All roads lead to Rome: Many ways to double spend your cryptocurrency

Zhiniang Peng of 360 Core Security
Yuki Chen of 360 Vulcan Team
@Zer0con 2019

360 INTERNET SECURITY CENTER

Who we are

Zhiniang Peng

Ph.D. in cryptography

Security researcher @Qihoo 360

Twitter: @edwardzpeng

Research areas:

- Software security
- Applied cryptography
- Threat hunting

Who we are

Yuki Chen (@guhe120): Sr. Director & Bug Hunter @ 360 Vulcan Team

CVE-2014-0290,CVE-2014-0321,CVE-2014-1753,CVE-2014-1769,CVE-2014-1782,CVE-2014-1804,CVE-2014-2768,
CVE-2014-2802,CVE-2014-2803,CVE-2014-2824,CVE-2014-4057,CVE-2014-4092,CVE-2014-4091,CVE-2014-4095,
CVE-2014-4096,CVE-2014-4097,CVE-2014-4082,CVE-2014-4105,CVE-2014-4129,CVE-2014-6369,CVE-2015-0029,
CVE-2015-1745,CVE-2015-1743,CVE-2015-3134,CVE-2015-3135,CVE-2015-4431,CVE-2015-5552,CVE-2015-5553,
CVE-2015-5559,CVE-2015-6682,CVE-2015-7635,CVE-2015-7636,CVE-2015-7637,CVE-2015-7638,CVE-2015-7639,
CVE-2015-7640,CVE-2015-7641,CVE-2015-7642,CVE-2015-7643,CVE-2015-8454,CVE-2015-8059,CVE-2015-8058,
CVE-2015-8055,CVE-2015-8057,CVE-2015-8056,CVE-2015-8061,CVE-2015-8067,CVE-2015-8066,CVE-2015-8062,
CVE-2015-8068,CVE-2015-8064,CVE-2015-8065,CVE-2015-8063,CVE-2015-8405,CVE-2015-8404,CVE-2015-8402,
CVE-2015-8403,CVE-2015-8071,CVE-2015-8401,CVE-2015-8406,CVE-2015-8069,CVE-2015-8070,CVE-2015-8440,
CVE-2015-8409,CVE-2015-8047,CVE-2015-8455,CVE-2015-8045,CVE-2015-8441,CVE-2016-0980,CVE-2016-1015,
CVE-2016-1016,CVE-2016-1017,CVE-2016-4120,CVE-2016-4160,CVE-2016-4161,CVE-2016-4162,CVE-2016-4163,
CVE-2016-4185CVE-2016-4249,CVE-2016-4180,CVE-2016-4181,CVE-2016-4183,CVE-2016-4184,CVE-2016-4185,
CVE-2016-4186,CVE-2016-4187,CVE-2016-4233,CVE-2016-4234,CVE-2016-4235,CVE-2016-4236, CVE-2016-4237,
CVE-2016-4238,CVE-2016-4239,CVE-2016-4240,CVE-2016-4241,CVE-2016-4242, CVE-2016-4243,CVE-2016-4244,
CVE-2016-4245,CVE-2016-4246,CVE-2016-4182,CVE-2016-3375,CVE-2017-3001,CVE-2017-3002,CVE-2017-3003,
CVE-2017-0238,CVE-2017-0236,CVE-2017-8549,CVE-2017-8619, CVE-2017-11887,CVE-2017-11913,CVE-2017-11846,
CVE-2017-8753,CVE-2018-0951,CVE-2018-0953,CVE-2018-0954,CVE-2018-0955,CVE-2018-1022,CVE-2018-8144,
CVE-2018-8122,CVE-2018-0981,CVE-2018-0987,CVE-2018-0988,CVE-2018-0989,CVE-2018-0994,CVE-2018-0997,
CVE-2018-1000,CVE-2018-1004,CVE-2018-0872,CVE-2018-0834,CVE-2018-0798,CVE-2018-0801,CVE-2018-0802,
CVE-2018-0804,CVE-2018-0805,CVE-2018-0806,CVE-2018-0807,CVE-2018-0812,CVE-2018-0845,CVE-2018-0848,
CVE-2018-0849,CVE-2018-0862,CVE-2018-8367, CVE-2018-8544, CVE-2018-8618, CVE-2019-0567, CVE-2019-0591,
CVE-2019-0605, CVE-2019-0606, CVE-2019-0610, CVE-2019-0649, CVE-2019-0651, CVE-2019-0652, CVE-2019-0655,
CVE-2019-0658, CVE-2019-0592, CVE-2019-0611, CVE-2019-0639, CVE-2019-0665, CVE-2019-0666, CVE-2019-0667,
CVE-2019-0680, CVE-2019-0680, CVE-2019-0756, CVE-2019-0771, CVE-2019-0772, CVE-2019-0784, CVE-2019-0752,
CVE-2019-0753,CVE-2019-0790,CVE-2019-0791,CVE-2019-0792,CVE-2019-0793,CVE-2019-0794,CVE-2019-0795,
CVE-2019-0806,CVE-2019-0810,CVE-2019-0829,CVE-2019-0835,CVE-2019-0842,CVE-2019-0862

About the topic

Introduction to blockchain system

Attack surface of blockchain system

Double spend attack on

- CCA attack on BFV
- d

Other issues

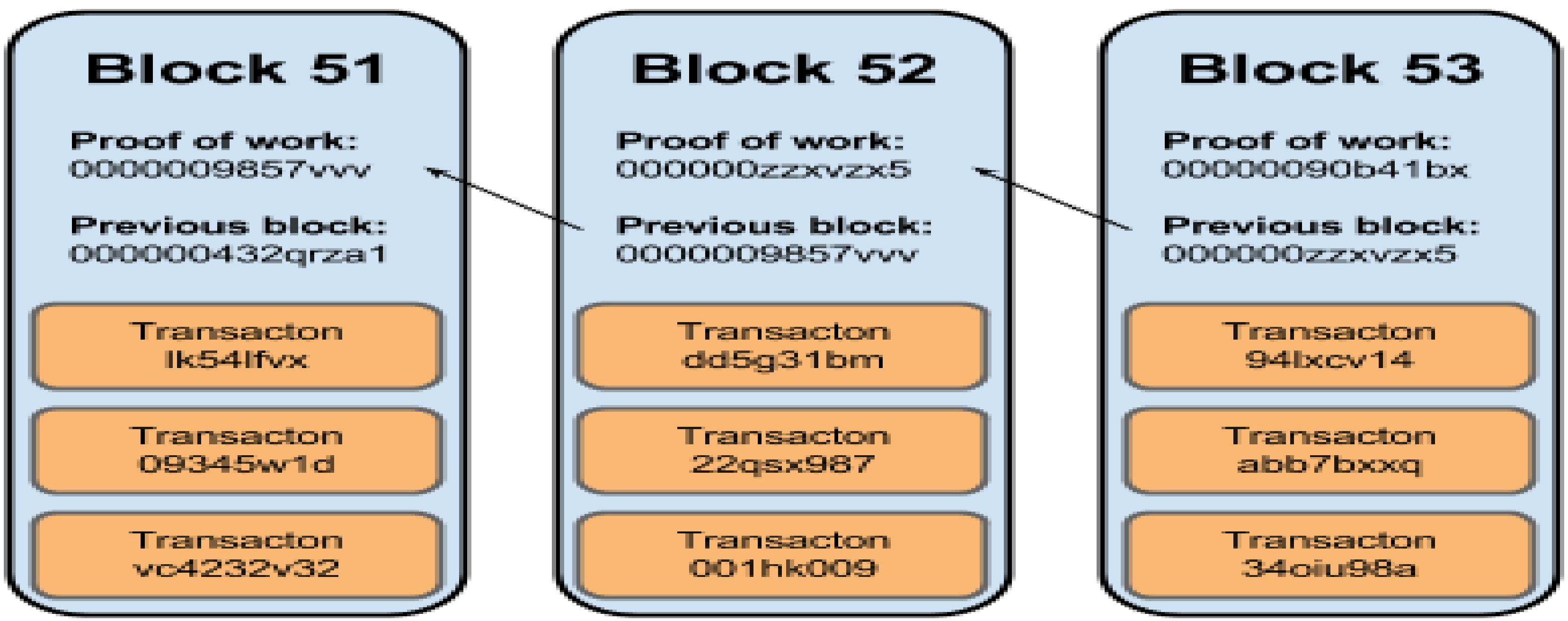
Conclusion

Introduction to Blockchain

Chain of Blocks

A blockchain is a chain of blocks….

