CLC_____ Number_____

UDC_____ Classification Level_____

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Undergraduate Thesis

Finding EOSIO Smart Contracts Security Issues by Comparing History Version

Author :	Yubin Hu
Student ID :	11712121
Department :	Computer Science and Technology
Major :	Computer Science
Supervisor :	Prof. Yepang Liu
Acad. Supervisors:	Prof. Haoyu Wang
Finished Time :	May, 2021

诚信承诺书

1. 本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2. 除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名: _____

_____ 年___ 月___ 日

Contents

ABSTRACT

Unlike Ethereum, the EOSIO blockchain platform is unique in that the features and characteristics of the blockchain built on it are flexible, which means they can be changed and modified completely to suit business case requirements. Thus, many developers update the code when bugs are detected. In this paper, we propose a method to find security issues of EOSIO smart contracts by comparing the history version. After analyzing all the security issues in EOSIO smart contracts, we integrate the causes, patterns, and solutions to the security issues.

Keywords: EOSIO, Smart Contracts, Security

Chapter 1 Introduction

The EOSIO is the most popular public blockchain platform supporting smart contracts and has grown rapidly in recent years. As of 2019, the total value of on-chain transactions of EOSIO has exceeded 6 billion USD. Furthermore, EOSIO is still in its early stages and has some experimental characteristics. As a result, as new bugs and security are discovered and new features are developed, the security threats we face constantly change. The vulnerabilities within smart contracts have led to severe financial loss. EOS Bet —a gambling dApp which uses EOS tokens, lost at least $338,000 from its operational wallets to hackers on 2018-10-15[1].

Therefore, we need practical vulnerability detection tools to safeguard the ecosystem of blockchain. However, the vulnerability detection tools for EOSIO smart contracts are limited. EOSAFE[2] and WANA[3] support detecting specific bugs, such as Fake EOS, Fake Receipt, Rollback, and Missing Permission Check.[4] Although the current detection effect has reached the expected level, it will not have a good effect on new vulnerabilities or some artificially maliciously implanted vulnerabilities.

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. On top of the EOSIO core layer, a WebAssembly virtual machine, *EOS VM*, executes smart contract code, and it is designed from the ground up for the high demands of blockchain applications which require far more from a WebAssembly engine than those designed for web browsers or standards development. The EOSIO smart contracts are developed using cpp or rust. And they can be compiled into WebAssembly code for execution in the corresponding WebAssembly VM implementation with *eosio.cdt* (EOSIO Contract Development Toolkit)

For Ethereum, the only way to destroy a smart contract on the blockchain system is using the *Selfdestruct* function when attackers find the bugs. Unlike Ethereum, applications deployed on EOSIO-based blockchains are upgradeable. This means you can deploy code fix, add features, and change the application's logic as long as sufficient authority is provided. As a developer, you can iterate your application without the risk of being locked into a software bug permanently. We can easily collect several versions of the same smart contract. Generally speaking, the code updated by the developer may be vulnerabilities, new functions, or even artificially implanted code with fraud.

This paper first crawls all the verified (open-sourced) smart contracts with several versions from EOSIO. Then we compare the differences between different versions of the same smart contract from the symbolic execution path. Finally, we analyze the differences to summarize the security issues and other reasons for smart contract updates.

Chapter 2 Background

2.1 WebAssembly

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications[5]. The WebAssembly virtual machines can be embedded into Web browsers or blockchain platforms. The EOSIO blockchain has supported Wasm. Furthermore, in Ethereum 2.0, Wasm VM is the replacement of Ethereum VM (EVM).

The design choice of using Wasm enables EOSIO to reuse optimized and battle-tested compilers and toolchains which are being maintained and improved by a broader community. In addition, adopting the Wasm standard also makes it easier for compiler developers to port other programming languages onto the EOSIO platform.

There are two convertible and equivalent representations for WebAssembly. We have a binary format - ".wasm" as the suffix. To enable WebAssembly to be read and edited by humans, there is a textual representation of the Wasm binary format - ".wast" as the suffix.

2.2 EOSIO

EOSIO is a free, open-source blockchain software protocol that provides developers and entrepreneurs with a platform on which to build, deploy and run high-performing blockchain applications.

