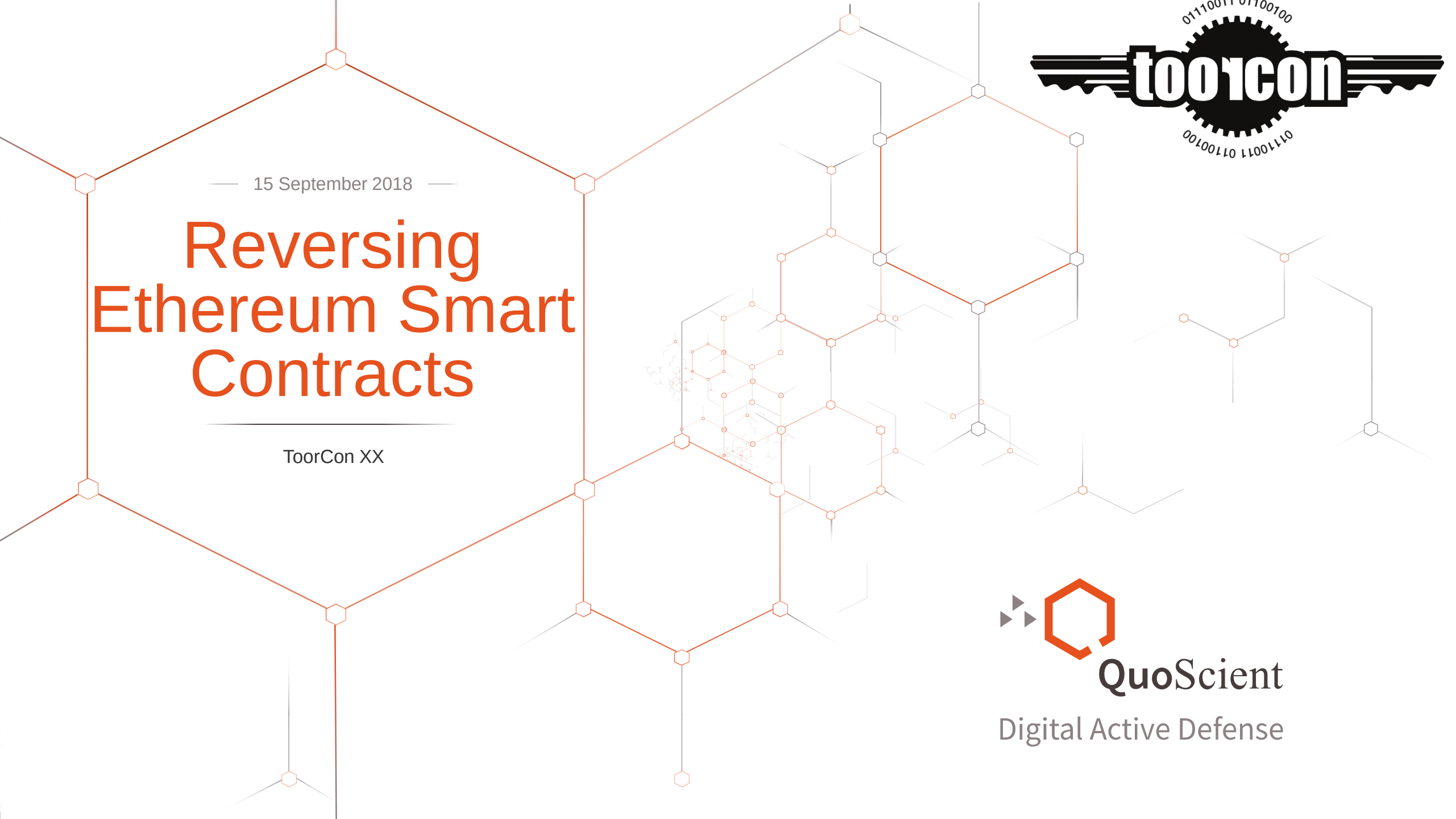

15 September 2018
Reversing Ethereum Smart Contracts
ToorCon XX
toorcon
QuoScient
Digital Active Defense

ethereum

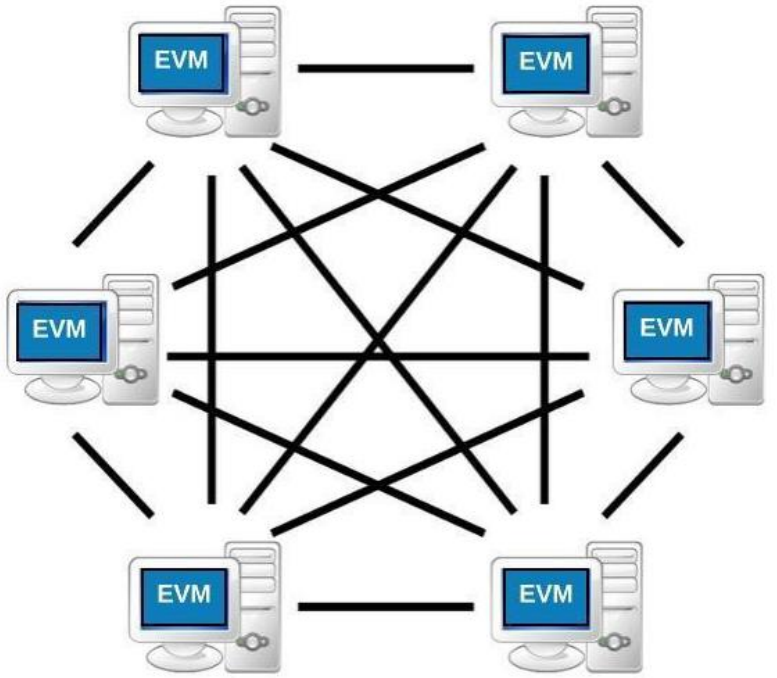
What is Ethereum?

- *“Ethereum is a decentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference.”*

- Create by Vitalik Buterin & Gavin Wood - 2013
  - White paper: Description of the project
  - Yellow paper: Ethereum's formal specification (Technical)
  - Open source

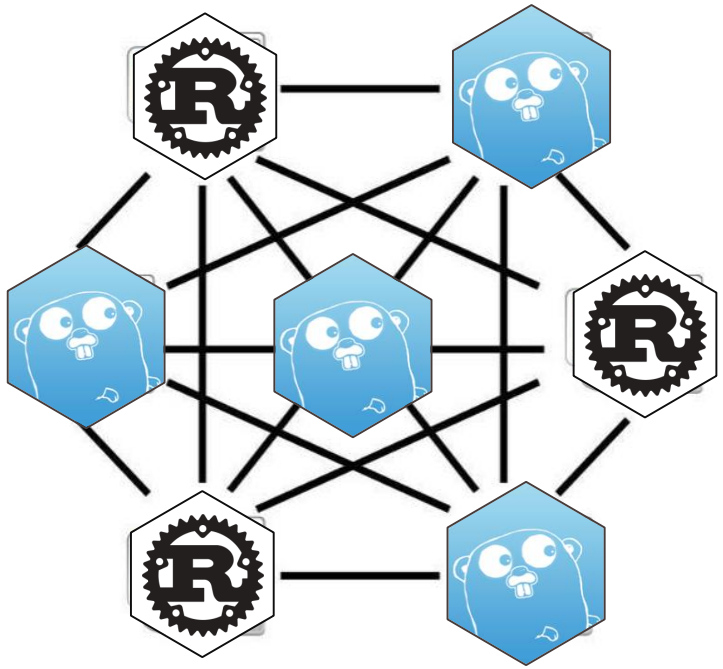
- Ethereum Virtual machine (EVM)

- Smart contracts
  - Application stored & execute on the blockchain
  - ĐApps (Decentralized Application)



Ethereum full node

- A node is:
  - Piece of Software
  - Connected to other nodes
  - Maintains locally a copy of the blockchain