- Proof of work (POW)
- Miners compute the hash
- Transactions in each block (ledger)
- Irreversible and unchangeable



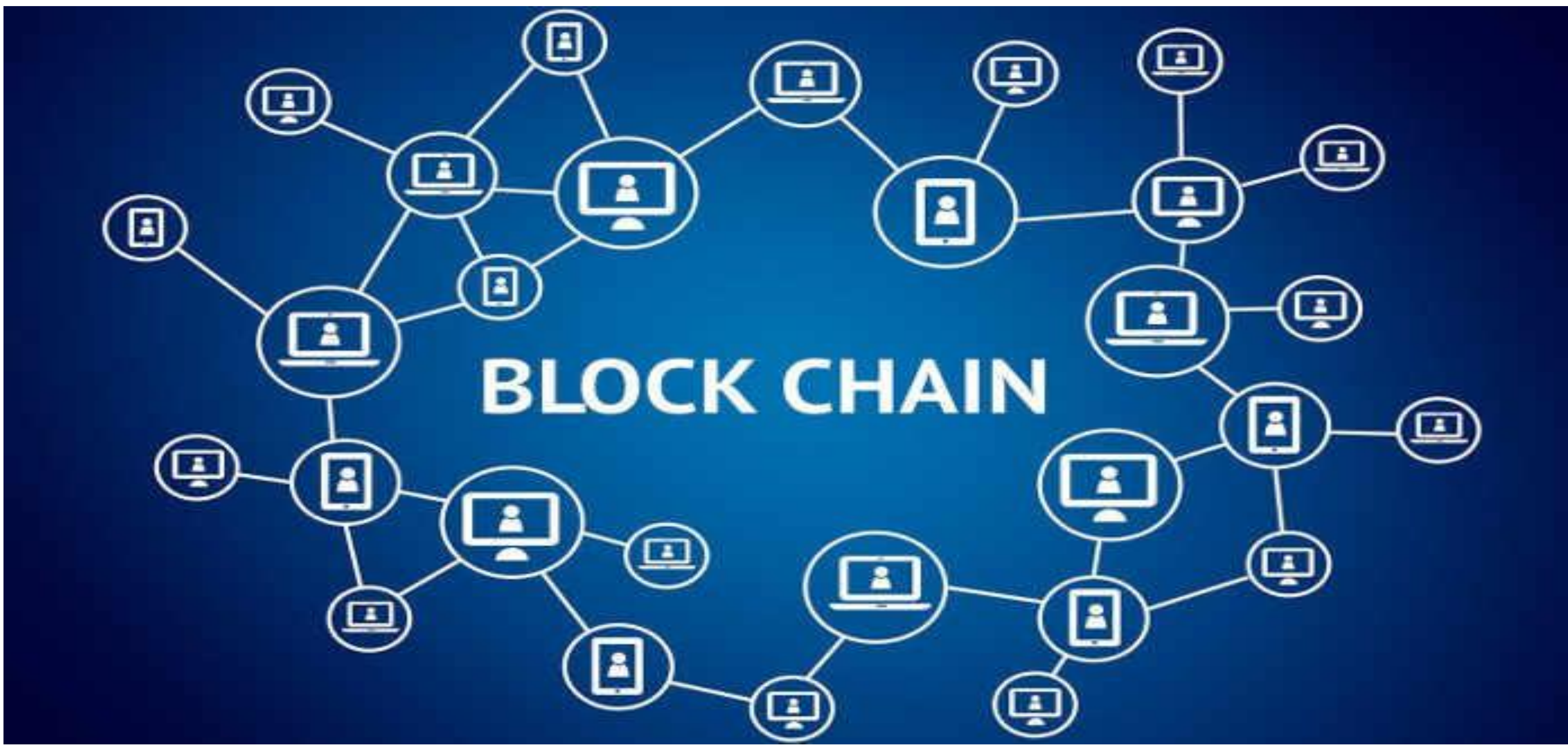
Decentralized network

P2P network

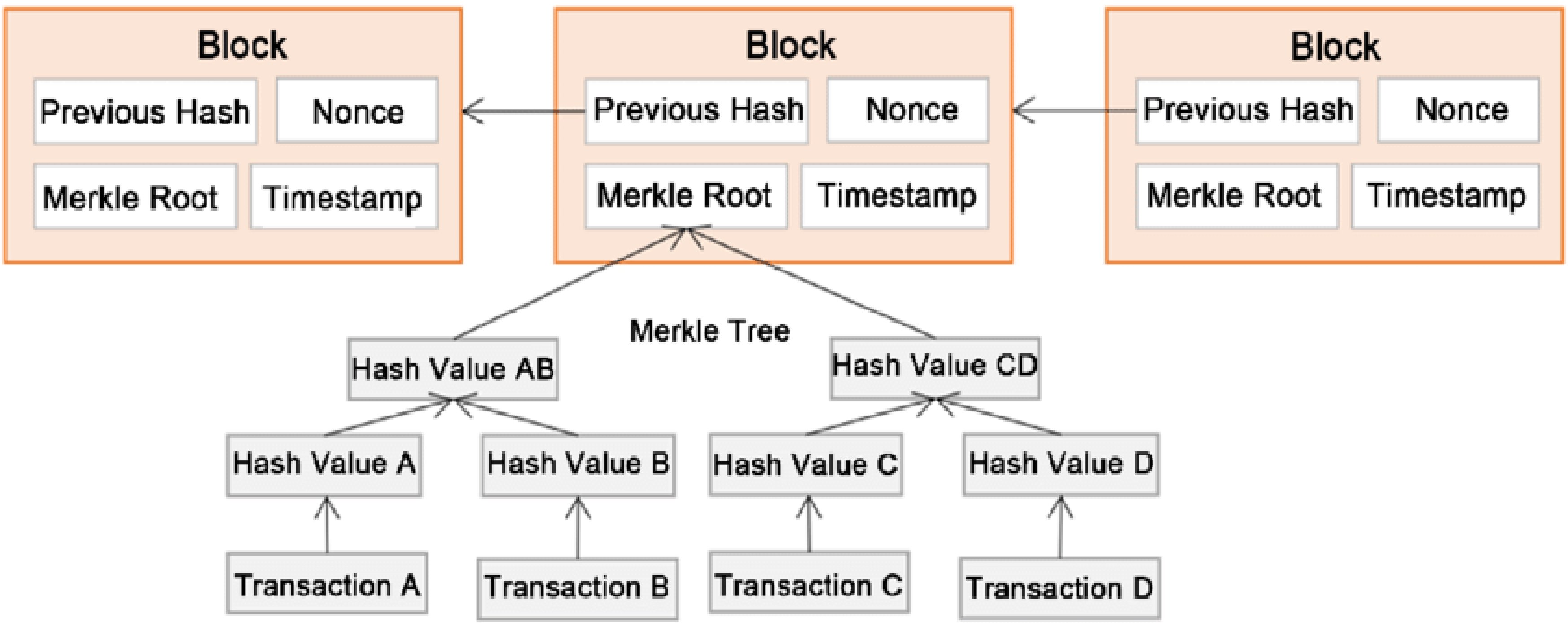
Every node is born equal

Agree with some rules

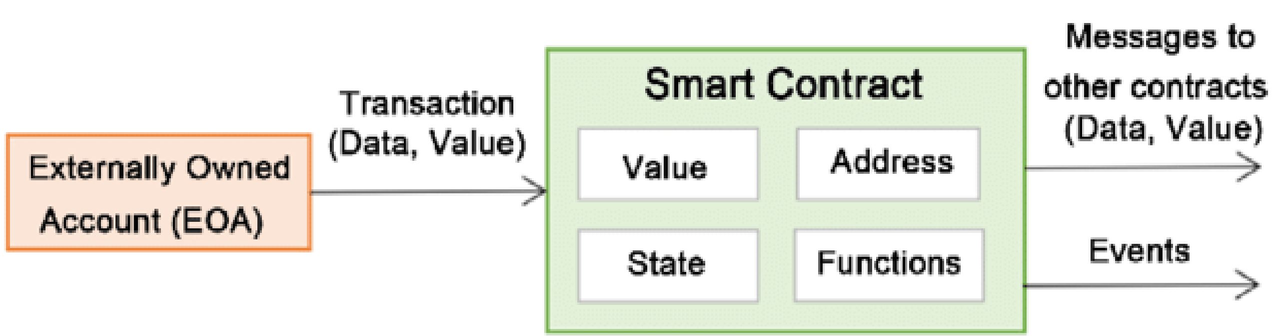
Decentralized



Smart Contract



(a)



(b)

Other kinds of blockchain

Above is a typical cryptocurrency system

Bitcoin, Ethereum

There are also many other kinds of blockchain

other forms

other applications

other consensus mechanism

We will focus on double spend on cryptocurrency in this talk

Attack surface of Blockchain

Attack surfaces

Blockchain security

Hot topic (last year) & emergency

Attack surfaces of public blockchain:

Smart contract virtual machine

Consensus mechanism

P2P network

Smart contract

Smart contract virtual machine

Some contract:

Turing complete programming language.

Run on every full node of the public chain.

Virtual Machine.

Security risk in nature:

Suppose you can run your JavaScript on everyone's computer.

The ideal place to achieve every hackers' dream:

One bug to rule the world, or crash the world.

Example: NEO VM DoS

Written in C# (memory safety)

Try catch everything

```
try
{
    ExecuteOp(opcode, CurrentContext);
}
catch
{
    State |= VMState.FAULT;
}
```

Example: NEO VM DoS

```
private void SerializeStackItem(StackItem item, BinaryWriter writer)
{
    switch (item)
    {
        case ByteArray _:
            writer.Write((byte)StackItemType.ByteArray);
            writer.WriteVarBytes(item.GetByteArray());
            break;
        case VMArray array:
            writer.WriteVarInt(array.Count);
            foreach (StackItem subitem in array)
                SerializeStackItem(subitem, writer, serialized);
            break;
```

Attack: Serialize(a[a])

Stack-overflow, cannot be caught.

