Porosity

Decompiling Ethereum Smart-Contracts

Matt Suiche (@msuiche)
Founder, Comae Technologies
m@comae.io

Whoami

- **@msuiche**
- Comae Technologies
- OPCDE - www.opcde.com
- First time in Vegas since BlackHat 2011
- Mainly Windows-related stuff
  - CloudVolumes (VMware App Volumes)
  - Memory Forensics for DFIR (Hibr2Bin, DumpIt etc.)
- `"looks like such fun guy" – TheShadowBrokers`
- Didn't know much about blockchain before this project

Just so you know…

- We won't talk about POW/POS stuff
- We won't talk about Merkle Trees
- We won't talk about how to becoming a crypto-currency millionaire

- We will talk about the Ethereum EVM
- We will talk about Solidity
- We will talk about smart-contract bytecodes

- And yes, the tool isn't perfect ☺

Agenda

- Ethereum Virtual Machine (EVM)
- Memory Management
- Addresses
- Call Types
- Type Discovery
- Smart-Contract
- Code Analysis
- Known Bugs
- Future

Solidity

- **Solidity** the quality or state of being firm or strong in structure.
  - *But also the name of Ethereum's smart-contracts compiler*

- **Porosity** is the quality of being porous, or full of tiny holes. Liquids go right through things that have porosity.
  - *But also the name of Comae's smart-contract decompiler.*

Accounts

- Normal Accounts: 3,488,419 (July 15)
- Contract Accounts: 930,889 (July 15)
- Verified Contract Accounts: 2285 (July 15)
  - Source code is provided.

- 40M$ lost in July
  - 10M$ in CoinDash's ICO
    - http://www.coindesk.com/coindash-ico-hacker-nets-additional-ether-theft-tops-10-million/
  - 30M$ due to Parity's wallet.sol vulnerability
    - https://blog.parity.io/security-alert-high-2/
    - https://github.com/paritytech/parity/pull/6103

Ethereum Virtual Machine (EVM)

- Account/Contract/Blockchain
- A Smart-Contract is made of bytecode stored in the blockchain.
- An address is a 160-bits value & corresponds to an “account”
- Operates 256-bits pseudo-registers
  - EVM does not really have registers, but uses a virtual stack to replace them.

Solidity & “Smart Contracts”

- Solidity compiles JavaScript-like code into Ethereum bytecode.

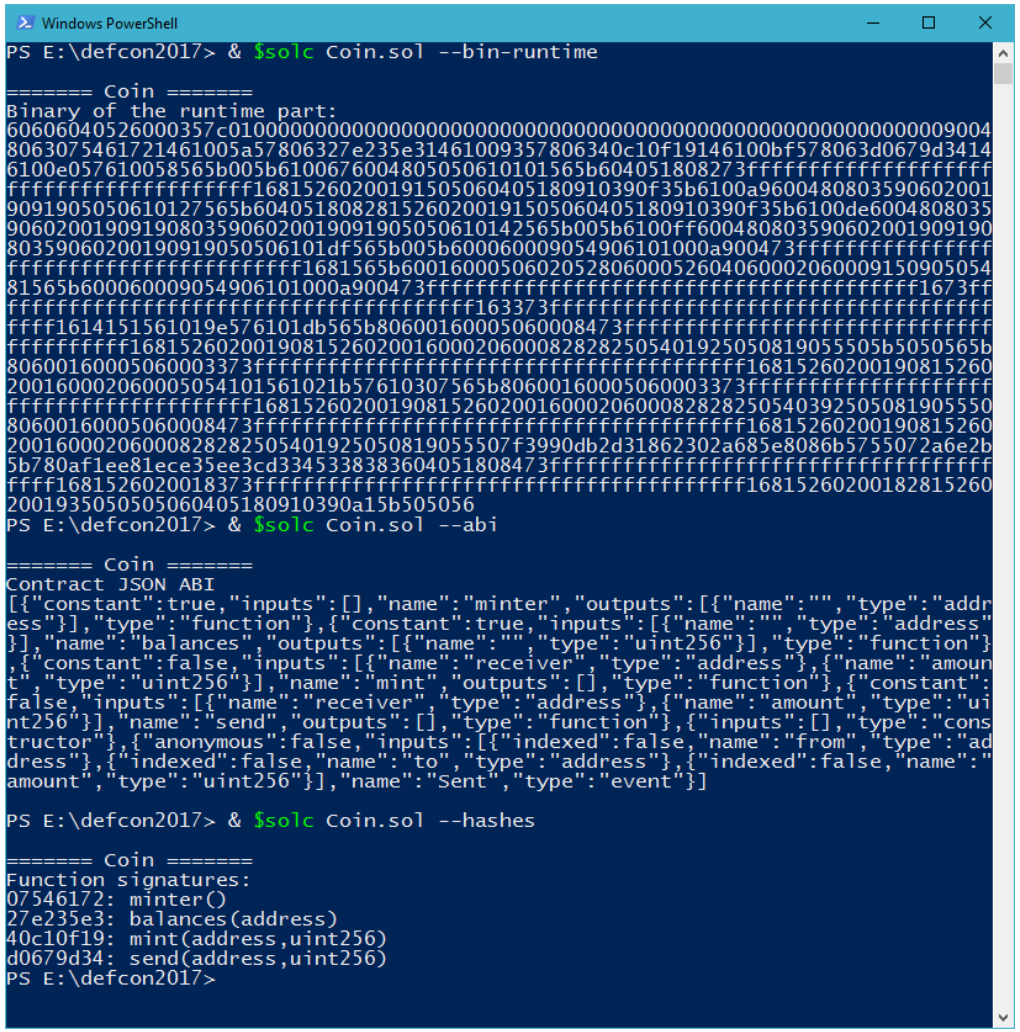
```
contract Coin {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    function Coin() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
    }
}
```