~ 70%

- Geth
  - Golang
  - Major full node
  - Ethereum foundation

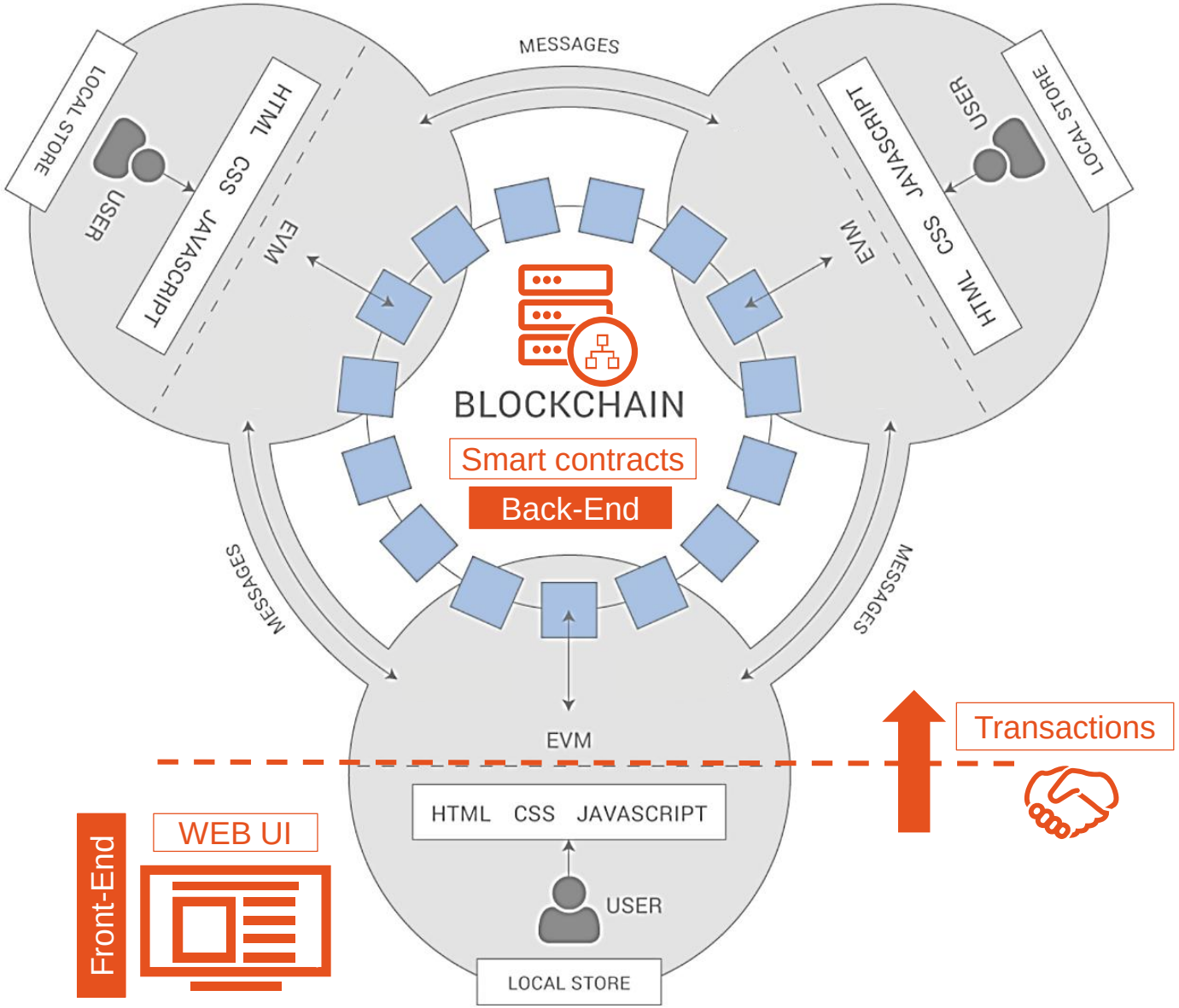
~ 25%

- Parity
  - Rust
  - Fast, secure, light
  - Parity team

EVM Implementations

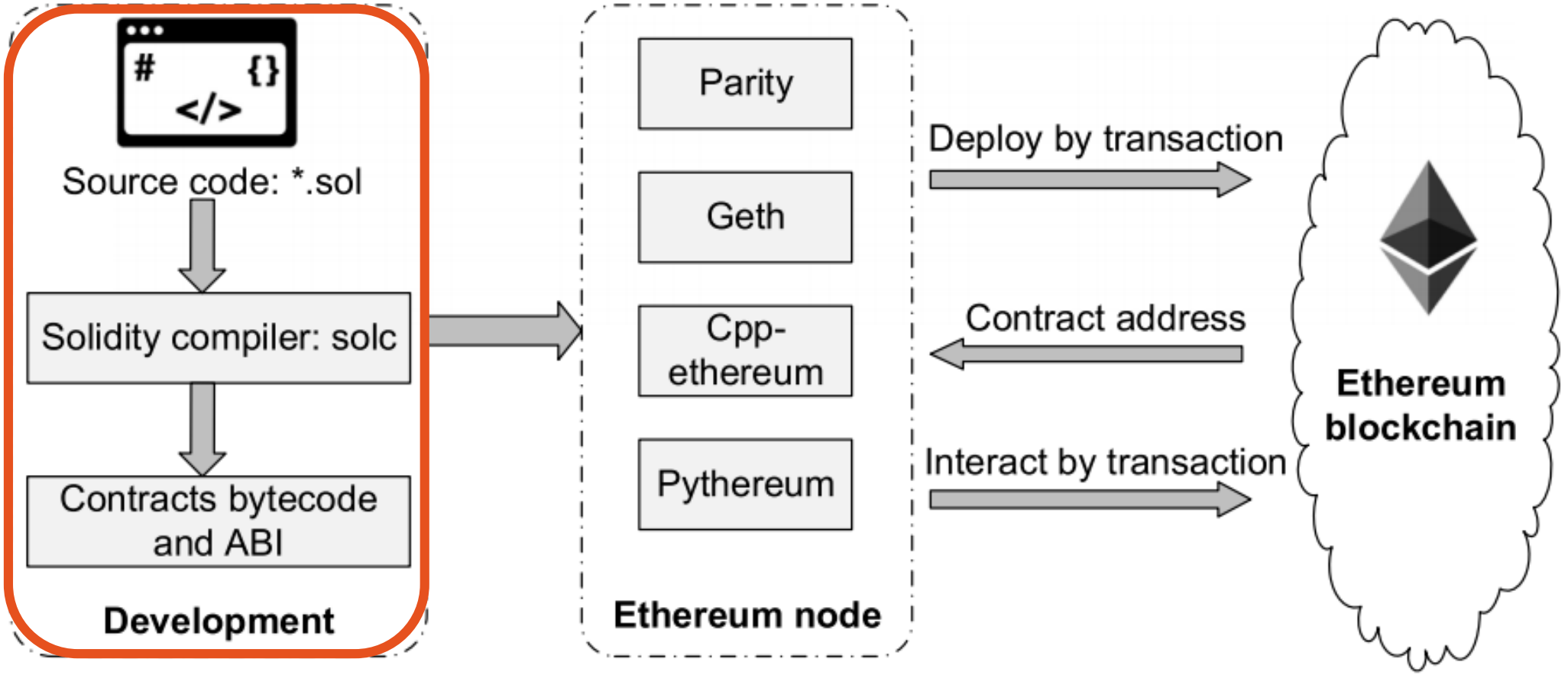
- Geth - Official Go implementation of the Ethereum protocol
  - https://github.com/ethereum/go-ethereum
- Parity – Rust implementation
  - https://github.com/paritytech/parity
- cpp-ethereum – Official C++ implementation
  - https://github.com/ethereum/cpp-ethereum
  - http://www.ethdocs.org/en/latest/ethereum-clients/cpp-ethereum/
- Pyethereum – Official Python implementation
  - https://github.com/ethereum/pyethereum

Smart contract application - ĐApps



Smart contract creation process

- Development, deployment & interaction



https://doi.org/10.1016/j.future.2017.08.020

Development languages

Solidity

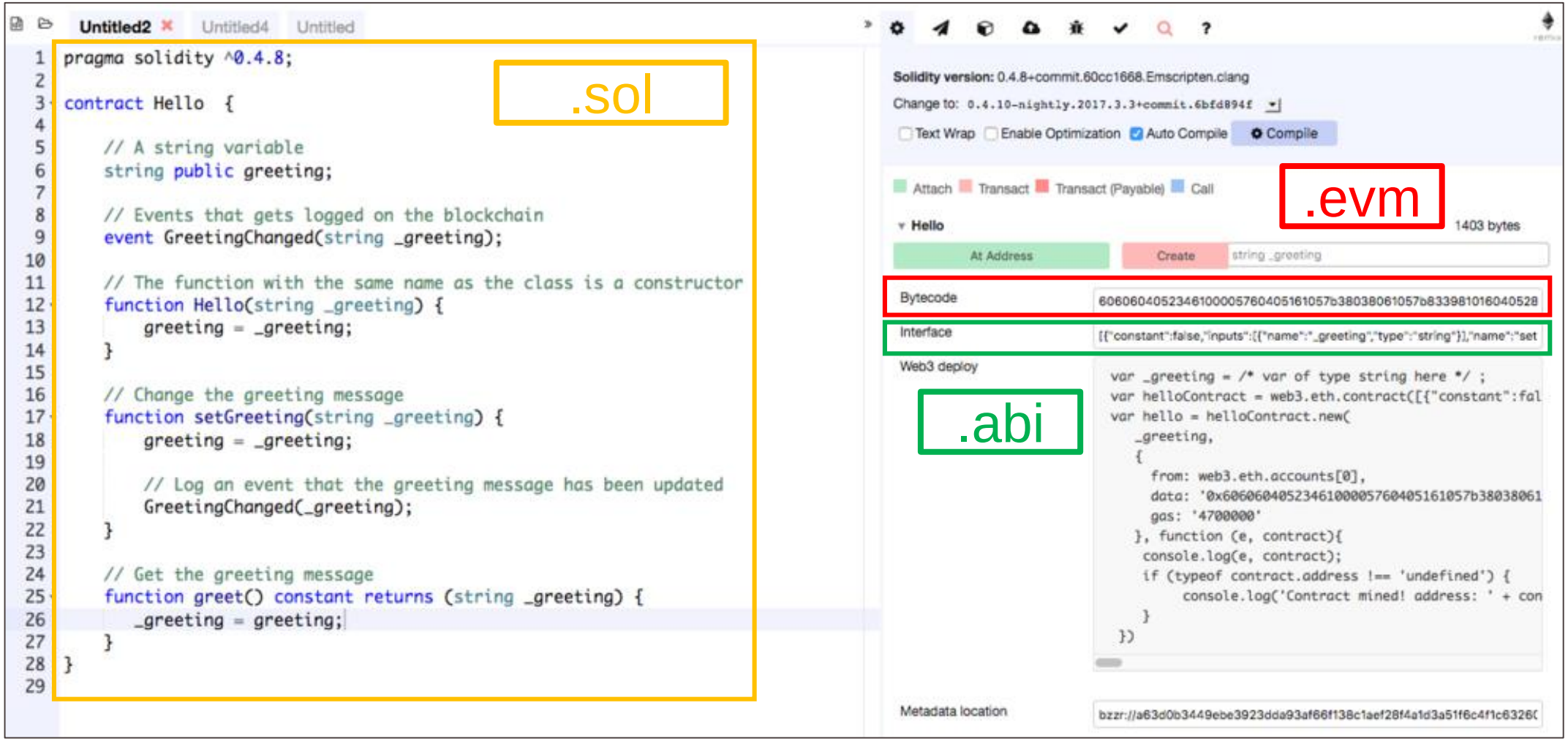
```
contract Mortal {
    /* Define variable owner of the type address */
    address owner;

    /* 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;
    }
}
```

Vyper

```
# Vyper Greeter Contract

greeting: bytes <= 20


@public
def __init__():
    self.greeting = "Hello"


@public
def setGreeting(x: bytes <= 20):
    self.greeting = x


@public
def greet() -> bytes <= 40:
    return self.greeting
```

