Smart != Secure - Breaking Ethereum Smart Contracts

Elliot Ward & Jake Humphries

AREA41

GOTHAM DIGITAL SCIENCE
A STROZ FRIEDBERG COMPANY

AON
Empower Results®

Elliot Ward

Senior Security Consultant I @elliotjward I eward@gdssecurity.com

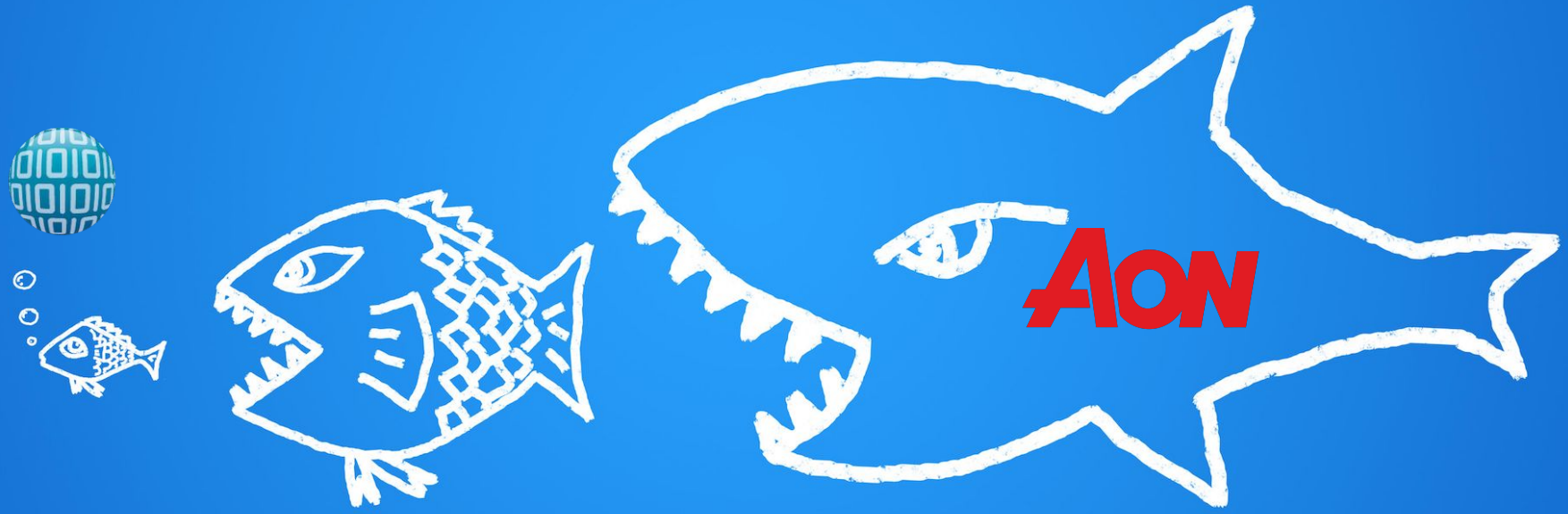
Jake Humphries

Security Consultant I @jake_151 I jhumphries@gdssecurity.com

What we’re talking about

- Ethereum Background
- Blockchain Scanning
- Automated Security Analysis
- Mass Blockchain Scanning
- Future Work

Why are there multiple company logos?



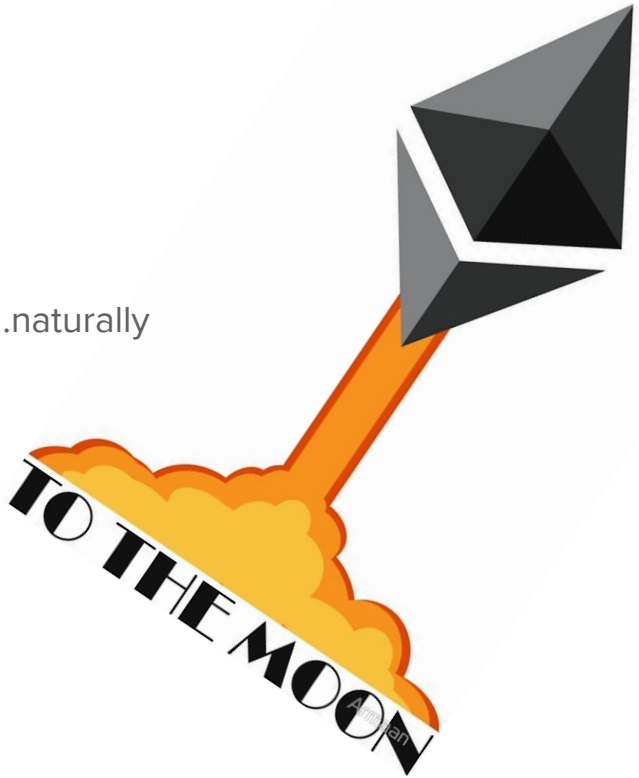
STROZ FRIEDBERG

GOTHAM DIGITAL SCIENCE
A STROZ FRIEDBERG COMPANY

AON
Empower Results®

What we’re not talking about

- Get rich quick schemes
- Breaking the Ethereum Virtual Machine (EVM)
- Breaking the crypto(graphy)
  - We’re not cryptographers…
- Politics around the different coins
  - Any and all blockchain tech is the solution of all problems…naturally



