Security in a Immutable Web3 World

Breaching Smart Contracts

Davide TheZero

Whoami

Security Researcher @ Shielder

osservatorionessuno.org

@Th3Zer0

Agenda

- Introduction
- Web3 from a Web2 developer PoV
- Web3 from a Web2 security researcher PoV
- Root Cause Analysis of some of the best web3 vulnerabilities
- Closing remarks

Talk's Objectives

- No **buzz-word** talk
- Understand the concepts behind web3 and smart contracts
- Learn new offensive and defensive concepts

Talk's Objectives

- No **buzz-word** talk
- Understand the concepts behind web3 and smart contracts
- Learn new offensive and defensive concepts

PS: **don't use** in production the code you will see in this talk.

Introduction

Blockchain

Nowadays, everyone knows what a **blockchain** is:

A public distributed immutable database.

Blockchain

Nowadays, everyone knows what a **blockchain** is:

A public distributed immutable database.

Blockchain

Nowadays, everyone knows what a **blockchain** is:

A public distributed immutable database.

Blockchain

Nowadays, everyone knows what a **blockchain** is:

A public distributed immutable database.

Blockchain

Ethereum-like cryptocurrencies introduced the concept of **Smart Contracts**.

Digital contracts where no third party is involved (ex. no public notary).

What is a Smart Contract?

A **Smart Contract** is a collection of code (functions) and data (state) residing at a specific address on the blockchain.

Its code 🔢 enforces a set of programmed rules 🚫.

What is a Smart Contract?

A **Smart Contract** is like a blockchain account (entity):

- Can hold and send money 💰
- Cannot be deleted, cannot be altered
- Interactions (function calls) are irreversible

What is a Smart Contract?


```
pragma solidity 0.6.0;

contract Greeter {
    string greet;

    constructor(string memory _greet) public {
        greet = _greet;
    }

    function greetings(string memory name) public view returns (string memory) {
        return string(abi.encodePacked(greet, name));
    }
}
```

What is a Smart Contract?


```
1  pragma solidity 0.6.0;
2
3  contract Greeter {
4      string greet;
5
6      constructor(string memory _greet) public {
7          greet = _greet;
8      }
9
10     function greetings(string memory name) public view returns (string memory) {
11         return string(abi.encodePacked(greet, name));
12     }
13 }
```

What is a Smart Contract?


```
1  pragma solidity 0.6.0;
2
3  contract Greeter {
4      string greet;
5
6      constructor(string memory _greet) public {
7          greet = _greet;
8      }
9
10     function greetings(string memory name) public view returns (string memory) {
11         return string(abi.encodePacked(greet, name));
12     }
13 }
```

What is a Smart Contract?


```
1  pragma solidity 0.6.0;
2
3  contract Greeter {
4      string greet;
5
6      constructor(string memory _greet) public {
7          greet = _greet;
8      }
9
10     function greetings(string memory name) public view returns (string memory) {
11         return string(abi.encodePacked(greet, name));
12     }
13 }
```

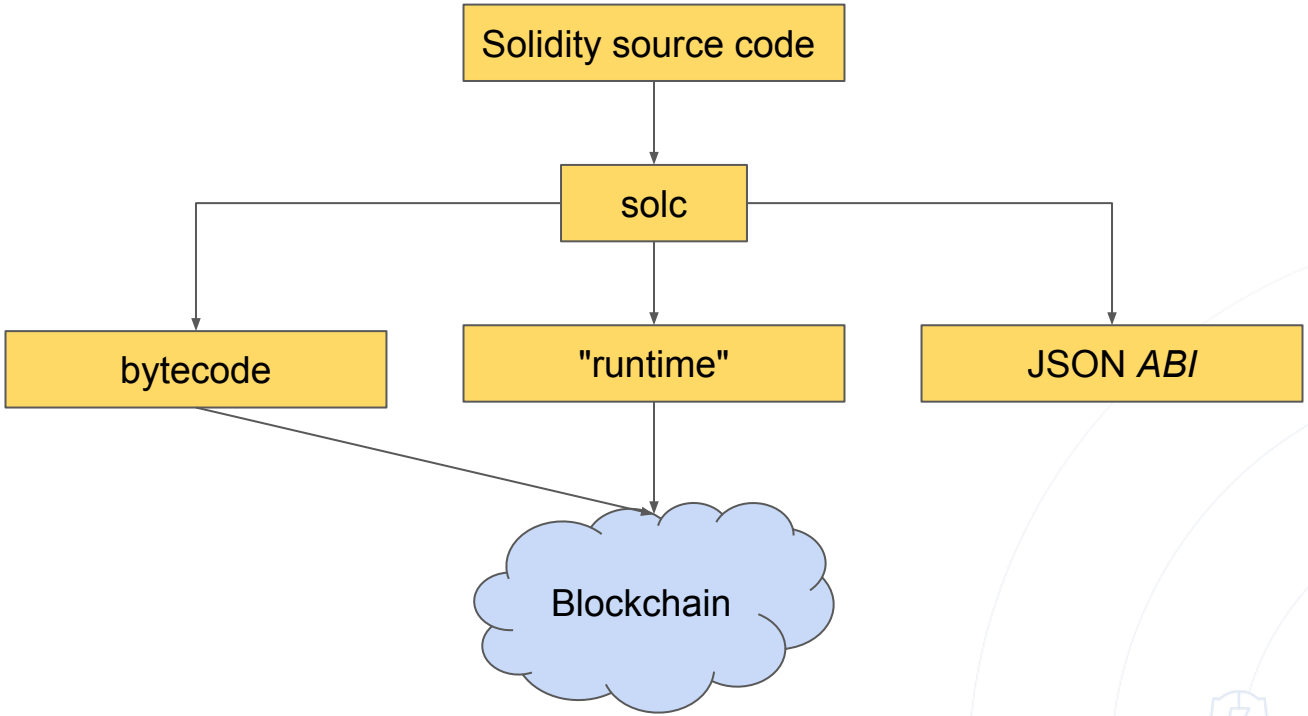
What is a Smart Contract?


```
1  pragma solidity 0.6.0;
2
3  contract Greeter {
4      string greet;
5
6      constructor(string memory _greet) public {
7          greet = _greet;
8      }
9
10     function greetings(string memory name) public view returns (string memory) {
11         return string(abi.encodePacked(greet, name));
12     }
13 }
```

What is a Smart Contract?


```
1  pragma solidity 0.6.0;
2
3  contract Greeter {
4      string greet;
5
6      constructor(string memory _greet) public {
7          greet = _greet;
8      }
9
10     function greetings(string memory name) public view returns (string memory) {
11         return string(abi.encodePacked(greet, name));
12     }
13 }
```

What is a Smart Contract?