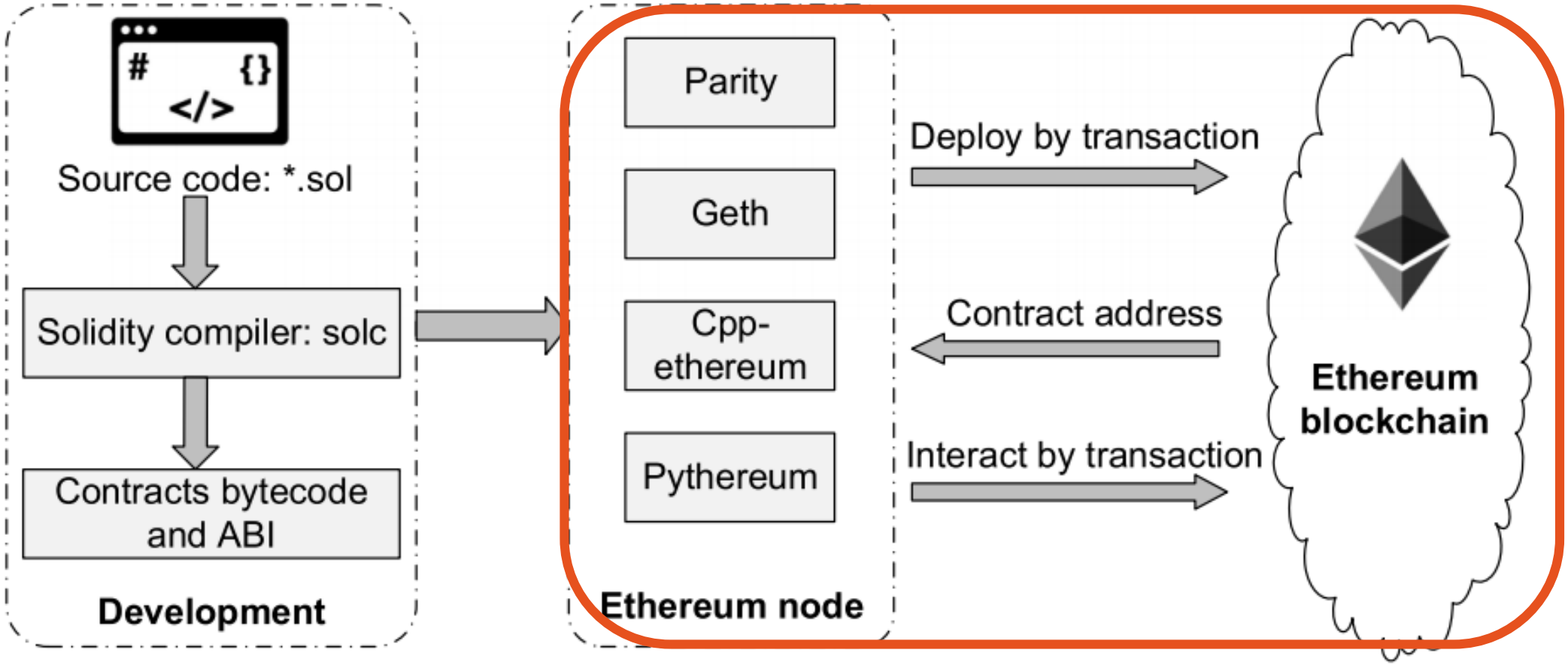
Remix - https://remix.ethereum.org/

- Online Web IDE for Solidity smart contracts development



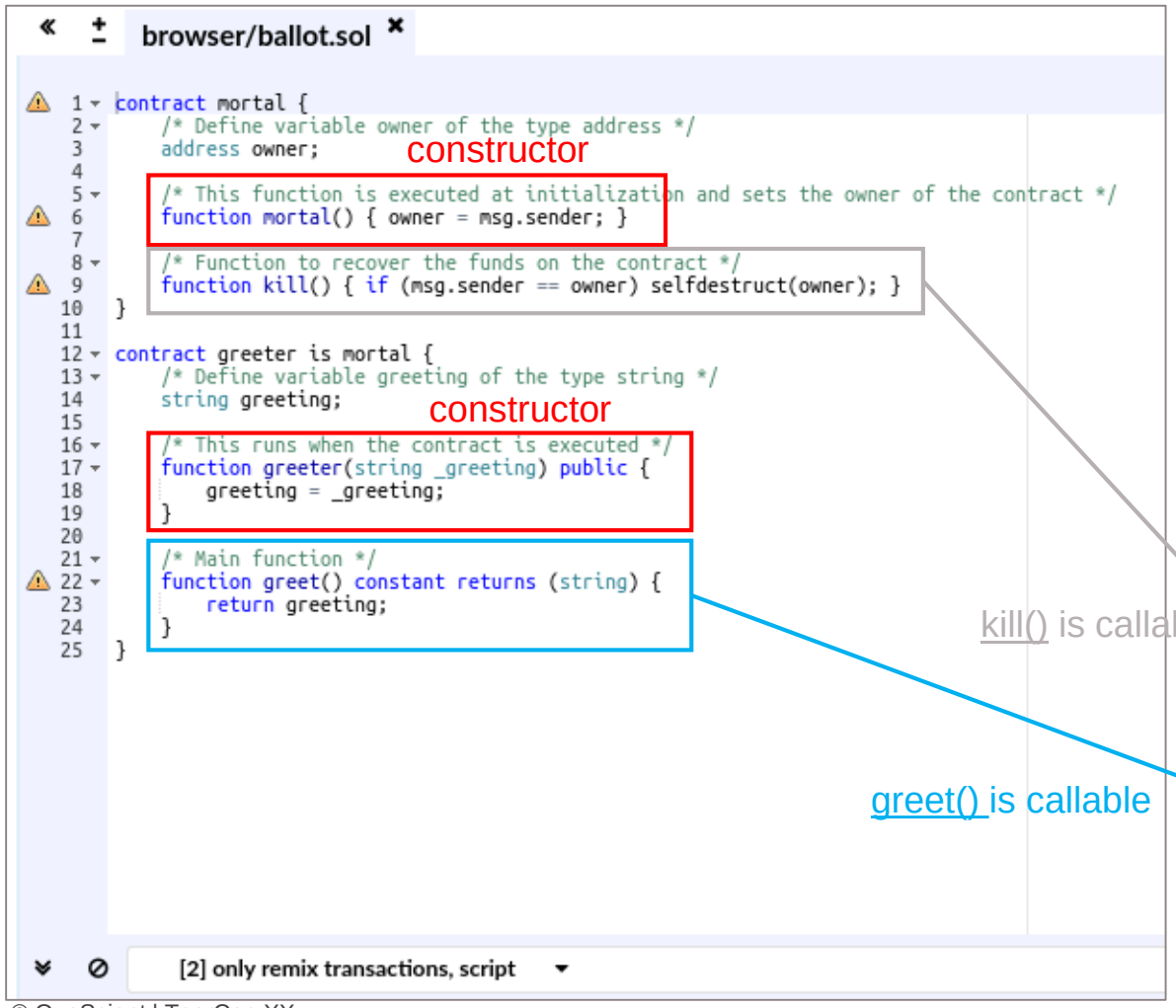
Smart contract creation process

- Development, deployment & interaction



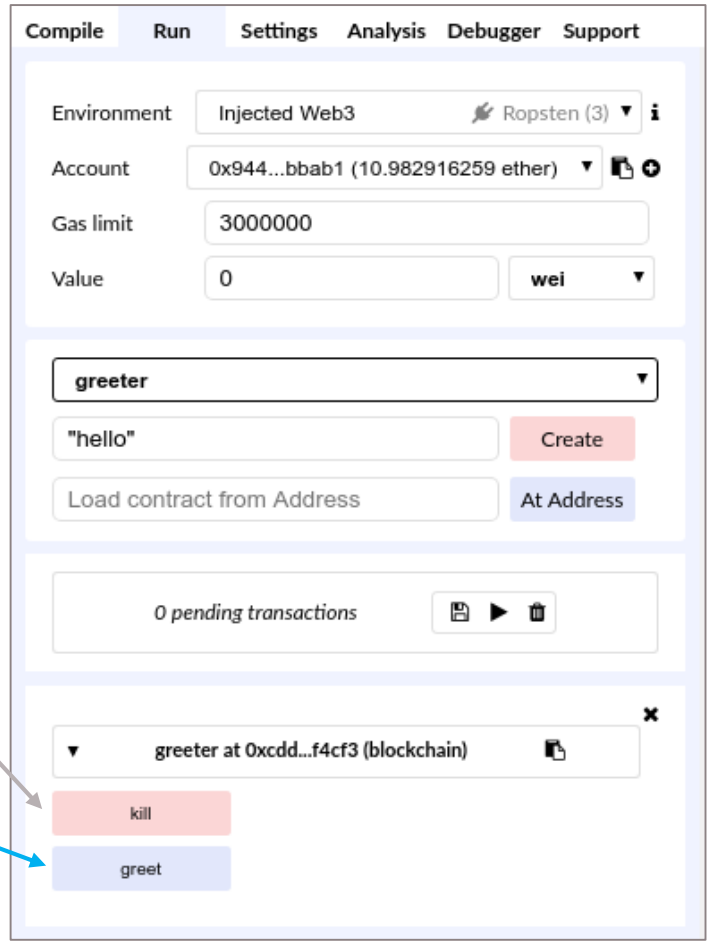
https://doi.org/10.1016/j.future.2017.08.020

Available functions of deployed contract



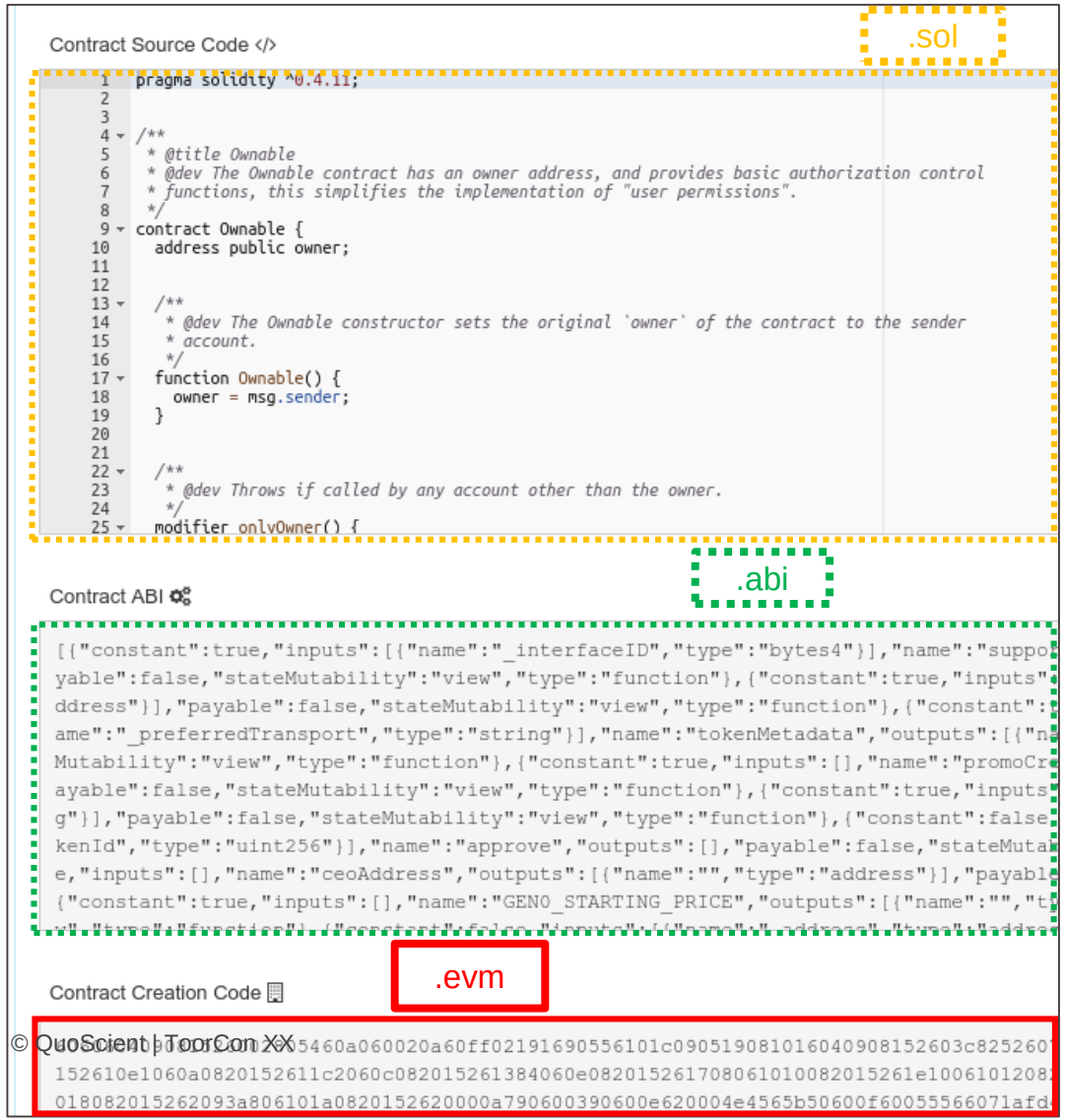
kill() is callable

greet() is callable



Existing contracts on etherscan.io

- Only bytecode is mandatory to create a contract account



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

Runtime code

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

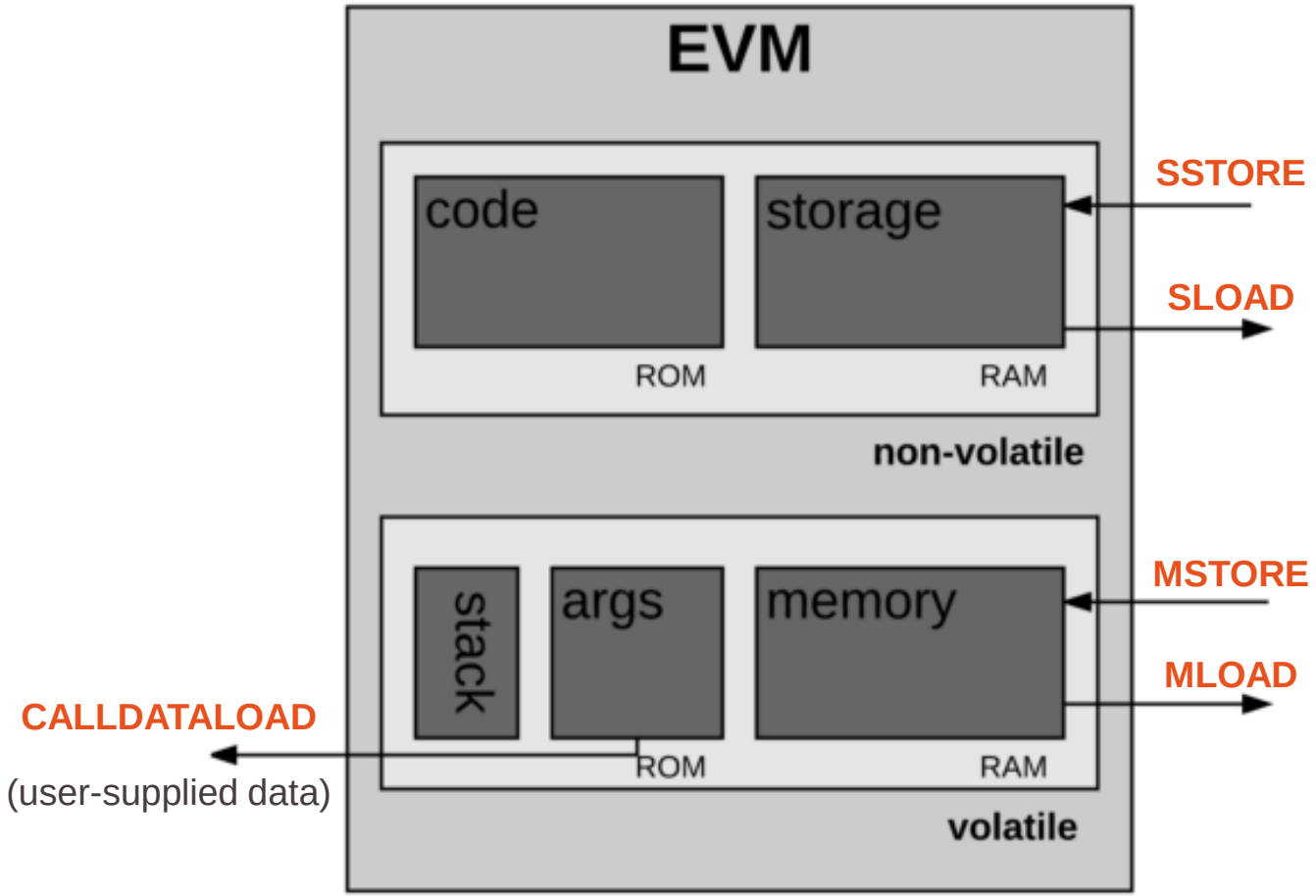
Swarm Hash (a.k.a. bzzhash)

- 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)

```
6080604052348015610010576000 80fd5b5060405161039b380
38061039b8339810180604052810190808051820192919050 50
5033600080610100 0a81548173ffffffffffffffffffffffff
ffffffffffffffff02191690837 3ffffffffffffffffffffff
ffffffffffffffff1602179055508060019080519060200190 6
1008992919061009056 5b5050610135565b8280546001816001
16156101000203166002900490600052602060002090601f016
0209004810192826 01f106100d15780516 0ff19168380011785
556100ff565b828001600101855582156100 ff579182015b828
111156100fe57825182559160200191906001019061 00e3565b
5b50905061010c9190610110565b5090565b61013291905b808
2111561012e576000816000905550600101610116565b509056
5b90565b6102578061014460003960 00f300 6080604052 60043
61061004c576000357c0100000000000000000000000000000000
0000000000000000000000000900463ffffffff16806341c0e
1b514610051578063cfae321714610068575b600080fd5b3480
1561005d57600080fd5b506100666100f8565b005b348015610
07457600080fd5b5061007d610189565b604051808060200182
8103825283818151815260200191508051906020019080838 36
0005b838110156100bd578082015181840152602081019050 61
00a2565b50505050905090810190601f1680156100ea578082 0
3805160018360200361010 00a031916815260200191505b5092
505050604051809103 90f35b600080905490610100 0a900473f
ffffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffffffffff163373ffffffffffffff
ffffffffffffffffffffffff1614156101875760008090 54
906101000a900473ffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff16
ff5b565b6060600180546001816001161561010002031660029
00480601f0160208091040260200160405190810160405280 92
919081815260200182805460018160011615610100020316600
2900480156102215780601f106101f6576101008083540402 83
5291602001916102215 65b8201919060005260206000 20905b8
154815290600101906020018083116102045782900360 1f1682
01915b5050505050905090560 0a165 627a7a72 305820df97826
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 {})



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
```

Control flow instructions

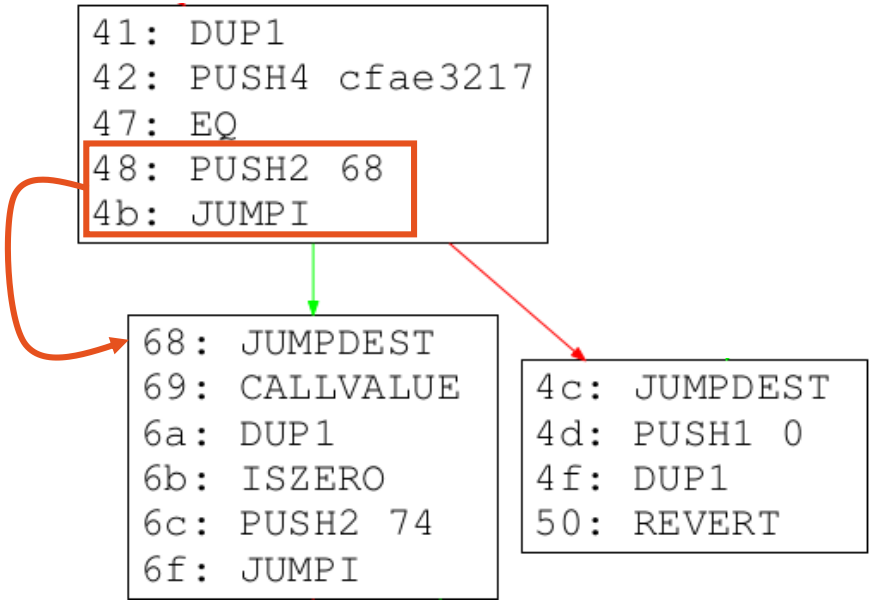
Opcode	Simplify description	Basic block position
JUMP	Unconditional jump	Last instruction
JUMPI	Conditional jump	Last instruction
RETURN , STOP INVALID SELFDESTRUCT , REVERT	Halt execution	Last instruction
JUMPDEST	Marks a position within the code that is a valid target destination for jumps	First instruction

EIP 615: Subroutines and Static Jumps for the EVM By *Greg Colvin*
New branch opcodes: JUMPTO, JUMPIF, JUMPSUB, JUMPSUBV, ….

