Reversing Ethereum Smart Contracts

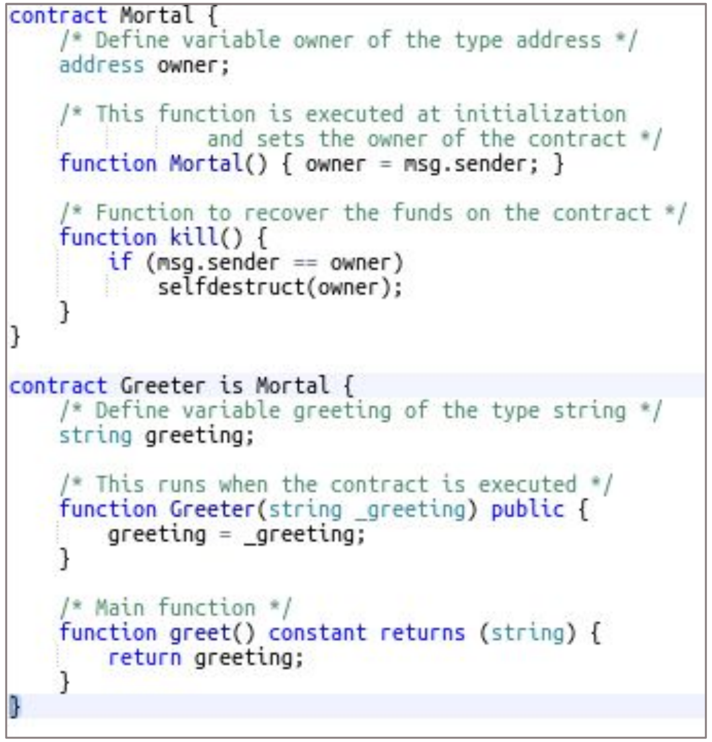
to find out what's behind EVM bytecode

Agenda

1. Introduction
2. Bytecode disassembly
3. Control flow graph (CFG) reconstruction
4. Functions identification
5. Functions name recovery
6. Why using reversing?
   - Closed-source analysis
   - Optimization
   - Vulnerability research
   - Post mortem analysis, …
7. Questions

Introduction

What is Reverse engineering?



```
contract Mortal {
    /* Define variable owner of the type address */
    address owner;

    /* This function is executed at initialization
                and sets the owner of the contract */
    function Mortal() { owner = msg.sender; }

    /* Function to recover the funds on the contract */
    function kill() {
        if (msg.sender == owner)
            selfdestruct(owner);
    }
}

contract Greeter is Mortal {
    /* Define variable greeting of the type string */
    string greeting;

    /* This runs when the contract is executed */
    function Greeter(string _greeting) public {
        greeting = _greeting;
    }

    /* Main function */
    function greet() constant returns (string) {
        return greeting;
    }
}
```

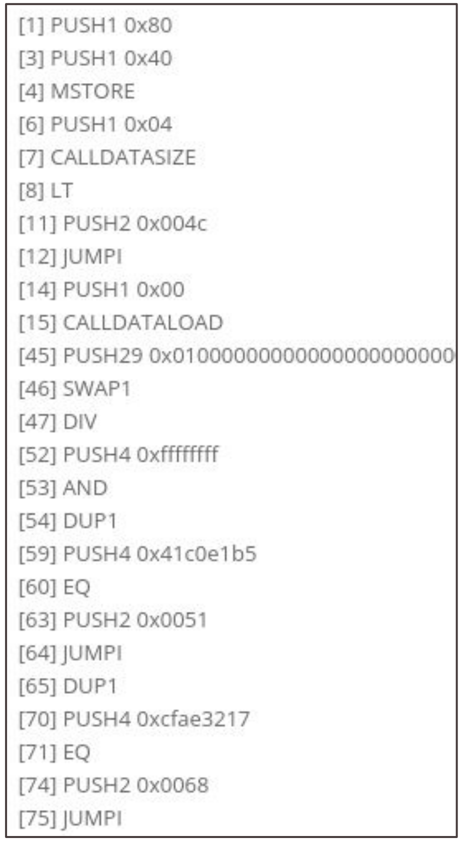
Solidity source code

```
6080604052600436106100 4c57
```

```
608060405260043610610 04c57
```

```
6080604052600436106100 4c57
```

EVM bytecode



```
[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[6] PUSH1 0x04
[7] CALLDATASIZE
[8] LT
[11] PUSH2 0x004c
[12] JUMPI
[14] PUSH1 0x00
[15] CALLDATALOAD
[45] PUSH29 0x010000000000000000000000000
[46] SWAP1
[47] DIV
[52] PUSH4 0xffffffff
[53] AND
[54] DUP1
[59] PUSH4 0x41c0e1b5
[60] EQ
[63] PUSH2 0x0051
[64] JUMPI
[65] DUP1
[70] PUSH4 0xcfae3217
[71] EQ
[74] PUSH2 0x0068
[75] JUMPI
```

EVM assembly

Bytecode decomposition

- Loader code
  - Run the contract constructor
  - Execute once to store the runtime code on the blockchain
  - Can be present in “Contract creation code” on etherscan.io
  - Present in Input Data of the deploying transaction

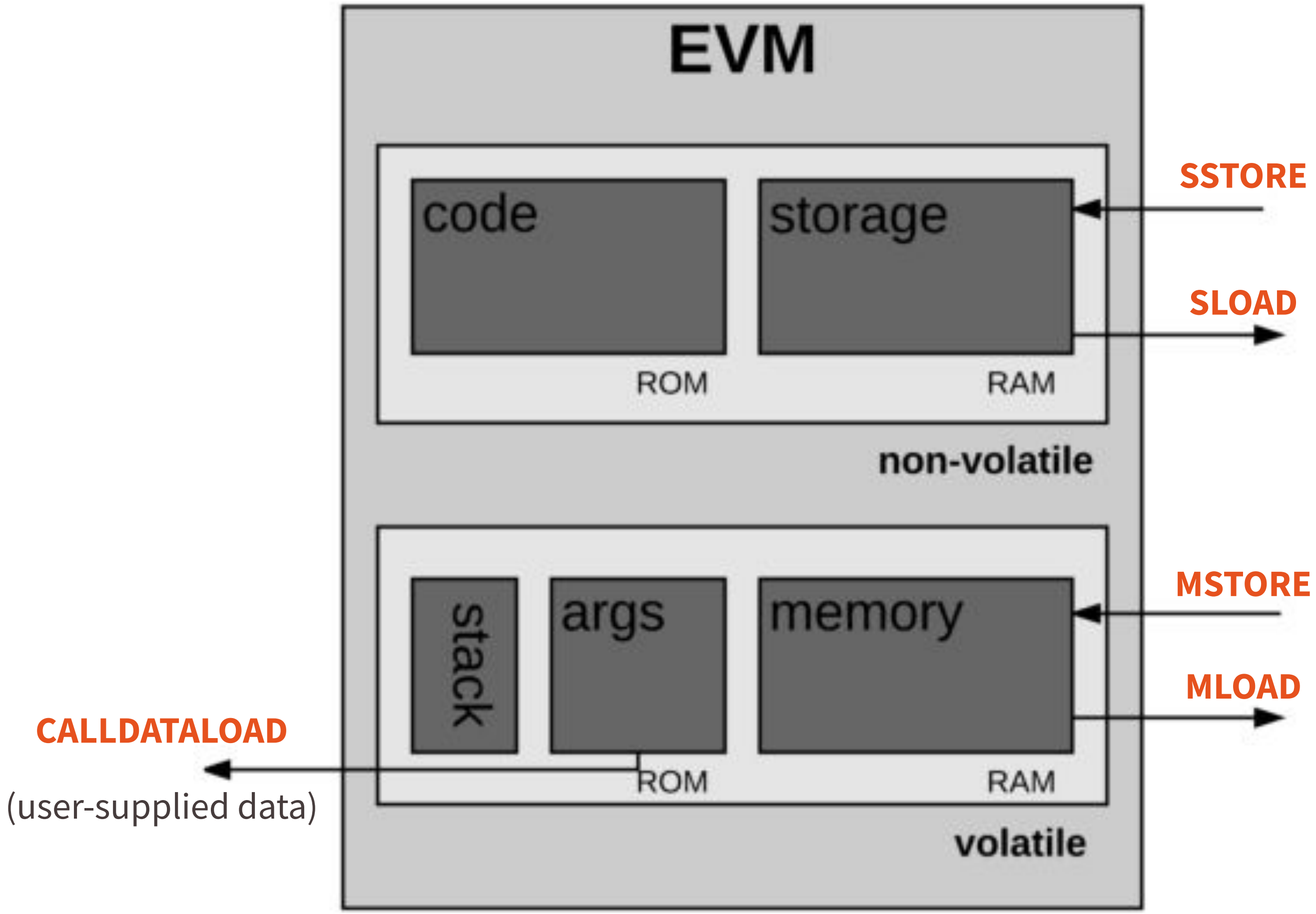
- Runtime code
  - Stored on the blockchain
  - Executed for each transaction with the contract

- Swarm Hash (a.k.a. bzzhash)
  - Merkle tree hash use to retrieve the content of the associated persistent storage of the contract
  - Concatenated at the end of the code
  - Magic number: 0x627a7a72 (bzzr)

```
608060405234801561001057600080fd5b5060405161039b380
38061039b833981018060405281019080805182019291905050
50336000806101000a81548173ffffffffffffffffffffffffff
ffffffffffffff021916908373ffffffffffffffffffffffff
ffffffffffffffff16021790555080600190805190602001906
10089929190610090565b505061013556 5b828054600181600 1
16156101000203166002900490600052602060002090601f016
020900481019282601f106100d157805160ff191683800117 85
556100ff565b828001600101855582156100ff579182015b828
111156100fe578251825591602001919060010190610 0e3565b
5b509050610 10c9190610110565b5090565b61013291905b808
2111561012e57600081600090555060010161011656 5b509056
5b90565b61025780610144600039600 0f30060806040526004 3
61061004c576000357c01000000000000000000000000000000
00000000000000000000000000900463ffffffff16806341c0e
1b514610051578063cfae321714610068575b600080fd5b3480
1561005d57600080fd5b506100666100f8565b005b348015610
07457600080fd5b5061007d610189565b60405180806020018 2
810382528381815181526020019150805190602001908083836
0005b838110156100bd578082015181840152602081019050 61
00a2565b50505050905090810190601f1680156100ea5780820
380516001836020036101000a031916815260200191505b5092
50505060405180910390f35b6000809054906101000a900473f
fffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffffffffff163373fffffffffffff
fffffffffffffffffffffffffff161415610187576000809054
906101000a900473fffffffffffffffffffffffffffffffffff
fffff1673ffffffffffffffffffffffffffffffffffffffff16
ff5b565b606060018054600181600116156101000203166002 9
00480601f016020809104026020016040519081016040528092
919081815260200182805460018160011615610100020316600
29004801561022157806 01f10610 1f6576101008083540402 83
5291602001916102215 65b820191906000526020600020905b8
154815290600101906020018083116102045782900360 1f1682
01915b505050505050905090 5600a165627a7a72305820df97826
8dd1593a7bbc753bfb0404d8353b4c6ced383d8107c926d5003
e40c060029
```

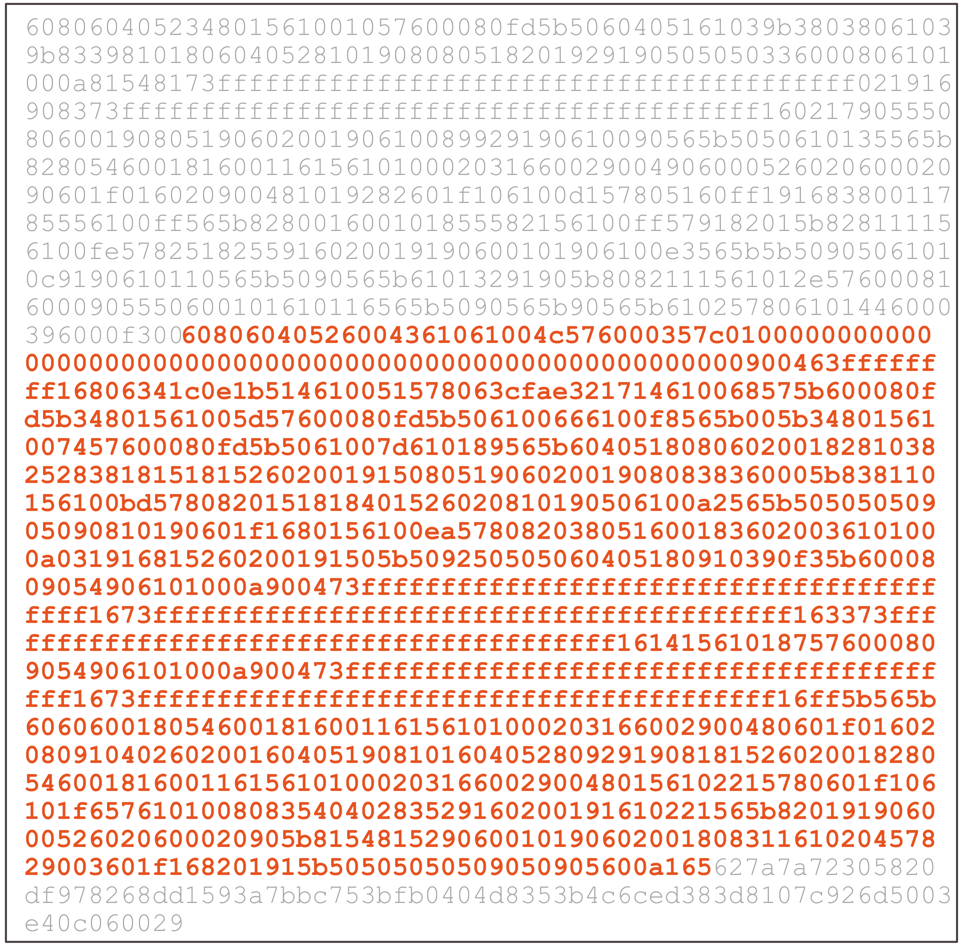
Ethereum Virtual Machine

Architecture		
Stack machine		
Turing complete		
Instruction set	~180 Opcodes	
Memory type		
Stack	volatile	byte-array (list [])
Memory	volatile	byte-array (list [])
Storage	persistent	key-value database (dictionary {})