Ethereum Virtual Machine

The **EVM** is the execution environment for Smart Contracts.

Ethereum Virtual Machine

The **EVM** is the execution environment for Smart Contracts.

Distributed computing infrastructure.

Ethereum Virtual Machine

The **EVM** is the execution environment for Smart Contracts.

Distributed computing infrastructure.
World state stored on the Ethereum blockchain.

Ethereum Virtual Machine

The **EVM** is the execution environment for Smart Contracts.

Distributed computing infrastructure.
World state stored on the Ethereum blockchain.
RISC instruction set.

Ethereum Virtual Machine

Every participant of the network can **broadcast** a request for an **arbitrary computation**.

This execution causes a **state change** which is committed and propagated inside a blockchain **transaction**.

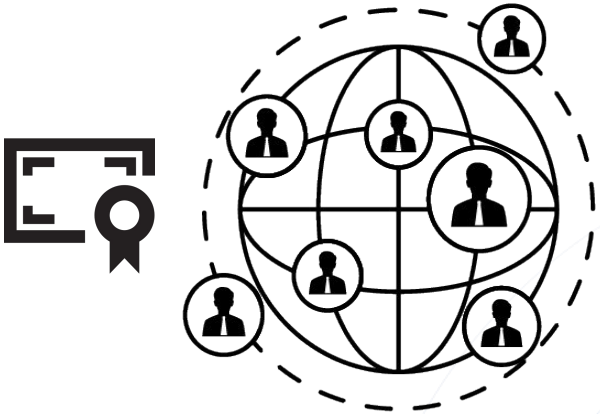
Ethereum Virtual Machine

Every participant of the network can **broadcast** a request for an **arbitrary computation**.

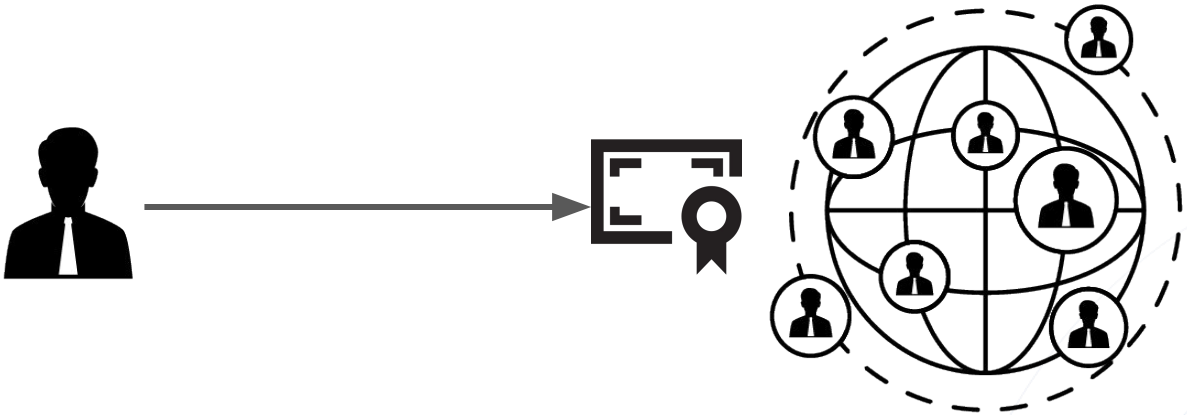
This execution causes a **state change** which is committed and propagated inside a blockchain **transaction**.

This results in ✨Distributed execution✨.

