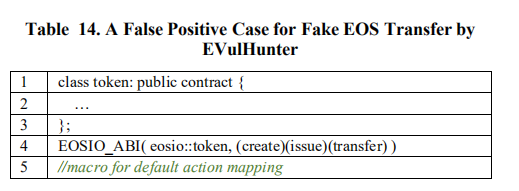
EVulHunter工具伪造的传输通知漏洞的一个假阴性情况是智能合约vigor。在分析过程中，EVulHunter工具无法找到任何指向间接调用的路径。然而，在模糊化过程中，EOSFuzzer工具确实找到了一条通向间接调用的路径。因此，EVulHunter在静态分析过程中似乎不够精确。

For the Fake EOS Transfer Vulnerability, EOSFuzzer identified 2 vulnerabilities while the EVulHunter tool reported 9 vulnerabilities. However, after manual check, we found 8 of the 9 vulnerabilities reported by EVulHunter are actually false positives. In contrast, only 1 of the 2 vulnerabilities reported by EOSFuzzer is false positive. Therefore, for Fake EOS Transfer Vulnerability, EOSFuzzer incurs much lower false positives than EVulHunter.

对于假EOS传输漏洞，EOSFuzzer发现了2个漏洞，而EVulHunter工具报告了9个漏洞。然而，经过手动检查，我们发现EVulHunter报告的9个漏洞中有8个实际上是误报的。相反，EOSFuzzer报告的2个漏洞中只有1个是假阳性。因此，对于假EOS传输漏洞，EOSFuzzer产生的误报率比EVulHunter低得多。

As shown in Table 14, the token smart contract is reported by EVulHunter as Fake EOS Transfer vulnerability. However, within the token contract, it uses the EOSIO\_ABI macro for default action mapping. The EOSIO\_ABI can only handle scenarios where code is equal to self. Therefore, the token contract cannot handle EOS transfer from the eosio.token contract. It is only a contract handling tokens other than EOS. Therefore, the token contract is a false positive case of EVulHunter.

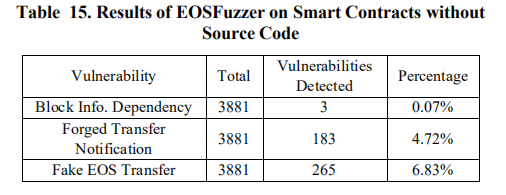
如表14所示，EVulHunter报告代币智能合约为假EOS传输漏洞。但是，在代币合约中，它使用EOSIO\_ABI宏进行默认操作映射。EOSIO\_ABI只能处理代码等于self的场景。因此，代币合约无法处理来自服务器的EOS传输eosio.token合约。这只是一个处理代币而不是EOS。因此，代币合约是EVulHunter的假阳性案例。

EOSFuzzer Results on Contract without Source Code. We have also performed fuzzing on 3881 EOSIO smart contract with bytecode and ABI only. As shown in Table 15, the EOSFuzzer has identified 3 Block Information Dependency vulnerabilities, 183 Forged Transfer Notification vulnerabilities, and 265 Fake EOS Transfer vulnerabilities. Among the 3881 smart contracts, the percentage of Block Information dependency, Forged Transfer Notification, and Fake EOS Transfer are 0.07%, 4.72%, and 6.83%, respectively. Without source code, it is hard to manually verify those reported smart contracts to identify the precise number of false positives and false negatives. But the results give us an estimation of the number of vulnerable smart contracts in the wild.

EOSFuzzer在没有源代码合约上的结果. 我们还用字节码和ABI对3881 EOSIO智能合约进行了模糊处理。如表15所示，EOSFuzzer已识别出3个块信息依赖漏洞、183个伪造的传输通知漏洞和265个伪造的EOS传输漏洞。在3881份智能合约中，区块信息依赖、伪造转让通知和伪造EOS转让的比例分别为0.07%、4.72%和6.83%。如果没有源代码，很难手动验证那些报告的智能合约，以确定误报和漏报的准确数量。但是研究结果给了我们一个易受攻击的智能合约数量的估计。

Considering the potential large number of Forged Transfer Notification and Fake EOS Transfer vulnerability, the developers are recommended to perform security check and harden their contract before releasing their EOSIO smart contract to public.