Crash all the world.

Consensus Mechanism

Crucial for a blockchain

Make sure everyone agree with the same blockchain.

May be insecure by design:

All PoS is vulnerable to long range attack.

PoW, may not secure as you think.

May have bugs in implementation:

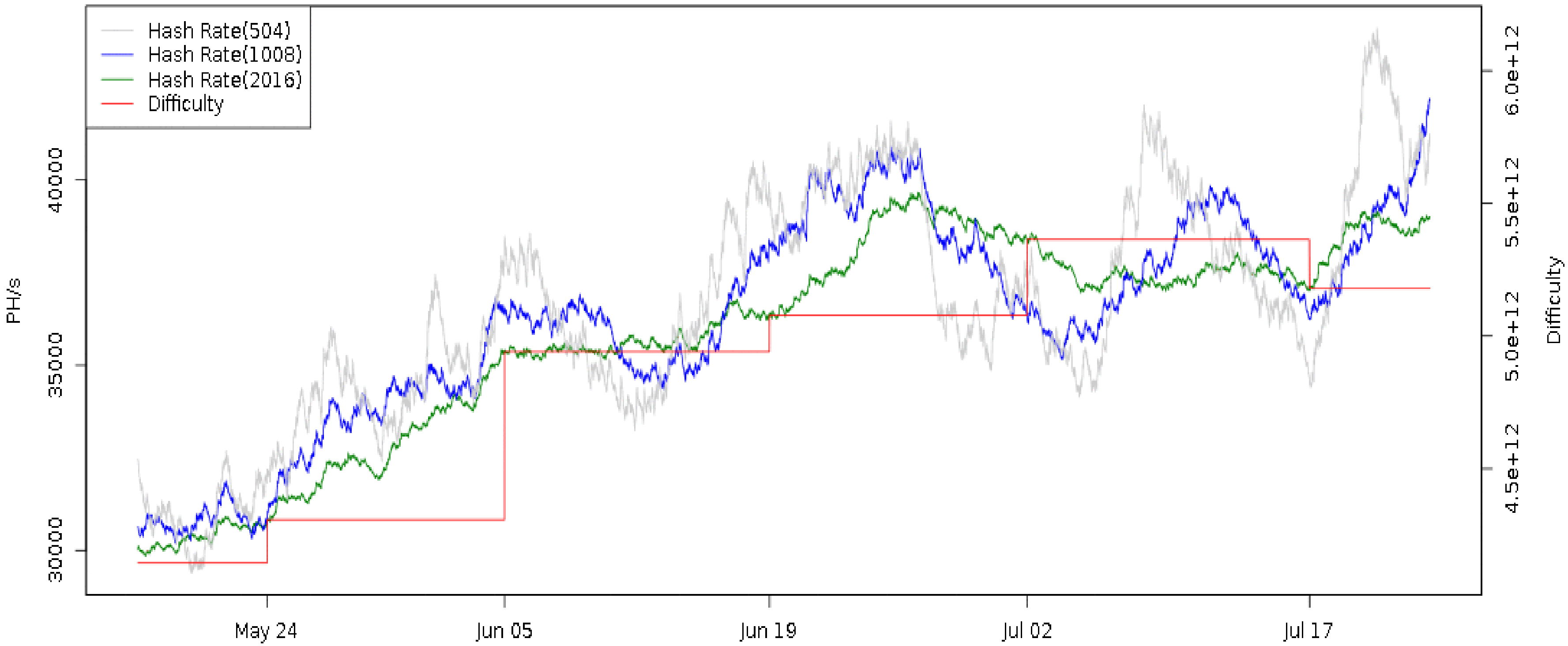
Software bugs.

Difficulty Adjustment Algorithm

Every M blocks ($M = 2016$ for Bitcoin) the difficulty is recalculated as

$$D_{i+1} = D_i \cdot \frac{M \cdot |\Delta|}{S_m}$$

Bitcoin Hash Rate vs Difficulty (2 Months)



Power based attack

Coin Hopping attack(Gain unfair income)

Of course ,attacker can

P2P network

Peer to peer network:

Block processing& transaction processing

Every node is both a server and a client.

Use Peer discovery mechanism to find all the peers in the network.

Again, one bug to kill them all.

RPC protocol:

Execute a specified procedure with supplied parameters.

May have some dangerous procedures.

Should not be accessed by untrusted users.

Exmaple:Json Parse in EOS, NEO

```diff
-        internal new static JArray Parse(TextReader reader)
+        internal new static JArray Parse(TextReader reader, int max_nest)
         {
+            if (max_nest < 0) throw new FormatException();
             SkipSpace(reader);
             if (reader.Read() != '[') throw new FormatException();
             SkipSpace(reader);
             JArray array = new JArray();
             while (reader.Peek() != ']')
             {
                 if (reader.Peek() == ',') reader.Read();
-                JObject obj = JObject.Parse(reader);
+                JObject obj = JObject.Parse(reader, max_nest - 1);
                 array.items.Add(obj);
                 SkipSpace(reader);
             }
```

Smart Contract

Many security accidents in real world:

Ethereum and EOS smart contract

Integer overflow, random number vulnerability, logical bugs ……

Gambling games , Tokens, financial scam …….

Cryptocurrency make bank robbery great again.

Many ways to double spent your cryptocurrency

Double spend attack

Above slides show cryptocurrency system is sensitive to security vulnerabilities

Denial-of-service, Information leakage, logic vulnerabilities all has serious consequences

There are more serious vulnerabilities

Double spend attack

The most critical vulnerability in cryptocurrency system

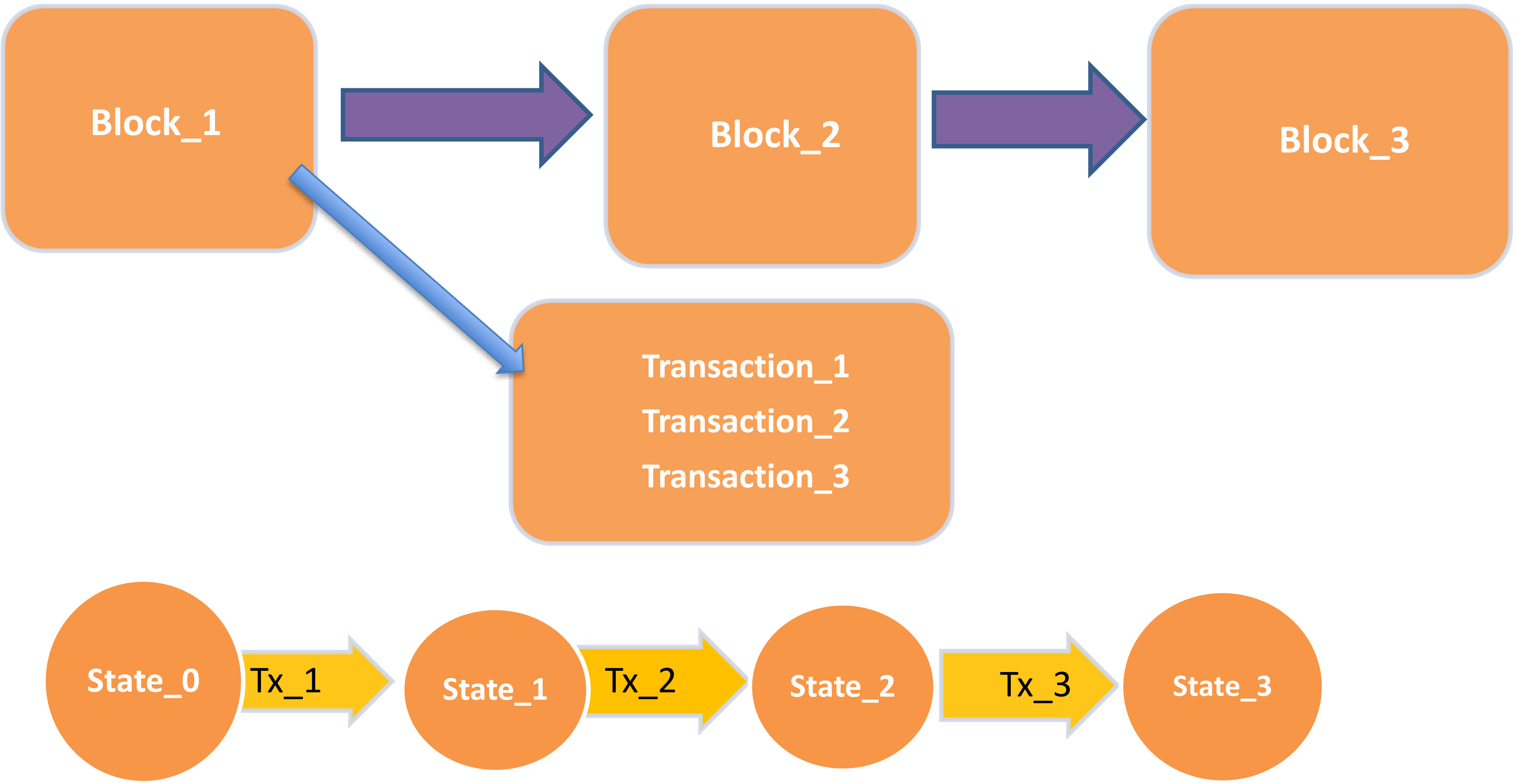
Discussion of double spend attack seems to still concentrate on 51% Attacks.