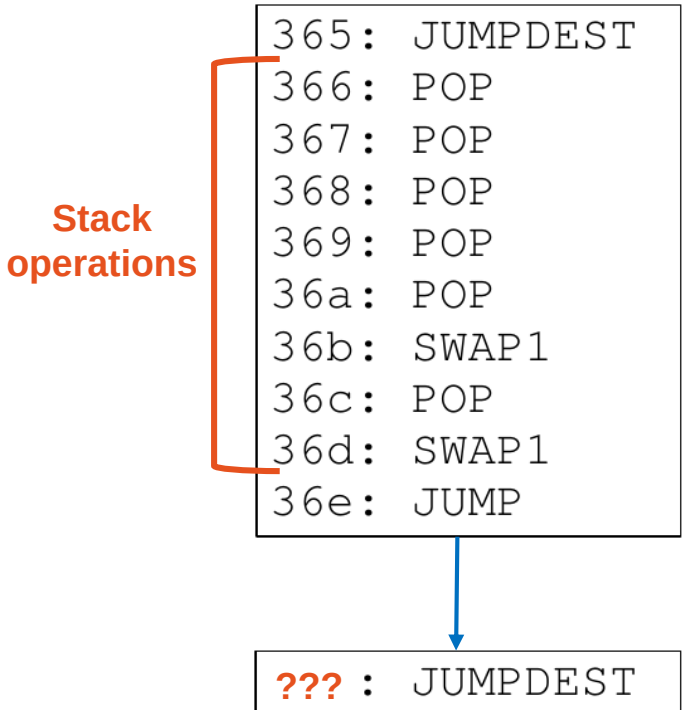
Edges identifications

Static analysis vs dynamic analysis

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

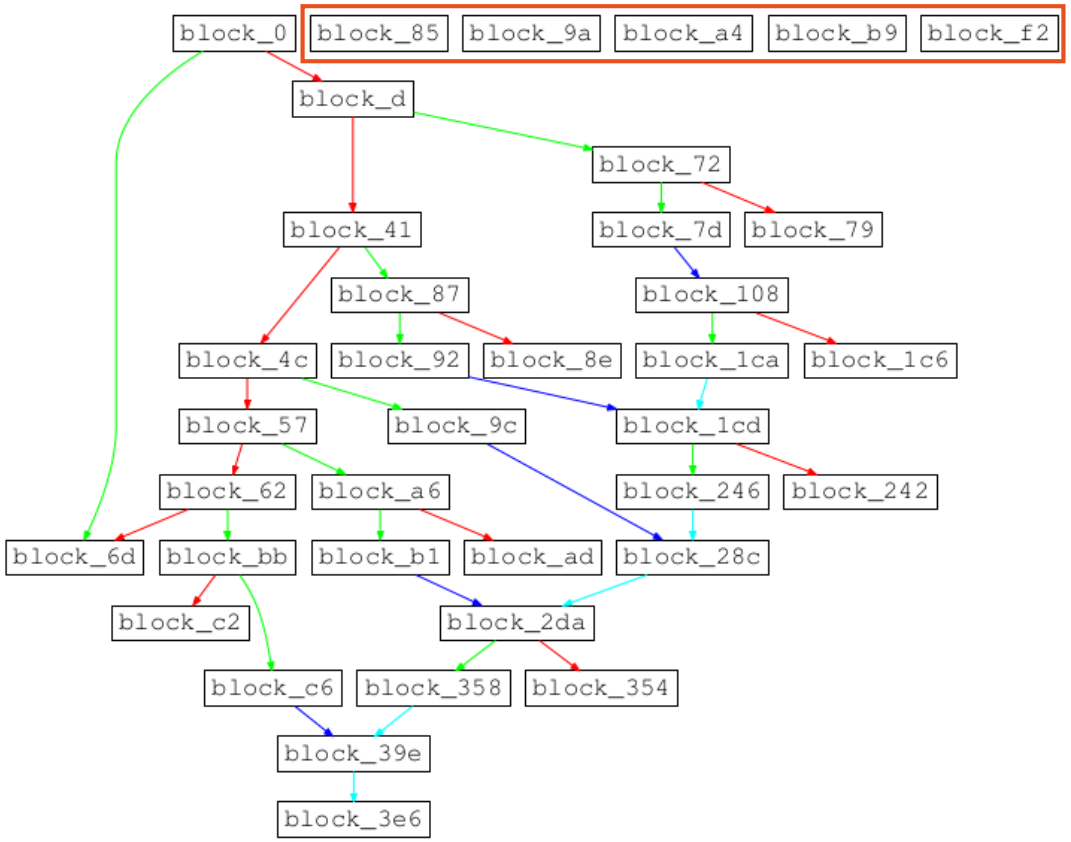


- And fails if:
  - Stack operations are used to put the jump target on top of the stack

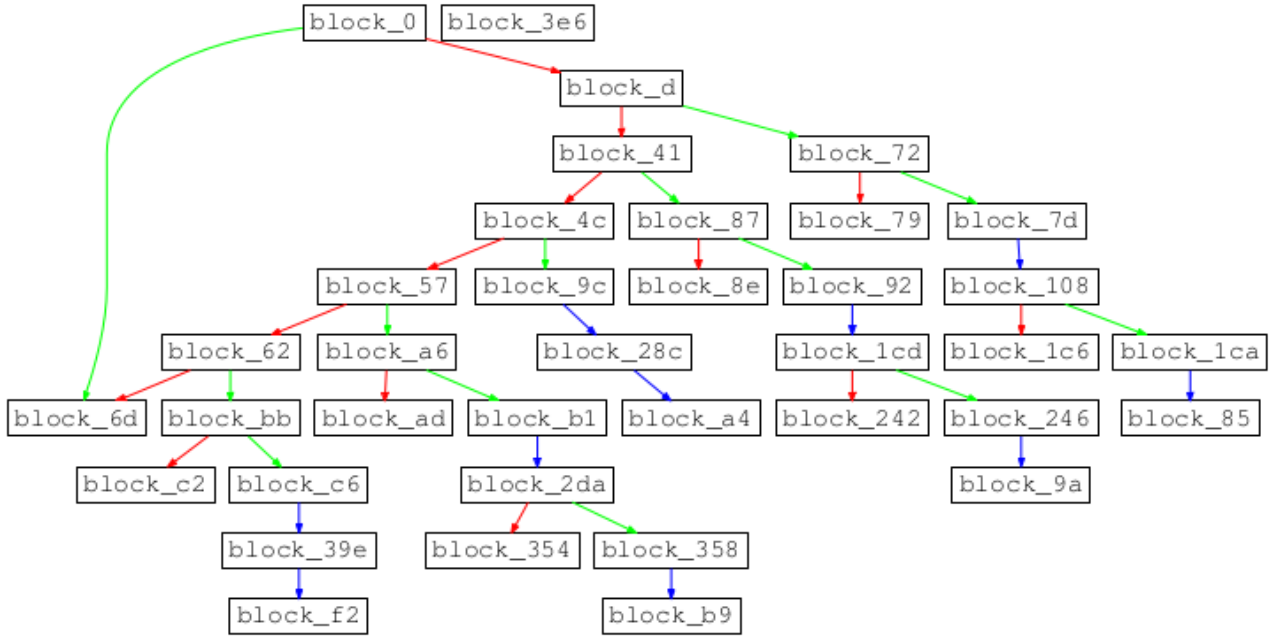


Control Flow Graph (CFG) reconstruction

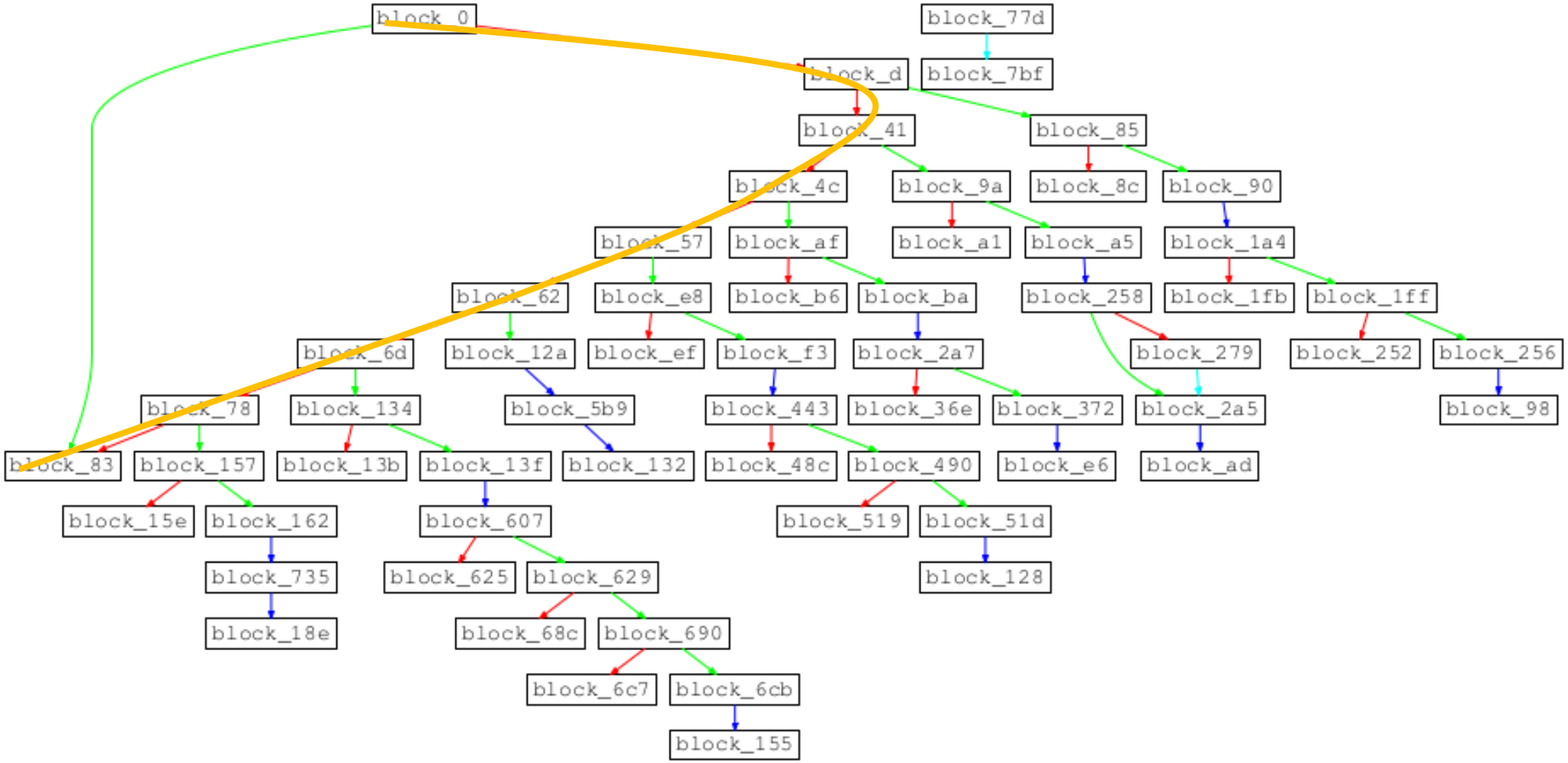
Static analysis



Dynamic analysis (stack evaluation)