2.2.1 Accounts

An *account* is a human-readable name stored on the blockchain[4]. In order to ensure account security and prevent identity fraud, EOSIO has implemented an advanced access control system based on permissions. An *account* is required to transfer or push any valid transaction to the blockchain.

2.2.2 Delegated Proof of Stake (DPOS)

The EOSIO platform implements a proven decentralized consensus algorithm capable of meeting the performance requirements of applications on the blockchain called the *Delegated Proof of Stake* (DPOS). Under this algorithm, if you hold tokens on a EOSIO-based blockchain, you can select block producers through a continuous approval voting system. Anyone can choose to participate in the block production and will be allowed to produce blocks, provided they can persuade token holders to vote for them[4].

2.2.3 Smart Contracts

Smart contract is a piece of code that can execute on a blockchain and keep the state of contract execution as a part of the immutable history of that blockchain instance. Therefore, developers can rely on that blockchain as a trusted computation environment in which inputs, execution, and the results of a *smart contract* are independent and free of external influence. Every EOSIO smart contract must use *apply* functions as input functions to process operation.

Transactions are composed of one or more actions, which are the basic units for triggering functions in smart contracts.

Dispatcher When one account needs to adjust the smart contract of another account, and the smart contract needs to be processed and dispatched, then we need a dispatcher.

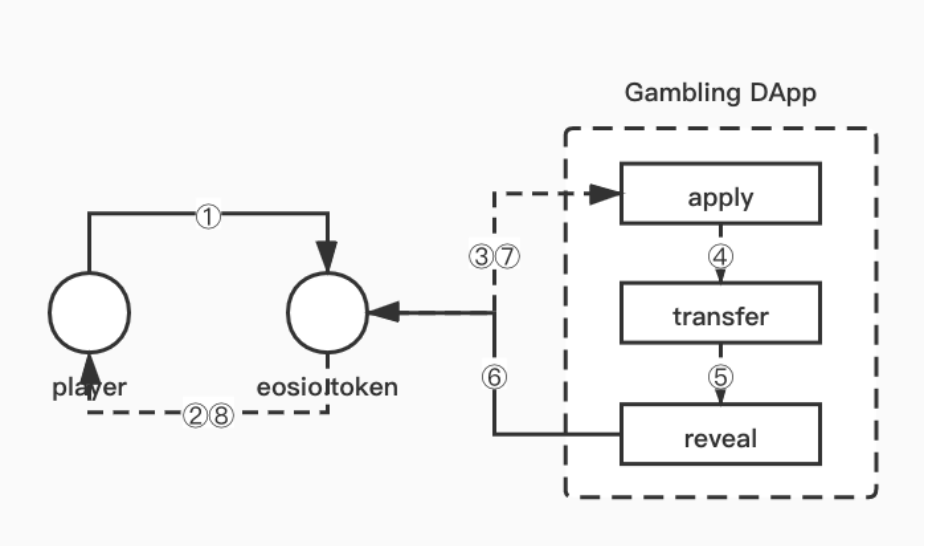

Figure 2.1: life-cycle of smart contract execution[2][6]

The life-cycle of smart contract execution[2]:

1. Invoke transfer to take part in game
2. Notify player (payer)
3. Notify DApp (payee)
4. Dispatch to transfer function
5. Invoke reveal to calculate jackpot
6. Invoke transfer to return prize
7. Notify DApp (payer)

8. Notify player (payee)

In this process, due to various reasons, such as failing to check the authentication parameters, security issues will occur.

Chapter 3 Security Issues in EOSIO Smart Contract

3.1 Fake EOS

Cause of vulnerability In EOSIO smart contract, we need *EOS* token at the 3rd step of the generate life-cycle of smart contract execution. However, anyone, including a hacker, can create a token through the public code of EOSIO.token. And the name of the token can be repeated. This means that the name of the token created by the hacker can also be called *EOS*.

Generally speaking, a hacker creates an *EOS* token and name is *EOS*, then transfer the amount of these fake EOS tokens to a EOSIO smart contract. If the DApp's code is robust enough, then the hacker's attack should have failed. Although the name is the same as the real EOS token, the issuer is different. The hacker transmits the EOS token to the DApp through the copy, and then the DApp will not receive the notified *code* EOS token. If it happens that the DApp does not check the value of the *code* at this time, the verification of the dispatcher will be bypassed.