Ethereum Background

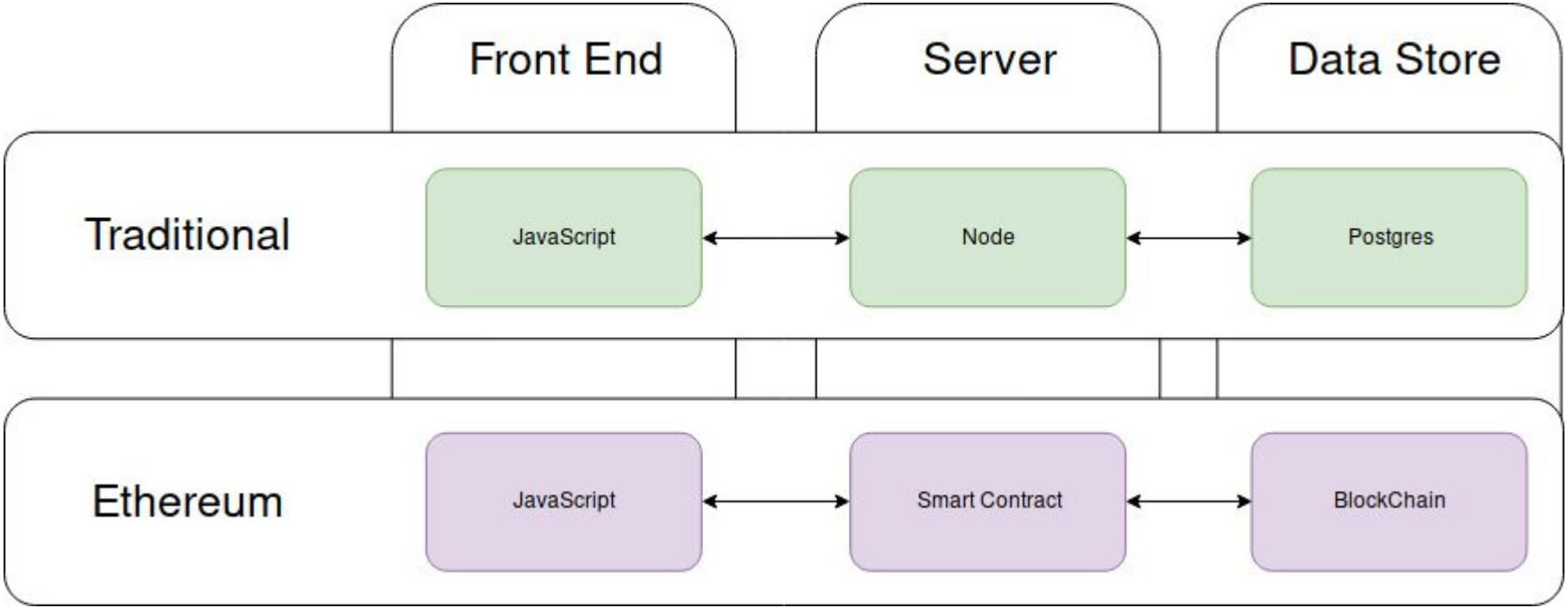
Ethereum Background - EVM

- *Quasi*-turing complete machine
  - Quasi as the EVM is intrinsically bound through the gas parameter, which limits the total amount of computational power available.
- Made up of a simple stack-based architecture with a word size of 256-bit
  - Chosen to facilitate the Keccak-256 (SHA-3) hash and elliptic-curve computation
- Memory model is a word-addressed byte array
- Stack has a maximum size of 1024
- EVM has an independant storage model
  - Similar to memory but is a word-addressable word array, non-volatile and stored as part of the system state

Ethereum Background - Restrictions/Security

- Every computational step must be paid for using *gas* that is always paid, even if the transaction fails.
  - Prevents DoS attempts against the network.
- Programs may only interact with each other in the form of a single arbitrary length byte array. Access to other program state is not possible.
- Sandboxed execution. The EVM program may access and modify its **own** internal state and may trigger **execution** of other EVM programs, but nothing else.
- Execution is fully deterministic and produces identical state transitions
  - This is to ensure all miners on the network produce the same results.