Simplify visualization of the smart contract CFG

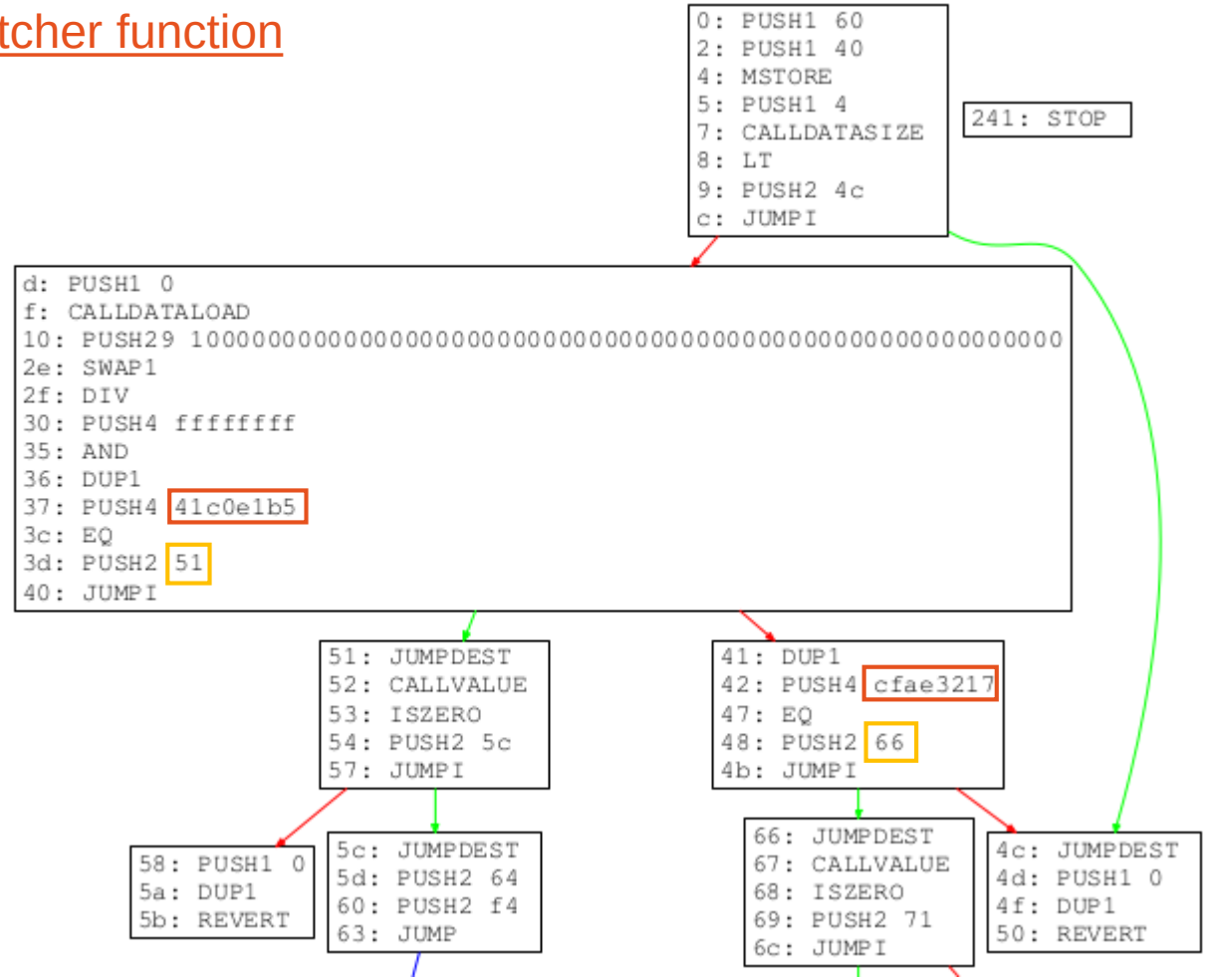


Dispatcher function

- 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
```

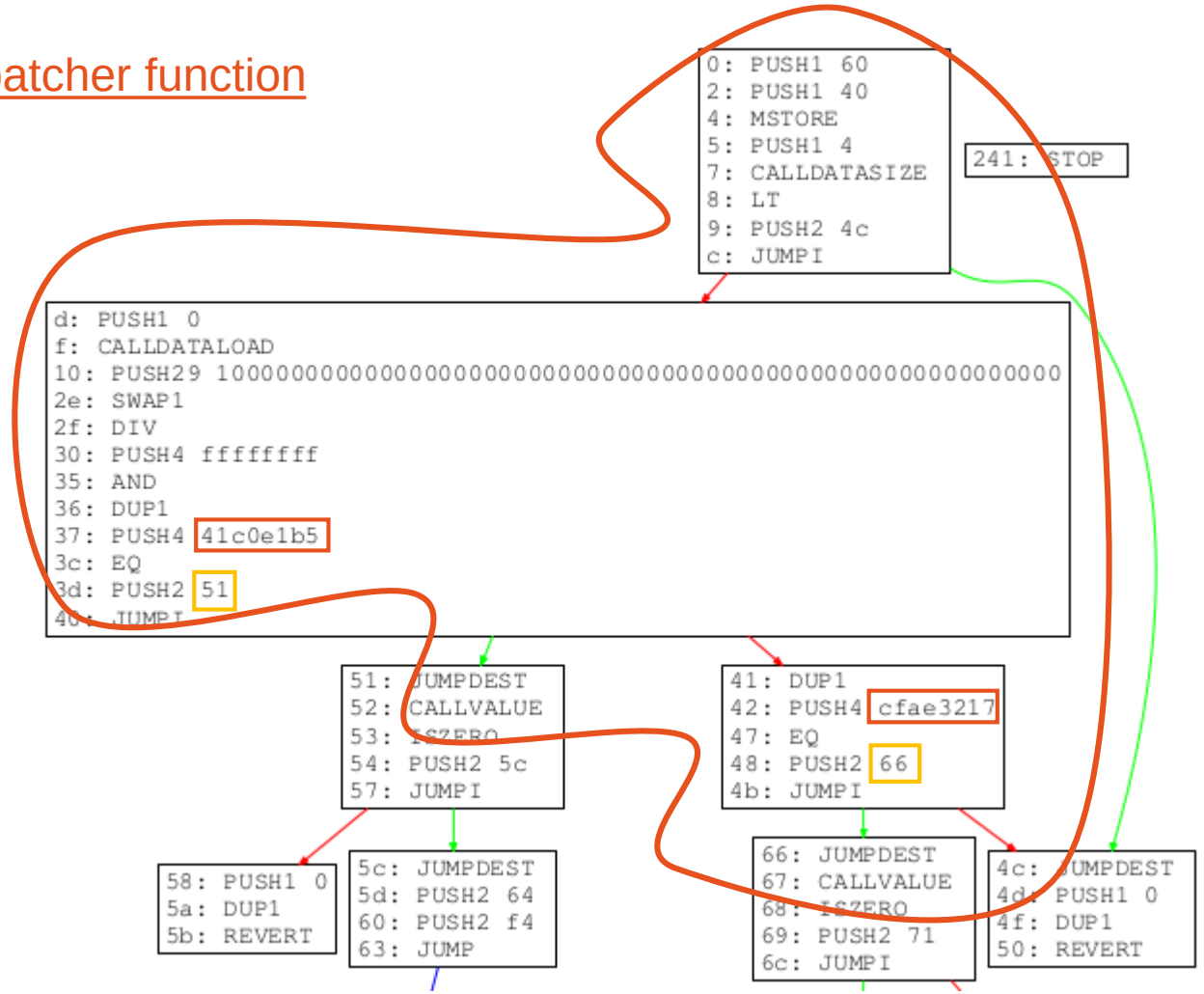


Dispatcher function

- 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.

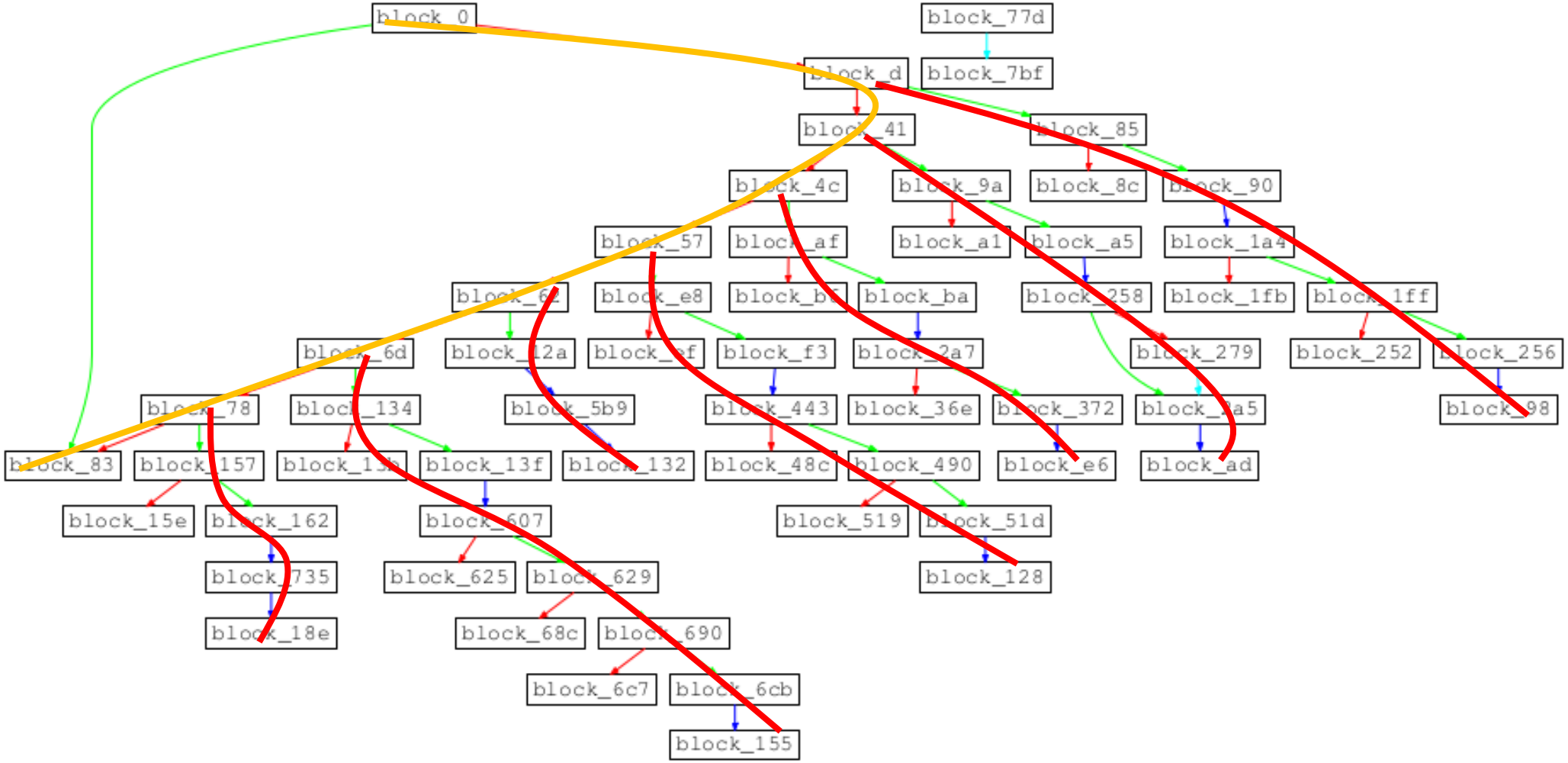
- Two functions signatures here:
  - **41c0e1b5**
  - **cfae3217**

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



Functions identification - Depth First Search

- Dispatcher function & 7 callable functions



Transaction payload

- 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'
```

- Composition :
  - Signature (function ID)
  - Argument #1 (address)
  - Argument #2 (uint)

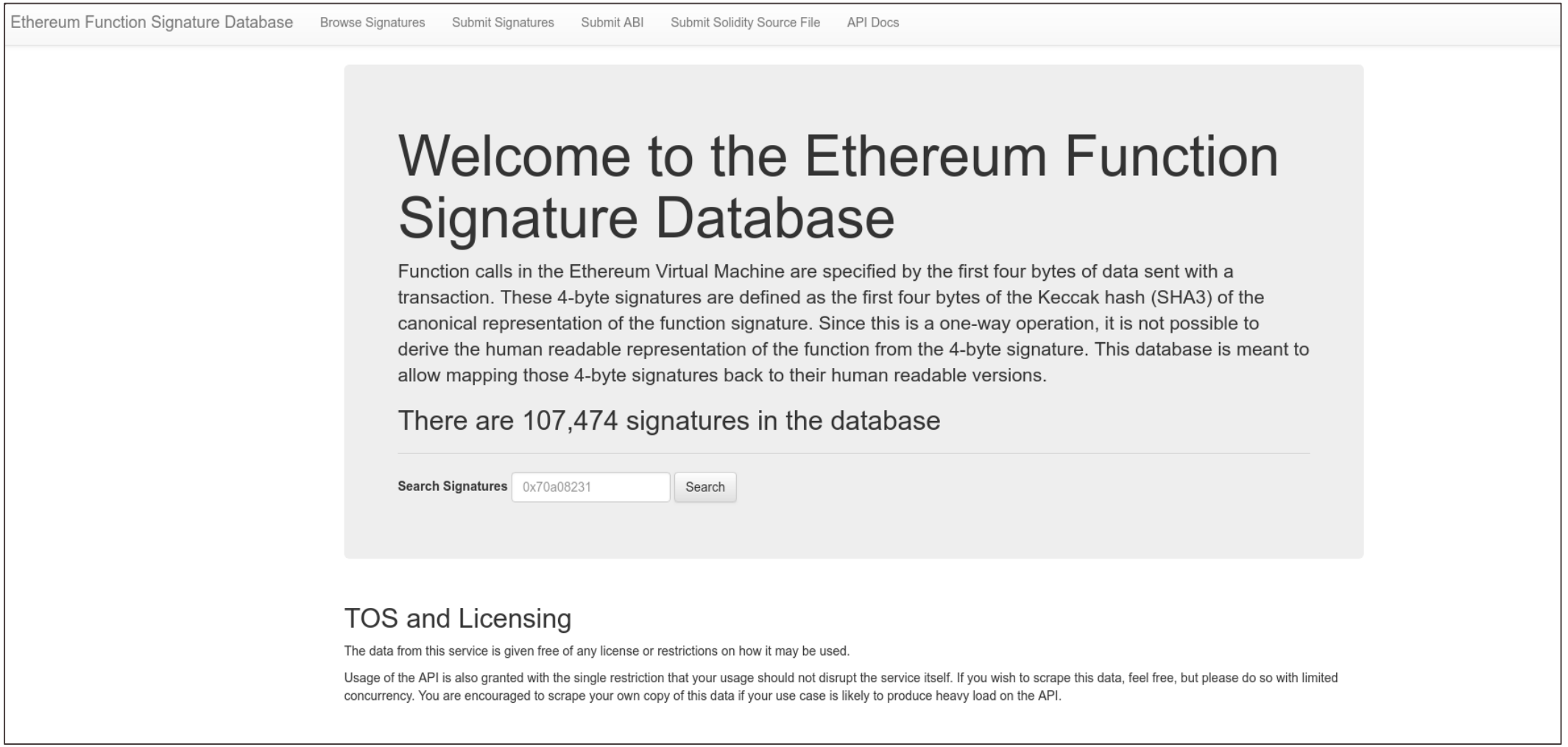
Function prototype string	4-bytes signature
“kill()”	0x41c0e1b5
“greet()”	0xcfae3217
“attack(address,uint)”	0x6ebb6d80