考虑到可能存在大量伪造的转让通知和伪造的EOS转让漏洞，建议开发者在向公众发布EOSIO智能合约之前进行安全检查并加固合约。

Efficiency of EOSFuzzer. When fuzzing each contract, we configured EOSFuzzer to perform around 1000 interleaved ABI function invocations and attacks from agent contracts in total for each smart contract. For each contract, the fuzzing experiment takes around 90 seconds on average, which is small based on the hardware configurations of our experiment. Furthermore, the average invocations per second is around 11, which is fast enough for comprehensive fuzzing. This is partly due to the high transaction speed of EOS based on the DPOS consensus protocol. Therefore, EOSFuzzer is also efficient for EOSIO smart contract vulnerability detection.

模糊器效率. 在对每个合约进行模糊化时，我们将EOSFuzzer配置为对每个智能合约执行大约1000次来自代理合约的交错ABI函数调用和攻击。对于每个合约，模糊化实验平均需要90秒左右，根据我们实验的硬件配置，这是很小的。此外，平均每秒调用11次左右，这对于全面的模糊化来说已经足够快了。这部分是由于基于DPOS共识协议的EOS具有很高的事务处理速度。因此，EOSFuzzer对于EOSIO智能合约漏洞检测也是有效的。

# 6 Mounting Attacks on Vulnerable Contracts

# 6针对易受攻击合约的攻击不断增加

We have also successfully mounted an attack on a smart contract called diamond1 to earn EOS within our testnet. Since diamond1 provides no source code, we performed fuzzing on its bytecode based on its ABI interfaces. In addition to the test oracles defined within EOSFuzzer, we also checked whether the EOS balance of the diamond1 is reduced after each ABI invocation (Initialized to be 1000 EOS before fuzzing). In another word, we wanted to ensure that the asset of the contract under fuzzing was lost through the attack.

我们还成功地对名为diamond1的智能合约发起了攻击，以在我们的测试网中获得EOS。由于diamond1不提供源代码，因此我们根据其ABI接口对其字节码执行模糊处理。除了在EOSFuzzer中定义的测试预言之外，我们还检查了diamond1的EOS余额是否在每次ABI调用后减少（在模糊化之前初始化为1000个EOS）。换言之，我们希望确保模糊化下的合约资产在攻击中丢失。

Within 7 ABI invocations, the EOSFuzzer not only successfully triggered the Forge Transfer Notification vulnerability, but it also successfully made a bet and earned a reward (26.46 EOS) without using a single EOS. The detailed attacking process is as follows:

在7次ABI调用中，EOSFuzzer不仅成功触发了伪造传输通知漏洞，而且还成功下注并获得了奖励（26.46个EOS），而无需使用单个EOS。具体攻击过程如下：

During the first ABI invocation, the EOSFuzzer performed an EOS transfer to the diamond1 smart contract and checked whether its transfer function was invoked. This was to check whether diamond1 could receive EOS. During the following 3 ABI invocations, the EOSFuzzer performed the Forged Transfer Notification attacks by transferring EOS to the notifier agent contract from a sender account. The notifier agent contract will send the notification to the diamond1 smart contract through the require\_recipient function. For all 3 invocations, the Forged Transfer Notification were all triggered, and EOSFuzzer successfully made a bet without spending any EOS. During the 5th invocation, the EOSFuzzer performed the Fake EOS Transfer attack with the agent contract to perform an inline invocation to the diamond1 smart contract. However, no Fake EOS Transfer vulnerability was triggered. During the 6th invocation, EOSFuzzer invoked the deposit ABI function of diamond1. However, its balance was not changed. Finally, during the 7th ABI invocation, EOSFuzzer invoked the endlottery ABI function of the diamond1 smart contract, which ended the betting process and announced the betting results. Note that the endlottery ABI function can also be triggered from the website of the gambling game.