Ethereum Background - Application Architecture



Ethereum Background - Solidity

- Contract-oriented high-level programming language
- Influenced by C++, Python and JavaScript
- Designed to target the Ethereum Virtual Machine
- Statically Typed, with support for inheritance, libraries and other features expected from a high-level programming language.

Ethereum Background - Solidity Code Example

```
pragma solidity ^0.4.0;
contract SampleContract {
    uint storageData;
    function set(uint x) {
        storageData = x;
    }
    function get() public view returns (uint) {
        return storageData;
    }
}
```

Blockchain Scanning

Blockchain Scanning

- Objectives
  - Collect binary opcodes in a sensible manner
  - Store them in a way that can be easily queried
  - Fairly fast and repeatable
- Context
  - Block numbers, transaction hashes, data to give meaning around the contract data collected
- Simple script written in Python3 using the IPC interface for GETH
  - Worked surprisingly well.

Blockchain Scanning - Issues

- Over 5 million blocks
- Hardware can’t keep up :(
  - The Ethereum blockchain is heavily reliant on fast hardware to sync
  - SSD is required and a decent internet connection
  - A `--fast` sync took 3 days to complete, ~80GB in size

Blockchain Scanning - Stats

- Contracts found
  - 1,441,589
- Empty Contracts
  - 59,257
  - Most likely from self destruction or out of gas exceptions during contract creation
- Unique Contracts Found
  - 76,466
- Largest Contract
  - Block Number: 5,246,520
  - Contract Hash: "0x10C621008B210C3A5d0385e458B48af05BF4Ec88"
  - Length: 49,100 bytes
- Most duplicated contract
  - Contract Hash: 0x246381b015702F7042D1cc44bfcaE99A08b915F3
  - First Seen: Block#4241111
  - Total: 361,126!!

Automated Security Analysis - TreacleMine


MOWZER
THE RESEARCH CAT

TreacleMine - Overview

- Automated Security Analysis for EVM bytecode
  - Hybrid approach between static + dynamic analysis
- Opensource - GPL 2.0
- Extensible Architecture
  - Multiple core components
  - Plugin architecture to extend functionality

TreacleMine - Architecture

TreacleMine - Architecture

- Written in pure JavaScript....
  - ~~Never again~~
  - Available as “NodeJS” module with a simple API
- Accessible to Ethereum developers
  - dApps typically comprised of JS frontend + SC “backend”
  - Easy to integrate into development toolchains
  - Designed to run in the browser
  - No need to setup complex environment / dependencies to run tool
- Three core components (currently)
  - EVM Disassembler
  - EVM Emulation Engine
  - **EVM Vulnerability Scanner**

TreacleMine - Architecture

- EVM Disassembler
  - Disassembles EVM bytecode
  - Converts to instructions and parses out basic blocks
- EVM Emulation Engine
  - Emulates EVM state
  - Executes opcodes and tracks variables on the stack using TAGs
- EVM Vulnerability Scanner
  - Plugs into the EVM emulator
  - Processes each opcode during execution
    - “Detector” modules get executed and check for insecure patterns
      - Modules derive a base class and implement the “check” function with self contained detection logic
    - Currently have detectors for 5 unique issues - but still early days