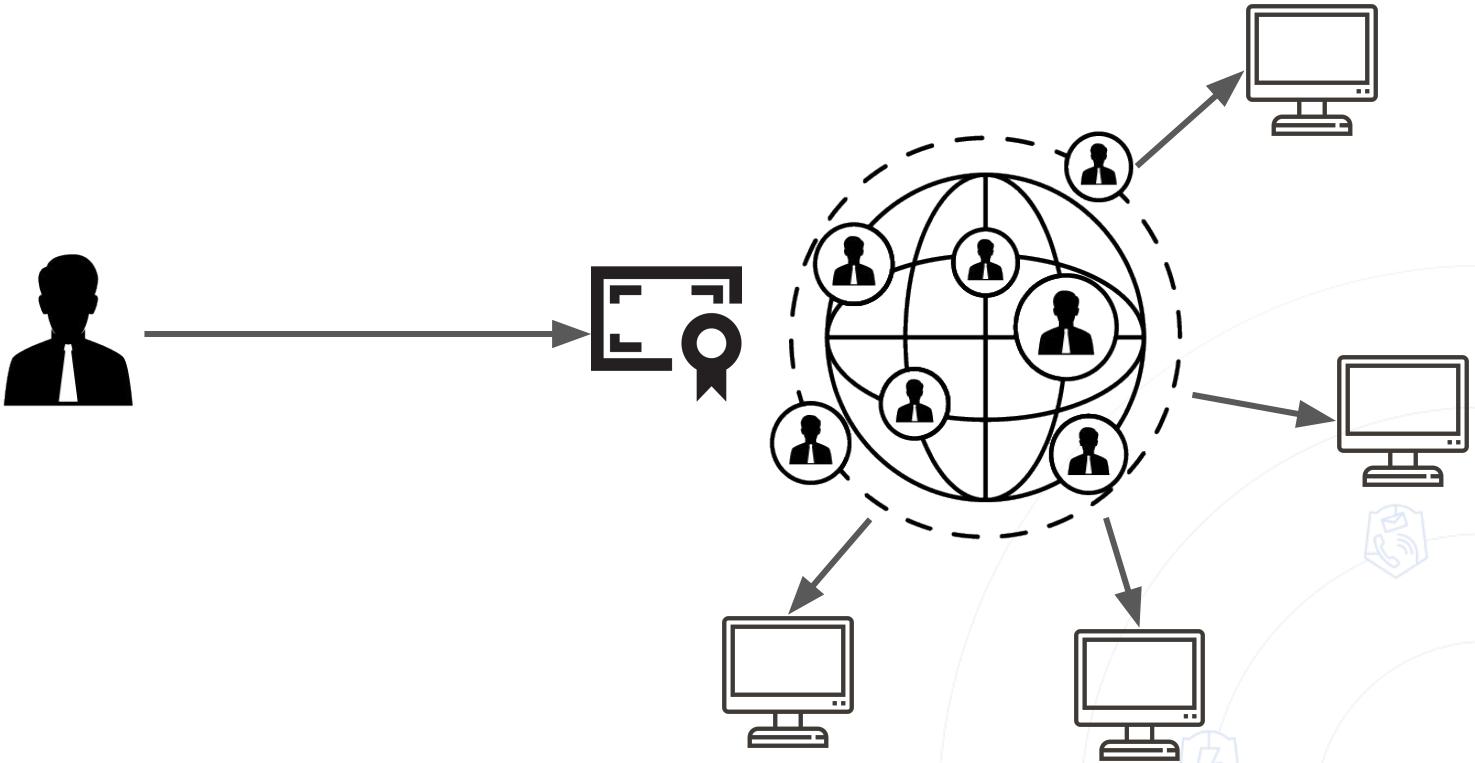
Ethereum Virtual Machine



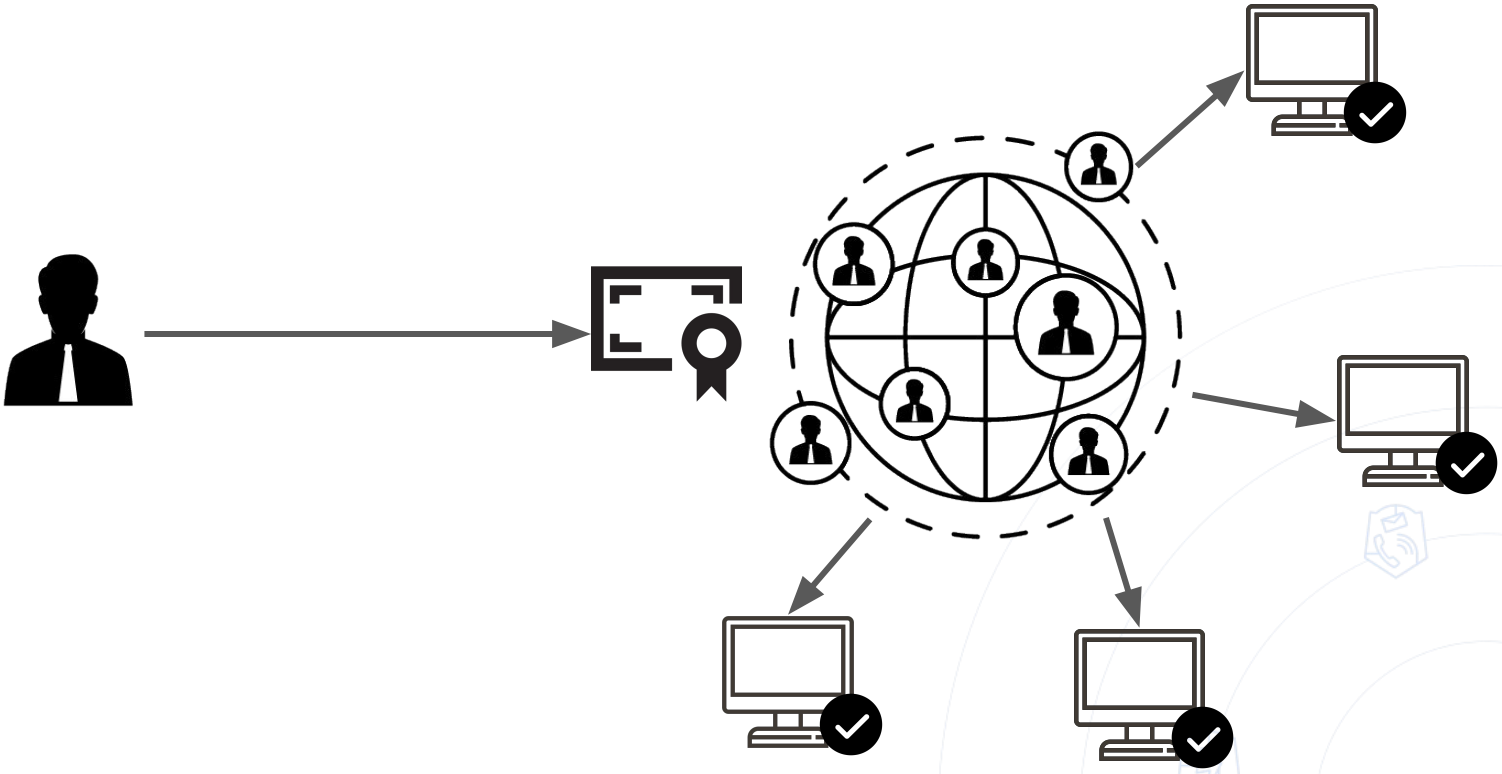
Ethereum Virtual Machine



Ethereum Virtual Machine



Ethereum Virtual Machine



How can we prevent the flooding of the network with execution requests?

Ethereum Virtual Machine

How can we prevent the flooding of the network with execution requests?

Through **Gas**: **Weight** computational effort and state changes (CPU cycles and storage space).

The user pays to execute accordingly.

Ethereum Virtual Machine

Gas makes the EVM a **quasi-Turing complete** machine.

There is no **halting problem**. Smart contracts' execution is (and must be) **deterministic**.

Web3 from a Web2 developer PoV


come up with web3 ideas for our product
crypto kittens
gambling
stick with web2
imgflip.com

Public distributed execution state

Everybody can 👀 the smart contract's **state** and **bytecode**.

Smart contract bytecode should be treated as "client" code.

Public distributed execution state

Everybody can 👀 the smart contract's **state** and **bytecode**.

Smart contract bytecode should be treated as "client" code.

You wouldn't check an admin password inside the browser.

Forget updates

Once your code is deployed on the blockchain it will be available **forever**… until the **`selfdestruct`** opcode is called.

Forget updates

How can you prevent pushing vulnerable code on the blockchain forever?

Forget updates

How can you prevent pushing vulnerable code on the blockchain forever?

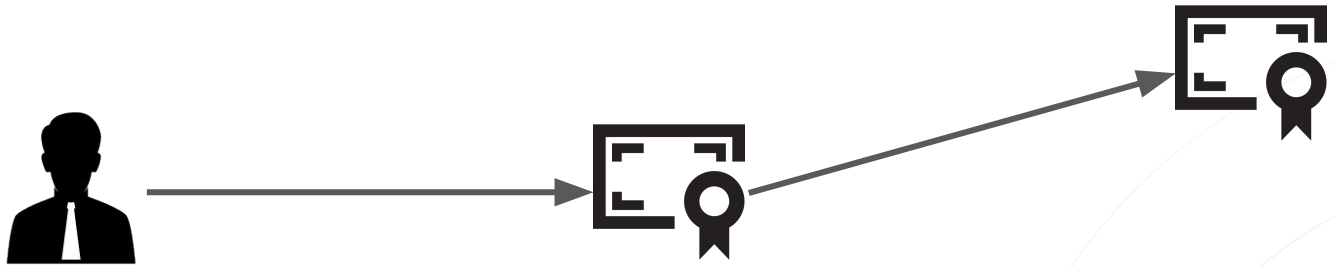
Test before deploy.
Use development tools, testnet/offline/fork blockchain, etc.