在第一次ABI调用期间，EOSFuzzer执行到diamond1智能合约的EOS传输，并检查其传输函数是否被调用。这是为了检查diamond1是否可以接收EOS。在接下来的3次ABI调用中，EOSFuzzer通过将EOS从发送方帐户传输到notifier-agent合约来执行伪造的传输通知攻击。通知代理合约将通过require\_recipient函数向diamond1智能合约发送通知。对于所有3个调用，伪造的转移通知都被触发，并且EOSFuzzer在没有花费任何EOS的情况下成功地进行了下注。在第5次调用期间，EOSFuzzer使用代理合约执行了假EOS传输攻击，以执行对diamond1智能合约的内联调用。但是，没有触发假EOS传输漏洞。在第6次调用中，EOSFuzzer调用了diamond1的取款ABI函数。然而，它的余额没有改变。最后，在第7次ABI调用期间，EOSFuzzer调用diamond1智能合约的EndLockit ABI函数，该函数结束了下注过程并宣布了下注结果。请注意，EndLockting ABI功能也可以从赌博游戏的网站触发。

At the end of the last invocation, EOSFuzzer found that the balance of diamond1 smart contract reduced to 973.54 EOS while the balance of the sender account increased from 1000 to 1026.46. The logs further confirmed the success of the attack: the diamond1 smart contract considered the sender account had successfully made a bet during the Forged Transfer Notification attack, it announced the sender account as the winner, and it had sent 26.46 EOS to the sender account as the reward. However, the sender account only sent the notifier agent contract some EOS for conspiracy, it earned 26.46 EOS without any stake.

在最后一次调用结束时，EOSFuzzer发现diamond1智能合约的余额减少到973.54 EOS，而发送方帐户的余额从1000增加到1026.46。日志进一步证实了攻击的成功：diamond1智能合约认为发送方账户在伪造转账通知攻击中成功下注，宣布发送方账户为赢家，并向发送方账户发送了26.46个EOS作为奖励。然而，发送者帐户只发送了通知代理合约的一些EOS，它赚了26.46 EOS却没有任何投注。

# 7 Related Work

# 7相关工作

In this section, we present closely related work on smart contract vulnerability detection and fuzzing techniques.

在本节中，我们将介绍与智能合约漏洞检测和模糊技术密切相关的工作。

## 7.1 Smart Contract Vulnerability Detection

## 7.1智能合约漏洞检测

Parizi et al. [15] carry out an extensive experimental assessment of current static smart contracts security testing tools for the Ethereum smart contracts. Tsankov et al. proposed the Securify tool for smart contract vulnerability detection. The Securify tool [18] is a scalable security analyzer for Ethereum smart contracts and it can prove contract behaviors as safe/unsafe with respect to a given property. Abdellatif and Brousmiche [1] used a formal modeling approach to verify a smart contract behavior in its execution environment. They further analyzed the security of contracts with a statistical model checking approach.

Parizi等人[15]对当前用于以太坊智能合约的静态智能合约安全测试工具进行了广泛的实验评估。Tsankov等人提出了智能合约漏洞检测的安全工具。Securify工具[18]是用于以太坊智能合约的可扩展安全分析器，它可以证明合约行为相对于给定的属性是安全/不安全的。Abdellatif和Brousmiche[1]使用了一种形式化的建模方法来验证智能合约在其执行环境中的行为。他们用统计模型检验方法进一步分析了合约的安全性。

Luu et al [12] designed Oyente, a symbolic verification tool for Ethereum smart contract. Oyente builds the control-flow graph of smart contracts and then performs symbolic execution on the control flow graph while checking whether there exist any vulnerable patterns. Nikolic et al. [14] designed MAIAN, a symbolic execution tool for reasoning about tracing properties to detect vulnerable Ethereum smart contracts. It specified three typical smart contracts vulnerabilities based on trace properties. The MAIAN can efficiently detect the greedy, the prodigal and the suicidal contracts through symbolic execution. Hirai [11] used Isabelle/HOL tool to verify the smart contract called Deed, which is part of the Ethereum Name Service implementation. Specifically, the work verifies the oracle that only the owner of Deed could decrease its balance. Furthermore, they also found the EVM implementation is poorly tested during the verification process.