Memory Management

- Stack
  - Virtual stack is being used for operations to pass parameters to opcodes.
  - 256-bit values/entries
  - Maximum size of 1024 elements
- Storage (Persistent)
  - Key-value storage mapping (256-to-256-bit integers)
  - Can't be enumerated.
  - SSTORE/SLOAD
- Memory (Volatile)
  - 256-bit values (lots of AND operations, useful for type discovery)
  - MSTORE/MLOAD

Basic Blocks

- Usually start with the JUMPDEST instruction – except few cases.
- JUMP* instruction jump to the address contained in the 1st element of the stack.
- JUMP* instruction are (almost always) preceded by PUSH instruction
  - This allows to push the destination address in the stack, instead of hardcoding the destination offset.
- SWAP/DUP/POP stack manipulation instructions can make jump destination address harder to retrieve
  - This requires dynamic analysis to rebuild the relationship between each basic block.

EVM functions/instructions

- EVM instructions are more like functions, such as:
- Arithmetic, Comparison & Bitwise Logic Operations
- SHA3
- Environmental & Block Information
- Stack, Memory, Storage and Flow Operations
- Logging & System Operations

Instruction call - Addition

Offset	Instruction	Stack[0]	Stack[2]
n	PUSH1 0x1	0x1	
n + 1	PUSH1 0x2	0x2	0x1
n + 2	ADD	0x3	

- The above translates at the EVM-pseudo code:
  - `add(0x2, 0x1)`

EVM Call

- Can be identified by the CALL instruction.
  - Call external accounts/contracts pointed by the second parameter
  - The second parameter contains the actual 160 address of the external contract
- With the exception of 4 hardcoded contracts:
  - 1 – elliptic curve public key recovery function
  - 2- SHA2 function
  - 3- RIPEMD160 function
  - 4- Identity function

```
call(
 gasLimit,
 to,
 value,
 inputOffset,
 inputSize,
 outputOffset,
 outputSize
)
```

User-Defined functions (Solidity)

- CALLDATALOAD instruction is used to read the Environmental Information Block (EIB) such as parameters.
- First 4 bytes of the EIB contains the 32-bits hash of the called function.
- Followed by the parameters based on their respective types.
  - e.g. **`int`** would be a 256 bits word.

```
function foo(int a, int b) {
 return a + b;
}
```

- `a = calldataload(0x4)`
- `b = calldataload(0x24)`
- `add(calldataload(0x4), calldataload(0x24)`

Type Discovery - Addresses

- As an example, addresses are 160-bit words
- Since stack registers are 256-bit words, they can easily be identified through AND operations using the `0xffffffffffffffffffffffffffffffffffffffff` mask.
- Mask can be static or computed dynamically

Ethereum Assembly	Translation (msg.sender)
CALLER PUSH1 0x01 PUSH 0xA0 PUSH1 0x02 EXP SUB AND	`and(reg256, sub(exp(2, 0xa0), 1))` *(EVM)*
	`reg256 & (2 ** 0xA0) - 1)` *(Intermediate)*
	`address` *(Solidity)*

Bytecode

- The bytecode is divided in two categories:
  - Pre-loader code
    - Found at the beginning that contains the routine to bootstrap the contract
  - Runtime code of the contract
    - The core code written by the user that got compiled by Solidity
    - Each contract contain a dispatch function that redirects the call to the corresponding function based on the provided hash function.