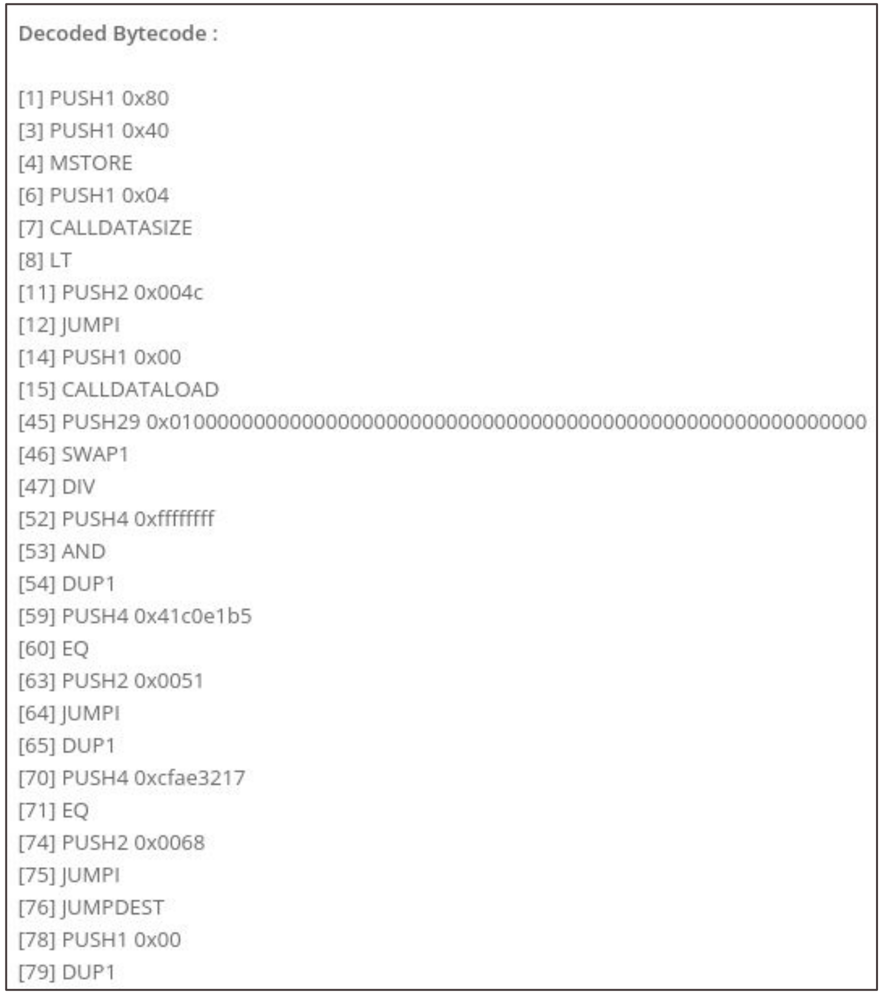


Bytecode disassembly

Disassembling



```
6080604052348015610010576000 80fd5b5060405161039b3803806103
9b83398101806040528101908080518201929190505050336000806101
000a81548173ffffffffffffffffffffffffffffffffffffffff021916
908373ffffffffffffffffffffffffffffffffffffffff160217905550
806001908051906020019061008992919061009056 5b5050610135565b
8280546001816001161561010002031660029004906000526020600020
90601f016020900481019282601f106100d157805160ff191683800117
85556100ff565b828001600101855582156100ff579182015b82811115
6100fe578251825591602001919060010190610 0e3565b5b5090506101
0c919061011056 5b5090565b61013291905b808211156101 2e57600081
6000905550600101610116565b5090565b90565b6102578061014460 00
396000f30060806040526004361061004c576000357c0100000000000
000000000000000000000000000000000000000000000900463ffffff
ff16806341c0e1b514610051578063cfae321714610068575b600080f
d5b34801561005d57600080fd5b506100666100f8565b005b34801561
007457600080fd5b5061007d610189565b6040518080602001828103 8
252838181518152602001915080519060200190808383600 05b838110
156100bd578082015181840152602081019050 6100a2565b505050509
050908101906 01f168015 6100ea5780820380516001836020036101 00
0a031916815260200191505b5092505050604051809103 90f35b60008
09054906101000a900473ffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff163373fff
fffffffffffffffffffffffffffffffffffff16141561018757600080
9054906101000a900473fffffffffffffffffffffffffffffffffffff
fff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b
606060018054600181600116156101000203166002900480601f01602
080910402602001604051908101604052809291908181526020018280
5460018160011615610100020316600290048015610221578 0601f106
101f6576101008083540402835291602001916102215 65b820191906 0
005260206000209 05b815481529060010190602001808311610204578
29003601f168201915b5050505050905090560 0a165627a7a72305820
df978268dd1593a7bbc753bfb0404d8353b4c6ced383d8107c926d5003
e40c060029
```

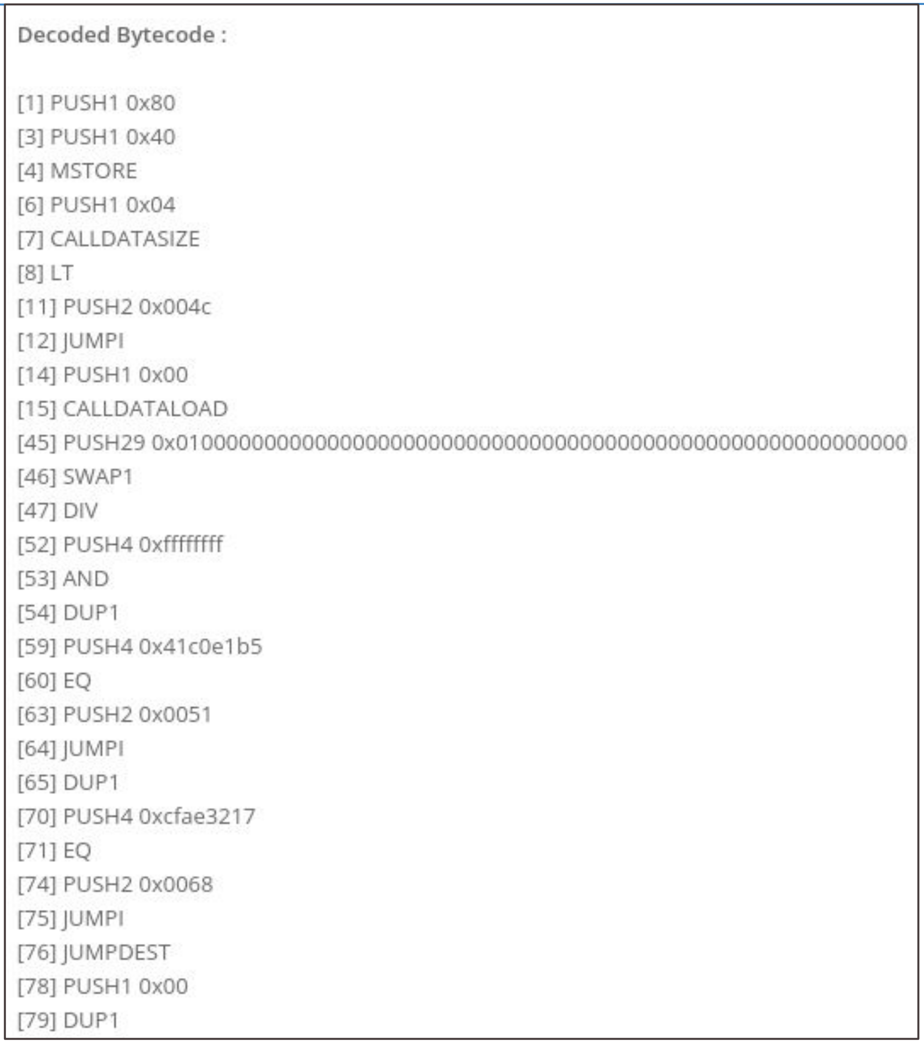


Decoded Bytecode :

```
[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[6] PUSH1 0x04
[7] CALLDATASIZE
[8] LT
[11] PUSH2 0x004c
[12] JUMPI
[14] PUSH1 0x00
[15] CALLDATALOAD
[45] PUSH29 0x0100000000000000000000000000000000000000000000000000000000
[46] SWAP1
[47] DIV
[52] PUSH4 0xffffffff
[53] AND
[54] DUP1
[59] PUSH4 0x41c0e1b5
[60] EQ
[63] PUSH2 0x0051
[64] JUMPI
[65] DUP1
[70] PUSH4 0xcfae3217
[71] EQ
[74] PUSH2 0x0068
[75] JUMPI
[76] JUMPDEST
[78] PUSH1 0x00
[79] DUP1
```

EVM Instructions set

Opcodes value	Family	Examples
0x00 – 0x0B	Stop and Arithmetic Operations	STOP, ADD, SUB, MUL, DIV, EXP
0x10 – 0x1A	Comparison & Bitwise Logic Operations	LT, GT, EQ, ISZERO, AND, XOR
0x20	SHA3	SHA3
0x30 – 0x3E	Environmental Information	ADDRESS, CALLER, CALLDATALOAD
0x40 – 0x45	Block Information	BLOCKHASH, COINBASE, NUMBER
0x50 – 0x5B	Stack, Memory, Storage and Flow Operations	POP, MSTORE, JUMP, JUMPI, JUMPDEST
0x60 – 0x7F	Push Operations	PUSH1 – PUSH32
0x80 – 0x8F	Duplication Operations	DUP1 – DUP16
0x90 – 0x9F	Exchange Operations	SWAP1 – SWAP16
0xA0 – 0xA4	Logging Operations	LOG0 – LOG4
0xF0 – 0xFF	System operations	CALL, RETURN, DELEGATECALL



```
Decoded Bytecode :

[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[6] PUSH1 0x04
[7] CALLDATASIZE
[8] LT
[11] PUSH2 0x004c
[12] JUMPI
[14] PUSH1 0x00
[15] CALLDATALOAD
[45] PUSH29 0x0100000000000000000000000000000000000000000000000000000000
[46] SWAP1
[47] DIV
[52] PUSH4 0xffffffff
[53] AND
[54] DUP1
[59] PUSH4 0x41c0e1b5
[60] EQ
[63] PUSH2 0x0051
[64] JUMPI
[65] DUP1
[70] PUSH4 0xcfae3217
[71] EQ
[74] PUSH2 0x0068
[75] JUMPI
[76] JUMPDEST
[78] PUSH1 0x00
[79] DUP1
```

https://etherscan.io/opcode-tool

CFG Reconstruction

Control flow graph (CFG)

- Control flow graph (CFG) is a graphical representation of the program logic using graph.

- Represented using:
  - Set of Basicblock (i.e. vertices or nodes)
  - Set of Edges

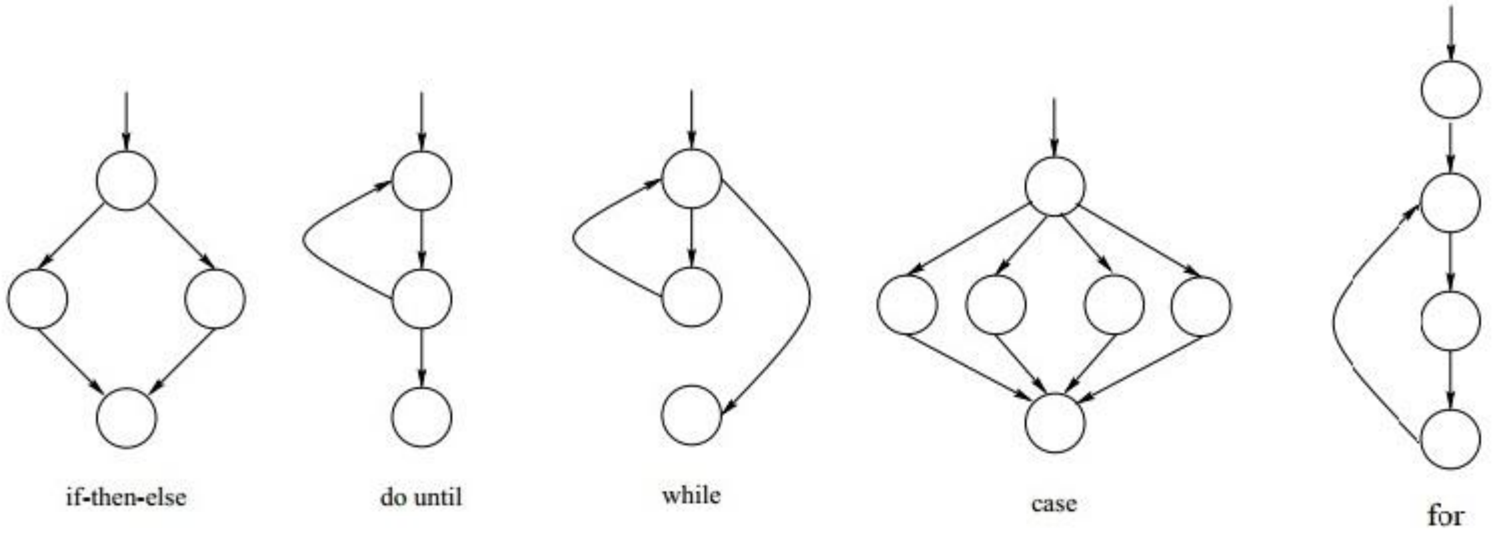


Figure 1: Flow graph representation.