Luu等人[12]设计了Oyente，一个用于以太坊智能合约的符号验证工具。Oyente构建智能合约的控制流图，然后在控制流图上执行符号执行，同时检查是否存在任何易受攻击的模式。Nikolic等人[14]设计了MAIAN，这是一个符号执行工具，用于推理跟踪属性以检测易受攻击的以太坊智能合约。它基于跟踪属性指定了三个典型的智能合约漏洞。通过象征性的执行，可以有效地发现贪婪、挥霍和自杀的合约。Hirai[11]使用Isabelle/HOL工具来验证名为Deed的智能合约，它是以太坊名称服务实现的一部分。具体来说，这项工作验证了预期想法，只有所有者的合约可以减少其余额。此外，他们还发现EVM的实现在验证过程中没有得到很好的测试。

Chen et al. [5] proposed the TokenScope tool, which automatically checks whether the behaviors of the token contracts are consistent with the ERC–20 standards. Nguyen et al. proposed the sFuzz tool [7], which combined the strategy in the AFL fuzzer and an efficient lightweight multi-objective adaptive strategy targeting those hard-to-cover branches. The evaluation shows the sFuzz tool is efficient and effective to achieve high code coverage and to detect vulnerabilities. Wang et al. [19] proposed the VULTRON tool, which can precisely detect irregular transactions in smart contracts due to various types of adversarial exploits. It provided a general way to solve the test oracle problem for smart contract vulnerability detection.

Chen等人[5]提出了TokenScope工具，它自动检查代币合约的行为是否符合ERC–20标准。Nguyen等人提出了sFuzz工具[7]，它将AFL fuzzer中的策略与针对难以覆盖分支的高效轻量级多目标自适应策略相结合。评估结果表明，sFuzz工具在实现高代码覆盖率和检测漏洞方面是有效的。Wang等人[19]提出了VULTRON工具，它可以精确地检测由于各种类型的对抗性攻击而导致的智能合约中的不规则交易。为智能合约漏洞检测提供了一种解决测试预言问题的通用方法。

The techniques reviewed above are in general effective to detect the vulnerabilities or bugs within Ethereum smart contracts. However, they are not specifically designed to support the detection of vulnerabilities within EOSIO smart contracts.

上述技术通常能有效地检测以太坊智能合约中的漏洞或bug。但是，它们并不是专门为支持EOSIO智能合约中的漏洞检测而设计的。

The EVulHunter [17] is a static analysis tool to detect vulnerabilities within EOSIO smart contracts. However, it fails to generate results for a number of EOSIO smart contracts and it generates many false negatives and false positives for vulnerability reporting. EOSAFE [10] is another static analysis framework to detect vulnerabilities within EOSIO smart contracts based on symbolic execution. We believe dynamic fuzzing and static analysis are two complementary techniques for EOSIO smart contract vulnerability detection.

EVulHunter[17]是一个静态分析工具，用于检测EOSIO智能合约中的漏洞。但是，它不能为许多EOSIO智能合约生成结果，并且会为漏洞报告生成许多误报和误报。EOSAFE[10]是另一个静态分析框架，用于检测基于符号执行的EOSIO智能合约中的漏洞。我们认为动态模糊和静态分析是EOSIO智能合约漏洞检测的两种互补技术。

## 7.2 Fuzzing Techniques for Vulnerability Detection

## 7.2漏洞检测的模糊化技术

There are many works on fuzzing techniques for vulnerability detection.

有许多关于漏洞检测的模糊技术的工作。

GodeFroid [9] et al. proposed to enhance whitebox fuzzing of complex structured-input applications with a grammar-based specification of their valid inputs. Their test data generation algorithm combines symbolic execution and constraint solving to improve the fuzzing process. Wang et al. proposed TaintScope [20], an automatic fuzzing system using dynamic taint analysis and symbolic execution techniques. TaintScope can identify checksum fields in input instances, locate checksum-based integrity checks by using branch profiling techniques, and bypass such checks via control flow alteration.

GodeFroid[9]等人提出了一种基于语法的有效输入规范，以增强复杂结构化输入应用程序的白盒模糊化。他们的测试数据生成算法结合符号执行和约束求解来改进模糊化过程。Wang等人提出了TaintScope[20]，一个使用动态污点分析和符号执行技术的自动模糊系统。TaintScope可以识别输入实例中的校验和字段，使用分支分析技术定位基于校验和的完整性检查，并通过控制流更改绕过此类检查。

Dai et al. [6] proposed the configuration fuzzing technique, which randomly modifies the configuration of the running application at certain execution points to check for vulnerabilities. Ganesh realized the BuzzFuzz tool [8], which uses dynamic taint tracing to automatically locate regions of original seed input files that influence values used at key program attack points. The BuzzFuzz tool then automatically generates new fuzzed test input files by fuzzing these identified regions of the original seed input files.