TreacleMine - EVM Vulnerability Scanner

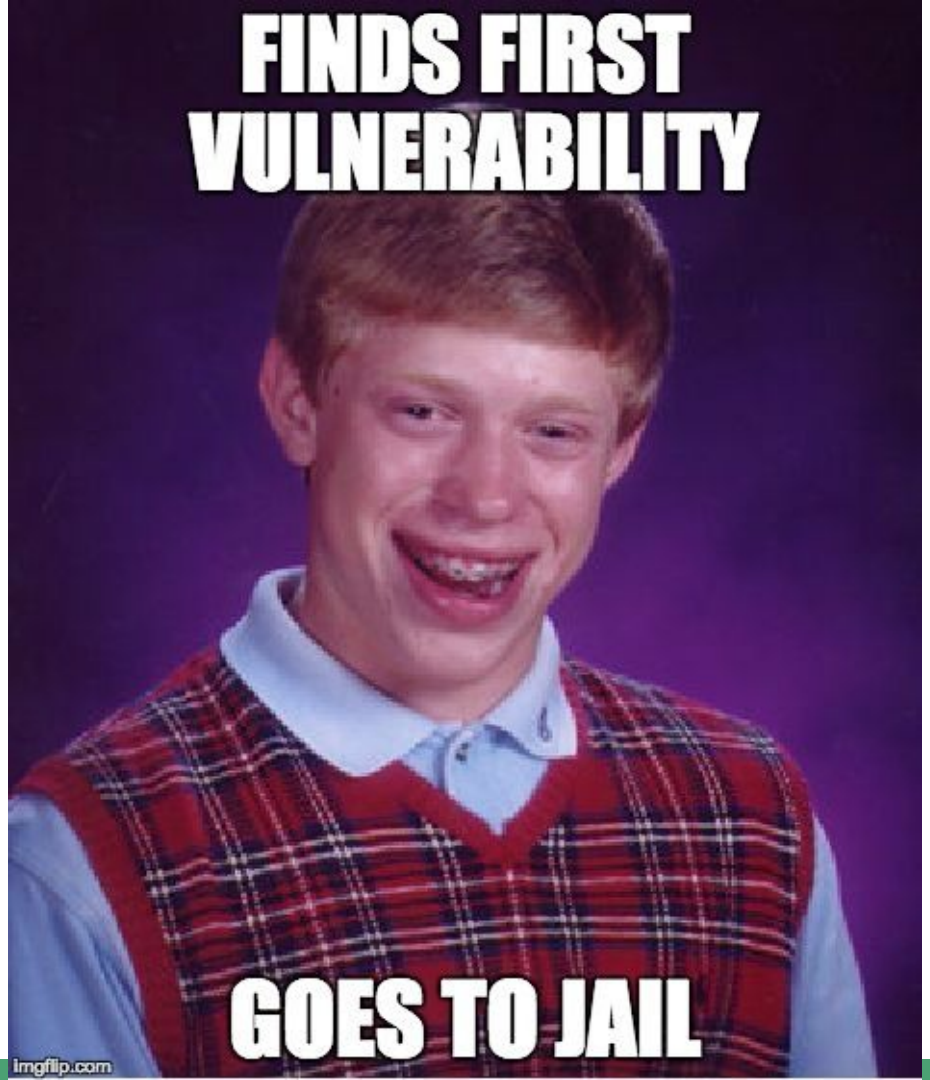
```
module.exports = function() {
  this.issues = [];
  this.signatures = [];

  this.registerSignature = function(signature) {
    this.signatures.push(signature);
  }

  this.processInstruction = function(instruction, stack) {
    for (let i = 0; i < this.signatures.length; i++) {
      this.signatures[i].check(instruction, stack, this.issues);
    }
  }

  this.checkDeferedIssues = function() {
    for (let i = 0; i < this.signatures.length; i++) {
      this.signatures[i].checkDeferedIssues(this.issues);
    }
  }
}
```

Responsible Disclosure


FINDS FIRST VULNERABILITY
GOES TO JAIL
imgflip.com

Let’s Talk - Responsible Disclosure

- *Code is Law (Ethereum mantra)*
- In the outside world applications and services will (commonly) have an owner
  - Due to the Ethereum ecosystem identifying an owner is difficult.
- Patching
  - Deployed Solidity code is immutable
- Question
  - In a world where blockchain tech becomes more widespread, how do we, as an industry see responsible disclosure happening?
- Result
  - No specific contracts will be discussed, only figures.
  - Discussions are being held internally.

Automated Security Analysis - Results

Results - Unique Contracts

- 51 hours to complete the scan
- Using the full Blockchain scan data
- Contract code hashed and the distinct values moved to a new table
- Scan took place over all the distinct contracts bytecode.

63%

Unique Contracts with Vulnerabilities*

*Data correct as of May 2018

Results - On-Chain Contracts

- Full blockchain scan
- Results were correlated from the unique contracts to the on-chain contracts including duplicate contracts