Many other ways to double spent your cryptocurrency

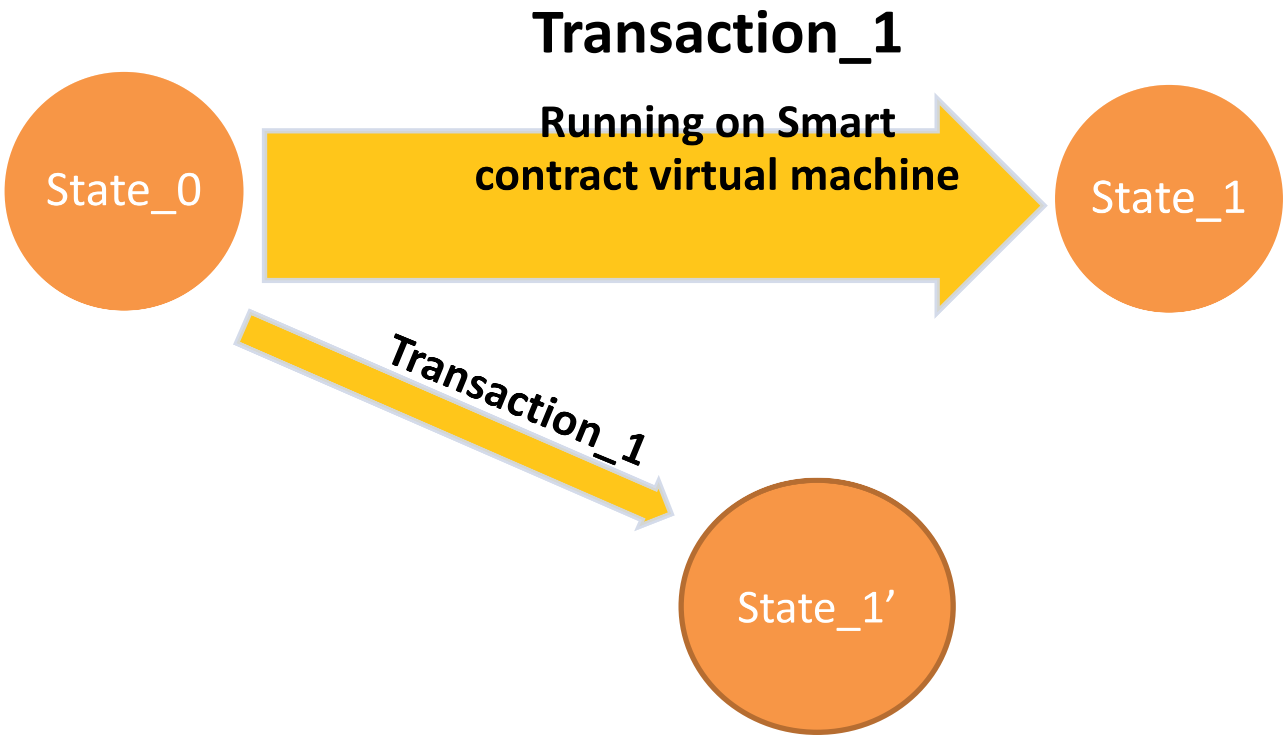
Layers of double spend attacks

Transaction	• Inconsistent execution of Smart contract
Block	• Logic bugs in processing Block
Consensus	• Attack on Consensus Mechanism
Network	• Network split

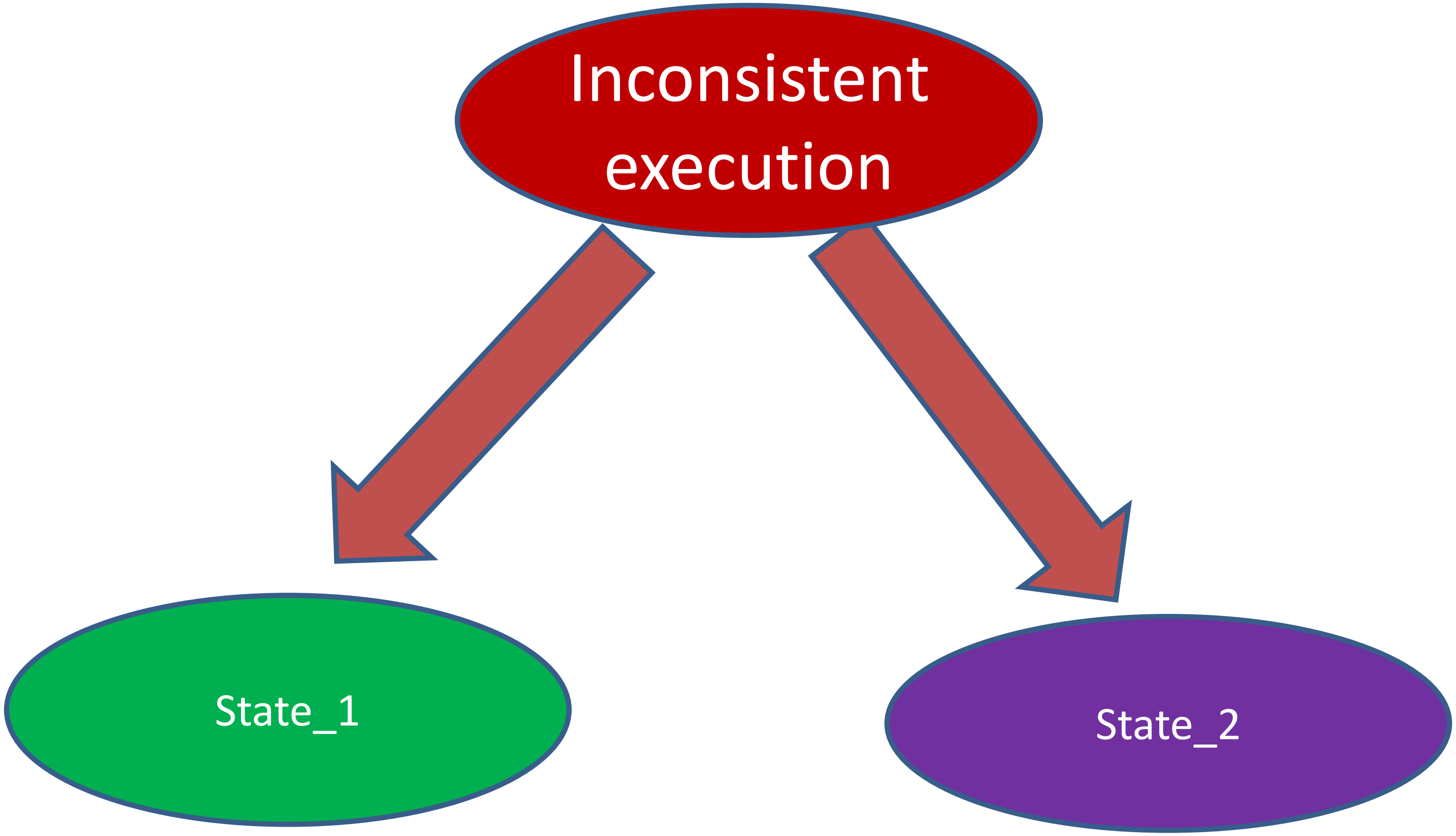
State machine replication



Inconsistent execution



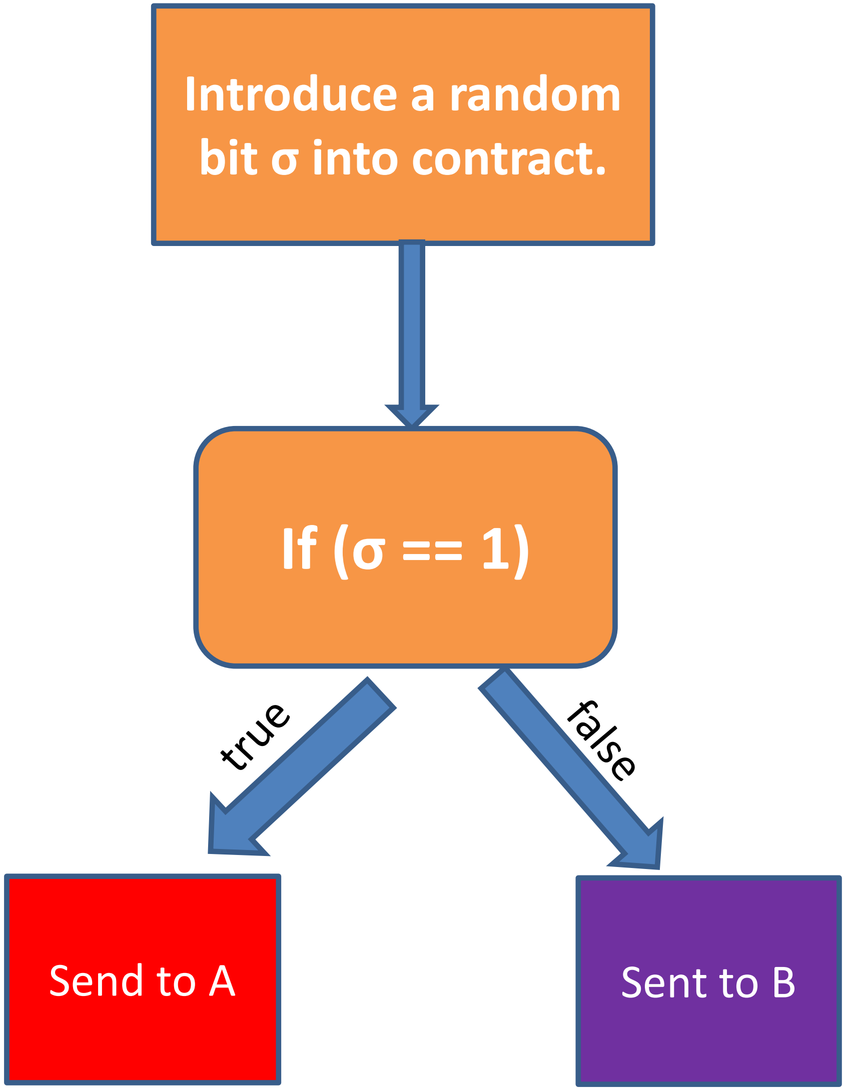
Double spend based on inconsistent execution