Dai等人[6]提出了配置模糊化技术，它在某些执行点随机修改正在运行的应用程序的配置，以检查漏洞。Ganesh实现了BuzzFuzz工具[8]，它使用动态污点跟踪自动定位原始种子输入文件的区域，这些区域会影响关键程序攻击点使用的值。然后，BuzzFuzz工具通过模糊原始种子输入文件的这些标识区域，自动生成新的模糊测试输入文件。

Chen et al. [3] proposed a mutation-based fuzzer called Angora, which proposed to use scalable byte-level taint tracking, contextsensitive branch count, search based on gradient descent, and input length exploration techniques to solve path constraints efficiently. Lyu et al. [13] proposed a novel mutation scheduling scheme, which made mutation-based fuzzer to discover vulnerabilities more efficiently. Chen et al. [4] proposed a globally asynchronous and locally synchronous (GALS) seed synchronization mechanism to seamlessly ensemble base fuzzer for better performance.

Chen等人[3]提出了一种基于变异的模糊器Angora，它利用可伸缩字节级污点跟踪、上下文敏感分支计数、基于梯度下降的搜索和输入长度探索技术来有效地解决路径约束问题。Lyu等人[13]提出了一种新的变异调度方案，使得基于变异的模糊器能够更有效地发现漏洞。Chen等人[4]提出了一种全局异步和局部同步（GALS）种子同步机制，无缝集成基本模糊器以获得更好的性能。

You et al. [21] proposed the ProFuzzer, which automatically recovered and understood input fields of critical importance to vulnerability discovery during a fuzzing process. The fuzzer can intelligently adapt the mutation strategy to enhance the chance of hitting zero-day targets.

You等人[21]提出了ProFuzzer，它可以自动恢复并理解在模糊化过程中对漏洞发现至关重要的输入字段。模糊器可以智能地调整变异策略，提高命中零日目标的几率。

In general, different from EOSFuzzer, the fuzzing techniques reviewed aboved are not specifically designed for detecting vulnerabilities within EOSIO smart contracts.

一般来说，不同于EOSFuzzer，上面讨论的模糊技术并不是专门为检测EOSIO智能合约中的漏洞而设计的。

There are also interesting works on stateful fuzzing techniques. The AFLnet takes a mutational approach and uses state-feedback to guide the fuzzing process of network protocols [16]. The REST-ler [2] is a stateful automatic intelligent REST API security-testing tool. REST-ler generates tests intelligently by inferring dependencies among request types declared in the Swagger specification. It also analyzes dynamic feedback from responses observed during prior test executions in order to generate new tests. Currently, the fuzzing performed by EOSFuzzer is stateless. We may extend EOSFuzzer to perform stateful fuzzing with the support of data-flow analysis in the future work.

还有一些关于状态模糊技术的有趣的工作。AFLnet采用了一种变异的方法，并使用状态反馈来指导网络协议的模糊化过程[16]。REST-ler[2]是一个有状态的自动智能REST API安全测试工具。REST-ler通过推断Swagger规范中声明的请求类型之间的依赖关系来智能地生成测试。为了生成新的测试，它还分析了在先前测试执行期间观察到的响应的动态反馈。目前，EOSFuzzer执行的模糊化是无状态的。在未来的工作中，我们可以在数据流分析的支持下扩展EOSFuzzer来执行有状态模糊化。

# 9 Conclusions and Future Work

# 9结论和今后的工作

The vulnerabilities within the EOSIO smart contracts have resulted in significant loss for its users. Therefore, effective tools for detecting the vulnerabilities within EOSIO smart contract are needed. In this work, we present EOSFuzzer, a black-box fuzzing tool to automatically detect vulnerabilities within EOSIO smart contracts. Within EOSFuzzer, we have proposed effective attacking scenarios as well as invocations to ABI interfaces to perform comprehensive fuzzing on EOSIO smart contracts. Furthermore, we have also defined and realized test oracles to detect three typical vulnerabilities for EOSIO smart contracts. Our fuzzing experiment showed that EOSFuzzer is effective and efficient to detect vulnerabilities within EOSIO smart contracts for practical use. In particular, EOSFuzzer has successfully mounted attacks on EOSIO smart contract without source code to make bets without spending any EOS.