Forget updates

"Updates" can be done with "proxy" contracts.

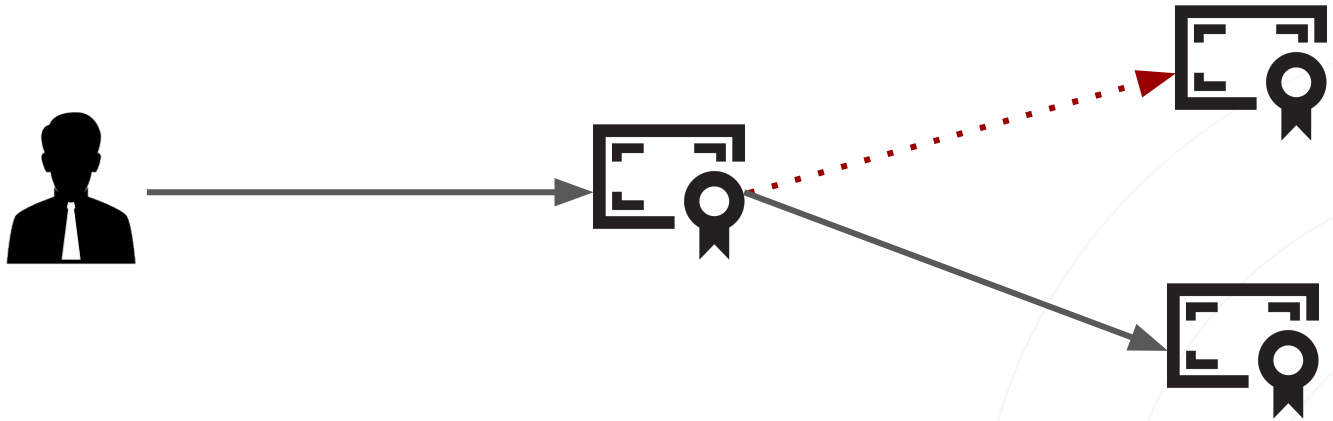
Forget updates

"Updates" can be done with "proxy" contracts.



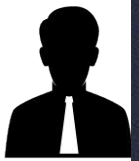
Forget updates

"Updates" can be done with "proxy" contracts.



Forget updates

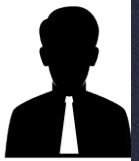
"Updates



```solidity
pragma solidity ^0.5.0;

contract Proxy {

    address public proxyOwner;
    address public implementation;

    constructor(address imp) public {
        proxyOwner = msg.sender;
        implementation = imp;
    }

    modifier onlyProxyOwner() {
        require(msg.sender == proxyOwner);
        _;
    }

    function upgrade(address imp) external onlyProxyOwner {
        implementation = imp;
    }
```

Forget updates

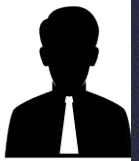
"Updates



```solidity
pragma solidity ^0.5.0;

contract Proxy {

    address public proxyOwner;
    address public implementation;

    constructor(address imp) public {
        proxyOwner = msg.sender;
        implementation = imp;
    }

    modifier onlyProxyOwner() {
        require(msg.sender == proxyOwner);
        _;
    }

    function upgrade(address imp) external onlyProxyOwner {
        implementation = imp;
    }
```

Forget updates

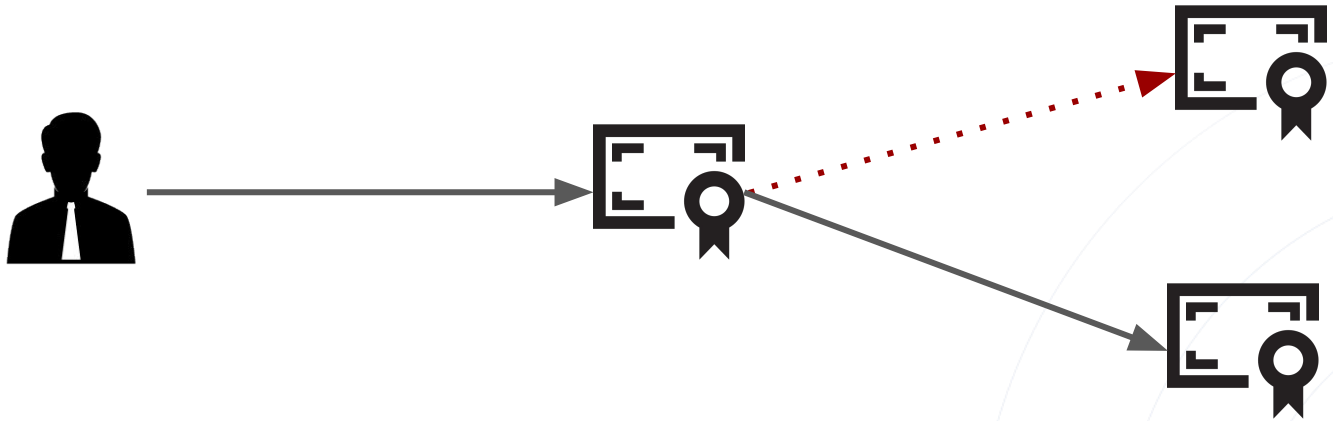
"Updates



```solidity
pragma solidity ^0.5.0;

contract Proxy {

    address public proxyOwner;
    address public implementation;

    constructor(address imp) public {
        proxyOwner = msg.sender;
        implementation = imp;
    }

    modifier onlyProxyOwner() {
        require(msg.sender == proxyOwner);
        _;
    }

    function upgrade(address imp) external onlyProxyOwner {
        implementation = imp;
    }
```

Forget updates

"Updates" can be done with "proxy" contracts.



the "proxy" contract **bytecode** is immutable.

Small code footprint

The lower ⬇️ the code size,
the higher ⬆️ a program's readability and efficiency.

Smart Contracts are usually small.

Solidity is weird

Solidity: **"JavaScript"** but worse… much worse.

Solidity's type system is not safe …
contract interfaces are ***not consulted at compile-time****, and this makes the execution raise an exception and…*

Solidity is weird

Solidity: **"JavaScript"** but worse… much worse.