Bytecode – Pre-loader

- CODECOPY copies the runtime part of the contract into the EVM memory – which gets executed at base address 0x0

```
00000000 6060      PUSH1 60
00000002 6040      PUSH1 40
00000004 52        MSTORE
00000005 6000      PUSH1 00
00000007 6001      PUSH1 01
00000009 6000      PUSH1 00
0000000b 610001    PUSH2 0001
0000000e 0a        EXP
0000000f 81        DUP2
00000010 54        SLOAD
00000011 81        DUP2
00000012 60ff      PUSH1 ff
00000014 02        MUL
00000015 19        NOT
```

```
00000016 16        AND
00000017 90        SWAP1
00000018 83        DUP4
00000019 02        MUL
0000001a 17        OR
0000001b 90        SWAP1
0000001c 55        SSTORE
0000001d 50        POP
0000001e 61bb01    PUSH2 bb01
00000021 80        DUP1
00000022 612b00    PUSH2 2b00
00000025 6000      PUSH1 00
00000027 39        CODECOPY
00000028 6000      PUSH1 00
0000002a f3        RETURN
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000: calldataload(0x0) / exp(0x2, 0xe0)
0x00000000 60 60                PUSH1 60
0x00000002 60 40                PUSH1 40
0x00000004 52                   MSTORE
0x00000005 60 e0                PUSH1 e0
0x00000007 60 02                PUSH1 02
0x00000009 0a                   EXP
0x0000000a 60 00                PUSH1 00
0x0000000c 35                   CALLDATALOAD
0x0000000d 04                   DIV
0x0000000e 63 06  72  e9  ee    PUSH4 06 72 e9 ee
0x00000013 81                   DUP2
0x00000014 14                   EQ
0x00000015 60 24                PUSH1 24
0x00000017 57                   JUMPI

loc_00000018:
0x00000018 80                   DUP1
0x00000019 63 9d  04  0a  f4    PUSH4 9d 04 0a f4
0x0000001e 14                   EQ
0x0000001f 60 35                PUSH1 35
0x00000021 57                   JUMPI

loc_00000022:
0x00000022 5b                   JUMPDEST
0x00000023 00                   STOP
```

```
double(uint256):
0x00000024 5b                   JUMPDEST
0x00000025 60 45                PUSH1 45
0x00000027 60 04                PUSH1 04
0x00000029 35                   CALLDATALOAD
0x0000002a 60 00                PUSH1 00
0x0000002c 60 4f                PUSH1 4f
0x0000002e 82                   DUP3
0x0000002f 60 02                PUSH1 02

loc_00000031:
0x00000031 5b                   JUMPDEST
0x00000032 02                   MUL
0x00000033 90                   SWAP1
0x00000034 56                   JUMP

triple(uint256):
0x00000035 5b                   JUMPDEST
0x00000036 60 45                PUSH1 45
0x00000038 60 04                PUSH1 04
0x0000003a 35                   CALLDATALOAD
0x0000003b 60 00                PUSH1 00
0x0000003d 60 4f                PUSH1 4f
0x0000003f 82                   DUP3
0x00000040 60 03                PUSH1 03
0x00000042 60 31                PUSH1 31
0x00000044 56                   JUMP
```

Function Hashes

- The 4 bytes of the sha3 (keccak256) value for the string `functionName(param1Type, param2Type, etc)`

```
[
    {
        "constant":false,
        "inputs":[{ "name":"a", "type":"uint256" }],
        "name":"double",
        "outputs":[{ "name":"", "type":"uint256" }],
        "type":"function"
    }
]
```

```
keccak256("double(uint256)") =>
 eee972066698d890c32fec0edb38a360c32b71d0a29ffc75b6ab6d2774ec9901

double(uint256) -> 0xeee97206
triple(uint256) -> 0xf40a049d
```

Extracting function hash

- `calldataload(0x0) / exp(0x2, 0xe0)`
- `(0x12345678xxxx / 0x00000001xxxx) = 0x12345678`

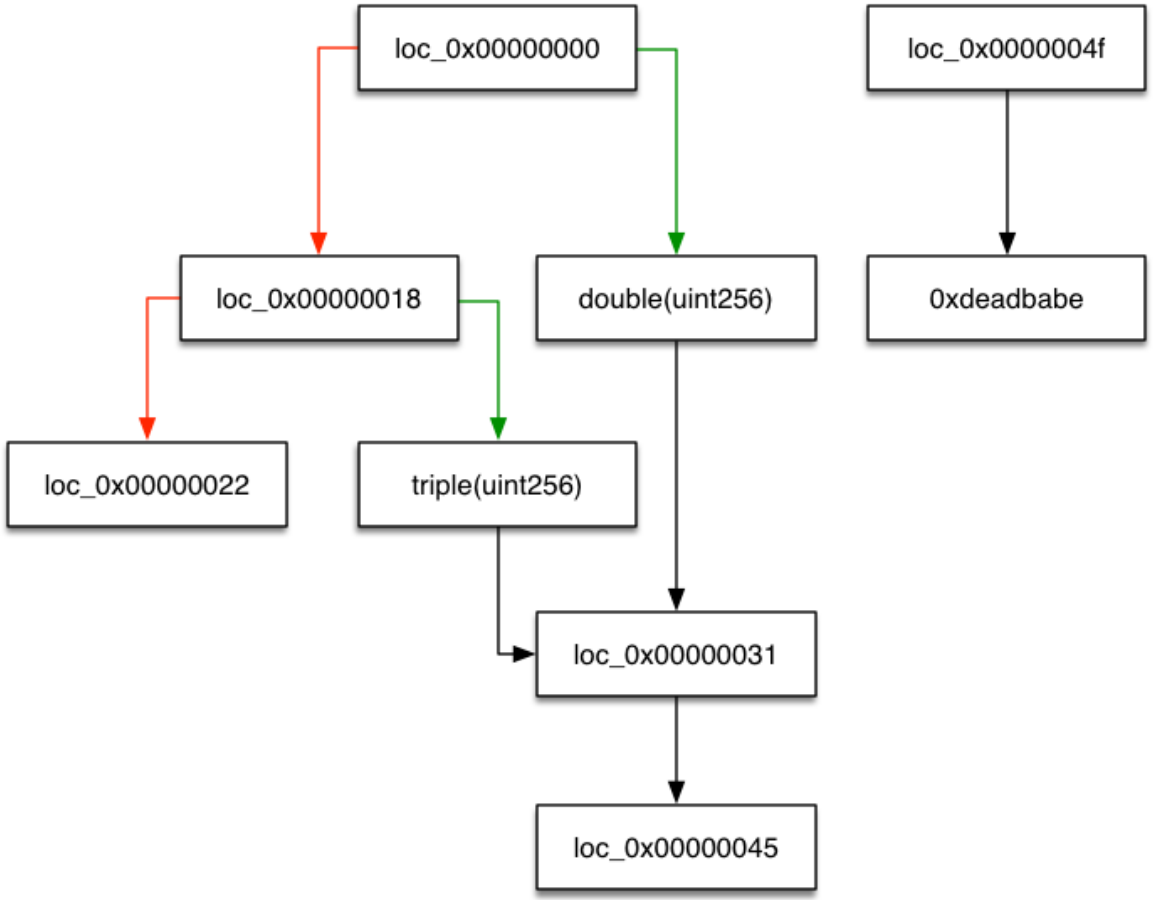
```
PS C:\Program Files\Geth> .\evm.exe \
 --code 60e060020a60003504 \
 --debug \
 --input 12345678aaaaaaaabbbbbbbbccccccccdddddddd
PC 00000009: STOP GAS: 9999999923 COST: 0
STACK = 1
0000: 0000000000000000000000000000000000000000000000000000000012345678
MEM = 0
STORAGE = 0
```