In order to avoid the above problems as much as possible, developers simply narrowed the scope of accepting code, but it can also be bypassed. Obviously, this approach is not rigorous enough. It is this place that gives us the possibility of detecting vulnerabilities.

Harm of vulnerability Early morning on 2018-10-31, online media reported that gambling platform EOSCast was attacked by hackers and lost more than 70K EOS tokens.[7]

3.2 Fake Receipt

Cause of vulnerability In the generate life-cycle of smart contract execution, the notification will be forwarded when the developer checks the *code*. Also, notification can be forwarded by the dispatcher even the code will not be changed. This is the key to this vulnerability being attacked.

Hackers play two roles in this game, and one is *initiator*, the other is *accomplice*. The initiator invokes a *transfer* to the accomplice through the EOS token. When the EOS token notifies the accomplice, the code is not changed and is directly forwarded to the DApp. However, if the parameters are not checked during *transfer*, the EOS token will be passed between the two roles of the hacker, which will lead to EOS loss.

3.3 Rollback

Cause of vulnerability The transaction can be rolled back in some malicious ways, which is not a good thing for some gambling DApps. Suppose that in the gambling DApp,

in the generate life-cycle of smart contract execution, when the eighth step, "Notify player" is executed, the hacker can immediately check whether the EOS balance of his account has decreased. If it reduces, the hacker will immediately call the interrupt to roll back the gambling. Such gambling is meaningless, and hackers will always win.

Harm of vulnerability On the day 2018–12–19, Most of the gambling dapp was suffered rollback attack, including Betdice, EOSMax, Tobet, etc. Caluclating at the price of 18 RMB each EOS, it had loss over 500 million rmb.[8]

Through the transaction records of the account, we can find that the account has only revealing records and no betting records.

3.4 Missing Permission Check

Cause of vulnerability In the generate life-cycle of smart contract execution, the fifth step, "Invoke reveal to calculate jackpot", the DApp should check whether the caller is actually the payer. We have such a function *require_auth()* in EOSIO, it is used to check whether the caller is authorized to call some methods. There is a pre-knowledge here, inlined actions inherit the permissions of the father. Therefore, hackers can execute inlined actions by calling functions to perform operations that the hacker does not have the authority to perform, which leads to security problems.

3.5 Feasibility & Versatility

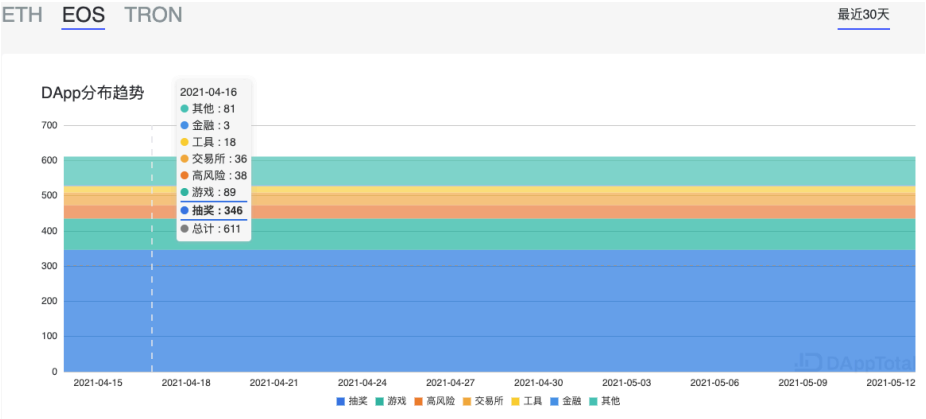

Figure 3.1: DApp distribution trend.
From the figure, we can see that in 2021-04-16, out of a total of 611 DApps, 346 DApps are of the gambling type, and 89 DApps are of the game type.

The above four vulnerabilities are common problems that may exist in EOS smart contracts, rather than special cases. According to research in EOSafe, 88.32% of EOSIO smart contracts are deployed using the transfer function. And Fake EOS and Fake receipt need to use the transfer function. Although Rollback is only found in gambling games,

from the perspective of DappRadar[9] and DAppTotal[10], gambling games are the most popular Dapp. And Missing Permission Check is a common and high-risk vulnerability.