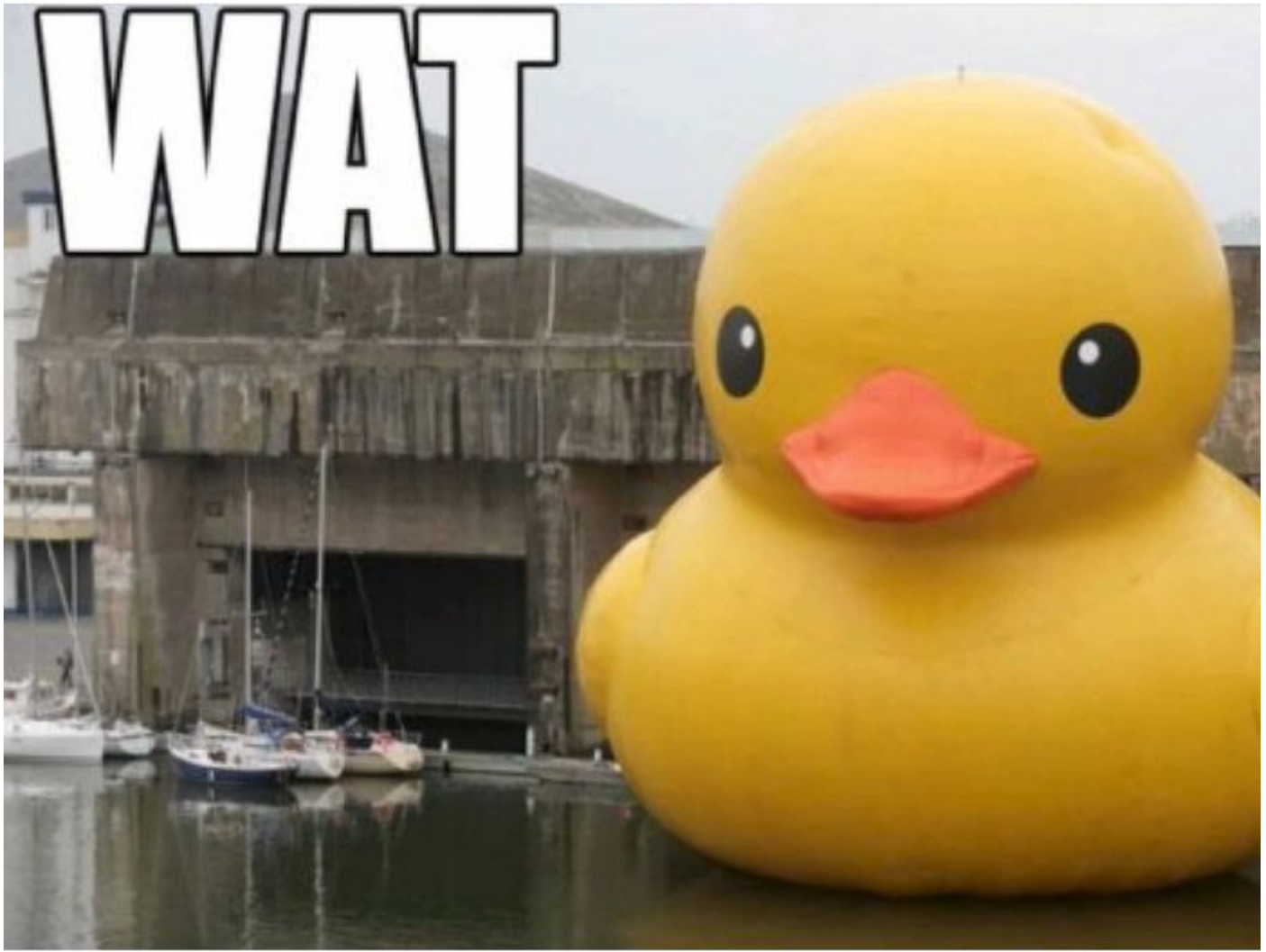
Solidity's type system is not safe …
*contract interfaces are **not consulted at compile-time**, and this makes the execution raise an exception and the user wastes money.*

Ryan Stortz
@withzombies

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

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

10:39 PM - 6 Mar 2018


WAT


WAT
Wat
A lightning talk by Gary Bernhardt from CodeMash 2012
https://www.destroyallsoftware.com/talks/wat

Fast evolving architecture

Unlike CPUs (🤭) the EVM/compilers can evolve at speed.

Kill vulnerability classes with breaking changes in the VM or in the Solidity compiler.

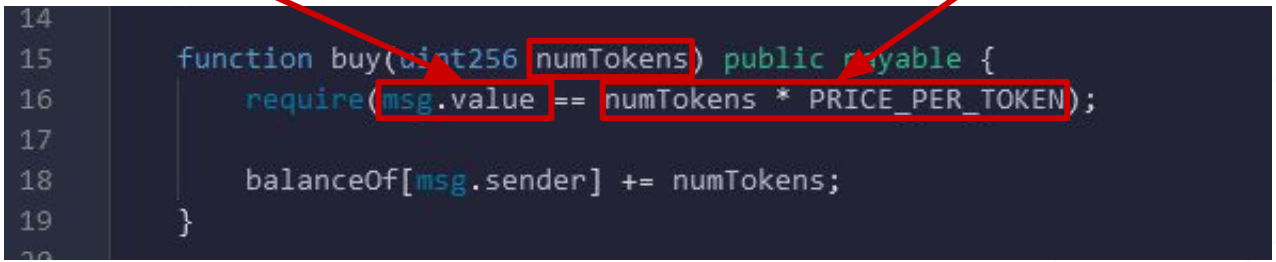
Fast evolving architecture


```solidity
pragma solidity ^0.4.21;

contract TokenSaleChallenge {
    mapping(address => uint256) public balanceOf;
    uint256 constant PRICE_PER_TOKEN = 1 ether;

    function TokenSaleChallenge(address _player) public payable {
        require(msg.value == 1 ether);
    }

    function isComplete() public view returns (bool) {
        return address(this).balance < 1 ether;
    }

    function buy(uint256 numTokens) public payable {
        require(msg.value == numTokens * PRICE_PER_TOKEN);

        balanceOf[msg.sender] += numTokens;
    }

    function sell(uint256 numTokens) public {
        require(balanceOf[msg.sender] >= numTokens);

        balanceOf[msg.sender] -= numTokens;
        msg.sender.transfer(numTokens * PRICE_PER_TOKEN);
    }
}
```

Fast evolving architecture

How many coins
we are sending

How many
tokens we want



Ask for a huge number of token,
multiplication will **wrap around (overflow)** to a low value,
then send just a few coins.

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

```
1  contract c0 {
2      function f2() public  returns (uint8) {
3          uint8 exp = uint8(2) ** uint8(8);
4          return uint8(0) ** exp;
5      }
6  }
```

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

```
1  contract c0 {
2      function f2() public  returns (uint8) {
3          uint8 exp = uint8(2) ** uint8(8);
4          return uint8(0) ** exp;
5      }
6  }
```

2^8=
256

uint8 can store up to 255

exp will wrap around and become 0

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

```
1  contract c0 {
2      function f2() public  returns (uint8) {
3          uint8 exp = uint8(2) ** uint8(8);
4          return uint8(0) ** exp;
5      }
6  }
```

so we are doing 0^0, right?