Ethereum Emulator

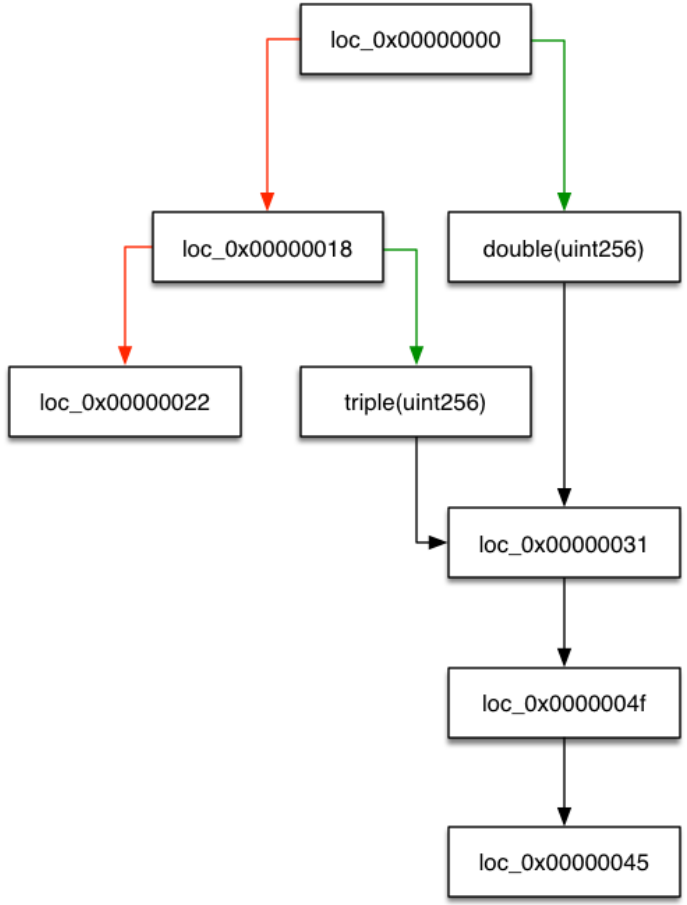
- `jumpi(eq(calldataload(0x0) / exp(0x2, 0xe0), 0xeee97206))`

Control Flow Graph

Static CFG *(--cfg)*



Emulated CFG *(--cfg-full)*



Dispatcher – pseudo code

```
hash = calldataload(0x0) / exp(0x2, 0xe0);
switch (hash) {
    case 0xeee97206: // double(uint256)
        memory[0x60] = calldataload(0x4) * 2;
        return memory[0x60];
    break;
    case 0xf40a049d: // triple(uint256)
        memory[0x60] = calldataload(0x4) * 3;
        return memory[0x60];
    break;
    default:
    // STOP
    break;
}
```

Pseudo-Code

```
contract C {
    function double(int arg_4) {
        return arg_4 * 2;
    }

    function triple(int arg_4) {
        return arg_4 * 3;
    }
}
```

Translated Code

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*

```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*