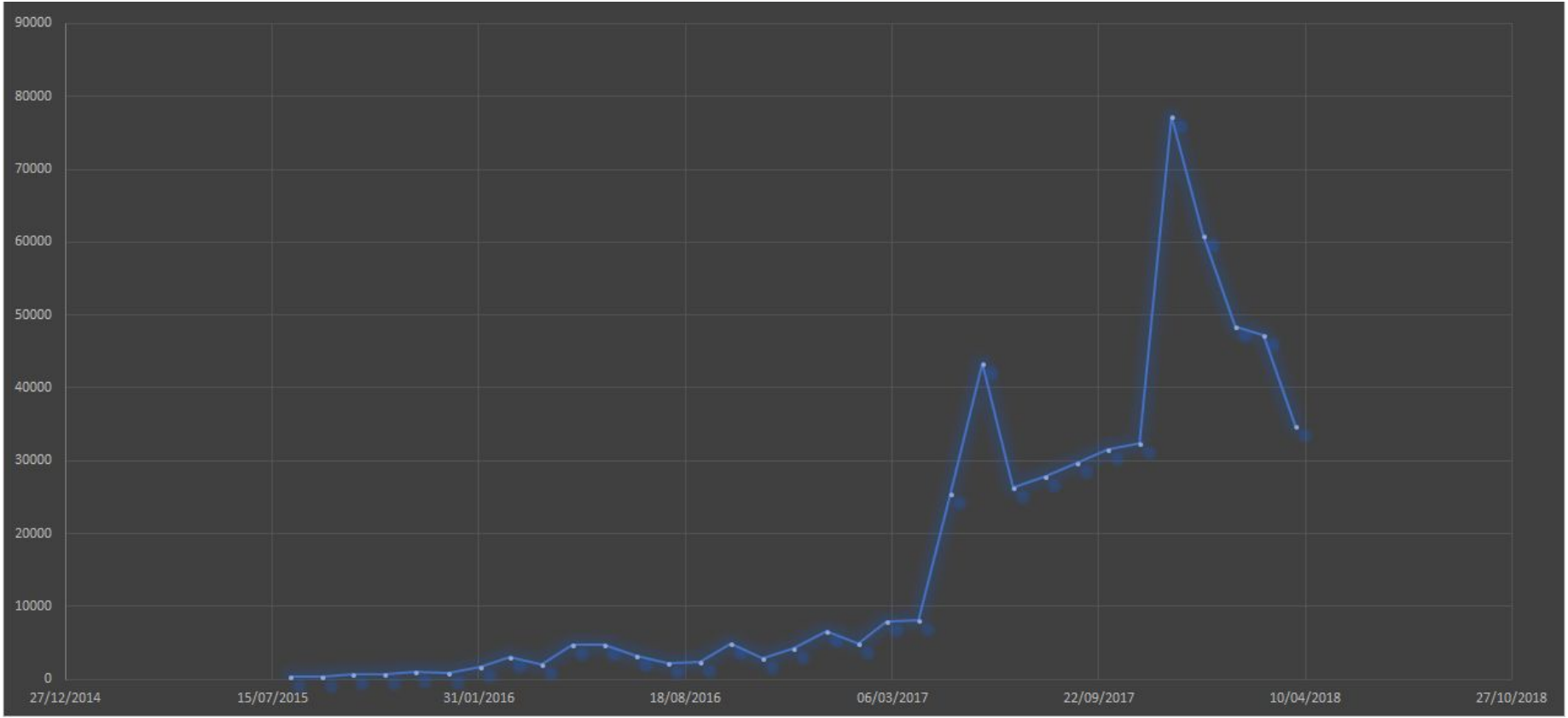
38%

All Deployed On-Chain Contracts*

*Data correct as of May 2018

Automated Security Analysis - Breakdown

Deployed Vulnerable Contracts Over Time



Insecure Random Number Generation

- Difficult to do securely in a decentralised environment
- Weak implementations typically rely on block properties
  - Provides an immediate result to smart-contract without external dependencies
    - block.number
    - block.coinbase
    - block.timestamp
    - block.blockhash
  - Simple to implement
- Incorrect assumption that only miners can manipulate these ‘seeds’
  - + that it is acceptable to do so if payout < block reward

Insecure RNG - Code example

```
function badRandom() public view returns(uint) {
    uint blockNo = block.number - 1;
    uint random_number = uint(block.blockhash(blockNo)) % 10;
    return random_number;
}
```

Insecure RNG - Bytecode example

```
0x00000233 40 BLOCKHASH
0x00000234 60 PUSH1 01
0x00000236 90 SWAP1
0x00000237 04  DIV
0x00000238 81 DUP2
0x00000239 15 ISZERO
0x0000023a 15 ISZERO
0x0000023b 61 PUSH2 0240
0x0000023e 57 JUMPI
0x0000023f fe ASSERT
0x00000240 5b JUMPDEST
0x00000241 06 MOD
0x00000242 90 SWAP1
```

```
IF (CURRENT_OPCODE == 'MOD')

    ARGS[0] = STACK.POP()
    ARGS[1] = STACK.POP()

    IF (VALUE_TAINTED_BY_BLOCKHASH(ARGS))

        SIGNATURE MATCH
```

0.8%

604 unique contracts vulnerable to Insecure Random Number Generation*

**Data correct as of May 2018*

Insecure CALL - Overview

- Solidity provides 3 primary ways to send ether
  - address.call.value(value)();
  - address.send(value);
  - address.transfer(value);

- At EVM level - implemented via CALL instruction
  - CALL(gasLimit, to, value, inputOffset, inputSize, outputOffset, outputSize)

- All these methods transfer control of execution to the recipient
  - EVM does not provide a way to send ether without transferring control flow

Insecure CALL - address.call.value(value)()

- Doesn’t propagate exceptions - returns true/false
- Forwards ALL remaining GAS to the recipient
  - If the recipient is a smart-contract, code execution can be triggered

Insecure CALL - address.call.value code example

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

function withdrawAll() {
    require(balances[msg.sender] > 0);
    msg.sender.call.value(balances[msg.sender])();
    balances[msg.sender] = 0;
}
```

Insecure CALL - Reentrancy attack contract example