Remember:

0 to the power of something is 0

0 to the power of 0 is an indeterminate form defined to be 1

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

solc < 0.4
outputs **0**

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

solc < 0.4
outputs **0**

solc > 0.4.25
outputs **1**

Fast evolving architecture

SOL-2018-4: **Cleanup of Exponent in Exponentiation**

solc < 0.4	solc > 0.4.25	solc > **0.8.x**
outputs **0**	outputs **1**	revert execution

Fast evolving architecture

How solc 0.8.x reverts execution on overflows?

Adds bytecode at **compile-time** to *guard*.

Fast evolving architecture

How solc 0.8.x reverts execution on overflows?

Adds bytecode at **compile-time** to *guard*. More gas-expensive.

Fast evolving architecture

How solc 0.8.x reverts execution on overflows?

Adds bytecode at **compile-time** to *guard*. More gas-expensive.
Use of **unchecked** to disable checks and save gas. 🤦

Gas optimization

It is important to **minimize gas requirements**:

- Minimize executed instructions (code).
- Minimize storage usage (data).
- Etc.

Gas optimization


```
1   contract c1 {
2       function dumb_function(uint[] memory arr) public returns (uint[] memory) {
3           assert(arr.length < 256);
4           for (uint8 i = 0; i < arr.length; i++) {
5               arr[i] = i;
6           }
7           return arr;
8       }
9   }
10
11  contract c2 {
12      function dumb_function(uint[] memory arr) public returns (uint[] memory) {
13          for (uint8 i = 0; i < arr.length; i++) {
14              arr[i] = i;
15          }
16          return arr;
17      }
18  }
```

Gas optimization


```
1  contract c1 {
2      function dumb_function(uint[] memory arr) public returns (uint[] memory) {
3          assert(arr.length < 256);
4          for (uint8 i = 0; i < arr.length; i++) {
5              arr[i] = i;
6          }
7          return arr;
8      }
9  }
10
11 contract c2 {
12     function dumb_function(uint[] memory arr) public returns (uint[] memory) {
13         for (uint8 i = 0; i < arr.length; i++) {
14             arr[i] = i;
15         }
16         return arr;
17     }
18 }
```

Gas optimization


```
1   contract c1 {
2       function dumb_function(uint[] memory arr) public returns (uint[] memory) {
3           assert(arr.length < 256);
4           for (uint8 i = 0; i < arr.length; i++) {
5               arr[i] = i;
6           }
7           return arr;
8       }
9   }
10
11  contract c2 {
12      function dumb_function(uint[] memory arr) public returns (uint[] memory) {
13          for (uint8 i = 0; i < arr.length; i++) {
14              arr[i] = i;
15          }
16          return arr;
17      }
18  }
```

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

`c2.dumb_function`:

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

input: [1,2,3,4,5]
gas used: 23,961

`c2.dumb_function`:

input: [1,2,3,4,5]
gas used: **23,935**

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

input: [1,2,3,4,5]
gas used: 23,961

`c2.dumb_function`:

input: [1,2,3,4,5]
gas used: **23,935**

26 gas units gained

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

`c2.dumb_function`:

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

input: [1,2,3,...256]
gas used: **67,208**

`c2.dumb_function`:

input: [1,2,3,...256]
gas used: **3,000,000***

Gas optimization


```
assert(arr.length < 256);
```

PUSH2 0100	3
DUP3	3
MLOAD	3*
LT	3
PUSH2 014b	3
JUMPI	10
INVALID	NaN

`c1.dumb_function`: (with assert)

input: [1,2,3,...256]
gas used: **67,208**

`c2.dumb_function`:

input: [1,2,3,...256]
gas used: **3,000,000***

user lose money

A bug could cost Millions

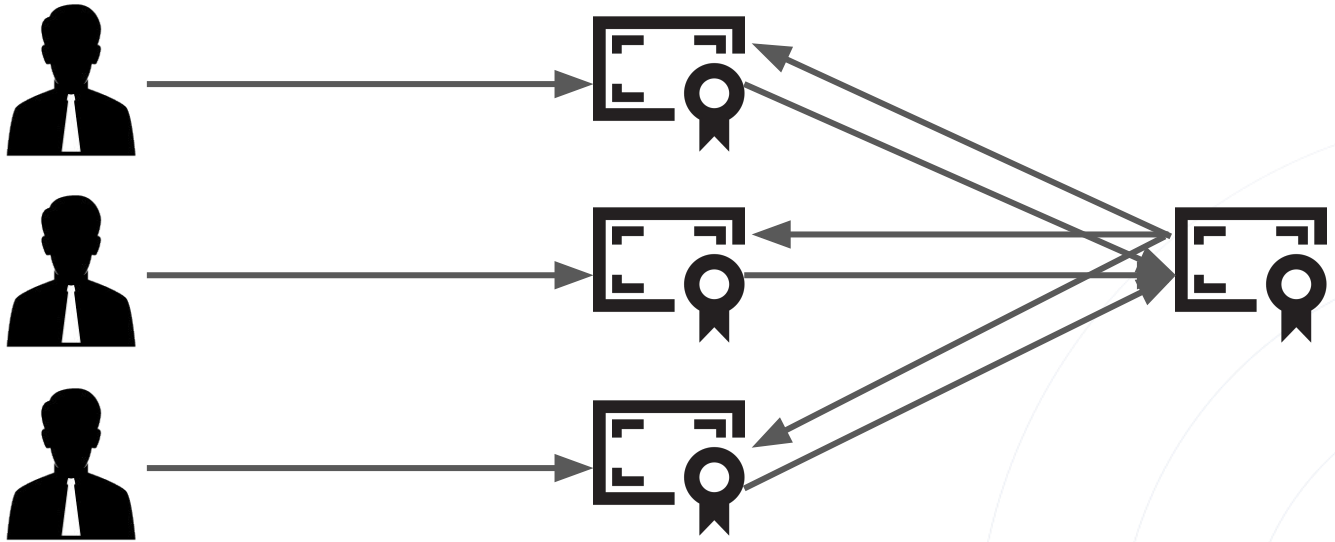
Try to explain to your boss that
a guy on the internet just deleted all your money.

A bug could cost Millions

Smart Contracts can be used as a "library" to minimize costs.

A bug could cost Millions

Smart Contracts can be used as a "library" to minimize costs.