```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

Bytecode – Dispatcher *(--list)*

```
loc_00000000:
0x00000000 60 60                    PUSH1 60
0x00000002 60 40                    PUSH1 40
0x00000004 52                       MSTORE
0x00000005 60 e0                    PUSH1 e0
0x00000007 60 02                    PUSH1 02
0x00000009 0a                       EXP
0x0000000a 60 00                    PUSH1 00
0x0000000c 35                       CALLDATALOAD
0x0000000d 04                       DIV
0x0000000e 63 06  72  e9  ee        PUSH4 06 72 e9 ee
0x00000013 81                       DUP2
0x00000014 14                       EQ
0x00000015 60 24                    PUSH1 24
0x00000017 57                       JUMPI

loc_00000018:
0x00000018 80                       DUP1
0x00000019 63 9d  04  0a  f4        PUSH4 9d 04 0a f4
0x0000001e 14                       EQ
0x0000001f 60 35                    PUSH1 35
0x00000021 57                       JUMPI

loc_00000022:
0x00000022 5b                       JUMPDEST
0x00000023 00                       STOP
```

```
double(uint256):
0x00000024 5b                       JUMPDEST
0x00000025 60 45                    PUSH1 45
0x00000027 60 04                    PUSH1 04
0x00000029 35                       CALLDATALOAD
0x0000002a 60 00                    PUSH1 00
0x0000002c 60 4f                    PUSH1 4f
0x0000002e 82                       DUP3
0x0000002f 60 02                    PUSH1 02

loc_00000031:
0x00000031 5b                       JUMPDEST
0x00000032 02                       MUL
0x00000033 90                       SWAP1
0x00000034 56                       JUMP

triple(uint256):
0x00000035 5b                       JUMPDEST
0x00000036 60 45                    PUSH1 45
0x00000038 60 04                    PUSH1 04
0x0000003a 35                       CALLDATALOAD
0x0000003b 60 00                    PUSH1 00
0x0000003d 60 4f                    PUSH1 4f
0x0000003f 82                       DUP3
0x00000040 60 03                    PUSH1 03
0x00000042 60 31                    PUSH1 31
0x00000044 56                       JUMP
```

DELEGATECALL & $30M Parity Bug

Abusing the dispatcher to reinitialize a wallet?

```
contract Wallet {
    address _walletLibrary;
    address owner;

    // initWallet() is only invoked by the constructor. WalletLibrary is hardcoded.
    function Wallet(address _owner) {
        _walletLibrary = 0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3;
        _walletLibrary.delegatecall(bytes4(sha3("initWallet(address)")), _owner);
    }

    (…)

    // fallback function behaves like a "generic forward" (WTF?)
    // Wallet.initWallet(attacker) becomes walletLibrary.initWallet(attacker)
    function () payable {
        _walletLibrary.delegatecall(msg.data);
    }
}
```

Generic-Forward & Fall back functions

- Typical issue introduce by the lack of upgradeability of smart-contract
- Software engineering is hard enough, but smart-contracts need to be:
  - Backward-Compatible
  - Forward-Compatible
- Does not help to make the language verifiable.
- Fall back functions are undefined behaviors.
- Imagine if your kernel would behave like that.
  - Scary, right ? Now, imagine if that's your bank…

Fixing the initialization bug

```
- function initMultiowned(address[] _owners, uint _required) {
+ function initMultiowned(address[] _owners, uint _required) internal {

- function initDaylimit(uint _limit) {
+ function initDaylimit(uint _limit) internal {

+ modifier only_uninitialized { if (m_numOwners > 0) throw; _; }
- function initWallet(address[] _owners, uint _required, uint _daylimit) {
+ function initWallet(address[] _owners, uint _required, uint _daylimit)
only_uninitialized {
```

Code Analysis – Vulnerable Contract

```
contract SendBalance {
    mapping ( address => uint ) userBalances ;
    bool withdrawn = false ;

    function getBalance (address u) constant returns ( uint ){
        return userBalances [u];
    }

    function addToBalance () {
        userBalances[msg.sender] += msg.value ;
    }

    function withdrawBalance (){
        if (!(msg.sender.call.value(
            userBalances [msg.sender])())) { throw ; }
        userBalances [msg.sender] = 0;
    }
}
```

Code Analysis – Vulnerable Contract

```
contract SendBalance {
    mapping ( address => uint ) userBalances ;
    bool withdrawn = false ;

    function getBalance (address u) constant returns ( uint ){
        return userBalances [u];
    }

    function addToBalance () {
        userBalances[msg.sender] += msg.value ;
    }

    function withdrawBalance (){
        if (!(msg.sender.call.value(
            userBalances [msg.sender])())) { throw ; }
        userBalances [msg.sender] = 0;
    }
}
```