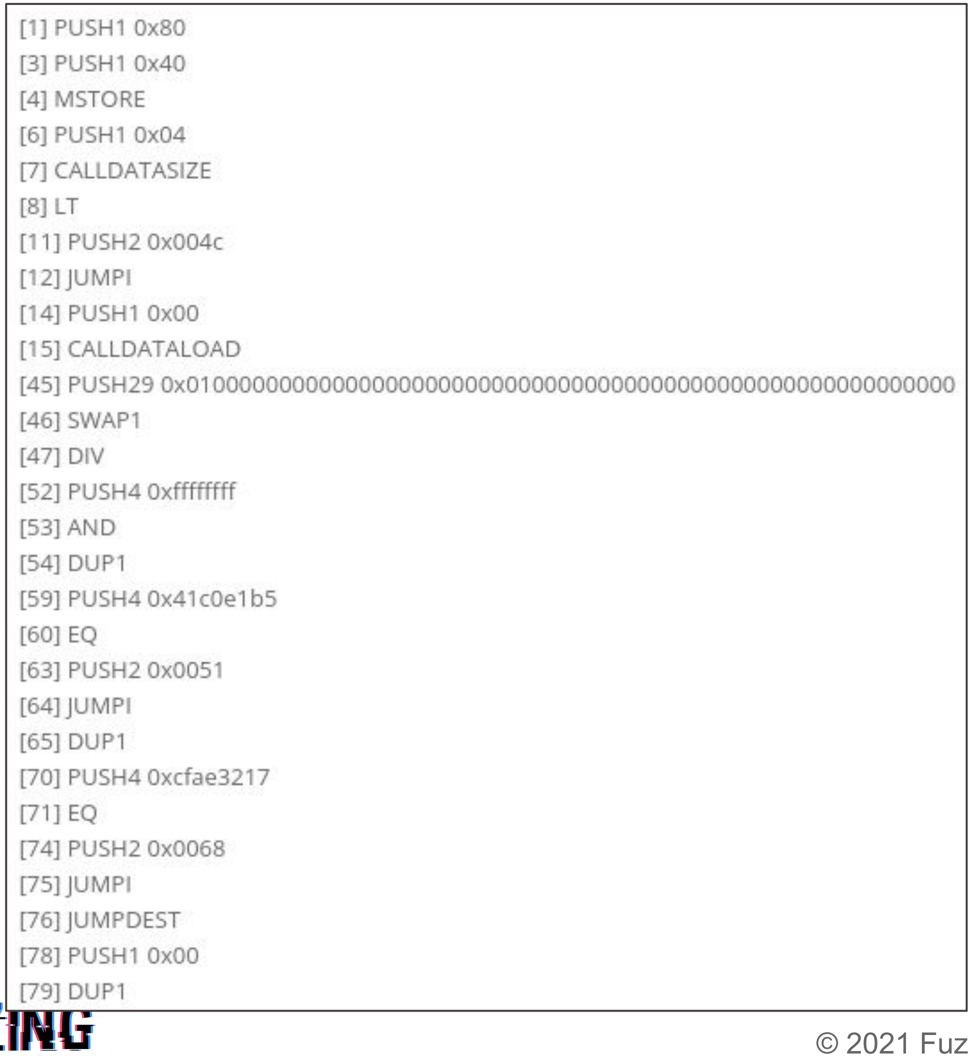
Control flow instructions

Opcodes	Simplify description	Position within a Basicblock
JUMP	Unconditional jump	Last instruction
JUMPI	Conditional jump	Last instruction
RETURN , STOP INVALID SELFDESTRUCT , REVERT	Halt execution	Last instruction
JUMPDEST	Marks a position within the code that is a valid target destination for jumps	First instruction

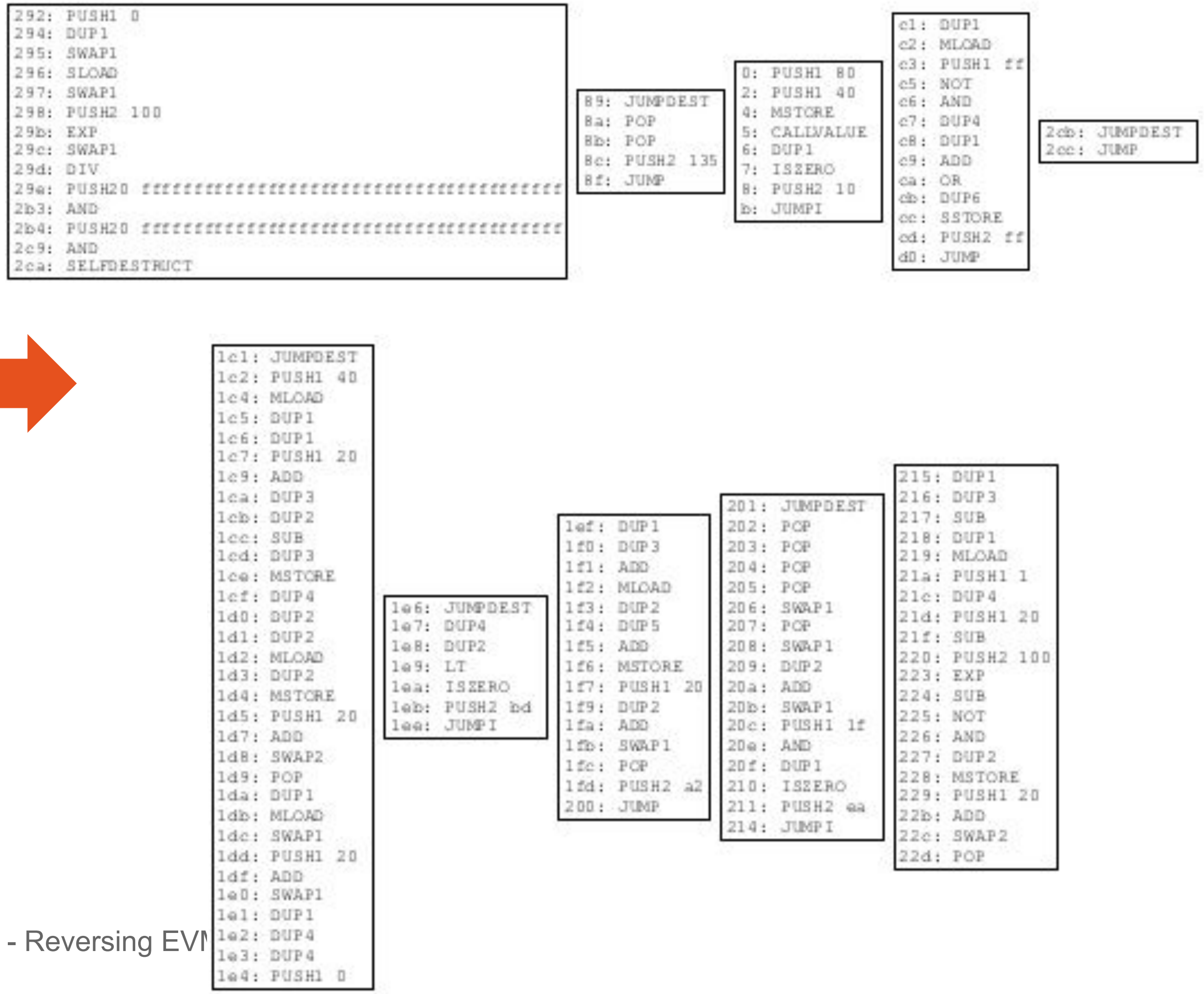
EIP 615: Subroutines and Static Jumps for the EVM By *Greg Colvin*

New branch opcodes: JUMPTO, JUMPIF, JUMPSUB, JUMPSUBV, ….

Decomposition into basic blocks



```
[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[6] PUSH1 0x04
[7] CALLDATASIZE
[8] LT
[11] PUSH2 0x004c
[12] JUMPI
[14] PUSH1 0x00
[15] CALLDATALOAD
[45] PUSH29 0x0100000000000000000000000000000000000000000000000000000000
[46] SWAP1
[47] DIV
[52] PUSH4 0xffffffff
[53] AND
[54] DUP1
[59] PUSH4 0x41c0e1b5
[60] EQ
[63] PUSH2 0x0051
[64] JUMPI
[65] DUP1
[70] PUSH4 0xcfae3217
[71] EQ
[74] PUSH2 0x0068
[75] JUMPI
[76] JUMPDEST
[78] PUSH1 0x00
[79] DUP1
```



```
292: PUSH1 0
294: DUP1
295: SWAP1
296: SLOAD
297: SWAP1
298: PUSH2 100
29b: EXP
29c: SWAP1
29d: DIV
29e: PUSH20 ffffffffffffffffffffffffffffffffffffffff
2b3: AND
2b4: PUSH20 ffffffffffffffffffffffffffffffffffffffff
2c9: AND
2ca: SELFDESTRUCT
```

```
89: JUMPDEST
8a: POP
8b: POP
8c: PUSH2 135
8f: JUMP
```

```
0: PUSH1 80
2: PUSH1 40
4: MSTORE
5: CALLVALUE
6: DUP1
7: ISZERO
8: PUSH2 10
b: JUMPI
```

```
c1: DUP1
c2: MLOAD
c3: PUSH1 ff
c5: NOT
c6: AND
c7: DUP4
c8: DUP1
c9: ADD
ca: OR
cb: DUP6
cc: SSTORE
cd: PUSH2 ff
d0: JUMP
```

```
2cb: JUMPDEST
2cc: JUMP
```

```
1c1: JUMPDEST
1c2: PUSH1 40
1c4: MLOAD
1c5: DUP1
1c6: DUP1
1c7: PUSH1 20
1c9: ADD
1ca: DUP3
1cb: DUP2
1cc: SUB
1cd: DUP3
1ce: MSTORE
1cf: DUP4
1d0: DUP2
1d1: DUP2
1d2: MLOAD
1d3: DUP2
1d4: MSTORE
1d5: PUSH1 20
1d7: ADD
1d8: SWAP2
1d9: POP
1da: DUP1
1db: MLOAD
1dc: SWAP1
1dd: PUSH1 20
1df: ADD
1e0: SWAP1
1e1: DUP1
1e2: DUP4
1e3: DUP4
1e4: PUSH1 0
```

```
1e6: JUMPDEST
1e7: DUP4
1e8: DUP2
1e9: LT
1ea: ISZERO
1eb: PUSH2 bd
1ee: JUMPI
```

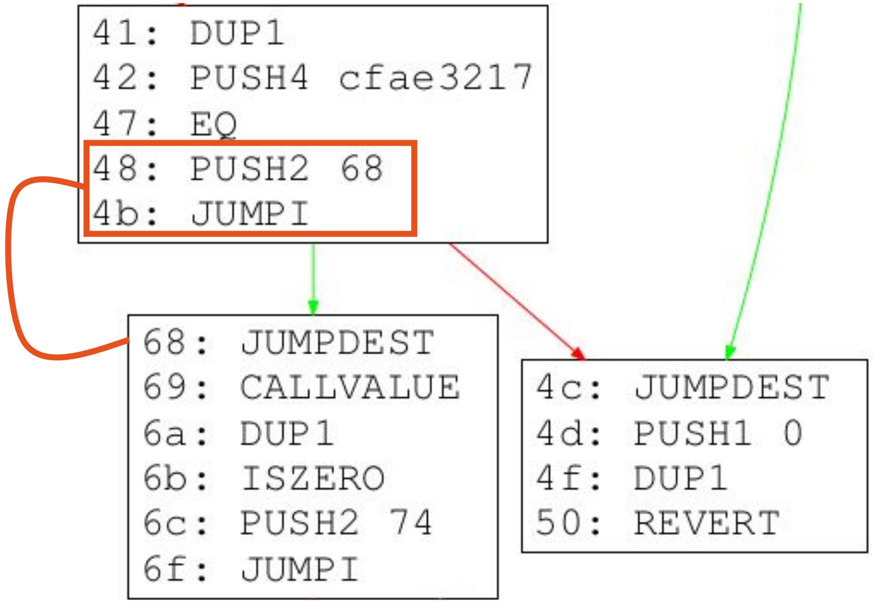
```
1ef: DUP1
1f0: DUP3
1f1: ADD
1f2: MLOAD
1f3: DUP2
1f4: DUP5
1f5: ADD
1f6: MSTORE
1f7: PUSH1 20
1f9: DUP2
1fa: ADD
1fb: SWAP1
1fc: POP
1fd: PUSH2 a2
200: JUMP
```

```
201: JUMPDEST
202: POP
203: POP
204: POP
205: POP
206: SWAP1
207: POP
208: SWAP1
209: DUP2
20a: ADD
20b: SWAP1
20c: PUSH1 1f
20e: AND
20f: DUP1
210: ISZERO
211: PUSH2 ea
214: JUMPI
```

```
215: DUP1
216: DUP3
217: SUB
218: DUP1
219: MLOAD
21a: PUSH1 1
21c: DUP4
21d: PUSH1 20
21f: SUB
220: PUSH2 100
223: EXP
224: SUB
225: NOT
226: AND
227: DUP2
228: MSTORE
229: PUSH1 20
22b: ADD
22c: SWAP2
22d: POP
```

Edges identifications – static analysis

- Basic static analysis works if:
  - Jump target offset is pushed on the stack
  - Just before the JUMP/I



```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 68
4b: JUMPI
```

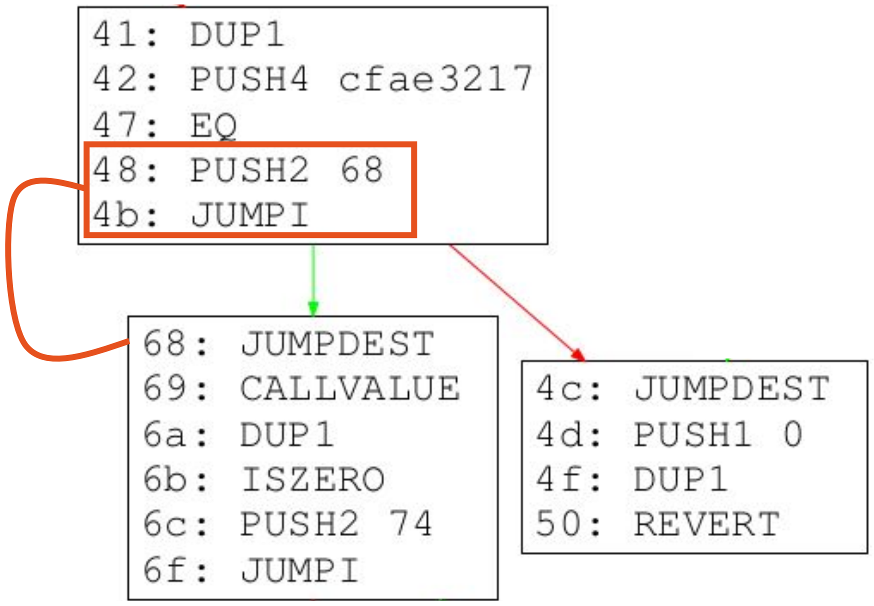
```
68: JUMPDEST
69: CALLVALUE
6a: DUP1
6b: ISZERO
6c: PUSH2 74
6f: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

Edges identifications – static analysis

- Basic static analysis works if:
  - ▸ Jump target offset is pushed on the stack
  - ▸ Just before the JUMP/I

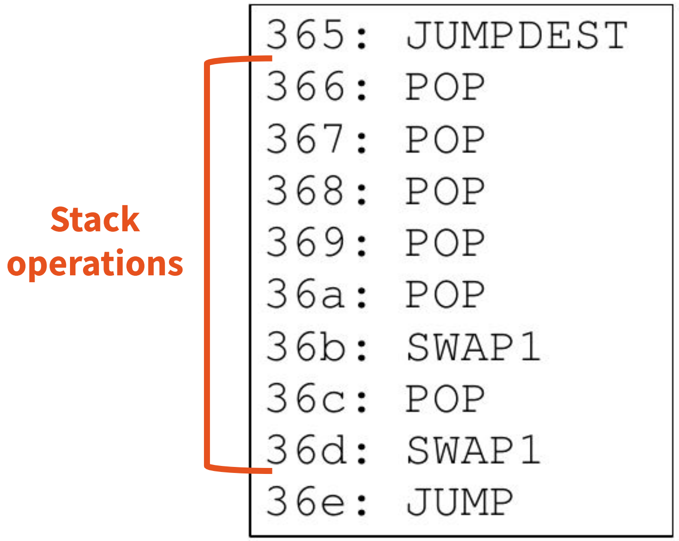
- But fails if:
  - ▸ Stack operations are used to put the jump target offset on top of the stack



```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 68
4b: JUMPI
```

```
68: JUMPDEST
69: CALLVALUE
6a: DUP1
6b: ISZERO
6c: PUSH2 74
6f: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