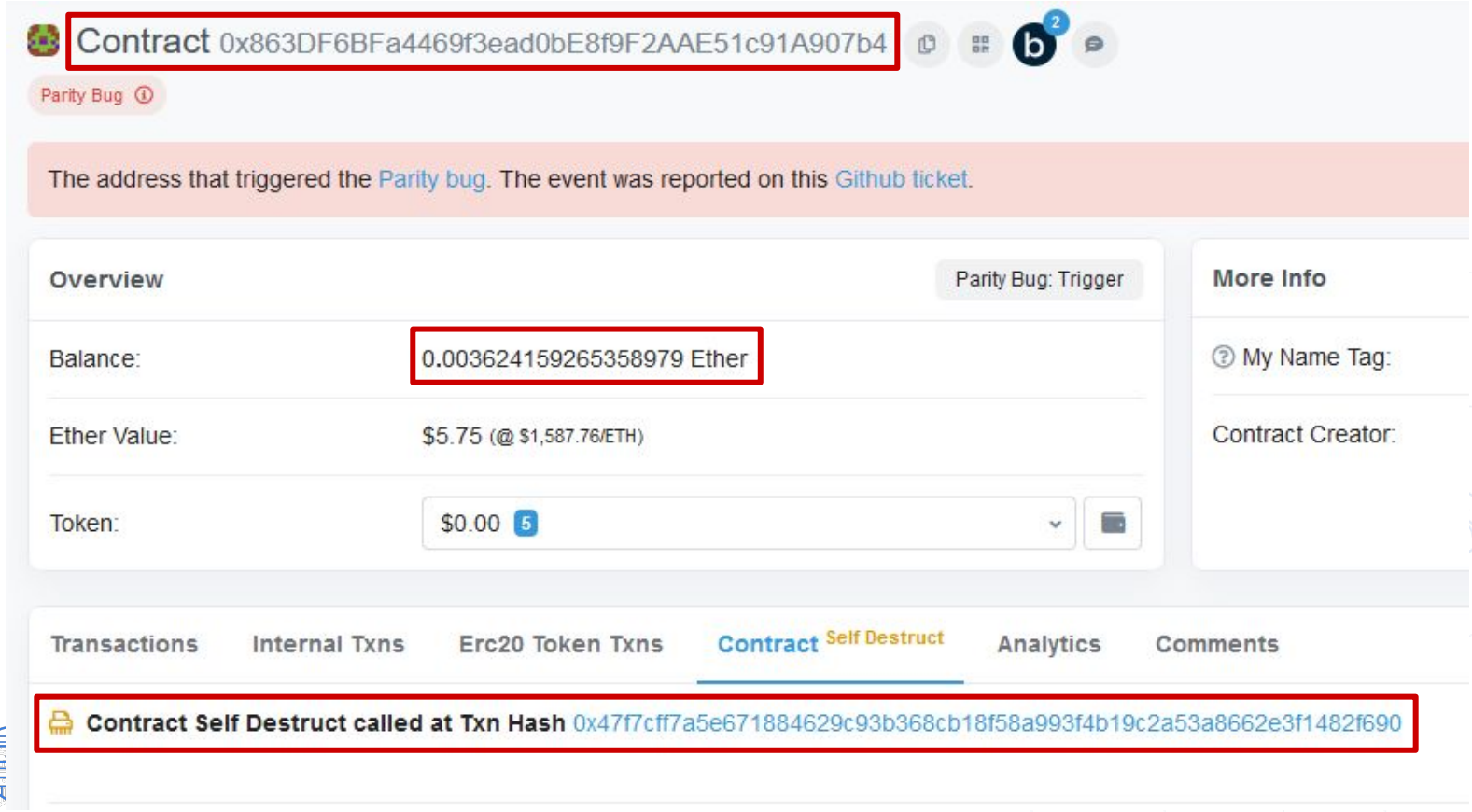


A bug could cost Millions

1. Initialize the contract as owner
2. Kill it
3. …
4. ~~Profit~~

0x47f7cff7a5e67188462…	Kill	4501969	2017-11-06 15:25:21	0xae7168deb525862f4f…	IN	Parity Bug: Trigger
0x05f71e1b2cb4f03e547…	Init Wallet	4501736	2017-11-06 14:33:47	0xae7168deb525862f4f…	IN	Parity Bug: Trigger
0x348ec4b5a396c95b4a…	0x60606040	4049249	2017-07-20 16:39:46	0x00caa64684700d2825…	IN	Create: WalletLibrary

A bug could cost Millions



Web3 from a Web2 security researcher PoV


I'VE FINALLY FOUND IT... AFTER 15 YEARS
THE SCROLL OF TRUTH!
RobotatertotComics
web3 exploitation is as low as binary exploitation
NYEHHH
imgflip.com

EVM is a RISC machine

EVM is a RISC Harvard-architecture stack machine.

~140 defined opcodes.
Put values on stack then perform operations.

Cutting edge binary-like research

Simple architecture → easy to prototype, emulate, etc.

Symbolic execution && Formal verification is the norm.

Cutting edge binary-like research

Simple architecture → easy to prototype, emulate, etc.

Symbolic execution && Formal verification is the norm.

Manticore, KEVM, hevm, Mythril, Pakala, **Qiling**, Echidna, … much more.

Automation is the key

Finding low hanging fruits is easy.

Symbolic execution can provide **exact input** to exploit vuln.

Scan new blocks and exploit at scale. 

Many logical and financial bugs

Developers kill low hanging fruits before production.

Most of live bugs are either logical or financial.

Time to learn something new… 👩‍🎓

Many logical and financial bugs

Shorting: selling an asset without owning it and buying it back later at a lower price.

Stablecoin: a crypto currency with value pegged / tied to that of another currency, typically USD.

Many logical and financial bugs

In order to short BTC → attack the **Terra(LUNA)** Stablecoin.

1. Loan some BTC.

Many logical and financial bugs

In order to short BTC → attack the Terra(LUNA) Stablecoin.

1. Loan some BTC.
2. Buy a lot of Terra with BTC.

Many logical and financial bugs

In order to short BTC → attack the Terra(LUNA) Stablecoin.

1. Loan some BTC.
2. Buy a lot of Terra with BTC.
3. Sell 35k Terra for 34k USDC (two USD stablecoins).

Many logical and financial bugs

In order to short BTC → attack the Terra(LUNA) Stablecoin.

1. Loan some BTC.
2. Buy a lot of Terra with BTC.
3. Sell 35k Terra for 34k USDC (two USD stablecoins).
4. People will go panic mode, crashing the market.

Many logical and financial bugs

Many logical and financial bugs

In order to short BTC → attack the Terra(LUNA) Stablecoin.

1. Loan some BTC.
2. Buy a lot of Terra with BTC.
3. Sell 35k Terra for 34k USDC (two USD stablecoins).
4. People will go panic mode, crashing the market.
5. Terra Foundation sell **big** BTC to increase Terra price.