Send money to A

Send the same money to B

Inconsistency introduce by randomness



Out-of-bound memory write

Out-of-memory write vulnerability

Traditional memory corruption bug

Lead to remote command execution

Out-of-memory write in EOS virtual machine

EOS smart contract (webaseembly bytecode)

Webaseembly engine is written in C++

We can use it to introduce randomness

Example: Out-of-bounds write in EOS leads to Remote-Code-Execution

2 libraries/chain/webassembly/binaryen.cpp View

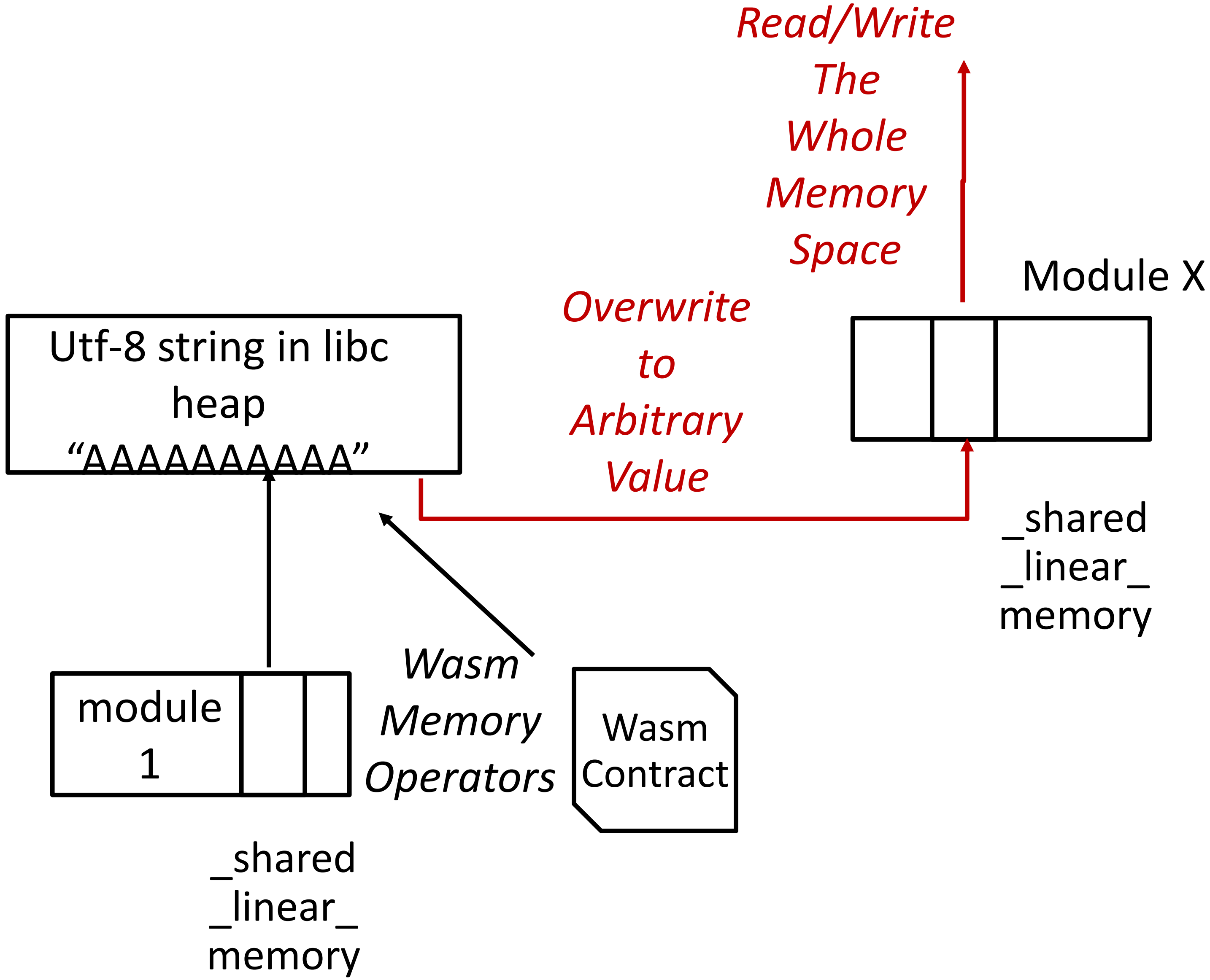
```diff
@@ -73,7 +73,7 @@ std::unique_ptr<wasm_instantiated_module_interface> binaryen_runtime::instantiat
      table.resize(module->table.initial);
      for (auto& segment : module->table.segments) {
         Address offset = ConstantExpressionRunner<TrivialGlobalManager>(globals).visit(segment.offset).value.geti32();
-        assert(offset + segment.data.size() <= module->table.initial);
+        FC_ASSERT(offset + segment.data.size() <= module->table.initial);
         for (size_t i = 0; i != segment.data.size(); ++i) {
            table[offset + i] = segment.data[i];
         }
```

1 comment on commit ea89dce

guhe120 commented on ea89dce 5 hours ago

Hi, there is still some problem with this patch. in 32-bits process, offset + segment.data.size() could overflow and bypass the FC_ASSERT check

Example: Out-of-bounds write in EOS leads to Remote-Code-Execution (.Cont)



Out-of-bound memory read

In traditional cybersecurity

Information discourse

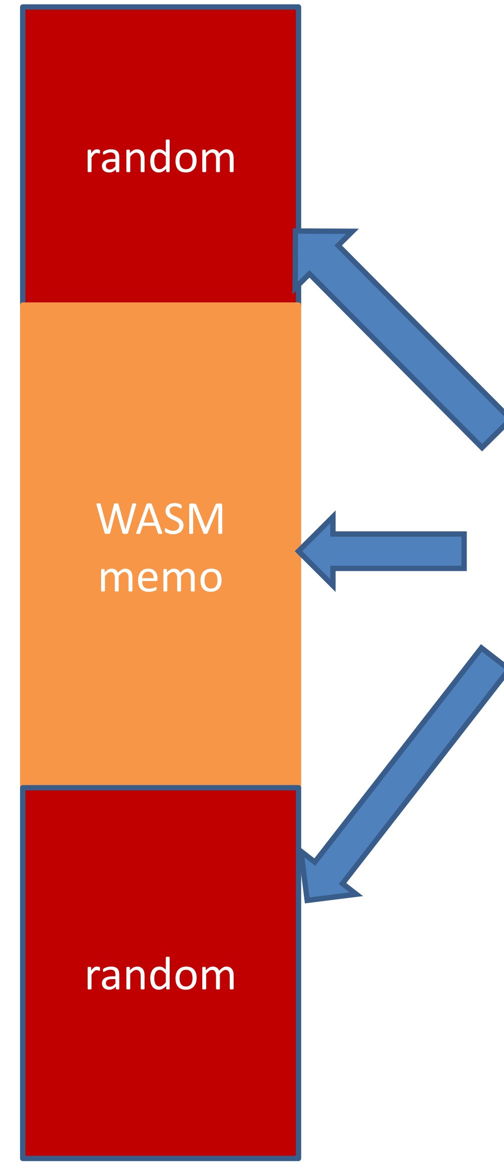
Help to bypass mitigations measure

In blockchain virtual machine

EOS smart contract (webaseembly bytecode)

Webaseembly engine is written in C++

We can use it to introduce randomness



Example: Out-of-bounds Read When Process Array Type

```
@@ -157,6 +157,7 @@ class binaryen_runtime : public eosio::chain::wasm_runtime_interface {
   template<typename T>
   inline array_ptr<T> array_ptr_impl (interpreter_interface* interface, uint32_t ptr, uint32_t length)
   {
+     FC_ASSERT( length < INT_MAX/(uint32_t)sizeof(T), "length will overflow" );
      return array_ptr<T>((T*)(interface->get_validated_pointer(ptr, length * (uint32_t)sizeof(T))));
   }
```

- When calling from wasm to the VM, it should verify all the parameters to make sure the memory access is in bound
- There is an integer overflow vulnerability in the checks
- We can read out of the bounds of wasm memory, which means we can have random values (values based on current process's memory state)