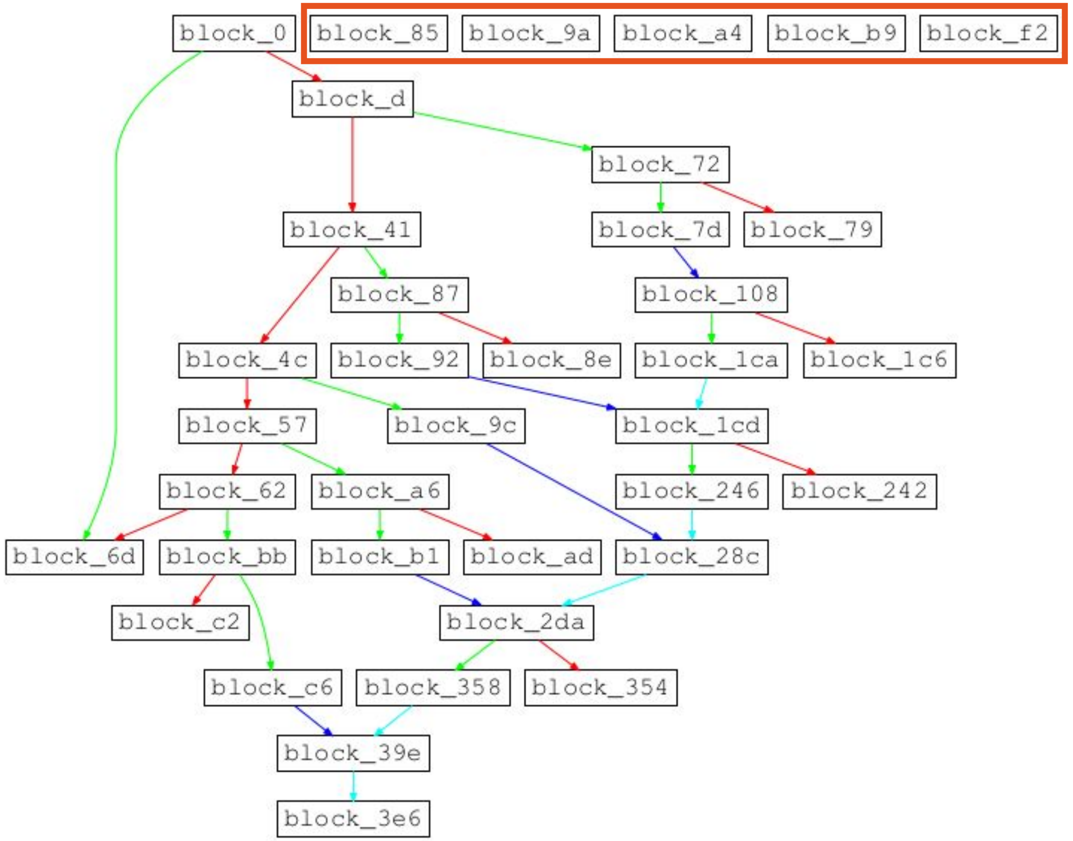


```
365: JUMPDEST
366: POP
367: POP
368: POP
369: POP
36a: POP
36b: SWAP1
36c: POP
36d: SWAP1
36e: JUMP
```

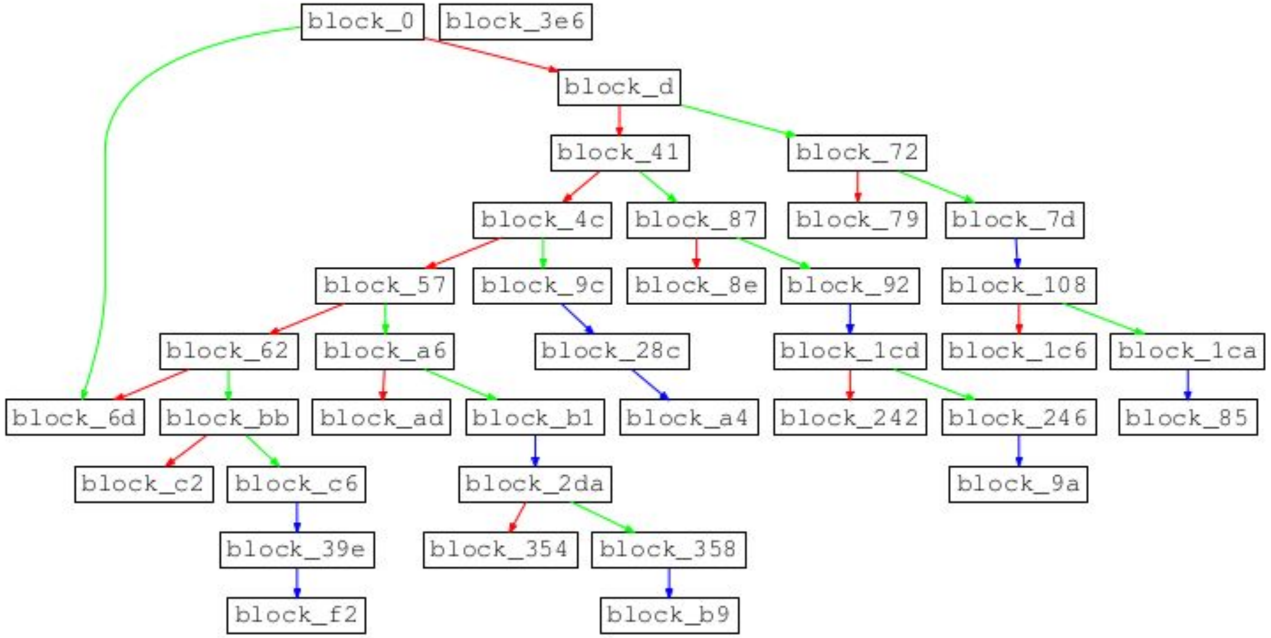
???

Control Flow Graph (CFG) reconstruction

Static analysis

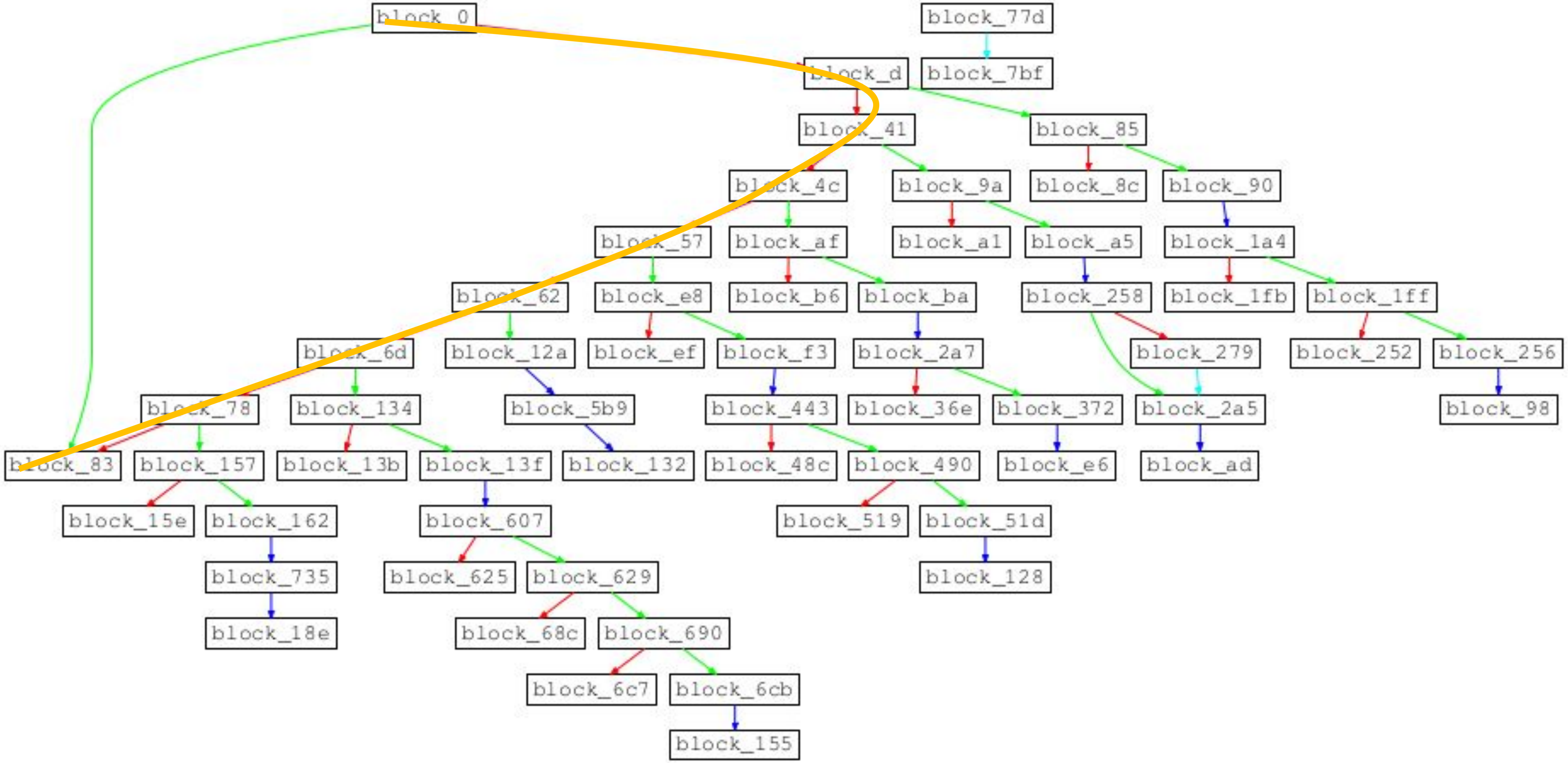


Dynamic analysis (stack evaluation)



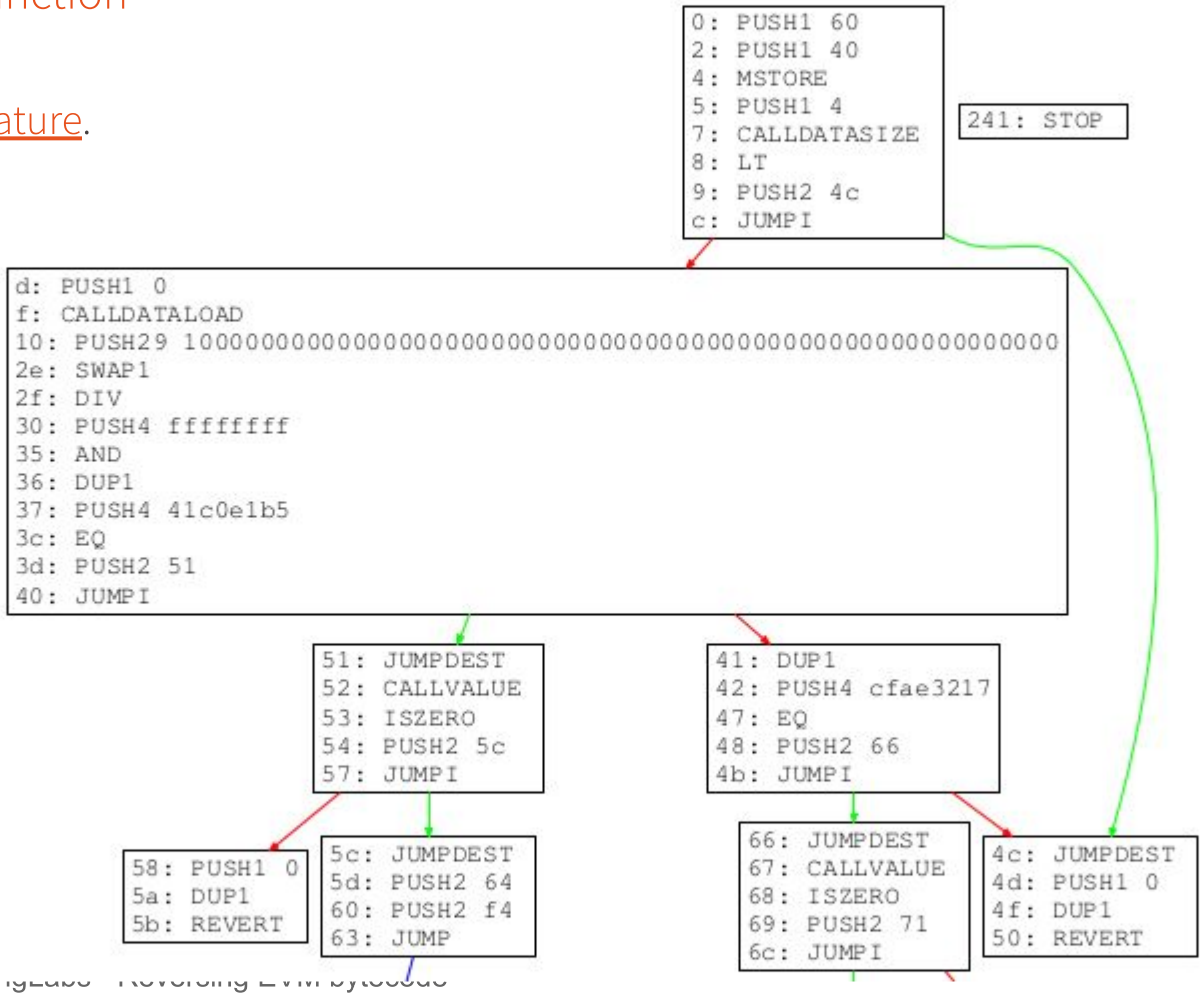
Functions identification

CFG pattern for a “switch”



Dispatcher function

- Runtime code entry point is usually a Dispatcher function
  - Switch on the first 4 bytes of the transaction payload
  - execute the associated code of the given function signature.



```
0: PUSH1 60
2: PUSH1 40
4: MSTORE
5: PUSH1 4
7: CALLDATASIZE
8: LT
9: PUSH2 4c
c: JUMPI
```

```
241: STOP
```

```
d: PUSH1 0
f: CALLDATALOAD
10: PUSH29 100000000000000000000000000000000000000000000000000000000
2e: SWAP1
2f: DIV
30: PUSH4 ffffffff
35: AND
36: DUP1
37: PUSH4 41c0e1b5
3c: EQ
3d: PUSH2 51
40: JUMPI
```

```
51: JUMPDEST
52: CALLVALUE
53: ISZERO
54: PUSH2 5c
57: JUMPI
```

```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 66
4b: JUMPI
```

```
58: PUSH1 0
5a: DUP1
5b: REVERT
```

```
5c: JUMPDEST
5d: PUSH2 64
60: PUSH2 f4
63: JUMP
```