Chapter 4 Research Challenges

4.1 Differences Between History Version

We aim to compare the differences between different versions of the same smart contract. So we are thinking about how to visually and effectively analyze the difference between the two codes.

4.1.1 Compare Codes Line By Line

Compare codes line by line is not a good choice. First, our code is in wasm bytecode format, which is not conducive to reading. Even if converted to wat format, it isn't easy to get specific meaning from the surface for codes similar to assembly language. So obviously, we will not take this approach.

4.1.2 Control Flow Graph

The control flow graph is an abstract representation of a process or program. It is an abstract data structure used in the compiler. It is maintained internally by the compiler and represents all the paths traversed during the execution of a program. It expresses the possible flow of execution of all basic blocks in a process in the form of a graph and can also reflect the process's real-time execution process.

4.1.3 Sybolic execution path

```
1  int twice(int v) {
2      return 2*v;
3  }
4
5  void testme(int x, int y) {
6      z = twice(y);
7      if(z == x) {
8          if(x > y + 10)
9              ERROR;
10     }
11 }
12
13 /* simple driver exercising testme() with sym */
14 int main() {
15     x = sym_input();
16     y = sym_input();
17     testme(x, y);
18     return 0;
19 }
```

Figure 4.1: Sybolic Code Example

Symbolic execution is a means of analyzing a program to determine what inputs cause each part of a program to execute. An interpreter follows the program, assuming symbolic

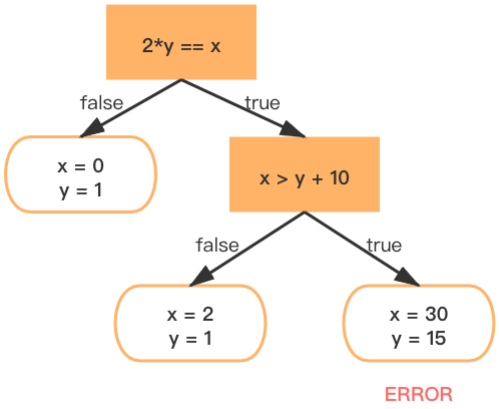

Figure 4.2: Sybolic Execution Tree

values for inputs rather than obtaining actual inputs as normal execution of the program would. It thus arrives at expressions in terms of those symbols for expressions and variables in the program, and constraints in terms of those symbols for the possible outcomes of each conditional branch.

The path constraint generated by symbolic execution makes it easier for us to detect the difference between the two codes.

4.2 Path Explosion

There are two instructions for branch jump in EOSIO, *br_if* and *br_table*. *br_if* will generate two branches. And *br_table* takes a vector of an arbitrary number *n* of label indices as its first immediate, and another label index as its second immediate. This means that *br_table* can generate *n* branches. Once the code has a deep call stack, the complexity of program analysis will increase exponentially.[2]

Under the framework of EOSafe, it has used heuristic pruning to try to solve the above problems, and added *call_depth* to limit the call depth. On this basis, we only focus on some critical functions, and other functions that have little impact on our research content can be ignored.

Finally, set the timeout period to prevent the analysis from being unable to continue due to excessive code complexity.

Chapter 5 Experimental Environment

5.1 Environment

The experiment was run on a server with Ubuntu 18.04.5 LTS x86_64, the CPU was Intel Xeon Gold 6238 (176) @ 3.700GHz, and the memory was 251GB.

The version of python is 3.6.9.

5.2 Limitation

According to the problem described in 4.2, we set the *call_depth* to *2* layers and the *timeout* to *10 minutes*. In the previous experiment, we found that there is almost no difference between the *timeout* set to *20 minutes* and the *timeout* set to *10 minutes*. Only one more EOSIO smart contract vulnerability was detected. So we set the *timeout* to *10 minutes*.