Uninitialized Memory

In traditional cybersecurity

Information leakage

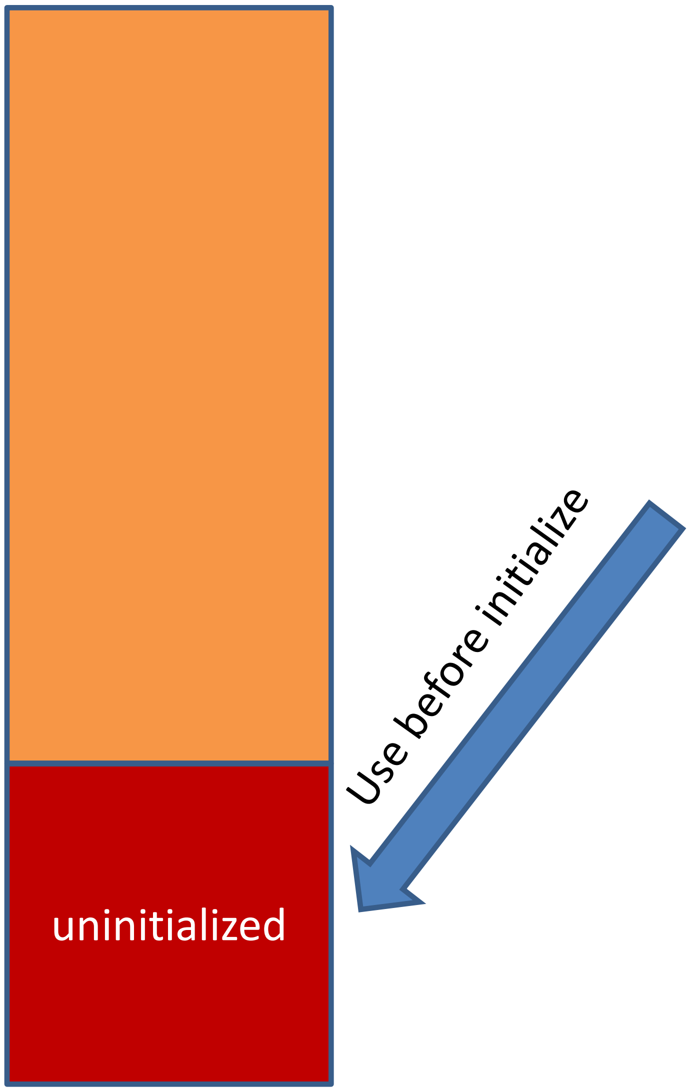
Type confusion

Help to bypass mitigations measure

In blockchain virtual machine

Directly lead to double spend

Introduce randomness



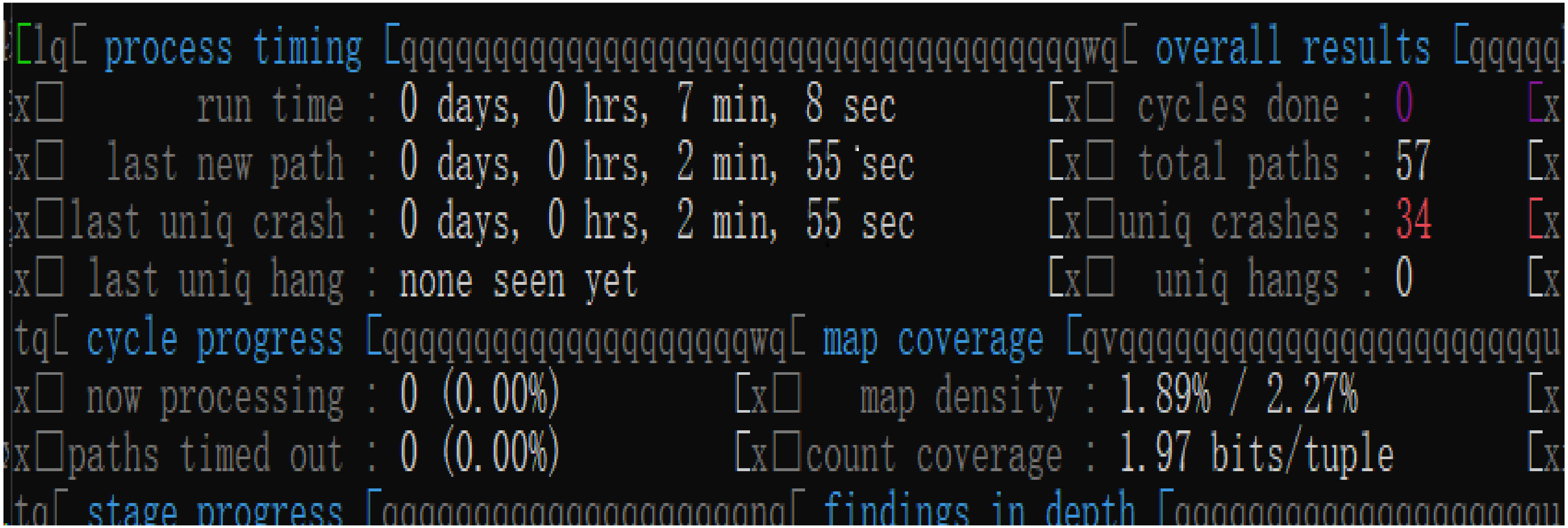
Example: Uninitialized Memory Read Wasm memory

```diff
1 libraries/chain/include/eosio/chain/webassembly/binaryen.hpp

@@ -114,6 +114,7 @@ struct interpreter_interface : ModuleInstance::ExternalInterface {
       }

       void growMemory(Address old_size, Address new_size) override {
+         memset(memory.data + old_size.addr, 0, new_size.addr - old_size.addr);
          current_memory_size += new_size.addr - old_size.addr;
       }

```

- All contracts share the same wasm memory block
- A contract calls grow_memory to allocate memory it needed
- grow_memory does not zero-out the memory before return it to the caller, so the returned memory may contain data written by previous contracts, which is not determinate

Fuzzing EOS VM with AFL

34 unique crashes in 7 mins:



All exploitable!

Beyond randomness

Inconsistent execution does not depend entirely on randomness

Logic vulnerability may also result in inconsistency

Subjective or objective differences are also ways to inconsistency

logical bugs

Inconsistency can be introduced by logical bugs

- In deterministic syscall
- Insufficient check
- other logical issuses

Example: MagnaChain ToString function

```c
static int luaB_tostring (lua_State *L) {
  luaL_checkany(L, 1);
  if (luaL_callmeta(L, 1, "__tostring"))  /* is there a metafield? */
    return 1;  /* use its value */
  switch (lua_type(L, 1)) {
    case LUA_TNUMBER:
      lua_pushstring(L, lua_tostring(L, 1));
      break;
    case LUA_TSTRING:
      lua_pushvalue(L, 1);
      break;
    case LUA_TBOOLEAN:
      lua_pushstring(L, (lua_toboolean(L, 1) ? "true" : "false"));
      break;
    case LUA_TNIL:
      lua_pushliteral(L, "nil");
      break;
    default:
      lua_pushfstring(L, "%s: %p", luaL_typename(L, 1), lua_topointer(L, 1));
      break;
  }
  return 1;
} « end luaB_tostring »
```

```lua
function tostring_poc()
    poc_t = {"a","b"}
    return (tostring(poc_t))
end
```

{
  "txid": "2c052be0e0d7450f72a068cbd201766530f4829762f9702a0978cbba9bed405f",
  "return": [
    "table: 0x7efd3000c770"
  ]
}

{
  "txid": "2c052be0e0d7450f72a068cbd201766530f4829762f9702a0978cbba9bed405f",
  "return": [
    "table: 0x7efd3000c770"
  ]
}

Difference is another way to inconsistency

Different running environment

- Different platform (windows vs Linux)
- Different hardware (4G memory vs 8G memory)

Different version

- Upgrade

Different implementation

- Python vs C#

Example: inconsistent memcmp return value

```diff
 int memcmp( array_ptr<const char> dest, array_ptr<const char> src, size_t length) {
-    return ::memcmp(dest, src, length);
+    int ret = ::memcmp(dest, src, length);
+    if(ret < 0)
+       return -1;
+    if(ret > 0)
+       return 1;
+    return 0;
 }
```

- The wasm runtime “memcmp” functions directly calls the system’s memcmp function
- What is the return value of memcmp?

Other issues

Other issues may cause inconsistent execution

- Timestamps
- Float point computation
- Random source
- More to discover

Layers of double spend attacks

Transaction	• Inconsistent execution of Smart contract
Block	• Logic bugs in processing Block
Consensus	• Attack on Consensus Mechanism
Network	• Network split

Hardness of Block Processing

Block1

Block2

Check transactions

- Is the reference spend before
- Is the amount of money correct
- Is the TX conflict with other TX in the block
- Is the TX have all the signature needed
- Is the TX legal
- …….

Check Block

- Is the block satisfy the consensus rule
- Is the block conflict with other block
- Is the block legal
- Is the block beat all the other block we get
- …….

Block3
I am better

Commit the best block

Commit and undo,

Small error, big mistake!