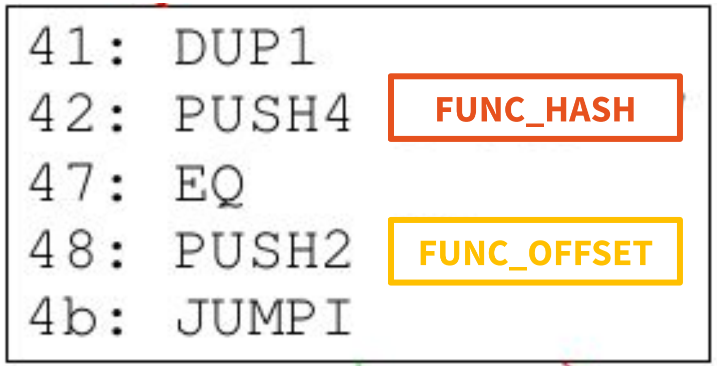
```
66: JUMPDEST
67: CALLVALUE
68: ISZERO
69: PUSH2 71
6c: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

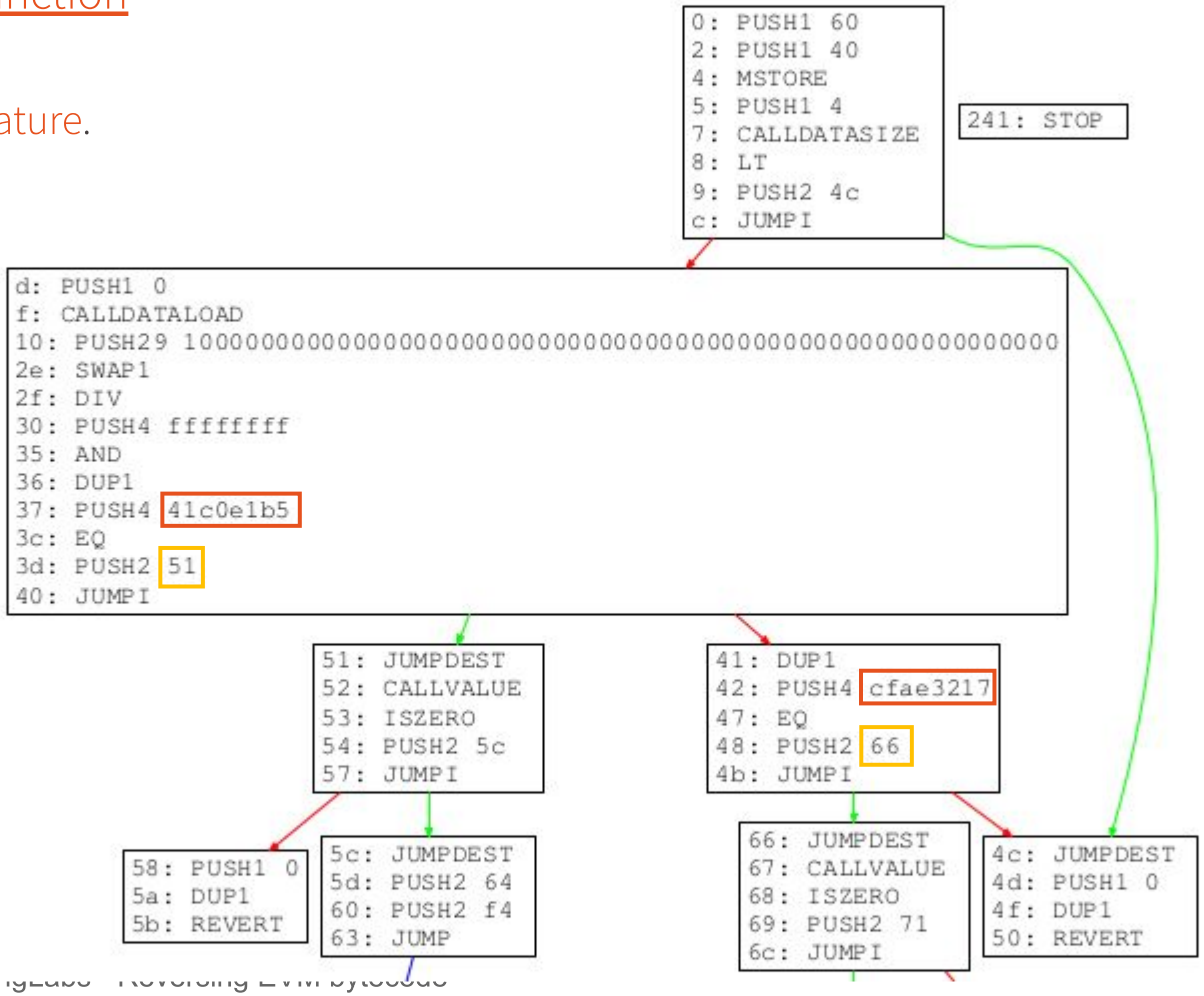
Dispatcher function

- Runtime code entry point is usually a Dispatcher function
  - Switch on the first 4 bytes of the transaction payload
  - execute the associated code of the given function signature.

- Two functions signatures here:
  - **`41c0e1b5`**
  - **`cfae3217`**



```
41: DUP1
42: PUSH4
47: EQ
48: PUSH2
4b: JUMPI
```



```
0: PUSH1 60
2: PUSH1 40
4: MSTORE
5: PUSH1 4
7: CALLDATASIZE
8: LT
9: PUSH2 4c
c: JUMPI
```

```
241: STOP
```

```
d: PUSH1 0
f: CALLDATALOAD
10: PUSH29 100000000000000000000000000000000000000000000000000000000
2e: SWAP1
2f: DIV
30: PUSH4 ffffffff
35: AND
36: DUP1
37: PUSH4 41c0e1b5
3c: EQ
3d: PUSH2 51
40: JUMPI
```

```
51: JUMPDEST
52: CALLVALUE
53: ISZERO
54: PUSH2 5c
57: JUMPI
```

```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 66
4b: JUMPI
```

```
58: PUSH1 0
5a: DUP1
5b: REVERT
```

```
5c: JUMPDEST
5d: PUSH2 64
60: PUSH2 f4
63: JUMP
```

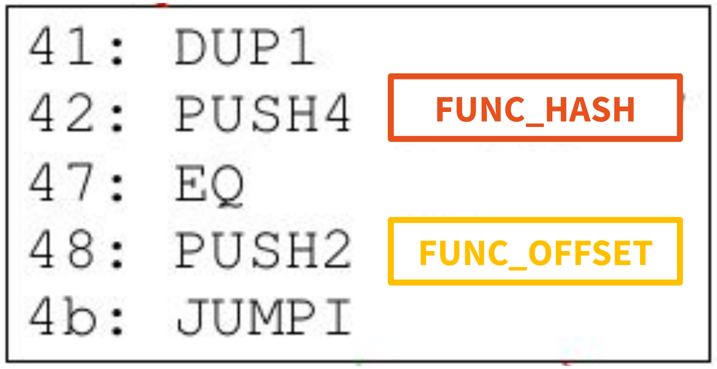
```
66: JUMPDEST
67: CALLVALUE
68: ISZERO
69: PUSH2 71
6c: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

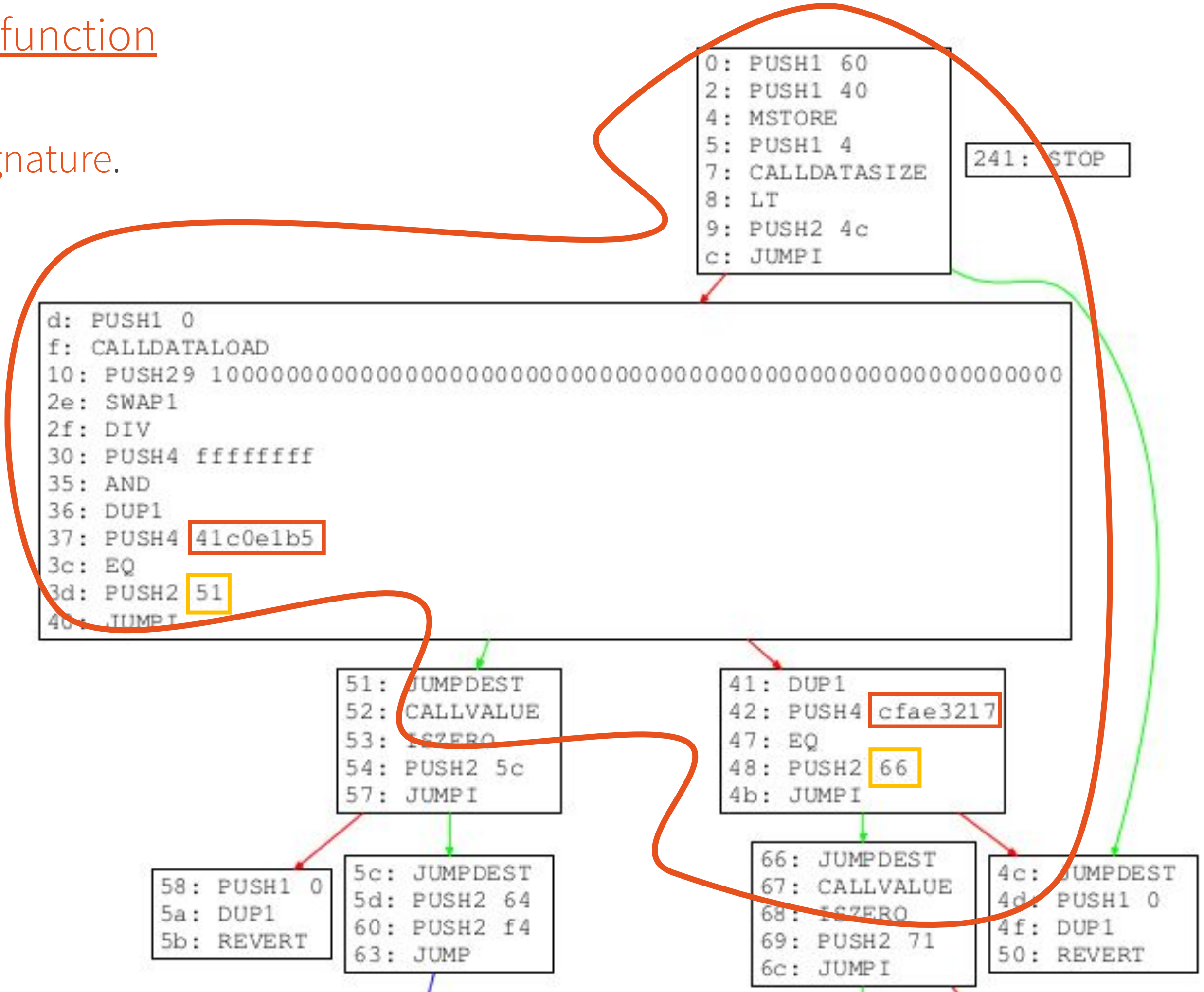
Dispatcher function

- Runtime code entry point is usually a Dispatcher function
  - Switch on the first 4 bytes of the transaction payload
  - execute the associated code of the given function signature.

- Two functions signatures here:
  - **41c0e1b5**
  - **cfae3217**



```
41: DUP1
42: PUSH4  FUNC_HASH
47: EQ
48: PUSH2  FUNC_OFFSET
4b: JUMPI
```



```
0: PUSH1 60
2: PUSH1 40
4: MSTORE
5: PUSH1 4
7: CALLDATASIZE
8: LT
9: PUSH2 4c
c: JUMPI
```

```
241: STOP
```

```
d: PUSH1 0
f: CALLDATALOAD
10: PUSH29 100000000000000000000000000000000000000000000000000000000
2e: SWAP1
2f: DIV
30: PUSH4 ffffffff
35: AND
36: DUP1
37: PUSH4 41c0e1b5
3c: EQ
3d: PUSH2 51
40: JUMPI
```

```
51: JUMPDEST
52: CALLVALUE
53: ISZERO
54: PUSH2 5c
57: JUMPI
```

```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 66
4b: JUMPI
```

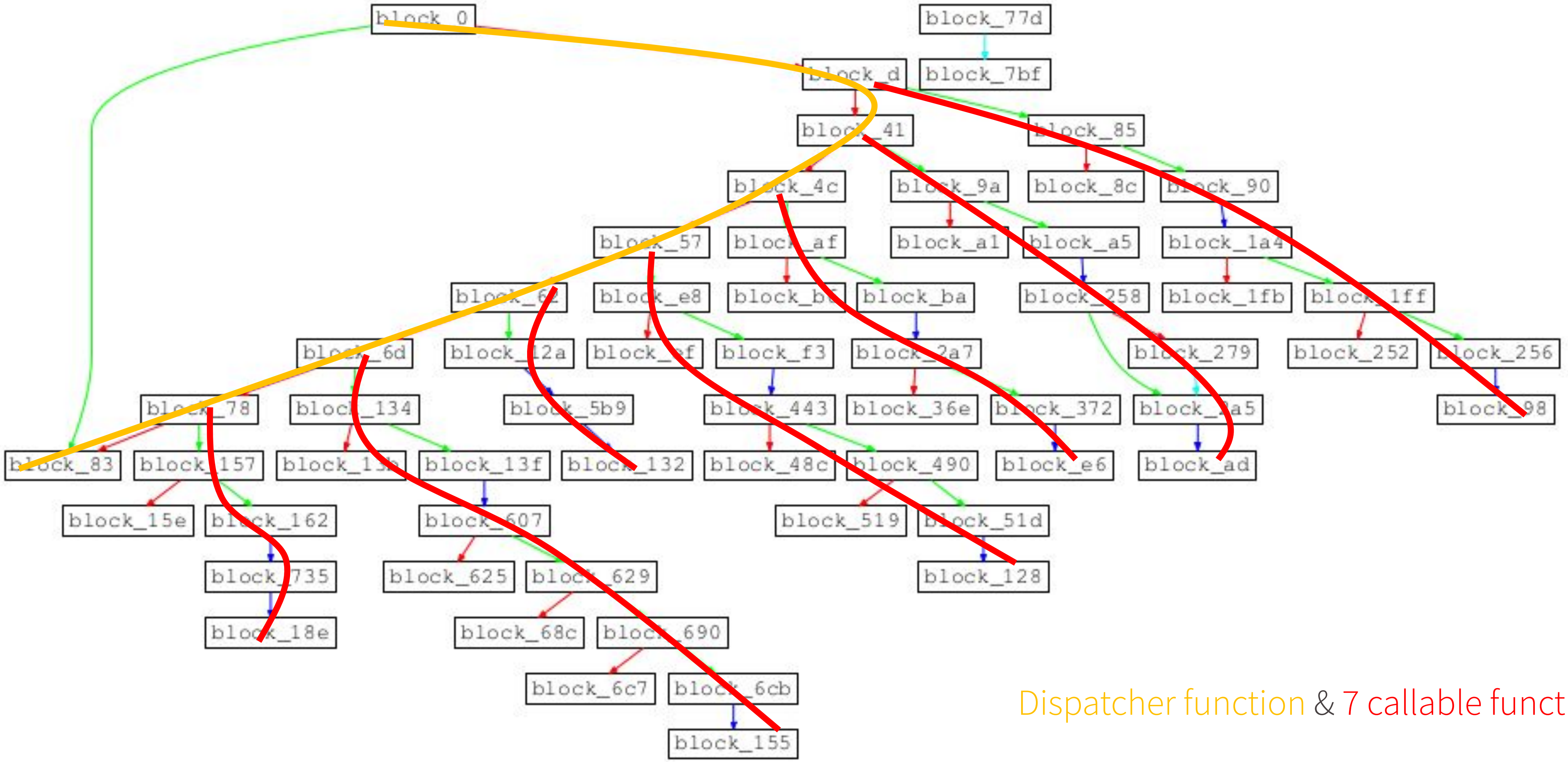
```
58: PUSH1 0
5a: DUP1
5b: REVERT
```

```
5c: JUMPDEST
5d: PUSH2 64
60: PUSH2 f4
63: JUMP
```

```
66: JUMPDEST
67: CALLVALUE
68: ISZERO
69: PUSH2 71
6c: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

Functions identification - Depth First Search

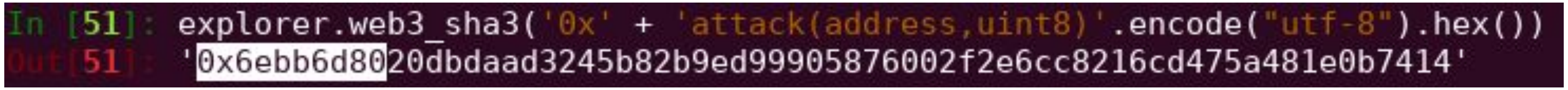


Dispatcher function & 7 callable functions

Functions name recovery

Functions signatures – 4byte identifiers

- Function signatures/identifiers: First 4 bytes of the sha3 (keccak256) of the function prototype text

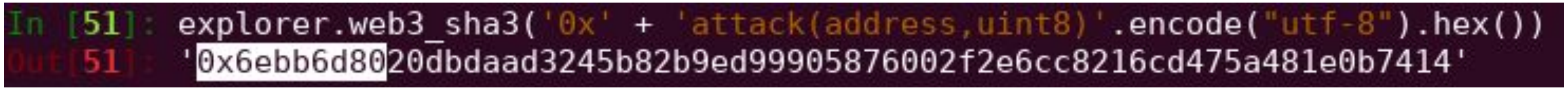