EOSIO智能合约中的漏洞已经给用户造成了巨大的损失。因此，需要有效的工具来检测EOSIO智能合约中的漏洞。在这项工作中，我们提出了EOSFuzzer，一个黑盒模糊工具，自动检测EOSIO智能合约的漏洞。在EOSFuzzer中，我们提出了有效的攻击场景以及对ABI接口的调用，以对EOSIO智能合约执行全面的模糊化。此外，我们还定义并实现了测试预言机来检测EOSIO智能合约的三个典型漏洞。我们的模糊实验表明，EOSFuzzer是有效的，有效的检测实际使用的EOSIO智能合约内的漏洞。尤其是，EOSFuzzer成功地在没有源代码的情况下对EOSIO智能合约发起攻击，从而在不花费任何EOS的情况下获利。

For future work, we would like to extend the EOSFuzzer to support the detection of new types of vulnerabilities in EOSIO smart contracts. We also plan to improve the search strategies of EOSFuzzer during fuzzing to improve its vulnerability detection effectiveness for complex smart contract.

对于未来的工作，我们希望扩展EOSFuzzer以支持EOSIO智能合约中新类型漏洞的检测。我们还计划改进模糊化过程中的搜索策略，以提高其对复杂智能合约的漏洞检测效率。

# REFERENCES

# 参考文献

[1] Abdellatif, T., Brousmiche, K.L. "Formal Verification of Smart Contracts Based on Users and Blockchain Behaviors Models," In Proceedings of 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328737.

[2] V. Atlidakis, P. Godefroid and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," In Proceedings of 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE'19), Montreal, QC, Canada, 2019, pp. 748-758, doi: 10.1109/ICSE.2019.00083.

[3] Chen, P. and Chen, H. "Angora: Efficient Fuzzing by Principled Search," In Proceedings of 2018 IEEE Symposium on Security and Privacy (SP'18), San Francisco, CA, 2018, pp. 711-725, doi: 10.1109/SP.2018.00046.

[4] Chen, Y., Jiang, Y., Ma, F., Liang, J., Wang, M., Zhou, C., Jiao, X., and Su, Z. EnFuzz: ensemble fuzzing with seed synchronization among diverse fuzzers. In Proceedings of the 28th USENIX Conference on Security Symposium (SEC' 19). USENIX Association, USA, 1967– 1983, 2019.

[5] Chen, T., Zhang, Y., Li, Z., Luo, X., Wang, T., Cao, R., Xiao, X., and Zhang, X. 2019. TokenScope: Automatically Detecting Inconsistent Behaviors of Cryptocurrency Tokens in Ethereum. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19). Association for Computing Machinery, New York, NY, USA, 1503–1520.

[6] Dai, H., Murphy, C., and Kaiser, G. "Configuration Fuzzing for Software Vulnerability Detection," In Proceedings of 2010 International Conference on Availability, Reliability and Security, Krakow, 2010, pp. 525-530, doi: 10.1109/ARES.2010.22.

[7] Duy Tai Nguyen, Long H. Pham, Jun Sun, Yun Lin and Minh Quang Tran: “sFuzz: An Efficient Adaptive Fuzzer for Solidity Smart Contracts”, in Proceedings of the 42nd International Conference on Software Engineering. IEEE Computer Society, 2020.

[8] Ganesh, V., Leek, T., and Rinard, M. "Taint-based directed whitebox fuzzing," in Proceedings of the IEEE 31st International Conference on Software Engineering, Vancouver, BC, 2009, pp. 474-484, doi: 10.1109/ICSE.2009.5070546.

[9] Godefroid, P., Kiezun, A., and Levin, M. Y. Grammar-based whitebox fuzzing. In Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '08). Association for Computing Machinery, New York, NY, USA, 206–215.

[10] He, N., Zhang, R., Wu, L., Wang, H., Luo, X., Guo, Y., Yu, T., Jiang, X. “Security Analysis of EOSIO Smart Contracts.” ArXiv abs/ 2003.06568, 2020.

[11] Hirai, Y. Formal verification of Deed contract in Ethereum name service. http://yoichihirai.com/deed.pdf, 2016.

[12] Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A. Making Smart Contracts Smarter. In Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16), 254269, Vienna, Austria, 2016.