- Arguments size: 256-bits words

```
'input': '0x6ebb6d80000000000000000000000002983eedfbd560b6d65ffa47de6a0f6d6fee6e136000000000000000000000000000000000000000000000000000000000000000000000004',
```

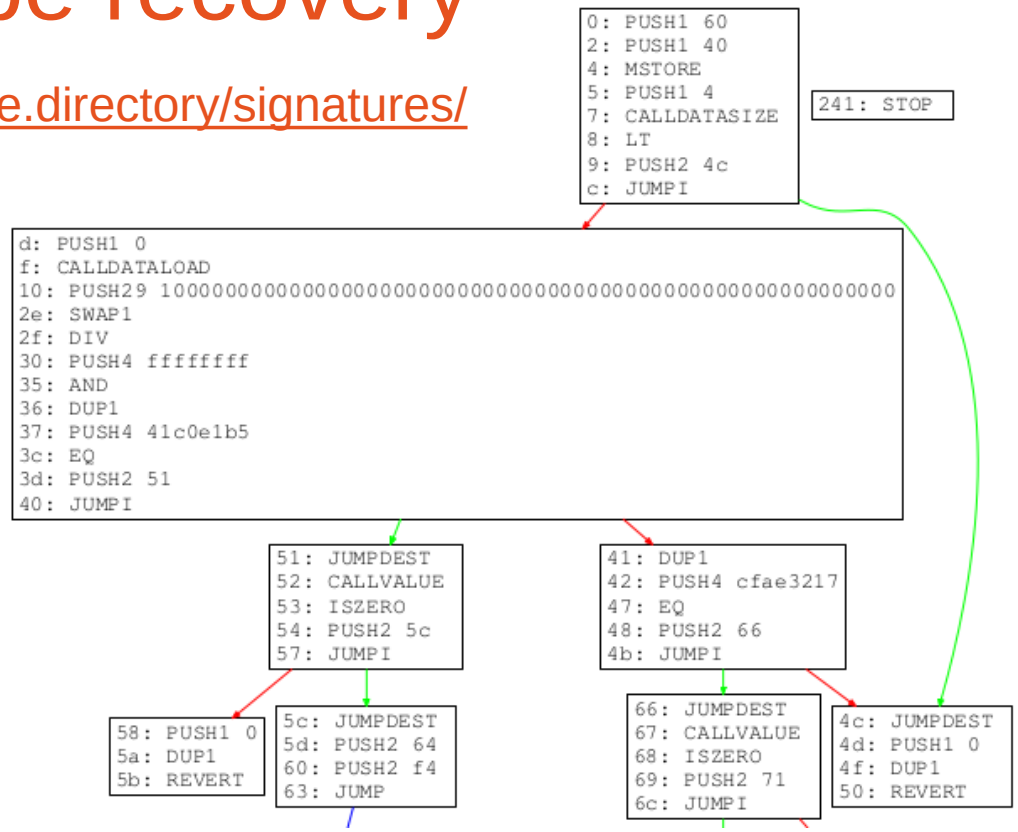
Functions name & arguments type recovery

- Function signature reverse lookup database - https://www.4byte.directory/signatures/



Functions name & arguments type recovery

- Function signature reverse lookup database - https://www.4byte.directory/signatures/



Search Signatures 0x70a08231 Search

ID	Text Signature	Bytes Signature
2009	greet()	0xcfae3217
1907	kill()	0x41c0e1b5

- Two functions signatures here:
  - `0x41c0e1b5 - "kill()"`
  - `0xcfae3217 - "greet()"`

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

EVM Disassembler available

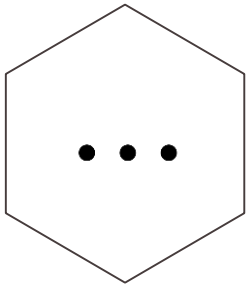
- Etherscan.io
  - ByteCode To Opcode Disassembler

- Quolab
  - Octopus/IDA/BinaryNinja integrate

- Capstone
  - Support EVM

- IDA-EVM
  - IDA Processor Module for the Ethereum Virtual Machine (EVM)

- Ethersplay
  - Binary ninja plugin

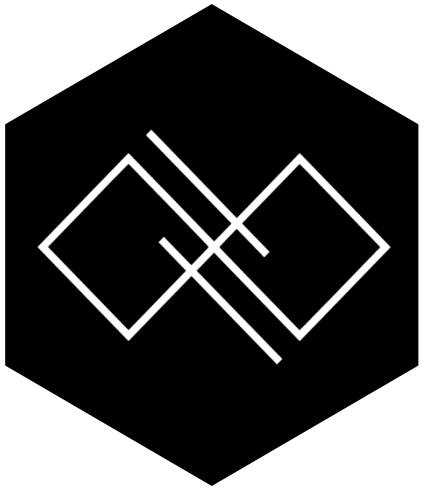


- evmdis
- ethdasm

Decompilation & IR SSA

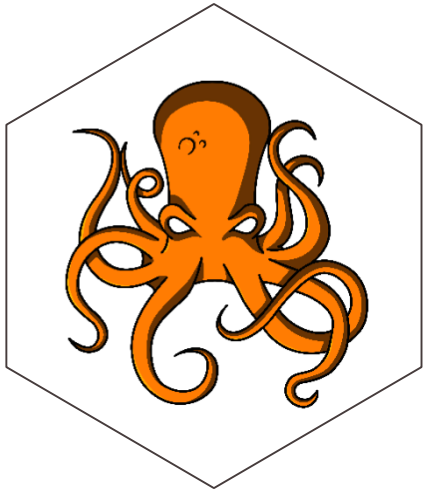
Porosity *by Comae*

- Decompiler
- Github



EthRays *by Ret2*

- Decompiler



Octopus *by QuoScient*

- IR SSA (WIP)
- Quolab or Github

Rattle *by Trail of bits*

- IR SSA
- Github

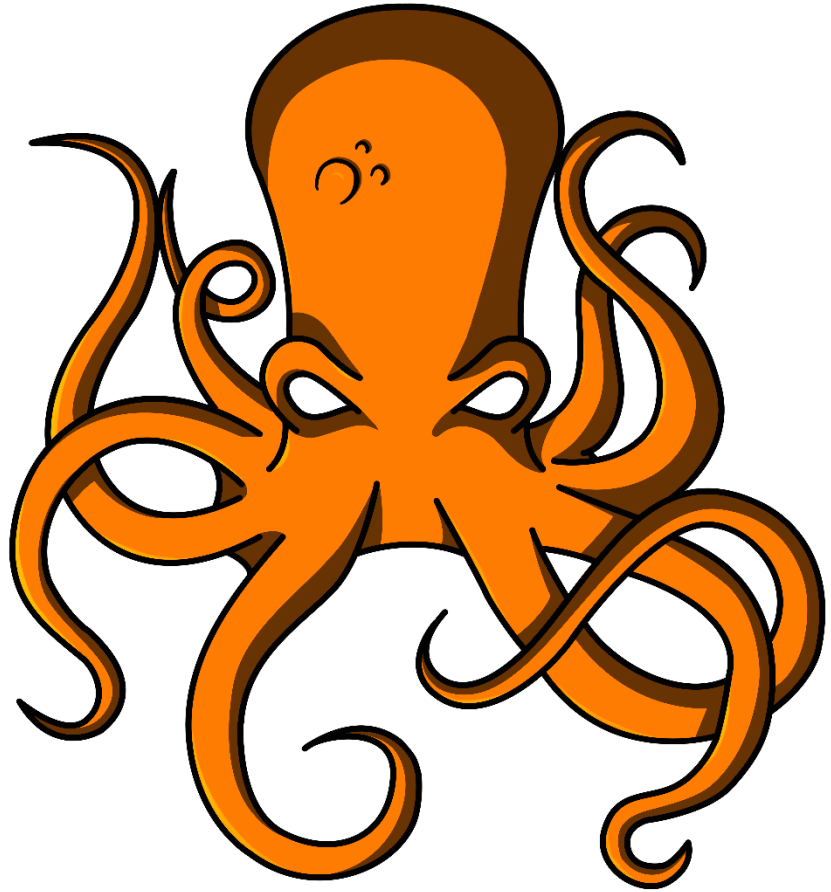
Simplify your analysis with IR

- Static single assignment (SSA)
  - IR (Intermediate representation)
  - each variable is assigned once
  - each variable is defined before being used

Instruction	SSA	Optimize SSA
PUSH1 0x03	%0 = #0x03	
PUSH1 0x05	%1 = #0x05	
ADD	%2 = ADD(%1, %0)	%0 = ADD(#0x05, #0x03)
PUSH1 0x09	%3 = #0x08	
MUL	%4 = EQ(%3, %2)	%1 = EQ(#0x08, %0)

Octopus - Ethereum

```python
from octopus.api.graph import CFGGraph
from octopus.platforms.ETH.cfg import EthereumCFG

# bytecode contract
bytecode_hex = "60606040526000357c...900480635f"

# create the CFG
cfg = EthereumCFG(bytecode_hex)

# generic visualization api
graph = CFGGraph(cfg)
graph.view()
```

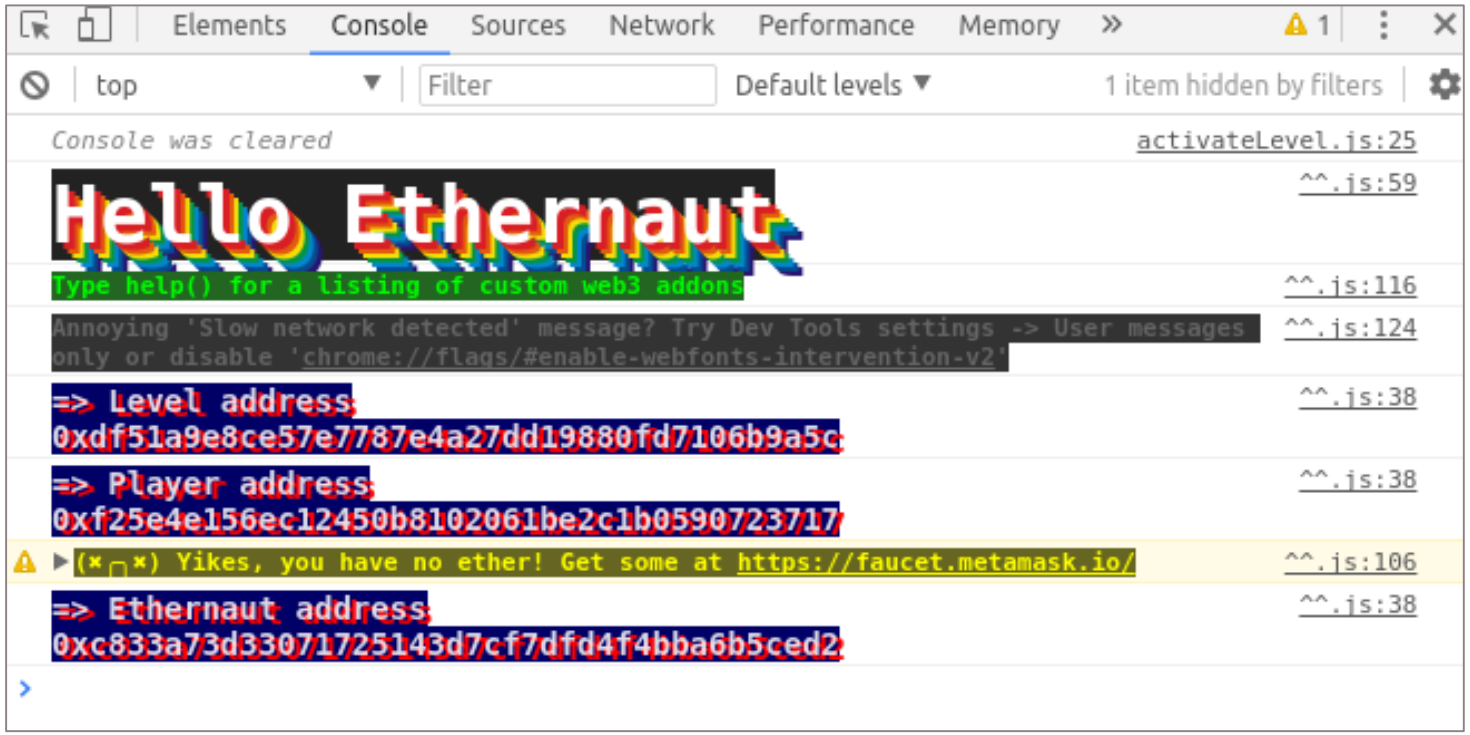


Extra resources

to learn and practice even more

Ethernaut CTF

- Web3/Solidity based wargame by Zeppelin
  - https://ethernaut.zeppelin.solutions/
  - Hands on: https://blog.trailofbits.com/2017/11/06/hands-on-the-ethernaut-ctf/