```
In [51]: explorer.web3_sha3('0x' + 'attack(address,uint8)'.encode("utf-8").hex())
Out[51]: '0x6ebb6d8020dbdaad3245b82b9ed99905876002f2e6cc8216cd475a481e0b7414'
```

- In the previous example:
  - kill() == 0x41c0e1b5eba5f1ef69db2e30c1ec7d6e0a5f3d39332543a8a99d1165e460a49e
  - greet() = 0xcfae3217c5b262aa4fd3346d6d110ec3c0361903298087be8626cb438090d274

Functions signatures – 4byte identifiers

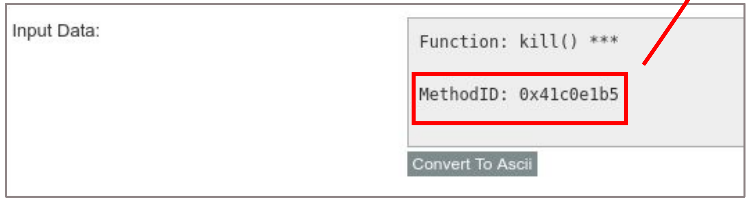
- Function signatures/identifiers: First 4 bytes of the sha3 (keccak256) of the function prototype text



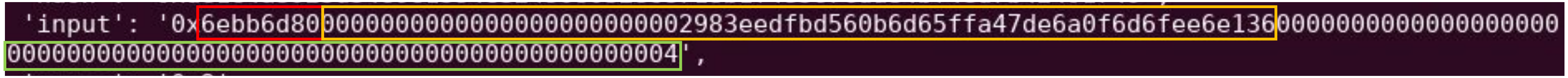
```
In [51]: explorer.web3_sha3('0x' + 'attack(address,uint8)'.encode("utf-8").hex())
Out[51]: '0x6ebb6d8020dbdaad3245b82b9ed99905876002f2e6cc8216cd475a481e0b7414'
```

- In the previous example:
  - kill() == 0x41c0e1b5eba5f1ef69db2e30c1ec7d6e0a5f3d39332543a8a99d1165e460a49e
  - greet() = 0xcfae3217c5b262aa4fd3346d6d110ec3c0361903298087be8626cb438090d274

- When you interact with a contract:
  - You send the function signature (MethodID) followed by the arguments
  - Signature, Argument #1, Argument #2 (256-bits words)

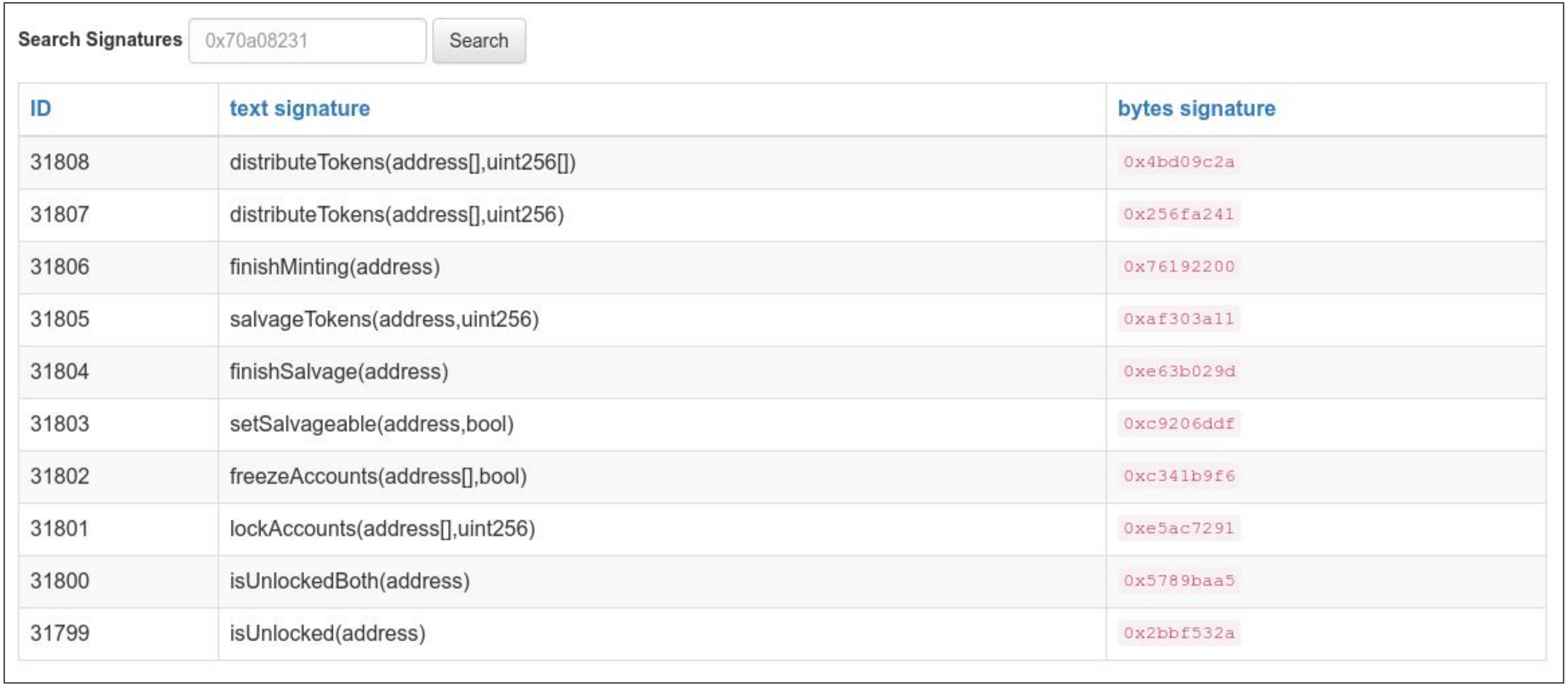


```
Input Data:
Function: kill() ***

MethodID: 0x41c0e1b5
```



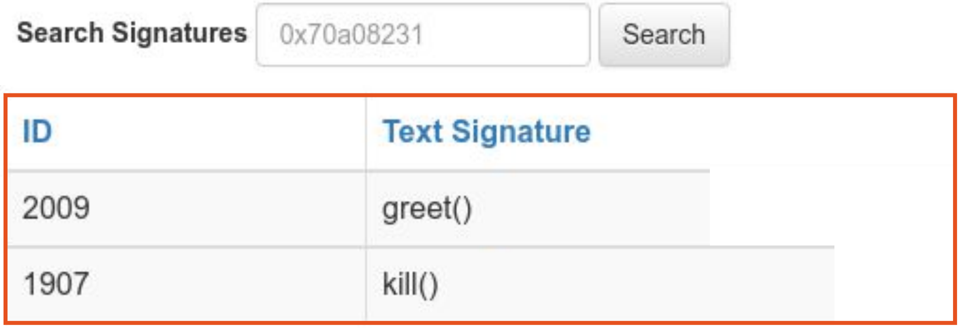
```
 'input': '0x6ebb6d80000000000000000000000000002983eedfbd560b6d65ffa47de6a0f6d6fee6e1360000000000000000000
0000000000000000000000000000000000000000000004',
```

Function signature reverse lookup database

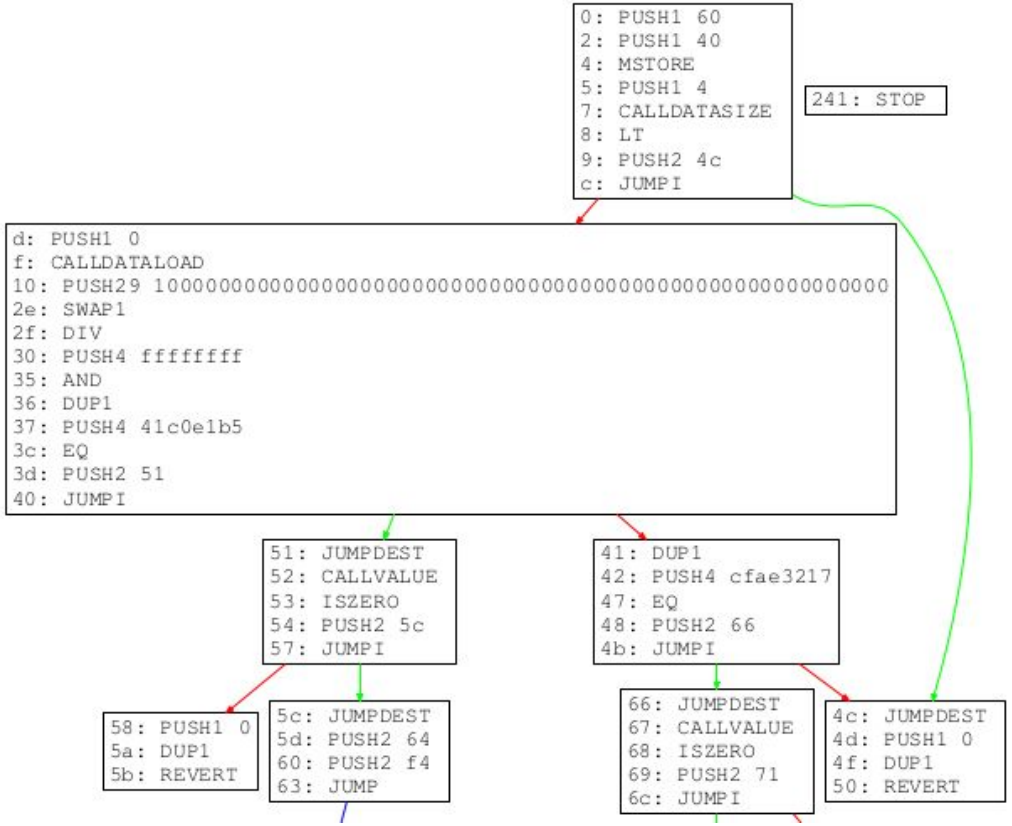


ID	text signature	bytes signature
31808	distributeTokens(address[],uint256[])	0x4bd09c2a
31807	distributeTokens(address[],uint256)	0x256fa241
31806	finishMinting(address)	0x76192200
31805	salvageTokens(address,uint256)	0xaf303a11
31804	finishSalvage(address)	0xe63b029d
31803	setSalvageable(address,bool)	0xc9206ddf
31802	freezeAccounts(address[],bool)	0xc341b9f6
31801	lockAccounts(address[],uint256)	0xe5ac7291
31800	isUnlockedBoth(address)	0x5789baa5
31799	isUnlocked(address)	0x2bbf532a

Functions name & arguments type recovery



ID	Text Signature
2009	greet()
1907	kill()



```
0: PUSH1 60
2: PUSH1 40
4: MSTORE
5: PUSH1 4
7: CALLDATASIZE
8: LT
9: PUSH2 4c
c: JUMPI
```

```
241: STOP
```

```
d: PUSH1 0
f: CALLDATALOAD
10: PUSH29 100000000000000000000000000000000000000000000000000000000
2e: SWAP1
2f: DIV
30: PUSH4 ffffffff
35: AND
36: DUP1
37: PUSH4 41c0e1b5
3c: EQ
3d: PUSH2 51
40: JUMPI
```

```
51: JUMPDEST
52: CALLVALUE
53: ISZERO
54: PUSH2 5c
57: JUMPI
```

```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 66
4b: JUMPI
```

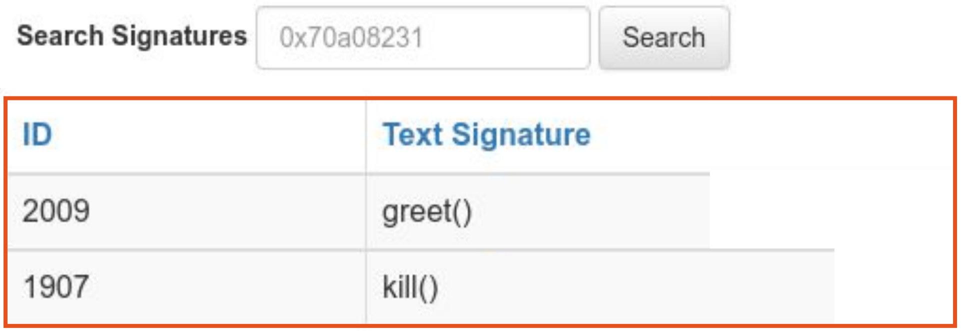
```
58: PUSH1 0
5a: DUP1
5b: REVERT
```