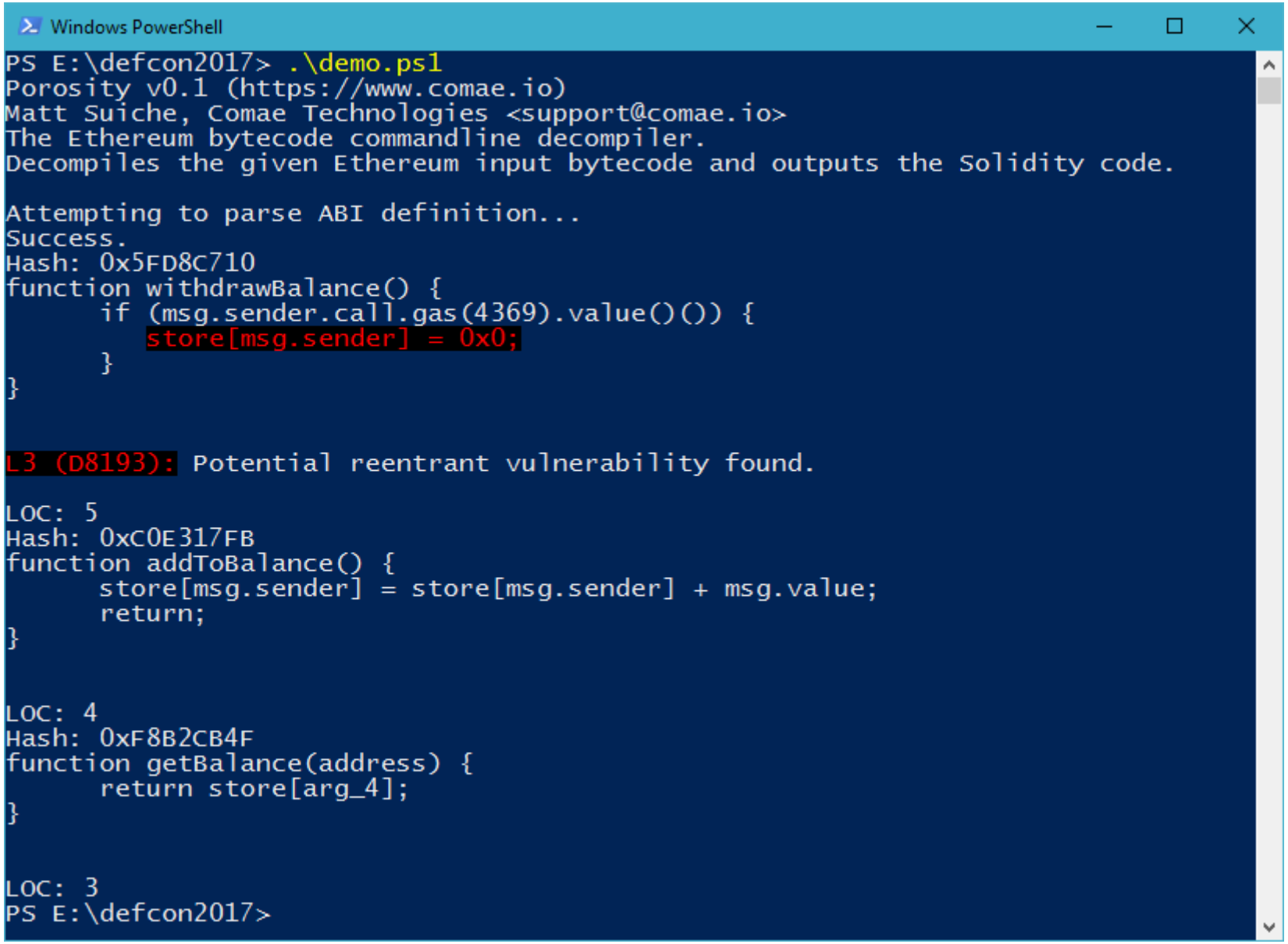
Caller contract can recall this function using its fallback function

Understanding the control flow

- In the case of reentrant vulnerability, since we can record the EVM state at each instruction using porosity.
- We can track in which basic block SSTORE instructions are called.
  - `userBalances [msg.sender] = 0;`
- And track states for each basic block.

.\demo.ps1

```
$porosity = 'E:\projects\porosity\Debug\porosity.exe'
$abi =
'[{\"constant\":false,\"inputs\":[],\"name\":\"withdrawBalance\",\"outputs\":[],\"ty
pe\":\"function\"},{\"constant\":false,\"inputs\":[],\"name\":\"addToBalance\",\"out
puts\":[],\"type\":\"function\"},{\"constant\":true,\"inputs\":[{\"name\":\"u\",\"ty
pe\":\"address\"}],\"name\":\"getBalance\",\"outputs\":[{\"name\":\"\",\"type\":\"ui
nt256\"}],\"type\":\"function\"}]'
$code =
'60606040526000357c0100000000000000000000000000000000000000000000000000000090048063
35fd8c7101461004f578063c0e317fb1461005e578063f8b2cb4f1461006d5761004d565b005b61005c6
004805050610099565b005b61006b6004805050610013e565b005b6100836004808035906020019091905
05061017d565b6040518082815260200191505060405180910390f35b3373ffffffffffffffffffffff
ffffffffffffff166111116000600050600033 73ffffffffffffffffffffffffffffffffffffffff1
6815260200190815260200160002060005054604051809050600060405180830381858888f1935050505
015156101065761000256 5b6000600060005060003373ffffffffffffffffffffffffffffffffffffff
f16815260200190815260200160002060005081905550 5b565b34600060005060003373ffffffffffff
ffffffffffffffffffff16815260200190815260200160002060008282825054019250508190550 5b56
5b6000600060005060008373ffffffffffffffffffffffffffffffffffffffff1681526020019
081526020016000206000505490506101b6565b91905056’
& $porosity --abi $abi --runtime-code $code --decompile --verbose 0
```