[13] Lyu, C., Ji, S., Zhang, C., Li, Y., Lee, W., Song, Y., and Beyah, R. MOPT: optimized mutation scheduling for fuzzers. In Proceedings of the 28th USENIX Conference on Security Symposium (SEC'19). USENIX Association, USA, 1949–1966, 2019.

[14] Nikolic, I., Kolluri, A., Sergey, I., Saxena, P., and Hobor, A. Ivica 2018. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. In Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18). Association for Computing Machinery, New

        York, NY, USA, 653–663.

DOI:https://doi.org/10.1145/3274694.3274743

[15] Parizi, R. M., Dehghantanha, A., Choo, K. K. R., and Singh, A. Reza 2018. Empirical vulnerability analysis of automated smart contracts security testing on blockchains. In Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering (CASCON '18). IBM Corp., USA, 103–113.

[16] Van-Thuan Pham, Marcel Böhme, Abhik Roychoudhury. AFLNet: A Greybox Fuzzer for Network Protocols. In Proceedings of IEEE International Conference on Software Testing, Verification and Validation : Testing Tools Track (ICST'20). 2020.

[17] Quan, Lijin, Wu, Lei and Wang, Haoyu. “EVulHunter: Detecting Fake Transfer Vulnerabilities for EOSIO's Smart Contracts at Webassemblylevel.” ArXiv abs/1906.10362, 2019.

[18] Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Bünzli, F., and Vechev, M. 2018. Securify: Practical Security Analysis of Smart Contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). Association for

        Computing Machinery, New York, NY, USA, 67–82.

DOI:https://doi.org/10.1145/3243734.3243780

[19] Wang, H., Li, Y., Lin, S., Ma, L., and Liu, Y."VULTRON: Catching Vulnerable Smart Contracts Once and for All," In Proceedings of IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Montreal, QC, Canada, 2019, pp. 1-4.

[20] Wang, T., Wei, T., Gu, G., and Zou. "TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection," In Proceedings of 2010 IEEE Symposium on Security and Privacy(SP'10), Berkeley/Oakland, CA, 2010, pp. 497-512, doi: 10.1109/SP.2010.37.

[21] You, W., Wang, X., Ma, S., Huang, J., Zhang, X., Wang, X., Liang, B. "ProFuzzer: On-the-fly Input Type Probing for Better Zero-Day Vulnerability Discovery," In Proceedings of 2019 IEEE Symposium on Security and Privacy (SP'19), San Francisco, CA, USA, 2019, pp. 769786, doi: 10.1109/SP.2019.00057.

[22] Attack on EOS Bet. https://medium.com/leclevietnam/hacking-in-eoscontracts-and-how-to-prevent-it-b8663c8bffa6. Last access, 2020.

[23] Cleos tool. https://eos.io/build-on-eosio/cleos/. Last access, 2020.

[24] EOSIO.TOKEN. https://developers.eos.io/welcome/latest/gettingstarted/smart-contract-development/deploy-issue-and-transfer-tokens. Last access, 2020.

[25] EOSIO ABI Macro and Apply. https://developers.eos.io/eosiocpp/v1.2.0/docs/abi. Last access, 2020.

[26] Fake EOS Transfer Vulnerabilities in EOS smart contracts. https://blog.peckshield.com/2018/11/02/eos/. Last access, 2020.

[27] Forged Transfer Notification in EOS smart contracts. https://blog.peckshield.com/2018/10/26/eos/. Last access, 2020.

[28] Random Number Generation Vulnerability by NoneAge. https://github.com/NoneAge/EOS\_dApp\_Security\_Incident\_Analysis/tr ee/master/NoneAge-20181103-EOSDICE-predictable-random-seed. Last access, 2020.

[29] Smart Contract Development on EOS. https://developers.eos.io/welcome/latest/getting-started/index. Last access, 2020.

[30] System Smart Contracts on EOSIO. https://developers.eos.io/manuals/eosio.contracts/latest/index. Last access, 2020.

[31] Transfer function of EOSIO smart contracts. https://developers.eos.io/manuals/eosio.cdt/latest/best-practices/abi/abicode-generator-attributesexplained/#eosioon\_notifyvalid\_eosio\_account\_namevalid\_eosio\_action \_name. Last access, 2020.

[32] Web Assembly. https://webassembly.org/. Last access, 2020.

[33] WebAssembly Binary Toolkit. https://github.com/WebAssembly/wabt. Last access, 2020.