```
5c: JUMPDEST
5d: PUSH2 64
60: PUSH2 f4
63: JUMP
```

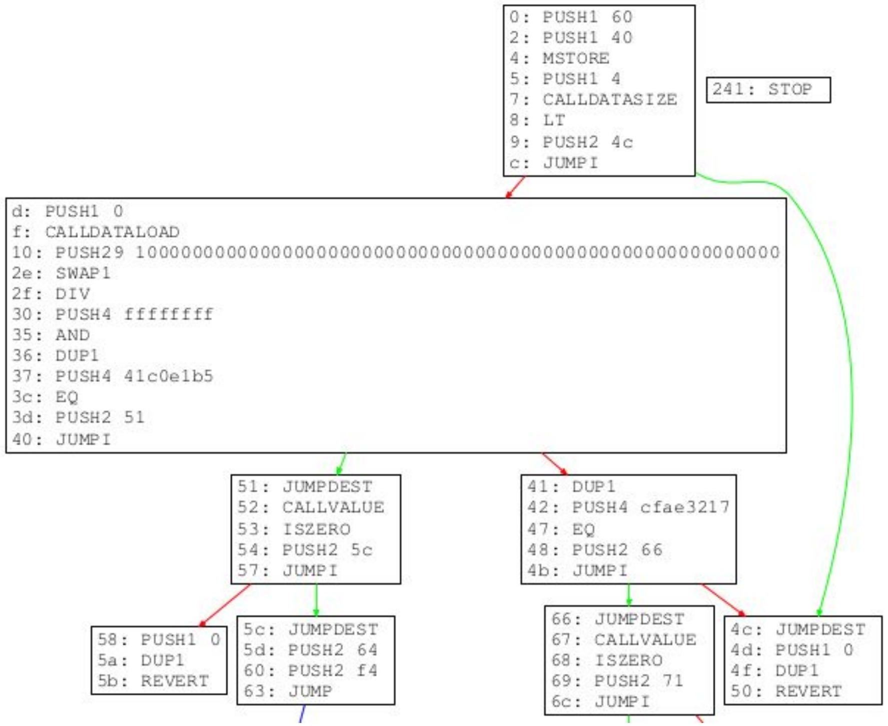
```
66: JUMPDEST
67: CALLVALUE
68: ISZERO
69: PUSH2 71
6c: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

Functions name & arguments type recovery



ID	Text Signature
2009	greet()
1907	kill()



```
0: PUSH1 60
2: PUSH1 40
4: MSTORE
5: PUSH1 4
7: CALLDATASIZE
8: LT
9: PUSH2 4c
c: JUMPI
```

```
241: STOP
```

```
d: PUSH1 0
f: CALLDATALOAD
10: PUSH29 100000000000000000000000000000000000000000000000000000000
2e: SWAP1
2f: DIV
30: PUSH4 ffffffff
35: AND
36: DUP1
37: PUSH4 41c0e1b5
3c: EQ
3d: PUSH2 51
40: JUMPI
```

```
51: JUMPDEST
52: CALLVALUE
53: ISZERO
54: PUSH2 5c
57: JUMPI
```

```
41: DUP1
42: PUSH4 cfae3217
47: EQ
48: PUSH2 66
4b: JUMPI
```

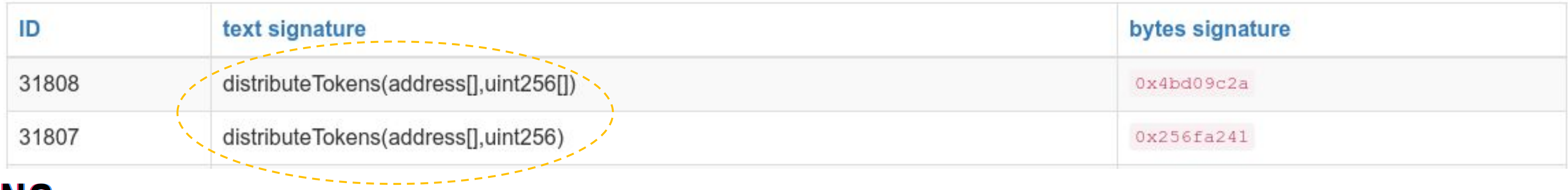
```
58: PUSH1 0
5a: DUP1
5b: REVERT
```

```
5c: JUMPDEST
5d: PUSH2 64
60: PUSH2 f4
63: JUMP
```

```
66: JUMPDEST
67: CALLVALUE
68: ISZERO
69: PUSH2 71
6c: JUMPI
```

```
4c: JUMPDEST
4d: PUSH1 0
4f: DUP1
50: REVERT
```

- Allow you to recover:
  - Function names
  - Arguments types



ID	text signature	bytes signature
31808	distributeTokens(address[],uint256[])	0x4bd09c2a
31807	distributeTokens(address[],uint256)	0x256fa241

Why using reversing?

Reversing & EVM bytecode analysis for…

- Users/ICO
  - ▶ Due diligence
  - ▶ Understand the Logic
  - ▶ CTF competition

- Security researcher
  - ▶ Bug hunting
  - ▶ Vulnerability research

- Company
  - ▶ Security audit
  - ▶ Bytecode Optimization

- Threat intelligence team
  - ▶ Transaction tracking
  - ▶ Analyze smart contract interactions

Reversing & EVM bytecode analysis for…

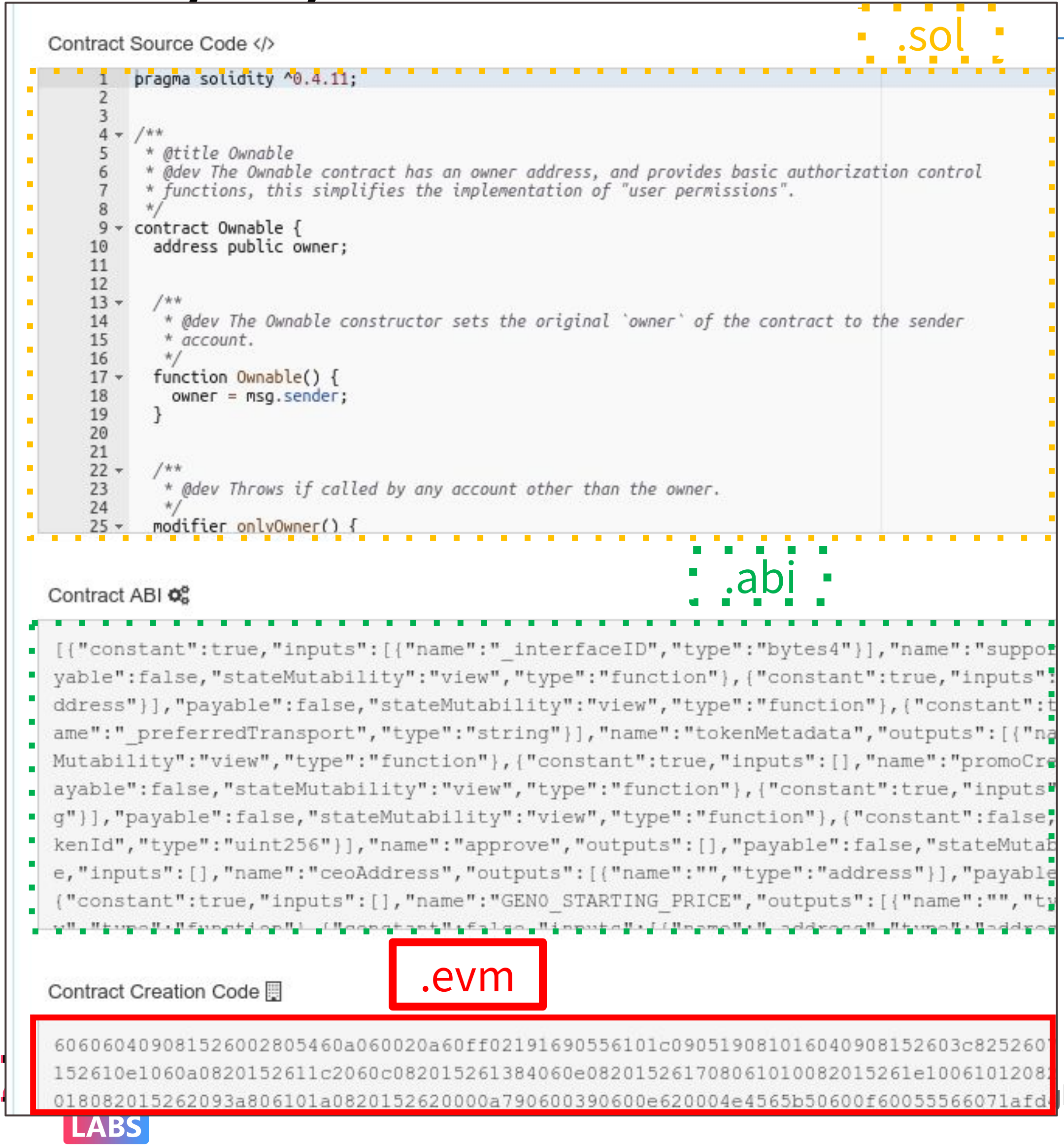
- **Users/ICO**
  - **Due diligence**
  - **Understand the Logic**
  - **CTF competition**

- Security researcher
  - Bug hunting
  - Vulnerability research

- Company
  - Security audit
  - Bytecode Optimization

- Threat intelligence team
  - Transaction tracking
  - Analyze smart contract interactions

Only bytecode is mandatory



Contract Source Code </>

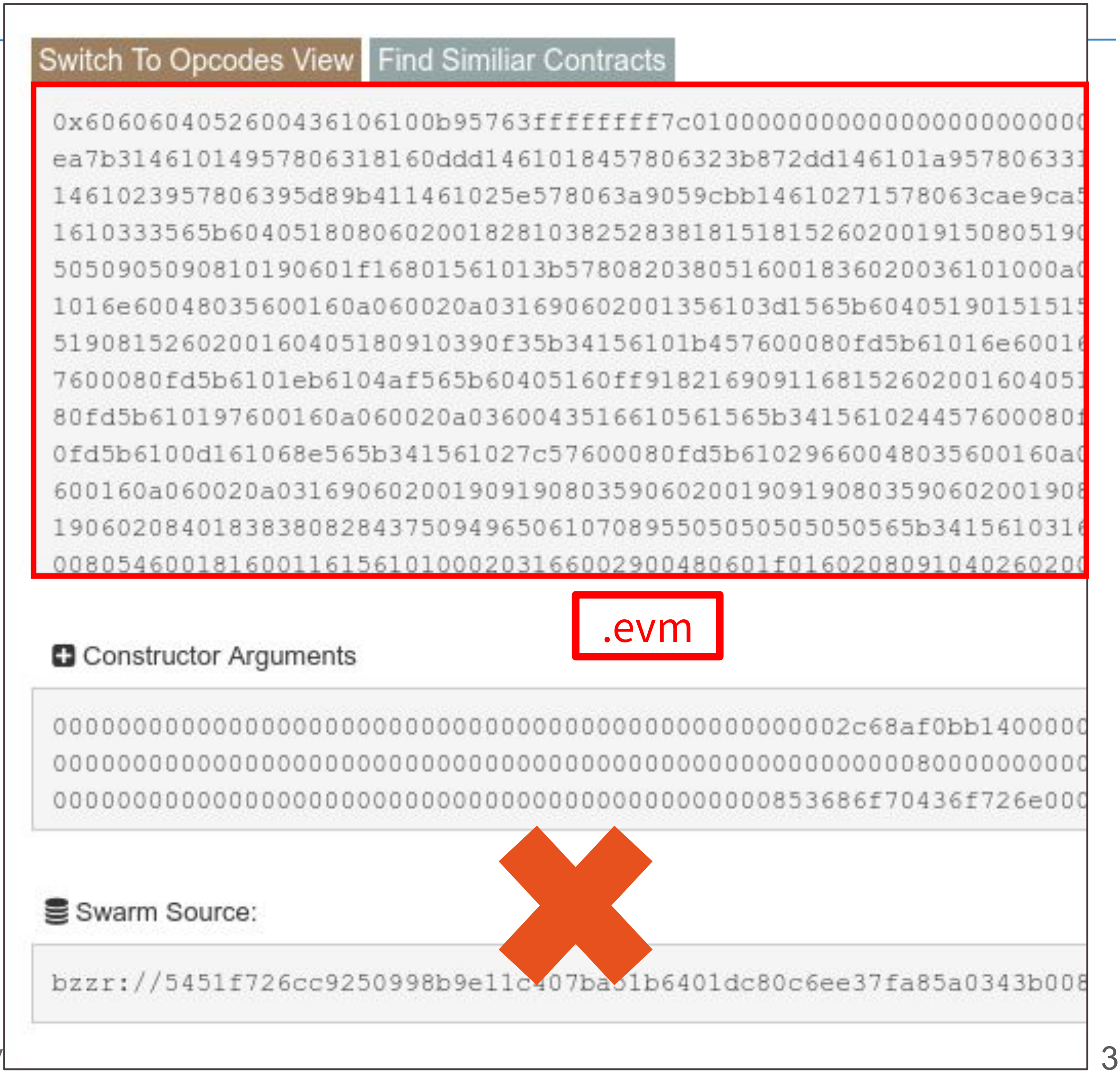
```
pragma solidity ^0.4.11;


/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;


  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  function Ownable() {
    owner = msg.sender;
  }


  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
```

Contract ABI

```
[{"constant":true,"inputs":[{"name":"_interfaceID","type":"bytes4"}],"name":"suppo
yable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":
ddress"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":t
ame":"_preferredTransport","type":"string"}],"name":"tokenMetadata","outputs":[{"na
Mutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"promoCre
ayable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs"
g"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,
kenId","type":"uint256"}],"name":"approve","outputs":[],"payable":false,"stateMuta
e,"inputs":[],"name":"ceoAddress","outputs":[{"name":"","type":"address"}],"payable
{"constant":true,"inputs":[],"name":"GEN0_STARTING_PRICE","outputs":[{"name":"","t
```

Contract Creation Code

```
606060409081526002805460a060020a60ff02191690556101c090519081016040908152603c825260
152610e1060a0820152611c2060c082015261384060e082015261708061010082015261e1006101208
018082015262093a806101a082015262000a790600390600e620004e4565b50600f60055566071afd
```



```
0x6060604052600436106100b95763ffffffff7c0100000000000000000000000
ea7b31461014957806318160ddd1461018457806323b872dd146101a95780633
1461023957806395d89b411461025e578063a9059cbb14610271578063cae9ca
1610333565b604051808060200182810382528381815181526020019150805190
5050905090810190601f16801561013b5780820380516001836020036101000a
1016e60048035600160a060020a031690602001356103d1565b604051901515
5190815260200160405180910390f35b34156101b457600080fd5b61016e6001
7600080fd5b6101eb6104af565b60405160ff918216909116815260200160405
80fd5b610197600160a060020a0360043516610561565b341561024457600080
0fd5b6100d161068e565b341561027c57600080fd5b61029660048035600160a
600160a060020a031690602001909190803590602001909190803590602001908
19060208401838380828437509496506107089550505050505050565b3415610316
008054600181600116156101000203166002900480601f0160208091040260200
```

Constructor Arguments

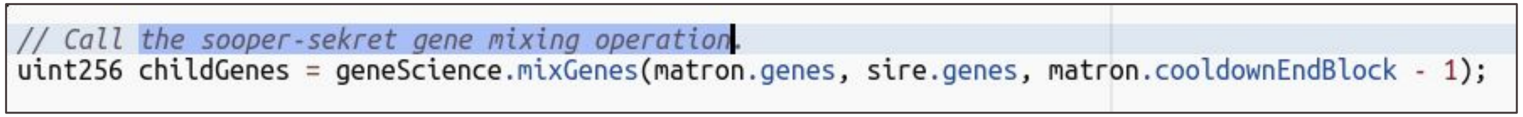
```
00000000000000000000000000000000000000000000002c68af0bb1400000
00000000000000000000000000000000000000000000000000800000000000
0000000000000000000000000000000000000000000853686f70436f726e000
```