```
contract ReentrancyAttacker {
  Reentrancy victim;

  function ReentrancyAttacker(address _victimContractAddress) public {
    victim = ReentrancyVulnerable(_victimContractAddress);
  }

  function depositInContract() public payable {
    require(msg.value > 0);
    victim.deposit.value(msg.value)();
  }

  function withdraw() public {
    victim.withdrawAll();
  }

  function() payable public {
    victim.withdrawAll();
  }
}
```

Insecure CALL - address.call.value detection

- Forwards all available GAS
- Remaining GAS is available via GAS opcode
- If CALL argument gasLimit := GAS opcode result
    - Then we match for **call.value** as **transfer** and **send** both only forward a stipend of **2300** gas

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

54%

40316 unique Contracts vulnerable to Insecure Call via address.call.value*

**Data correct as of May 2018*

Insecure CALL - Unchecked return value

- address.send(value)
  - Returns true||false depending on result of external call
- address.transfer(value)
  - Propagates errors and throws if call fails

- These should be handled appropriately...

Insecure CALL - Unchecked return value code

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

Insecure CALL - Unchecked return value detection

- First...lets understand how the compiled checks look

0x0000007c f1 CALL ← Return value pushed to stack
0x0000007d 93 SWAP4
0x0000007e 50 POP
0x0000007f 50 POP
0x00000080 50 POP
0x00000081 50 POP
0x00000082 15 ISZERO ← Check is performed with ISZERO opcode
0x00000083 60 PUSH1 a4
0x00000085 57 JUMPI

Insecure CALL - Unchecked return value detection

- First...lets understand how the compiled checks look

0x0000007c f1 CALL ← Return value pushed to stack

0x0000007d 93 SWAP4

0x0000007e 50 POP

0x0000007f 50 POP

0x00000080 50 POP

0x00000081 50 POP

0x00000082 15 ISZERO ← Check is performed with ISZERO opcode

0x00000083 60 PUSH1 a4

0x00000085 57 JUMPI

Track CALL return values by tagging stack item with CALLRESULT_$(CALL_INSTRUCTION_OFFSET)

For example: 'CALLRESULT_7C'

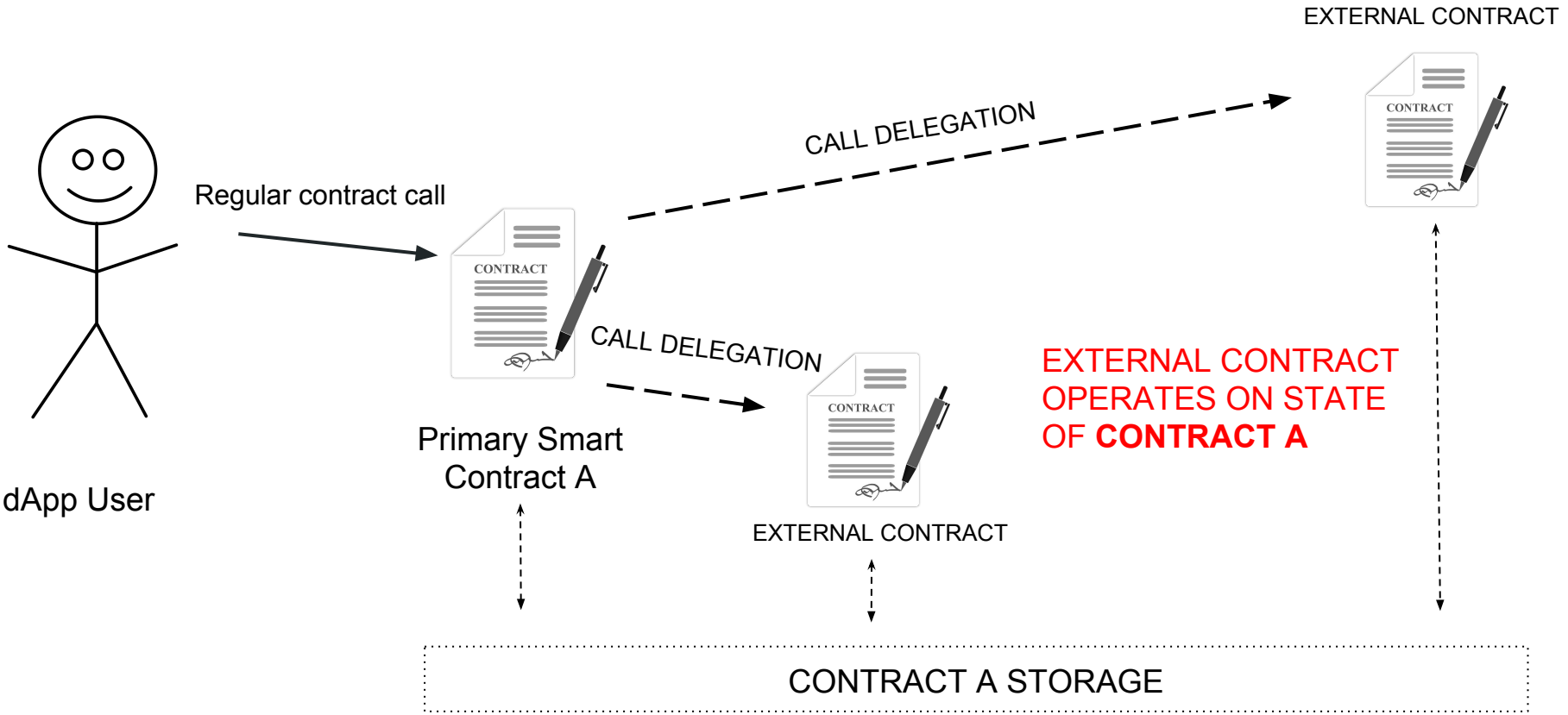
At end of execution - we can see what return values were checked or not

14%

10964 unique Contracts vulnerable to Insecure Call - Unchecked Return Value*

**Data correct as of May 2018*

Delegate Call



Code example - Parity multisig wallet contract