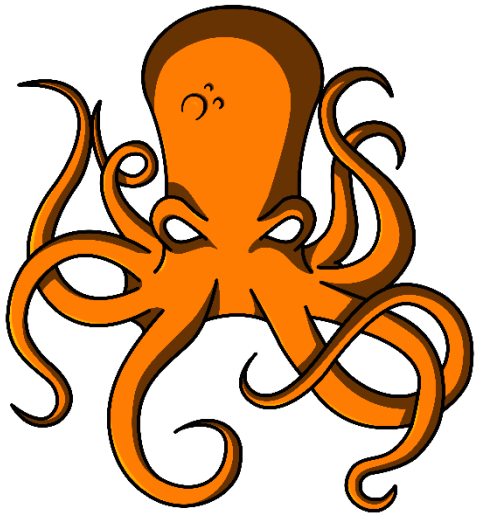
Vulnerable Smart Contracts Examples

- A CTF Field Guide for Smart Contracts
  - By Sophia D'Antoine
  - Video: https://www.youtube.com/watch?v=tI-m44Wcm7s
- Examples of Solidity security issues
  - https://github.com/trailofbits/not-so-smart-contracts
- GreHack 2017
  - https://github.com/trailofbits/ctf-challenges/tree/master/grehack-2017
- ZeroNights 2017
  - ZeroNights ICO Hacking Contest Writeup
  - https://blog.positive.com/zeronights-ico-hacking-contest-writeup-63afb996f1e3

Tutorials & talks

- Porosity - Decompiling Ethereum Smart-Contracts
  - https://www.comae.io/reports/dc25-msuiche-Porosity-Decompiling-Ethereum-Smart-Contracts.pdf
- Reversing Ethereum Smart Contracts
  - https://arvanaghi.com/blog/reversing-ethereum-smart-contracts/
- Diving Into The Ethereum VM
  - https://blog.qtum.org/diving-into-the-ethereum-vm-6e8d5d2f3c30

Thanks & Question



- Patrick Ventuzelo / @Pat_Ventuzelo / patrick.ventuzelo@quoscient.io
- Octopus - https://github.com/quoscient/octopus

Exercices

Bytecode 1

- Does this following bytecode contains a loader and a swarm hash?

- “0x60606040523415610000f57600080fd5b6040516103a93803806103a98339810160405280805182019190505033600080610100a8
1548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550806001908051906020019061008192919061006
88565b505061012d565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10
6100c957805160ff1916838001178555610000f7565b828001600101855582156100f7579182015b828111156100f657825182559160200191
9190600101906100db565b5b509050610104919061010856 5b5090565b61012a91905b8082111561012657600081600090555060010
161010e565b5090565b90565b61026d8061013c6000396000f300606060405260043610610004c576000357c0100000000000000000000
0000000000000000000000000000000000000900463ffffffff16806341c0e1b514610051578063cfae321714610066575b600080fd5b34
1561005c57600080fd5b6100646100f4565b005b34156100715760080fd5b610079610185565b60405180806020018281038252838181518152602001915080519060200190808383600005b838110156100b957808201518184015260208101905061009e565b505050509
050908101906601f1680156100e6578082038051600183602003610100 0a031916815260200191505b509250505060405180910390f35
b60008090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff1614156101835
760008090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b61018d61022d565b6001805460018
160011615610100020316600290048060 1f0160208091040260200160405190810160405280929190818152602001828054600181600
116156101000203166002900480156102235780601f106101f857610100808354040283529160200191610223565b820191906000526
0206000020905b8154815290600101906020018083116102065782900360 1f168201915b5050505050905090565b602060405190810
160405280600081525090560 0a165627a7a72305820c4498eaabe7598422b89a825ece27b0e5df8371a9d48cd33e9a25b0b6b4dcab5
0029”

Bytecode 1 – loader + runtime code

- Does this following bytecode contains a loader and a swarm hash?

- “0x606060405234156100f57600080fd5b6040516103a93803806103a98339810160405280805182019190505033600080610100a8
1548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550806001908051906020019061008192919061006
88565b505061012d565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f10
6100c957805160ff1916838001178555610100f7565b828001600101855582156100f7579182015b828111156100f657825182559160200191
9190600101906100db565b5b509050610104919061010856 5b5090565b61012a91905b8082111561012657600081600090555060010
161010e565b5090565b90565b61026d8061013c6000396000f300606060405260043610610 04c576000357c0100000000000000000000
0000000000000000000000000000000000000000900463ffffffff16806341c0e1b514610051578063cfae3217146100665 75b600080fd5b34
1561005c57600080fd5b6100646100f4565b005b34156100715760008 0fd5b6100796101855 65b60405180806020018281038252838 1
8151815260200191508051906020019080838360005b838110156100 b95780820151818401526020810190506100 9e565b505050509
050908101906 01f1680156100e6578082038051600183602003610 1000a031916815260200191505b509250505060405180910390f35
b6000809054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff1614156101835
76000809054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b61018d61022d565b6001805460018
160011615610100020316600290048060 1f0160208091040260200160405190810160405280929190818152602001828054600181600
011615610100020316600290048015610223578060 1f106101f8576101008083540402835291602001916102235 65b820191906000526
0206000020905b8154815290600101906020018083116102065782900360 1f168201915b5050505050905090565b602060405190810
160405280600081525090560 0a165627a7a72305820c4498eaabe7598422b89a825ece27b0e5df8371a9d48cd33e9a25b0b6b4dcab5
0029”

Bytecode 1

- How many instructions in this contract?

Bytecode 1

- How many instructions in this contract?
  - 342

```
Decoded Bytecode :

[1] PUSH1 0x60
[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 0x0066
[75] JUMPI
```

https://etherscan.io/opcode-tool

Bytecode 1

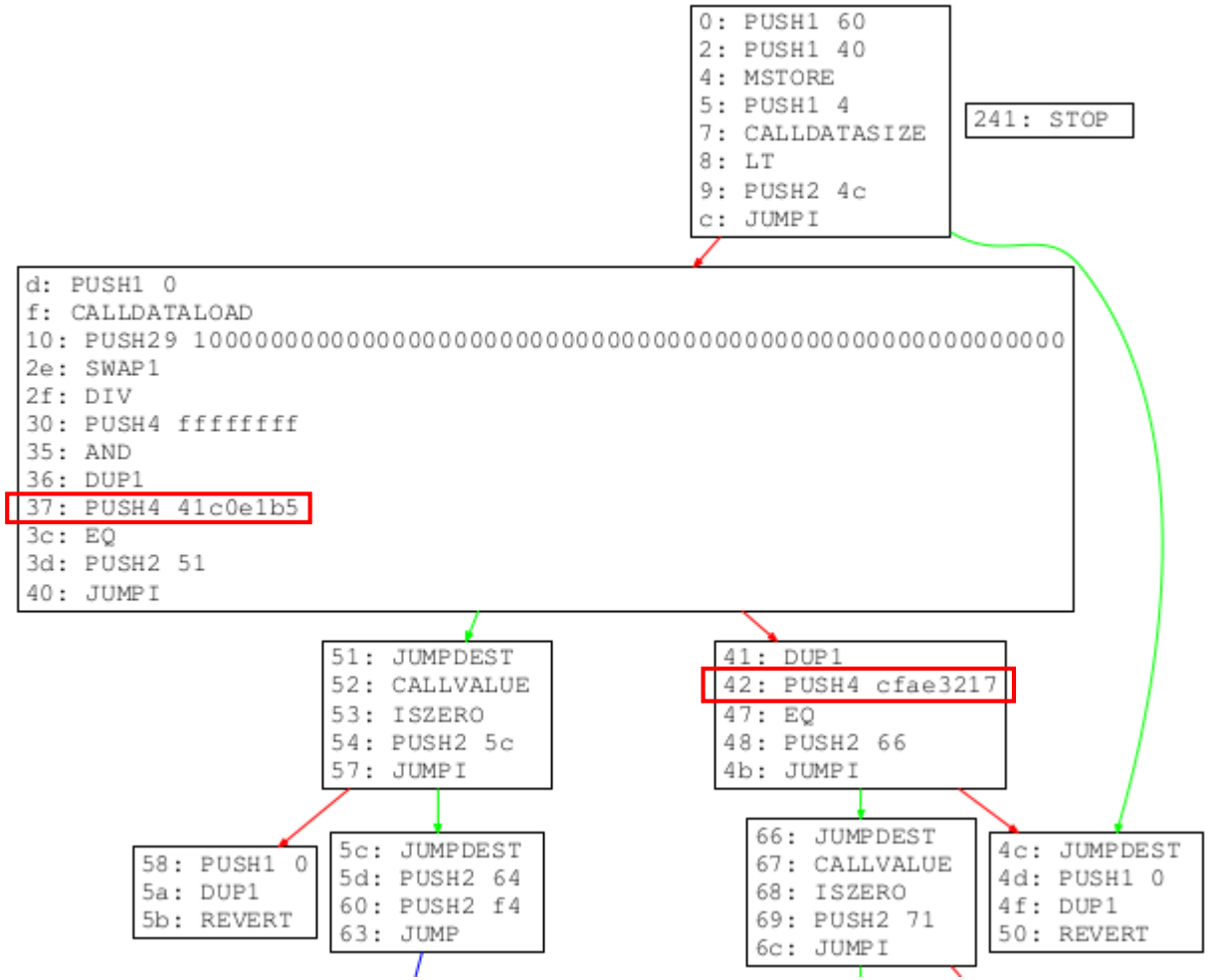
- How many functions in this contract?

Bytecode 1 – Dispatcher function

- How many functions in this contract?
  - `41c0e1b5`
  - `cfae3217`

- Pattern:
  - DUP1 = 0x80
  - PUSH4 = 0x63
  - EQ = 0x14
  - PUSH2 = 0x61
  - JUMPI = 0x57
  - 8063XXXXXXXX1461XXXX57

```
import re
regex = r'8063.{8}1461.{4}57'
re.findall(regex , bytecode)
```



Bytecode 1

- Can you find the names of those function??

Bytecode 1

- Can you find the names of those function??
  - Using octopus

```
In [11]: [x.prefered_name for x in cfg.functions]
Out[11]: ['Dispatcher', 'kill()', 'greet()']
```

  - Using 4byte website
    - https://www.4byte.directory/signatures/?bytes4_signature=41c0e1b5

Search Signatures 41c0e1b5 Search

ID	Text Signature	Bytes Signature
1907	kill()	0x41c0e1b5

    - https://www.4byte.directory/signatures/?bytes4_signature=cfae3217

Search Signatures cfae3217 Search

ID	Text Signature	Bytes Signature
2009	greet()	0xcfae3217

Bytecode 1

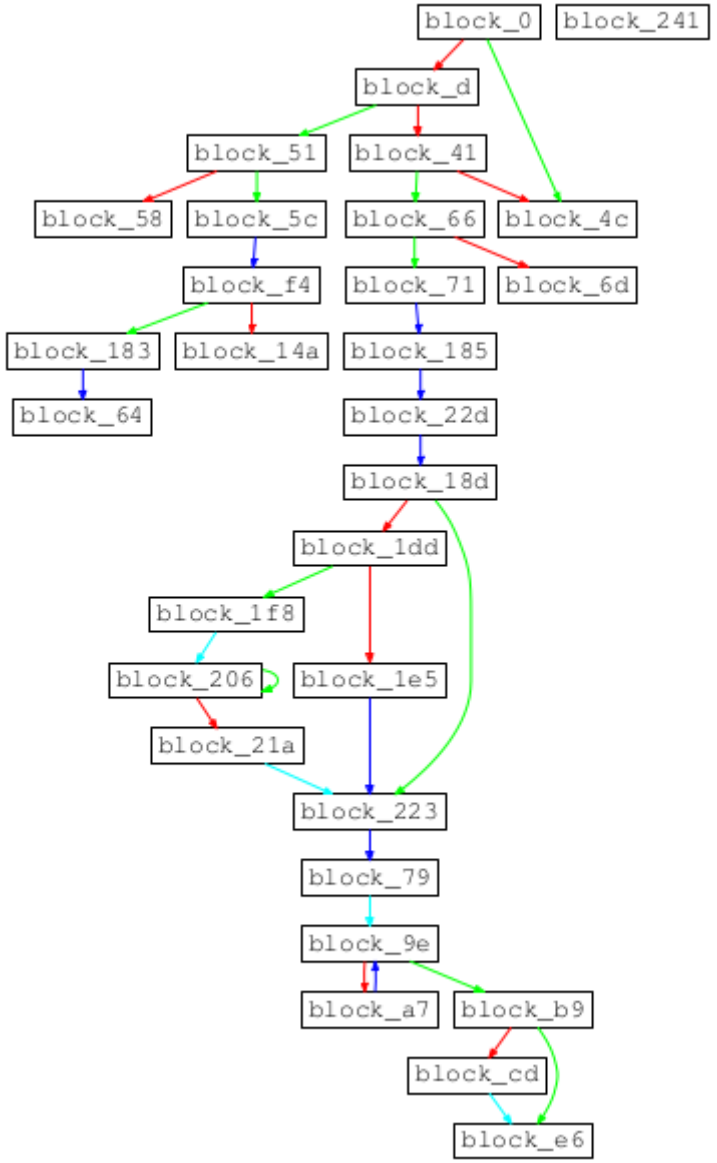
- How many basicblocks in this contract?