Extremely careful with block processing

- Validate many details
- Commit and undo, re-commit
- Complicate, and you need optimize performance!
- Small error, big mistake!
- Even Bitcoin in 2018 has block processing problem
  - CVE-2018-17144 (occur in performance optimizing)
  - Double spend
  - DOS
  - Inflation

Example: NEO MerlkeTree bypass

```
private static MerkleTreeNode Build(MerkleTreeNode[] leaves)
{
    if (leaves.Length == 0) throw new ArgumentException();
    if (leaves.Length == 1) return leaves[0];
    MerkleTreeNode[] parents = new MerkleTreeNode[(leaves.Length + 1) / 2];
    for (int i = 0; i < parents.Length; i++)
    {
        parents[i] = new MerkleTreeNode();
        parents[i].LeftChild = leaves[i * 2];
        leaves[i * 2].Parent = parents[i];
        if (i * 2 + 1 == leaves.Length)
        {
            parents[i].RightChild = parents[i].LeftChild;
        }
        else
        {
            parents[i].RightChild = leaves[i * 2 + 1];
            leaves[i * 2 + 1].Parent = parents[i];
        }
        parents[i].Hash = new UInt256(Crypto.Default.Hash256(parents[i].LeftC
    }
    return Build(parents); //TailCall
```

Example: NEO Transaction bind bypass

```
private bool Transaction_GetWitnesses(ExecutionEngine engine)
{
    if (engine.CurrentContext.EvaluationStack.Pop() is InteropInterface _interface)
    {
        Transaction tx = _interface.GetInterface<Transaction>();
        if (tx == null) return false;
        if (tx.Witnesses.Length > ApplicationEngine.MaxArraySize)
            return false;
        engine.CurrentContext.EvaluationStack.Push(tx.Witnesses.Select(p => StackItem.FromInterface(p)).ToArray());
        return true;
    }
    return false;
}
```

Get VerficationScript in NEO

- Syscall to get verification script of a contract
- Verification script of NEO does not binding with a NEO contract
  - We can generate different Verification scripts for one Contract
  - Because the Smart contract language is Turing complete!
- Result in double spend

Layers of double spend attacks

Transaction	• Inconsistent execution of Smart contract
Block	• Logic bugs in processing Block
Consensus	• Attack on Consensus Mechanism
Network	• Network split

Exmaple: Fork in NEO dBFT

dBFT consensus mechanism:

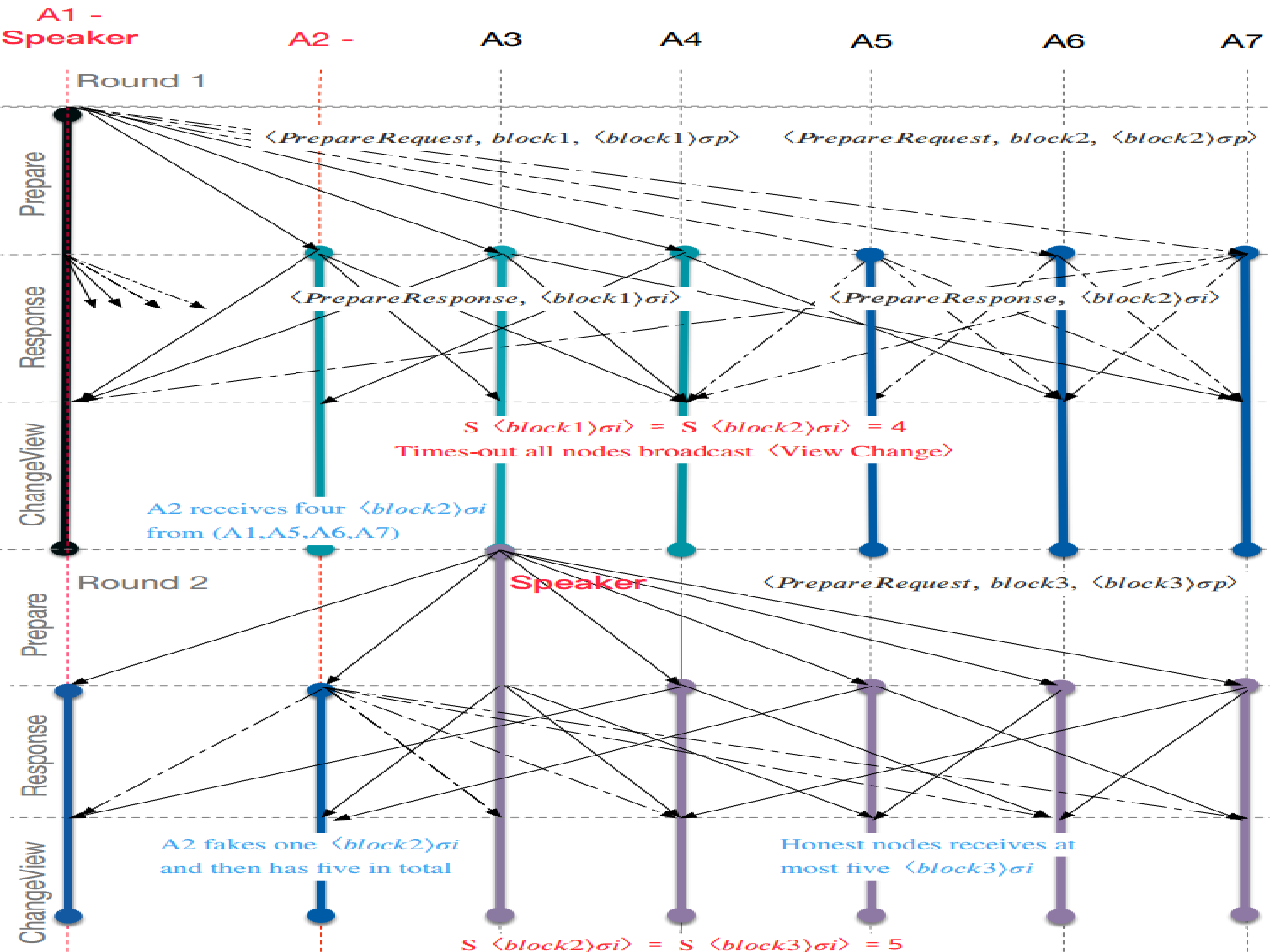
Byzantine Fault Tolerance

POS+pBFT:

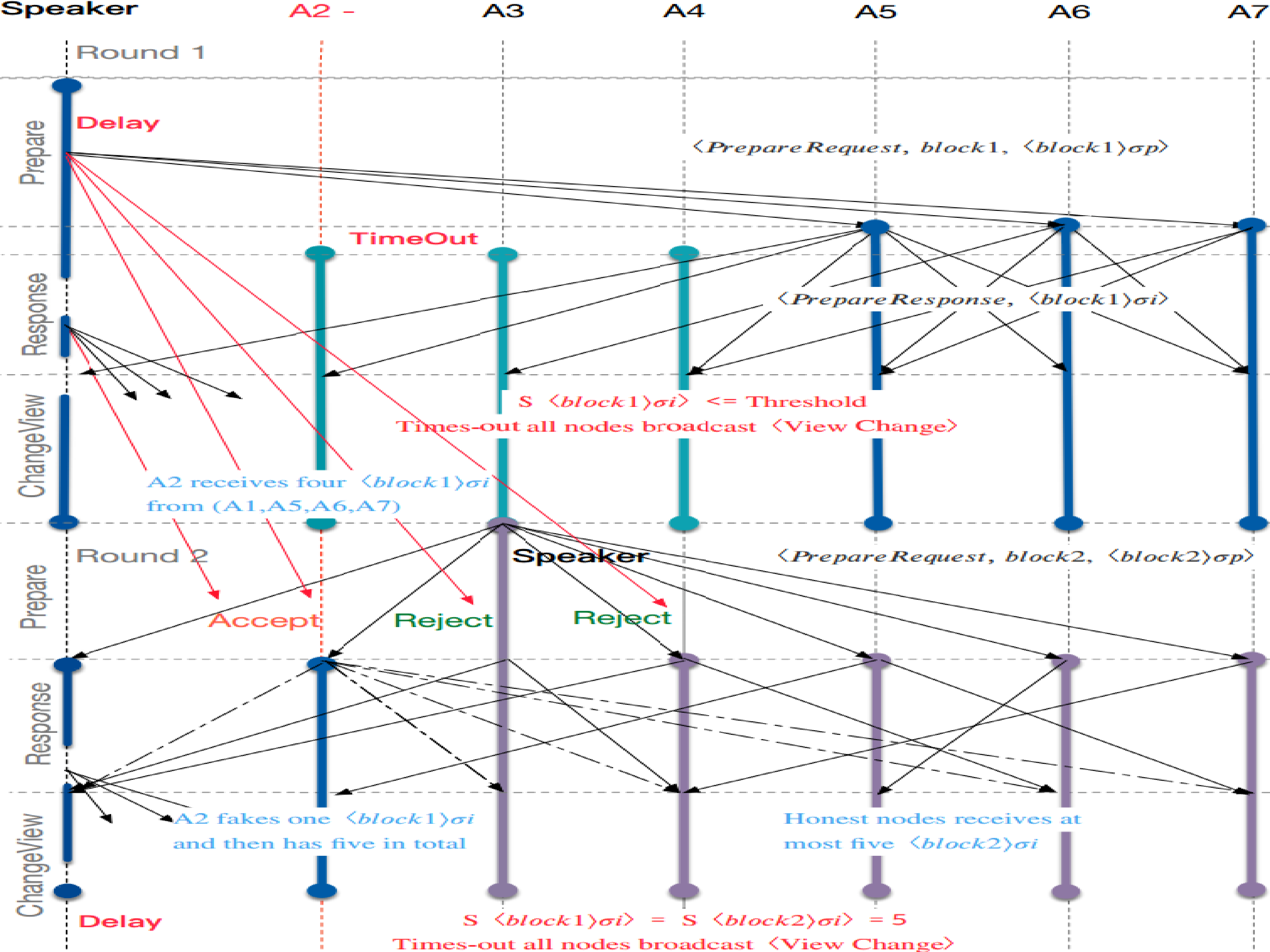
Choose a small number of committees by voting.