```
function() payable {
  // just being sent some cash?
  if (msg.value > 0)
    Deposit(msg.sender, msg.value);
  else if (msg.data.length > 0)
    _walletLibrary.delegatecall(msg.data);
}
```

CALLER CONTROLLABLE!

Therefore **all public** functions from the library to be callable by **anyone**

Delegate Call - detecting issues

0x0000041b 37 CALLDATACOPY ← Copy input data in current environment to memory

...

0x00000429 51 MLOAD ← Load CALLDATA from memory

0x0000042a 80 DUP1

0x0000042b 83 DUP4

0x0000042c 3 SUB

0x0000042d 81 DUP2

0x0000042e 85 DUP6

0x0000042f 5a GAS

0x00000430 f4 DELEGATECALL ← Invoke the delegate call to external library

6%

4526 unique Contracts using DELEGATECALL*

*Data correct as of May 2018

What Can Be Done?

What Can Be Done

- Code is law
  - Once it’s on the blockchain, there is nothing more you can do, how do we combat this?
- Education
  - Improve the tools and resources around for developers to make better choices in their programming
    - NCC DASP, OpenZeppelin & Securify (to name just a couple)
- SDLC
  - Tool improvement to spot bugs before they become a problem
    - Remix is going a long way to help with this, pointing out obvious flaws from a static analysis stand point

Future Work

Future Work

- Continuation of TreacleMine development
  - Add rules / detectors for more vulnerabilities
    - Frontrunning, DoS, Integer over/underflows, Unchecked Exceptions etc
  - Implement support for STORAGE and MEMORY in emulator
  - Fuzzer (maybe?)
  - More static analysis
    - Identifying which function vulns exist in
    - Do contracts implement any ERC standards?
- Dashboard for continual scanning of Ethereum blockchain contracts
- Web interface for easy use / reverse engineering contracts

evm-disassembler

Github Options

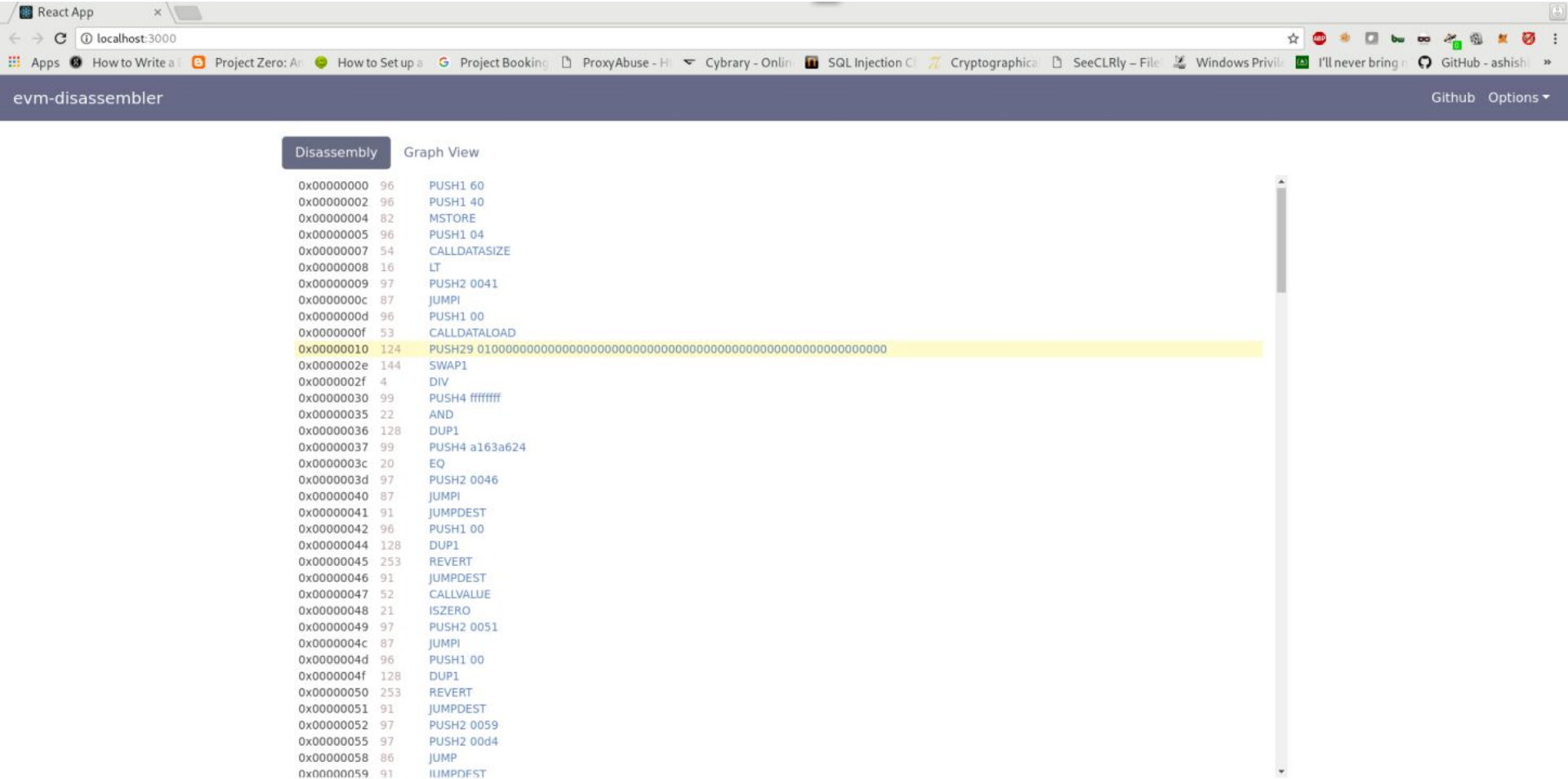
Ethereum Bytecode Disassembler

Open source JavaScript based disassembler and static analyzer for EVM bytecote.

Bytecode Contract Address