Chapter 6 System Design

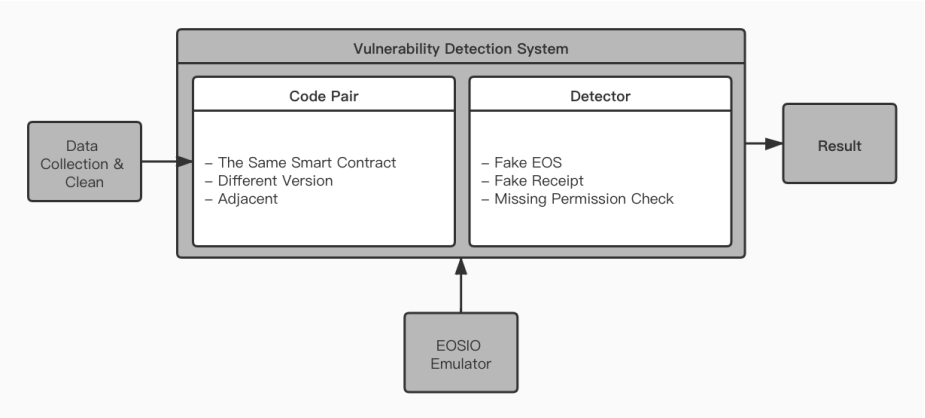

Figure 6.1: System architecture

6.1 Data Collection & Cleaning

For this stage, we collect data for the following stages. We crawl the verified smart contracts, which are the open-sourced contracts. We have synchronized the EOS mainnet nodes. The complete history node provides an analysis layer on top of historical data, making DApp easier to process and use. In order to process this type of data, nodes need to run full history plugins and solutions, which require a lot of RAM and high maintenance costs. DApp developers can use the complete history node to track and query the data of a specific user account, and we can extract all operations or transactions associated with a specific account to obtain every update of a specific smart contract.

Although there are some public EOS data sets on the Internet, we need the special data required for the experiment which are different versions of the same smart contract. And only the code API of the requesting node cannot get the historical information, it only returns the latest code. So we need to filter out the EOSIO 'setCode' action, grab all the 'setCode' information of the related account, then parse the data packet, and get the Wasm code through decoding.

But there are too many node history records, and there are many repetitive data, it isn't very meaningful for the experiment. So we chose to use the records before 2019-03-27, the data volume is top 50 million blocks, and the size is about 91GB.

6.2 Difference comparison

First, we select the cleaned data and choose EOSIO smart contracts with multiple versions.

Then, among multiple versions, we select two adjacent versions in turn for different analysis. The reason why two adjacent versions are chosen instead of any two versions is that one is that the number of codes to be compared increases, and the other is that the code modification of the two adjacent versions can best reflect the modified content. If the old version contains vulnerabilities and the new version fixes the vulnerabilities, we can clearly determine the type of vulnerabilities and how to fix them by analyzing the differences between the two codes.

After the two versions of the code are selected, emulate the code is executed, and the control flow diagram and symbolic execution path of the two versions are produced respectively. The control flow graph is convenient for experimenters to observe the difference of the code visually, and the symbolic execution path and some recorded states are used to automatically analyze whether there are vulnerabilities and the types of vulnerabilities.

We have implemented three types of vulnerability detectors in the experiment. They are Fake EOS Detector, Fake Receipt Detector, and Missing Permission Check Detector. The reason why the Rollback detector is not implemented is that the detection complexity of the rollback is extremely high. Generally, one can only try to substitute a particular method to perform the detection. Due to the time factor, the Rollback detector is temporarily not implemented.

6.3 Frame

We use Octopus and EOSafe as framework. Octopus is a security analysis framework for WebAssembly module and Blockchain Smart Contract. The purpose of Octopus is to provide an easy way to analyze closed-source WebAssembly modules and smart contracts bytecode to understand deeper their internal behaviors.

Chapter 7 Secruity Issues Detector

7.1 Fake EOS Detector

According to the life-cycle of smart contract execution in 2.2.3 and the cause of the Fake EOS vulnerability we know in 3.1, we found that the smart contract containing the vulnerability of Fake EOS must be called by the hacker to transfer the function, and successfully bypassed the code. An examination. Then we only need to perform symbolic execution on the apply function, generate a symbolic execution path, and determine whether the difference constraints between the new and the old version is:

$$\begin{aligned} & action == transfer, (\text{ in apply function}) \\ \bigwedge\ & code == account\ name,\ not\ self, (\text{ in apply function}) \end{aligned} \tag{7.1}$$

If the new and old versions of the EOSIO smart contract meet the above conditions, we will consider the old version of the EOSIO smart contract to be fragile and may contain the vulnerability of Fake EOS, and it has been fixed in the new version.