Windows PowerShell
PS E:\defcon2017> .\demo.ps1
Porosity v0.1 (https://www.comae.io)
Matt Suiche, Comae Technologies <support@comae.io>
The Ethereum bytecode commandline decompiler.
Decompiles the given Ethereum input bytecode and outputs the Solidity code.
Attempting to parse ABI definition...
Success.
Hash: 0x5FD8C710
function withdrawBalance() {
if (msg.sender.call.gas(4369).value()()) {
store[msg.sender] = 0x0;
}
}
L3 (D8193): Potential reentrant vulnerability found.
LOC: 5
Hash: 0xC0E317FB
function addToBalance() {
store[msg.sender] = store[msg.sender] + msg.value;
return;
}
LOC: 4
Hash: 0xF8B2CB4F
function getBalance(address) {
return store[arg_4];
}
LOC: 3
PS E:\defcon2017>

Known class of bugs

- Reentrant Vulnerabilities / Race Condition
  - Famously known because of the $50M USD DAO hack (2016) *[8]*
- Call Stack Vulnerabilities
  - Got 1024 frames but a bug ain't one – *c.f. Least Authority [12]*
- Time Dependency Vulnerabilities
  - @mhswende blogposts are generally awesome, particularly the roulette one [10]
- Unconditional DELEGATECALL

Porosity + Quorum = <3

- Quorum: Ethereum code fork created by J.P. Morgan
  - aimed at enterprises that want to use a permissioned blockchain that adds private smart contract execution & configurable consensus.
- Quorum now includes Porosity integrated directly into geth out of the box:
  - Scan private contracts sent to your node from other network participants
  - Incorporate into security & patching processes for private networks with formalized governance models
  - Automate scanning and analyze risk across semi-public Quorum networks
- https://github.com/jpmorganchase/quorum


```
function scanBlock(blockNumber) {
        var b = eth.getBlock(blockNumber);
        for (var i = 0; i < b.transactions.length; i++) {
                var tx = eth.getTransaction(b.transactions[i]);
                var code;
                if (tx.v == 37 || tx.v == 38) { // private
                        code = quorum.getPrivatePayload(tx.input);
                        if (code === "0x") {
                                continue // we weren't a party to this transaction
                        }
                } else {
                        // code = tx.input;
                        continue; // skip public transactions
                }
                var isVulnerable = quorum.runPorosity({"code": code, "decompile": true, "silent": true})
                if (isVulnerable) {
                        console.log("Reentrant vulnerability in block " + tx.blockNumber +
                                ":\nTransaction: " + tx.hash +
                                "\nFrom:        "  + tx.from +
                                "\nTo:          "  + (tx.to === null ? "Contract creation" : tx.to)
                               );
                }
        }
}

console.log("Scanning all private transactions for vulnerabilities");
for (var i = 0; i < eth.blockNumber; i++) {
        scanBlock(i);
}
```

```
~/quorum-examples/examples/7nodes: ./runscript.sh porosity/scan.js
Scanning all private transactions for vulnerabilities


Reentrant vulnerability in block 2:
Transaction: 0x4d5155cde81d71730be693b04e6e1aadc2e1381f08c18b774ccc154b53de82e3
From:        0xed9d02e382b34818e88b88a309c7fe71e65f419d
To:          Contract creation

Reentrant vulnerability in block 3:
Transaction: 0x6820cc1630587495386589798d2859c27a4c20a9c82837f128e7cefc9cdbe9e8
From:        0xed9d02e382b34818e88b88a309c7fe71e65f419d
To:          Contract creation

Reentrant vulnerability in block 4:
Transaction: 0xf0287b32d8e50508d5598069550b25c02a9a8df9217a1316f32c2a51282249fe
From:        0xed9d02e382b34818e88b88a309c7fe71e65f419d
To:          Contract creation
true
~/quorum-examples/examples/7nodes:
```

```
......... console.log("err creating contract", e);
......... } else {
......... if (!contract.address) {
.......... console.log("Contract transaction send: TransactionHash: " + contract.transactionHash + " waiting to be mined...");
.......... } else {
.......... console.log("Vulnerable contract mined! Address: " + contract.address);
.......... console.log(contract);
.......... }
......... }
...... });
Contract transaction send: TransactionHash: 0x31bf6bf870466467260797a1180aa6e626cebd1384cabe38991af0a888ecaa31 waiting to be mined...
undefined
> Vulnerable contract mined! Address: 0x3950943d8d86267c04a4bba804f9f0b8a01c2fb8
[object Object]
> quorum.runPorosity({"code": eth.getCode("0x3950943d8d86267c04a4bba804f9f0b8a01c2fb8"), "decompile": true})
Porosity v0.1 (https://www.comae.io)
Matt Suiche, Comae Technologies <support@comae.io>
The Ethereum bytecode commandline decompiler.
Decompiles the given Ethereum input bytecode and outputs the Solidity code.

Hash: 0x5FD8C710
executeInstruction: NOT_IMPLEMENTED: REVERT
executeInstruction: NOT_IMPLEMENTED: REVERT
executeInstruction: NOT_IMPLEMENTED: REVERT
function func_5fd8c710 {
      if (!msg.value) {
      }
      if (msg.sender.call.gas(4369).value()()) {
         store[msg.sender] = 0x0;
      }
      store[msg.sender] = 0x0;
}


L5 (D8193): Potential reentrant vulnerability found.

L7 (D8193): Potential reentrant vulnerability found.

LOC: 8
Hash: 0xC0E317FB
executeInstruction: NOT_IMPLEMENTED: REVERT
```