```
606060405260043610610041576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff168063a163a62414
610046575b600080fd5b341561005157600080fd5b6100596100d4565b604051808060200182810382528381815181526020019150805190602001 9
080838360005b838110156100995780820151818401526020810190506100 7e565b505050509050908101906 01f6801561 00c6578082038051600
183602003610100 0a031916815260200191505b509250505060405180910390f35b6100dc610123565b6100e4610123565b604080519081016040 5
280601081526020017f414141414141414141414141414141414100000000000000000000000000000000815250905080915050905 65b60206040519
081016040528060008152509056 00a165627a7a72305820ff6e676e29aa4fc4dfe64dd5c660d06fcc201087de6a2fb1707765c318c90f6e0029
```

Disassemble


React App
localhost:3000
Apps
evm-disassembler
Github
Options
Disassembly
Graph View
0x00000000 96 PUSH1 60
0x00000002 96 PUSH1 40
0x00000004 82 MSTORE
0x00000005 96 PUSH1 04
0x00000007 54 CALLDATASIZE
0x00000008 16 LT
0x00000009 97 PUSH2 0041
0x0000000c 87 JUMPI
0x0000000d 96 PUSH1 00
0x0000000f 53 CALLDATALOAD
0x00000010 124 PUSH29 0100000000000000000000000000000000000000000000000000000000
0x0000002e 144 SWAP1
0x0000002f 4 DIV
0x00000030 99 PUSH4 ffffffff
0x00000035 22 AND
0x00000036 128 DUP1
0x00000037 99 PUSH4 a163a624
0x0000003c 20 EQ
0x0000003d 97 PUSH2 0046
0x00000040 87 JUMPI
0x00000041 91 JUMPDEST
0x00000042 96 PUSH1 00
0x00000044 128 DUP1
0x00000045 253 REVERT
0x00000046 91 JUMPDEST
0x00000047 52 CALLVALUE
0x00000048 21 ISZERO
0x00000049 97 PUSH2 0051
0x0000004c 87 JUMPI
0x0000004d 96 PUSH1 00
0x0000004f 128 DUP1
0x00000050 253 REVERT
0x00000051 91 JUMPDEST
0x00000052 97 PUSH2 0059
0x00000055 97 PUSH2 00d4
0x00000058 86 JUMP

fin.

Thank You
Q&A

References/Links

https://ethereum.github.io/yellowpaper/paper.pdf
http://securify.ch/
https://www.dasp.co/