We only need to analyze the differences in the paths related to transfer, and the differences in other paths will hardly cause Fake EOS. This is a direction that can be optimized for pruning.

7.2 Fake Receipt Detector

According to the life-cycle of smart contract execution in 2.2.3 and the cause of the vulnerability of Fake Receipt we know in 3.2, we found that the critical factor of the smart contract that contains the vulnerability of Fake Receipt is insufficient verification in the transfer function. Then our approach is straightforward: find all the functions that may call/jump to the transfer function, perform symbolic execution, generate a symbolic execution path, and determine whether there are operations to modify the state of the blockchain between the new and old versions of the suspicious functions, and whether there is a function such as *eosio_assert()* to interrupt the path that does not jump.

$$\begin{aligned} & self == to,\ modification, (\text{ in suspicious function}) \\ \bigwedge\ & self! = to,\ eosio\_assert(), (\text{ in suspicious function}) \end{aligned} \tag{7.2}$$

7.3 Missing Permission Check Detector

According to the life-cycle of smart contract execution in 2.2.3 and the cause of the Missing Permission Check vulnerability we know in 3.4, we only need to pay attention

to whether functions check permissions before performing sensitive operations. And all functions are connected with apply function in the call graph. So we can perform symbolic execution on the apply function, find all functions, and check whether a permission check has been added before the instructions for some sensitive operations that we have counted. If permission check is added, we can think that the old version of EOSIO smart contract has the Missing Permission Check vulnerability, and it has been fixed in the new version.

$$\begin{aligned} sensitive\ operations &== db\_update_i64(), etc. \\ \exists sensitive\ operations &\bigwedge no\ require\_auth() \end{aligned} \tag{7.3}$$

Chapter 8 Experiment Result

8.1 Dataset

We use the EOSIO records before 2019-03-27, the data volume is top 50 million blocks. In the data set after data cleaning, there are *1373* smart contracts, and *610* smart contracts have multiple versions. There are *2849* pairs of old and new version codes in total.

8.2 The Accuracy of Vulnerability Detection

In order to evaluate the detection level of this system, we inquired about the reports issued by blockchain security companies such as PeckShield[11] and SlowMist[12] to confirm that the vulnerabilities we found were accurate and real. And at the same time, use EOSafe to assist judgment. EOSafe is an excellent detection framework. In its previous experiments, its precision and recall were 100% and 96.30%, respectively. We believe that EOSafe can serve as an effective referee.

Through comparison, we found that EOSIO smart contracts with vulnerabilities detected by this system can all be confirmed on EOSafe. However, there is a phenomenon of underreporting in this system, and the general reasons are as follows:

- The timeout setting time is relatively short, resulting in *587* EOSIO smart contracts that cannot be fully analyzed (the three detectors cannot be successfully executed within the specified time).

- The detection conditions of the detector need to be strengthened.

- Because the data is the first 50 million blocks, and the last block is dated 2019-03-27, EOSIO has just developed, many vulnerabilities have not been discovered, and no developers have fixed these blockchain security issues, so there are a lot of historical version codes. Both have the same problem, and it is impossible to check out the vulnerabilities by comparing the differences.

8.3 Distribution of vulnerabilities

Type	# Candidates	# Vulnerable (%)
Fake EOS	23	1(4.35%)
Fake Receipt	23	2(8.70%)
Missing Permission Check	23	4(17.39%)
Total	23	7(30.43%)

Table 8.1: Vulnerabilities Detection Result.
Candidates are smart contracts that can run under specified time and conditions. Vulnerable (%) is the proportion of smart contracts that have vulnerabilities in the smart contracts that are detected.

Since there are multiple versions in an EOSIO smart contract, as long as there is a pair of old and new versions of the code and vulnerabilities are detected, we can consider the EOSIO smart contract to be fragile and insecure. We found that among the EOSIO smart contracts that can fully execute the 3 vulnerability detectors, 30.43% of the smart contracts have security issues. The most frequent problem is Missing Permission Check, followed by Fake Receipt, and the least one is Fake EOS.