Bytecode 1 - CFG

- How many basicblocks in this contract?
  - 30

```
# create the CFG
cfg = EthereumCFG(bytecode_hex)

# generic visualization api
graph = CFGGraph(cfg)
graph.view()
```

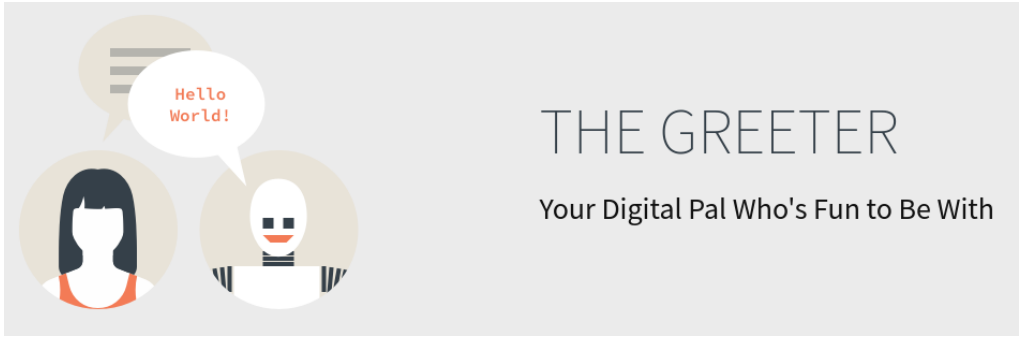
- You can do **`graph.view_simplify()`** to get this output



Bytecode 1 was in reality THE GREETER

- Building a smart contract using the command line
  - Keywords: solc, remix, geth, web3js
  - https://www.ethereum.org/greeter
  - tutorial

- Wiki: https://github.com/ethereum/go-ethereum/wiki/Contract-Tutorial#your-first-citizen-the-greeter




```
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;
    }
}
```

Bytecode 2

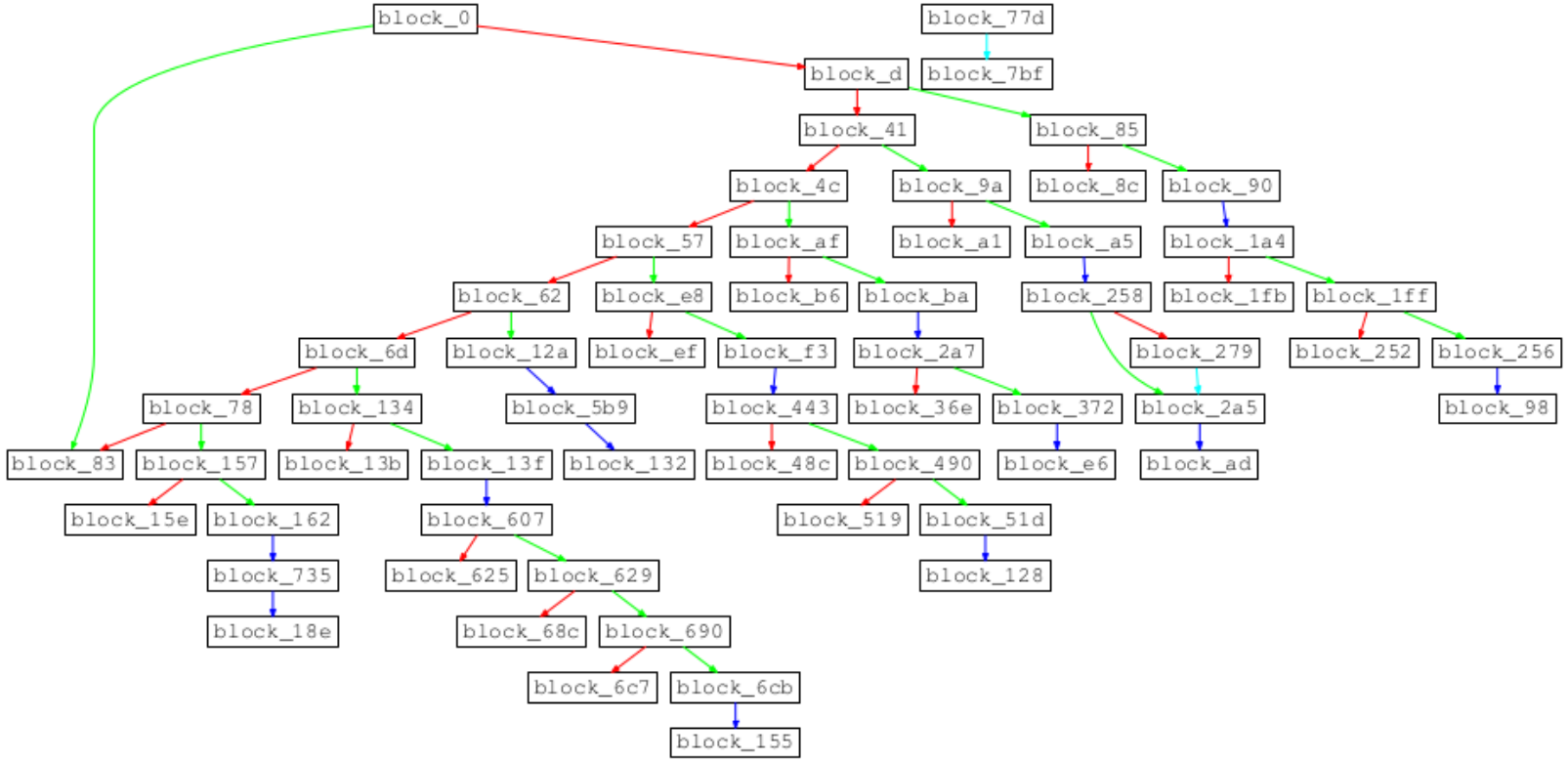
- 60606040526004361061008357600035 7c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806341c0e1b514610085578063
473ca96c1461009a578063a3a7e7f3146100af578063a9059cbb146100e8578063c0e317fb1461012a578063da76d5cd14610134578063f8b2cb4f14610157575b005
b341561009057600080fd5b6100986101a4565b005b34156100a557600080fd5b6100ad610258565b005b34156100ba57600080fd5b6100e6600480803573ffffffffff
ffffffffffffffffffffffffffffff16906020019091905050610 2a7565b005b34156100f357600080fd5b610128600480803573ffffffffffffffffffffffffffffffffffffffff1690602001909190803590
6020019091905050610443565b005b6101326105b9565b005b341561013f57600080fd5b6101556004808035906020019091905050610607565b005b34156101625
7600080fd5b61018e600480803573ffffffffffffffffffffffffffffffffffffffff16906020019091905050610735565b60405180828152602001915050604051809103 90f35b6001809
054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16141515 6101ff57600080fd5b3373ffffffffffffffffffff
ffffffffffffffffffff166108fc3073ffffffffffffffffffffffffffffffffffffffff16319081150290604051600060405180830381858888f19350505050151561025657600080fd5b565b60003073
ffffffffffffffffffffffffffffffffffffffff163114156102a5577f29822734f2619c764b25a111ff2328da08e02d6652fc891eb80227a6ec58774360405160405180910390a15b565b60
00808273ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054600080 3373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffffff168152602001908152602001600020546000808473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020540
1101515156103725 7600080fd5b6000803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020546000808373ffffffffff
ffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600082825401925050819055506000806000 3373ffffffffffffffffffffffffffffffffffffffff1
673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020819055505 0565b806000803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815
26020019081526020016000205410151515610490 57600080fd5b6000808373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020
0160002054816000808573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054011015151561051d57600080fd5b806
000808473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600082825401925050819055508060008033 73fffffffffffffff
fffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282540392505081905550505 0565b346000803373ffffffffffffffffffffffffffffffffff
ffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282540192505081905550565b6000151560016000905490610100 0a900460ff16151
5141515610629 57600080fd5b600180600061010 00a81548160ff021916908315150217905550806000803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffff
fffff16815260200190815260200160002054101515156106905760 0080fd5b3373ffffffffffffffffffffffffffffffffffffffff168160405160006040518083038185876187965a03f192
50505015156106cb57600080fd5b806000803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1 68152602001908152602001600020600082825403925
050819055506000600160006101000a81548160ff02191690831515021790555050565b60008060008373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1
68152602001908152602001600020549050919050565b336001806101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790
55505600a165627a7a72305820e6c90b0d41a108d1ea58939205012475060e1c2fb26db4fde12e5ffc5ea2919f0029

Bytecode 2

- How many functions in this contract?
- How many basicblocks in this contract?

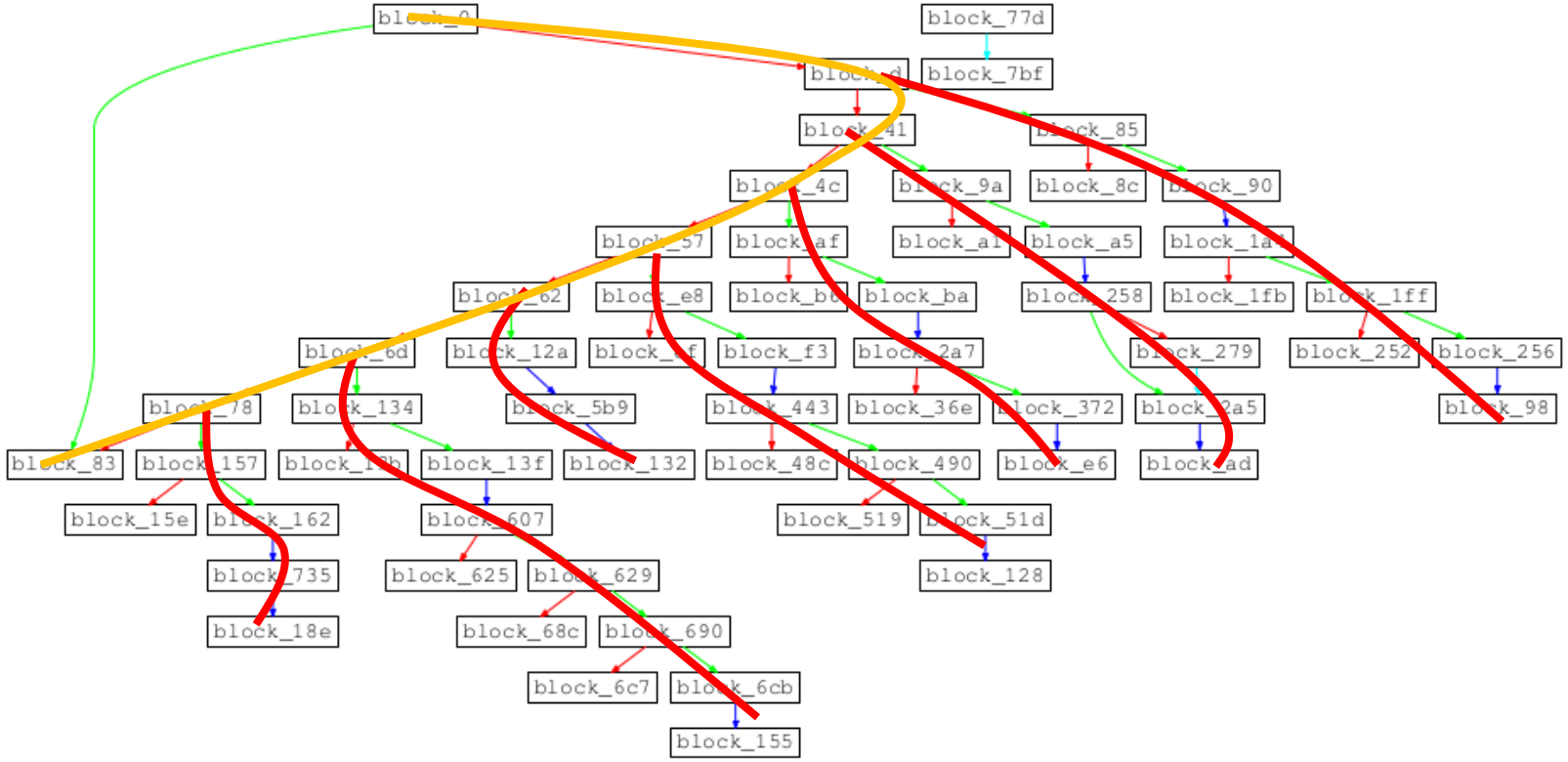
Bytecode 2

- How many functions in this contract?
  - `'Dispatcher' + 'func_41c0e1b5', 'func_473ca96c', 'func_a3a7e7f3', 'func_a9059cbb', 'func_c0e317fb', 'func_da76d5cd', 'func_f8b2cb4f'` == 1 + 7
- How many basicblocks in this contract?
  - 62



Bytecode 2

- How many functions in this contract?
  - `'Dispatcher' + 'func_41c0e1b5', 'func_473ca96c', 'func_a3a7e7f3', 'func_a9059cbb', 'func_c0e317fb', 'func_da76d5cd', 'func_f8b2cb4f'` == 1 + 7
- How many basicblocks in this contract?
  - 62



Continued your analysis

- What does this smart contract do?
- Try to find the original solidity source code?
- Is this smart contract is optimize?
- How many path on this smart contract?

- Try some tools on it:
  - Octopus
  - Rattle
  - Manticore
  - Mythril
  - …