Future

- Ethereum DApps created a new segment for softwares.
- Porosity
  - Improving support for conditional and loop statements
- Ethereum / Solidity & Security
  - Fast growing community and more tools such as OYENTE or Porosity
  - Personally, looking forward seeing the Underhanded Solidity Coding Contest [10] results
  - EVM vulnerabilities triggered by malicious bytecode? *CLOUDBURST on the blockchain*
- More Blockchain VMs ?
  - Swapping the EVM for Web Assembly (WASM) for 2018

Future - WebAssembly

- New Ethereum Roadmap (March 2017) mentions making the EVM obsolete and replacing it with WebAssembly for 2018.
- *"WebAssembly (or WASM) is a new, portable, size- and load-time-efficient format."*
  - WebAssembly is currently being designed as an open standard by a W3C Community Group.
  - WebAssembly defines an instruction set, intermediate source format (WAST) and a binary encoded format (WASM).
  - Major browser JavaScript engines will notably have native support for WebAssembly – including V8, Chakra and Spidermonkey.
- https://github.com/ewasm/design

WASM + DApps + Blockchain

- C++ -> WASM is already a thing
  - https://mbebenita.github.io/WasmExplorer/
  - And a similar thing is planned for eWASM

Goals of the eWASM project

- To provide a specification of *eWASM contract* semantics and the *Ethereum interface*
- To provide an *EVM transpiler*, preferably as an eWASM contract
- To provide a *metering injector*, preferably as an eWASM contract
- To provide a VM implementation for executing eWASM contracts
- To implement an eWASM backend in the Solidity compiler
- To provide a library and instructions for writing contracts in Rust
- To provide a library and instructions for writing contracts in C

Acknowledgements

- Mohamed Saher
- Halvar Flake
- DEFCON Review Board Team
- Max Vorobjov & Andrey Bazhan
- Martin H. Swende
- Gavin Wood
- Andreas Olofsson

References

[1] Woods, Gavin. "Ethereum: A Secure Decentralised Generalised Transaction Ledger." Web. https://github.com/ethereum/yellowpaper.pdf

[2] Olofsson, Andreas. "Solidity Workshop." Web. https://github.com/androlo/solidity-workshop

[3] Olofsson, Andreas. "Solidity Contracts." Web. https://github.com/androlo/standard-contracts

[4] Velner, Yarn, Jason Teutsch, and Loi Luu. "Smart Contracts Make Bitcoin Mining Pools Vulnerable." Web. https://eprint.iacr.org/2017/230.pdf

[5] Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Aquinas Hobor. "Making Smart Contracts Smarter." Web. https://www.comp.nus.edu.sg/%7Ehobor/Publications/2016/Making%20Smart%20Contracts%20Smarter.pdf

[6] Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. " A Survey of Attacks on Ethereum Smart Contracts." Web. https://eprint.iacr.org/2016/1007.pdf

[7] Sarkar, Abhiroop. "Understanding the Transactional Nature of Smart-Contracts." Web. https://abhiroop.github.io/Exceptions-and-Transactions

[8] Siegel, David. "Understanding The DAO Attack." Web. http://www.coindesk.com/understanding-dao-hack-journalists

[9] Blockchain software for asset management. "OYENTE: An Analysis Tool for Smart Contracts." Web. https://github.com/melonproject/oyente

[10] Holst Swende, Martin. "Breaking the house." Web. http://martin.swende.se/blog/Breaking_the_house.html

[11] Buterin, Vitalik. "Thinking About Smart Contract Security."Web. https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security

[12] Least Authority. "Gas Economics: Call Stack Depth Limit Errors." Web. https://github.com/LeastAuthority/ethereum-analyses/blob/master/GasEcon.md#callstack-depth-limit-errors

[13] Underhanded Solidity Coding Contest, Web. http://u.solidity.cc/

[14] Quorum. "A permissioned implementation of Ethereum supporting data privacy." https://github.com/jpmorganchase/quorum

m@comae.io / @msuiche

https://github.com/comaeio/porosity