Among the smart contracts in the data set, the smart contract with the most historical versions is guydgnjygige, a popular gaming platform. Of course it also has the most vulnerable versions.

By manually inspecting the EOSIO smart contract for which vulnerabilities were detected, we found the following problems:

1. The vulnerabilities repaired by DApp developers are generally very fast. For example, the smart contract danakilblock introduced the vulnerability at 2018-09-05T 19:41:37.500, and updated the version at 2018-09-05T 20:00:42.000 to solve the Missing Permission Check.

2. When DApp developers update the code, they may fix vulnerabilities in the code and update the business code more often. Of course, as long as the code is updated, new vulnerabilities are likely to be introduced. According to our manual inspection, there are two EOSIO smart contracts that have not been detected vulnerabilities in the old version, but were detected after updating the business code. This reminds DApp developers to avoid creating new vulnerabilities as much as possible when developing new features.

3. After relevant blockchain security companies released security reports, DApp developers updated smart contracts more intensively.

Conclusions

In this paper, we proposed the idea of searching for security issues in EOSIO smart contracts by comparing historical versions, and implemented three types of vulnerability detectors. According to the experimental results, in the current EOSIO ecological environment, there are still many smart contracts facing the situation of being exploited by hackers to obtain illegal benefits. Our research results have further verified that EOSIO's blockchain security needs to be improved.

References

[1] LE Q. How hackers attack EOS contracts and ways to prevent it[EB/OL]. 2018. https://medium.com/leclevietnam/hacking-in-eos-contracts-and-how-to-prevent-it-b8663c8bffa6/.

[2] HE N, ZHANG R, WANG H, et al. EOSAFE: Security Analysis of EOSIO Smart Contracts[C/OL] // 30th USENIX Security Symposium (USENIX Security 21). [S.l.] : USENIX Association, 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/he-ningyu.

[3] WANG D, JIANG B, CHAN W K. WANA: Symbolic Execution of Wasm Bytecode for Cross-Platform Smart Contract Vulnerability Detection[J]. ArXiv, 2020, abs/2007.15510.

[4] EOSIO Developer Portal[EB/OL]. . https://developers.eos.io/.

[5] WebAssembly 1.0 has shipped in 4 major browser engines.[EB/OL]. . https://webassembly.org/.

[6] "Transactions Protocol," [EB/OL]. . https://developers.eos.io/welcome/v2.0/protocol/transactions_protocol.

[7] "Fake EOS Attack" Upgraded, 60K EOS Tokens Lost by EOSCast[EB/OL]. . https://blog.peckshield.com/2018/11/02/eos/.

[8] Roll Back Attack about blacklist in EOS[EB/OL]. . https://slowmist.medium.com/roll-back-attack-about-blacklist-in-eos-adf53edd8d69.

[9] "DappRadar, a DApp browser," [EB/OL]. 2020. https://dappradar.com/.

[10] DAppTotal,[EB/OL]. 2019. https://dapptotal.com/.

[11] "Blogs about blockchain security events," [EB/OL]. 2020. https://blog.peckshield.com/blog.html.

[12] "Blockchain security events," [EB/OL]. . https://hacked.slowmist.io/en/.

[13] CADAR C, SEN K. Symbolic execution for software testing[J]. Communications of the ACM, 2013.

[14] CHEN J. Finding Ethereum Smart Contracts Security Issues by Comparing History Versions[J]. 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020 : 1382 – 1384.

[15] HUANG Y, JIANG B, CHAN W K. EOSFuzzer: Fuzzing EOSIO Smart Contracts for Vulnerability Detection[J]. ArXiv, 2020, abs/2007.14903.

[16] pventuzelo/octopus[EB/OL]. . https://github.com/pventuzelo/octopus.

Acknowledgements

I cannot express enough thanks to my supervisors for their continued support and encourangement: Dr. Yepang Liu, Dr. Haoyu Wang. I offer my sincere appreciation for the learning opportunities provided by my supervisors.

My completion of this project could not have been accomplished without the support of my classmates.

Thanks to my parents as well. My heartfelt thanks.

Yubin Hu

May, 2021