Swarm Source:

```
bzzr://5451f726cc9250998b9e11c407ba51b6401dc80c6ee37fa85a0343b008
```

CryptoKitties “geneScience” contract

- *“the sooper-sekret gene mixing operation” (0xf97e0a5b616dffc913e72455fde9ea8bbe946a2b)*
  - ▶ Call to the `mixGenes` function of the `geneScience` external contract
  - ▶ `GeneScience.sol` not release on there bug-bounty github

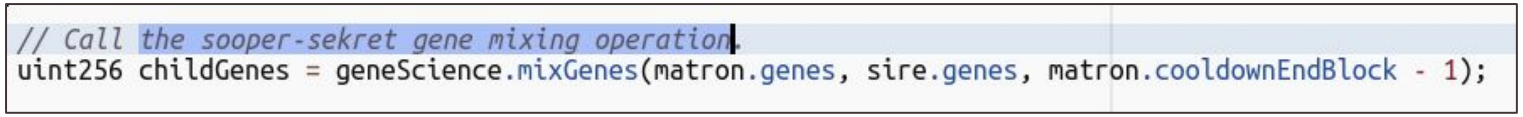


```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```

CryptoKittiesCore

CryptoKitties “geneScience” contract

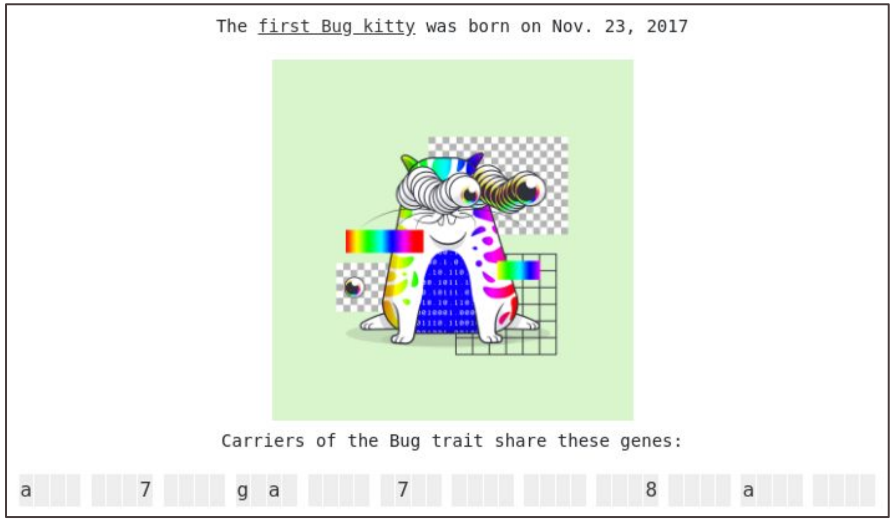
- “*the sooper-sekret gene mixing operation*”
  - Call to the `mixGenes` function of the `geneScience` external contract
  - `GeneScience.sol` not release on there bug-bounty github



```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```

CryptoKittiesCore

- Community started by diffing kitties genome DNA
  - in order to isolate the genes associated with a specific trait.
  - CryptoKittydex website



The first Bug kitty was born on Nov. 23, 2017

Carriers of the Bug trait share these genes:

```
a    7    g a    7    8    a
```

- Reversed & analyzed by the community
  - CryptoKitties GeneScience algorithm by *Alex Hegyi*
  - Towards Cracking Crypto Kitties’ Genetic Code by *Mo Dong*
  - CryptoKitties mixGenes Function by *Sean Soria*
  - The CryptoKitties Genome Project by kaigani

Reversing & EVM bytecode analysis for…

- Users/ICO
  - Due diligence
  - Understand the Logic
  - CTF competition

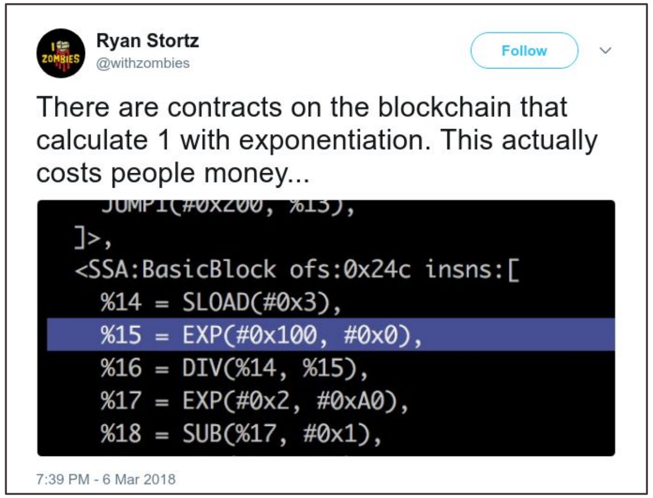
- Security researcher
  - Bug hunting
  - Vulnerability research

- **Company**
  - **Security audit**
  - **Bytecode Optimization**

- Threat intelligence team
  - Transaction tracking
  - Analyze smart contract interactions

Bytecode optimization – Exponentiation

- Exponentiation (`EXP`) can be optimize (save 10 GAS) in two cases:
  - If args of `EXP` are constants
    - compiler should calculate the result
    - replace directly `EXP` by a `PUSH result` instruction
  - If one of the `EXP` args == 0 | 1
    - result can be calculate at compilation
    - even if the other `EXP` argument are a runtime variable (`CALLDATALOAD/MLOAD/SLOAD/`…)



Ryan Stortz
@withzombies

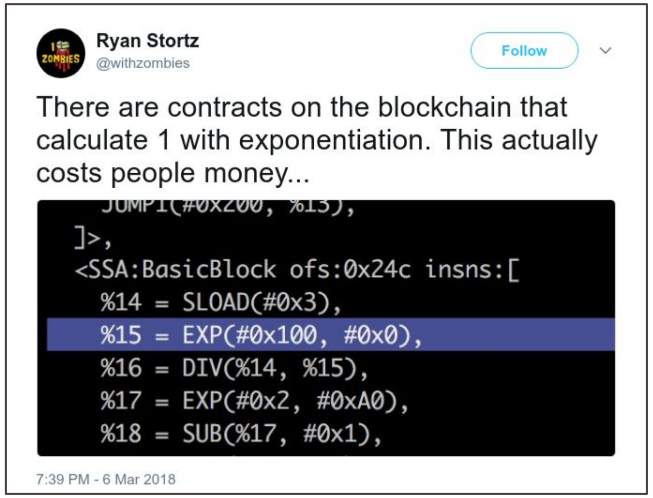
There are contracts on the blockchain that calculate 1 with exponentiation. This actually costs people money...

```
    JUMPI(#0x200, %13),
  ]>,
  <SSA:BasicBlock ofs:0x24c insns:[
    %14 = SLOAD(#0x3),
    %15 = EXP(#0x100, #0x0),
    %16 = DIV(%14, %15),
    %17 = EXP(#0x2, #0xA0),
    %18 = SUB(%17, #0x1),
```

7:39 PM - 6 Mar 2018

Bytecode optimization – Exponentiation

- Exponentiation (`EXP`) can be optimize (save 10 GAS) in two cases:
  - If args of `EXP` are constants
    - compiler should calculate the result
    - replace directly `EXP` by a `PUSH result` instruction
  - If one of the `EXP` args == 0 | 1
    - result can be calculate at compilation
    - even if the other `EXP` argument are a runtime variable (`CALLDATALOAD/MLOAD/SLOAD/`…)



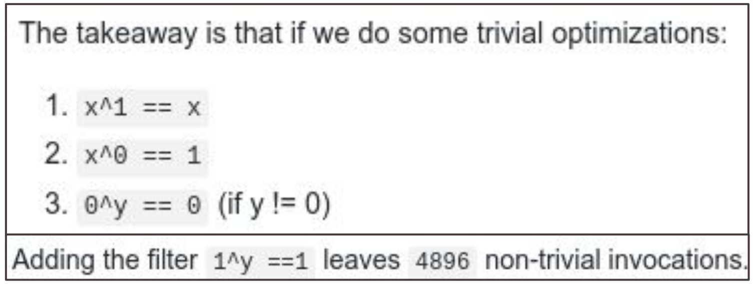
Ryan Stortz
@withzombies

There are contracts on the blockchain that calculate 1 with exponentiation. This actually costs people money...

```
    JUMPI(#0x200, %13),
  ]>,
  <SSA:BasicBlock ofs:0x24c insns:[
    %14 = SLOAD(#0x3),
    %15 = EXP(#0x100, #0x0),
    %16 = DIV(%14, %15),
    %17 = EXP(#0x2, #0xA0),
    %18 = SUB(%17, #0x1),
```

7:39 PM - 6 Mar 2018

- Martin Holst Swende (holiman) reproduces this on 16 random blocks:
  - 18538 invocations of `EXP`
  - 4896 are non-trivial
  - 13642 can be optimize == 73.5 % of EXP invocations



The takeaway is that if we do some trivial optimizations:

1. `x^1 == x`
2. `x^0 == 1`
3. `0^y == 0` (if y != 0)

Adding the filter `1^y ==1` leaves `4896` non-trivial invocations.

- Maybe those optimization are done using `solc --optimize`
  - But compiler should optimize that behavior by default without any extra flag

Reversing & EVM bytecode analysis for…

- Users/ICO
  - Due diligence
  - Understand the Logic
  - CTF competition

- **Security researcher**
  - **Bug hunting**
  - **Vulnerability research**

- Company
  - Security audit
  - Bytecode Optimization

- Threat intelligence team
  - Transaction tracking
  - Analyze smart contract interactions

Vulnerability research / Bug finding

- All those tools uses directly the EVM bytecode to perform their analysis

- Octopus: Security Analysis tool for Blockchain Smart Contracts (BTC/ETH/NEO/EOS)
- Mythril: Security analysis tool for Ethereum smart contracts
- Securify: Security Scanner for Ethereum Smart Contracts
- Rattle: evm binary static analysis
- Echidna: Ethereum fuzz testing framework
- ethervm.io: Online Solidity Decompiler
- …

Reversing & EVM bytecode analysis for…

- Users/ICO
  - Due diligence
  - Understand the Logic
  - CTF competition

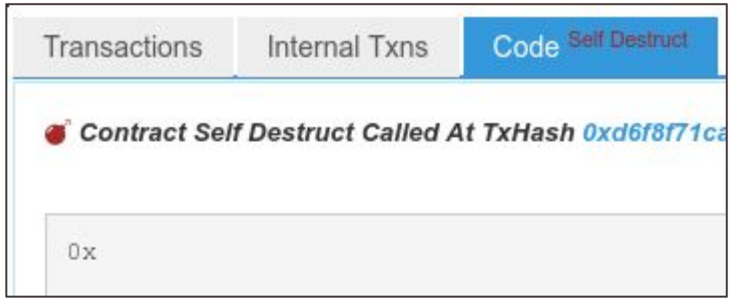
- Security researcher
  - Bug hunting
  - Vulnerability research

- Company
  - Security audit
  - Bytecode Optimization

- **Threat intelligence team**
  - **Transaction tracking**
  - **Analyze smart contract interactions**

Post-mortem smart contract analysis

- If you want to analyze post-mortem interactions of a destroyed smart contract
  - When a smart contract is destroyed, code is no longer available

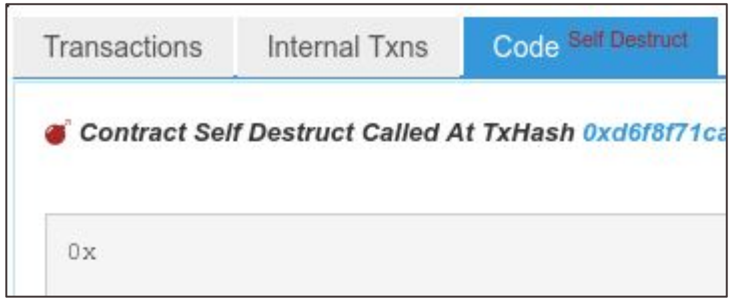


Contract Self Destruct Called At TxHash 0xd6f8f71ca

```
0x
```

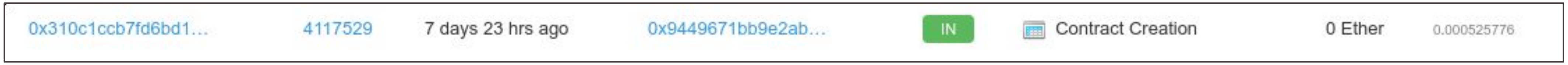
Post-mortem smart contract analysis

- If you want to analyze post-mortem interactions of a destroyed smart contract
  - ▶ When a smart contract is destroyed, code is no longer available
  - ▶ but the smart contract bytecode is still present in the contract creation transaction
  - ▶ So you can recover the smart contract bytecode
  - ▶ and you can analyze the smart contract & replay locally the transactions

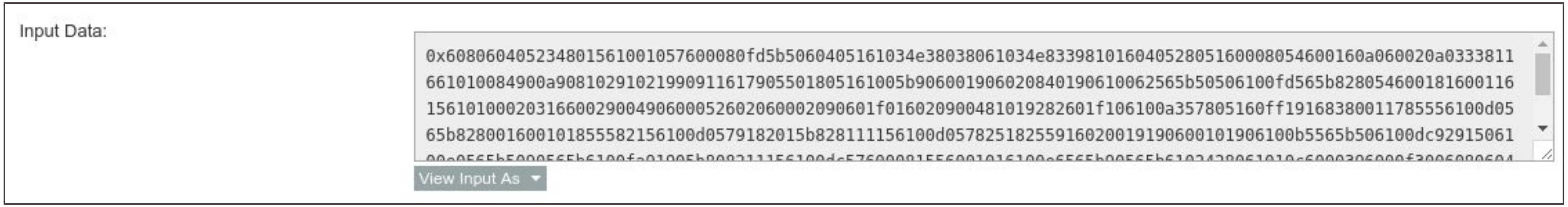


Contract Self Destruct Called At TxHash 0xd6f8f71ca

```
0x
```



0x310c1ccb7fd6bd1...	4117529	7 days 23 hrs ago	0x9449671bb9e2ab...	IN	Contract Creation	0 Ether	0.000525776



Input Data:

```
0x608060405234801561001057600080fd5b5060405161034e38038061034e833981016040528051600080546001600160a060020a0333811
661010084900a908102910219909116179055018051610 05b906001906020840190610062565b50506100fd565b828054600181600116
156101000203166002900490600052602060002090601f016020900481019282601f106100a357805160ff19168380011785556100d05
65b828001600101855582156100d0579182015b828111156100d057825182559160200191906001019061 00b5565b506100dc92915061
```

Loader code + Runtime code

Conclusion