Use pBFT algorithm to reach consensus among committees.

Only guarantees a consensus between honest consensus nodes.



A1 - Speaker
A2 -
A3
A4
A5
A6
A7
Round 1
Prepare
Response
ChangeView
⟨PrepareRequest, block1, ⟨block1⟩σp⟩
⟨PrepareRequest, block2, ⟨block2⟩σp⟩
⟨PrepareResponse, ⟨block1⟩σi⟩
⟨PrepareResponse, ⟨block2⟩σi⟩
S ⟨block1⟩σi⟩ = S ⟨block2⟩σi⟩ = 4
Times-out all nodes broadcast ⟨View Change⟩
A2 receives four ⟨block2⟩σi
from (A1,A5,A6,A7)
Round 2
Speaker
⟨PrepareRequest, block3, ⟨block3⟩σp⟩
Prepare
Response
ChangeView
A2 fakes one ⟨block2⟩σi
and then has five in total
Honest nodes receives at
most five ⟨block3⟩σi
S ⟨block2⟩σi⟩ = S ⟨block3⟩σi⟩ = 5



Speaker
A2 -
A3
A4
A5
A6
A7
Round 1
Prepare
Delay
⟨PrepareRequest, block1, ⟨block1⟩σp⟩
TimeOut
Response
⟨PrepareResponse, ⟨block1⟩σi⟩
ChangeView
S ⟨block1⟩σi⟩ <= Threshold
Times-out all nodes broadcast ⟨View Change⟩
A2 receives four ⟨block1⟩σi
from (A1,A5,A6,A7)
Round 2
Speaker
⟨PrepareRequest, block2, ⟨block2⟩σp⟩
Prepare
Accept
Reject
Reject
Response
ChangeView
A2 fakes one ⟨block1⟩σi
and then has five in total
Honest nodes receives at
most five ⟨block2⟩σi
Delay
S ⟨block1⟩σi⟩ = S ⟨block2⟩σi⟩ = 5
Times-out all nodes broadcast ⟨View Change⟩

Example: EOS “consensus mechanism”

The consensus mechanism of EOS is just:

All the super nodes are good man

We should trust good man, should we?

I am tried of diss EOS

Exmaple: VRF bypassed in ONT vBFT

```
ECVRF_prove(y, x, alpha)

Input:

   y - public key, an EC point

   x - private key, an integer

   alpha - VRF input, an octet string

Output:

   pi - VRF proof, octet string of length m+3n

Steps:

1.  h = ECVRF_hash_to_curve(y, alpha)

2.  gamma = h^x

3.  choose a random integer nonce k from [0, q-1]

4.  c = ECVRF_hash_points(g, h, y, gamma, g^k, h^k)

5.  s = k - c*x mod q (where * denotes integer multiplication)

6.  pi = EC2OSP(gamma) || I2OSP(c, n) || I2OSP(s, 2n)

7.  Output pi
```

Don’t roll your own crypto.

Layers of double spend attacks

Layer	Attack
Transaction	• Inconsistent execution of Smart contract
Block	• Logic bugs in processing Block
Consensus	• Attack on Consensus Mechanism
Network	• Network split

We will not talk network layer here

Why?

Because we didn't found any vulnerability in this category

But there are some cases:

Network splitting

Eclipse attack

Time modification attacks

General mitigation for inconsistent execution

A simple mitigation

Block producer hashes the $state_{n+1}$ after running the transactions

pack the hash into the block

Ordinary nodes compare the hash of the local $state'_{n+1}$ with the hash of $state_{n+1}$.

If the two hashes are equal, there is no inconsistency

Linear complexity in size of the local ledger

Performance unacceptable ☹

Ethereum use Merkle Tree structure to improve the performance

Specialized, can not generalized to other ledger ☹

Our mitigation

1. In the block generating phase, the block producer records the write sequence of the database [$write_1$ $write_2$ …. $write_n$]. And compute a hash of it ($write_{hash}$).
2. Normal node also compute their $write'_{hash}$ of [$write'_1$ $write'_2$ …. $Write'_n$].
3. If $write'_{hash}$ equals $write_{hash}$, no inconsistent execution happends.

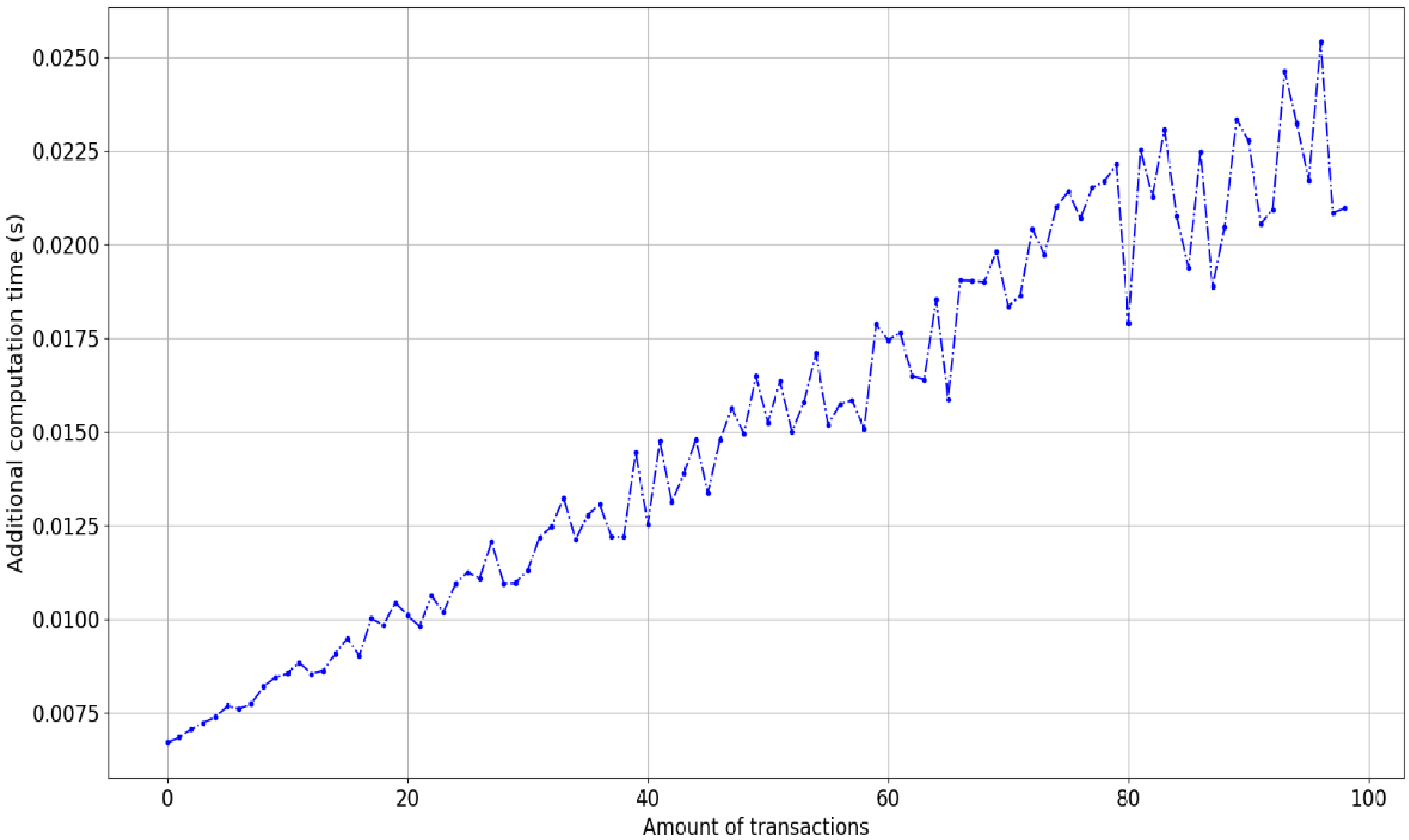
Experiments

We implemented this mitigation on EOS and tested 5 inconsistent execution vulnerability we found.

CVE number	Project	Vulnerability Type	Result
CVE-2018-20689	EOS	memory corruption	reject
CVE-2018-20696	EOS	uninitialized memory	reject
CVE-2018-20690	EOS	out-of-bound read	reject
CVE-2018-20692	EOS	inconsistent version	reject
CVE-2018-14439	EOS	inconsistent floating-point calculation	reject

We can prevent them all.

Performance



Almost no storage penalty

The computation overhead is about 3.8%

Whitepaper

All roads lead to Rome: Many ways to double spend your cryptocurrency

http://blogs.360.cn/post/double-spending-attack_EN.html

Thanks

360
WWW.360.cn
SAFETY FIRST