Many logical and financial bugs

In order to short BTC → attack the Terra(LUNA) Stablecoin.

1. Loan some BTC.
2. Buy a lot of Terra with BTC.
3. Sell 35k Terra for 34k USDC (two USD stablecoins).
4. People will go panic mode, crashing the market.
5. Terra Foundation sell **big** BTC to increase Terra price.
6. **Buy back BTC at lower price.**

Not limited to smart contracts

- Bugs in SOLC or in the EVM
- Bugs in other VMs (OptimismVM attack by saurik)
- Cross-chain interactions bugs
- Flash Loans
- Cross-chain Flash Loans
- **… ?**

Root Cause Analysis of some of the best web3 vulnerabilities

DAO Hack

What is a DAO? Decentralized Autonomous Organization.

A Smart Contract where:

- **Funds raised** from the investors.
- Holders can **submit proposals** to the consensus.
- Profits from proposals **return** to holders.

DAO Hack

If a minority objected a proposal that is going to be funded, the DAO splits and allow them to retrieve their funds.

DAO Hack


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

    function buyTokens(){
        balances[msg.sender] += msg.value;
    }

    function transferTokens(address _to, uint _amount){
        if (balances[msg.sender] < _amount)
            throw;
        balances[_to]=_amount;
        balances[msg.sender]-=_amount;
    }

    function withdraw(address _recipient) returns (bool) {
        if (balances[msg.sender] > 0){
            if (_recipient.call.value(balances[msg.sender])()) {
                balances[msg.sender] = 0;
                return true;
            }
        }
        throw;
    }
}
```

DAO Hack

Sending coins
add "pot" balance

Fund a "proposal"

Retrieve coins

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

    function buyTokens(){
        balances[msg.sender] += msg.value;
    }

    function transferTokens(address _to, uint _amount){
        if (balances[msg.sender] < _amount)
            throw;
        balances[_to]=_amount;
        balances[msg.sender]-=_amount;
    }

    function withdraw(address _recipient) returns (bool) {
        if (balances[msg.sender] > 0){
            if (_recipient.call.value(balances[msg.sender])()) {
                balances[msg.sender] = 0;
                return true;
            }
        }
        throw;
    }
}
```

DAO Hack

Sending coins
add "pot" balance

Fund a "proposal"

Retrieve coins

Transfer coins
before updating
balance

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

    function buyTokens(){
        balances[msg.sender] += msg.value;
    }

    function transferTokens(address _to, uint _amount){
        if (balances[msg.sender] < _amount)
            throw;
        balances[_to]=_amount;
        balances[msg.sender]-=_amount;
    }

    function withdraw(address _recipient) returns (bool) {
        if (balances[msg.sender] > 0){
            if (_recipient.call.value(balances[msg.sender])()) {
                balances[msg.sender] = 0;
                return true;
            }
        }
        throw;
    }
}
```

DAO Hack

`_recipient` can contain the address of a Contract.

```
15    function withdraw(address _recipient) returns (bool) {
16        if (balances[msg.sender] > 0){
17            if (_recipient.call.value(balances[msg.sender])()) {
18                balances[msg.sender] = 0;
19                return true;
20            }
21        }
22        throw;
23    }
```

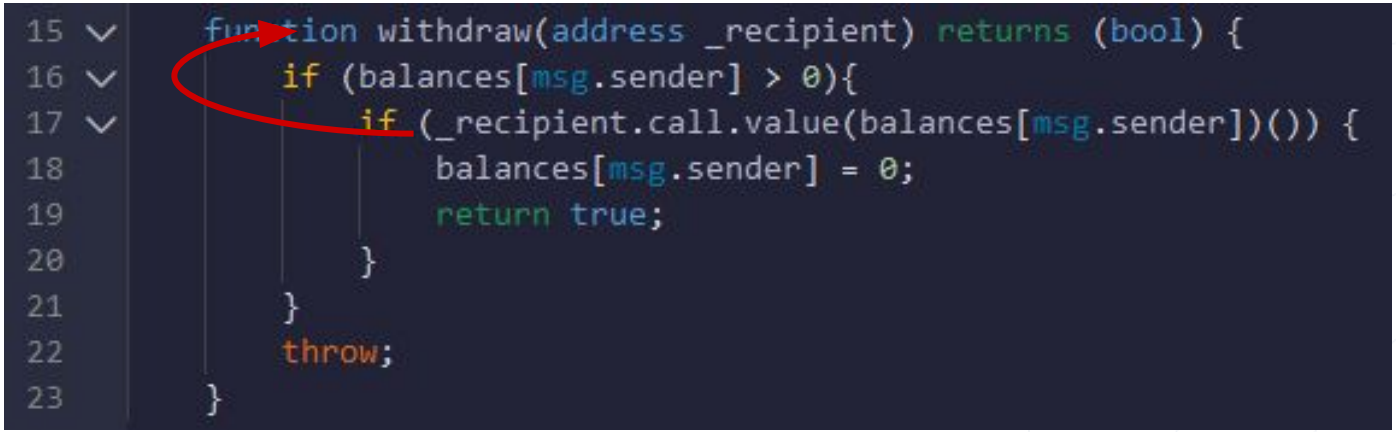
DAO Hack

`call` will transfer coins and execution flow to **`_recipient`** fallback function.

```
15    function withdraw(address _recipient) returns (bool) {
16        if (balances[msg.sender] > 0){
17            if (_recipient.call.value(balances[msg.sender])()) {
18                balances[msg.sender] = 0;
19                return true;
20            }
21        }
22        throw;
23    }
```

DAO Hack

`_recipient` fallback function can call withdraw recursively.

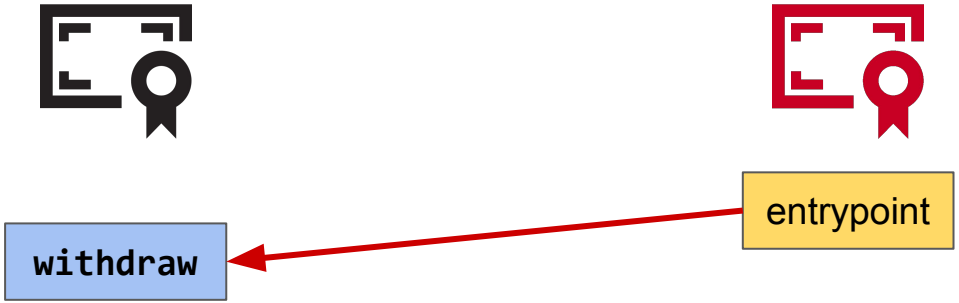


DAO Hack

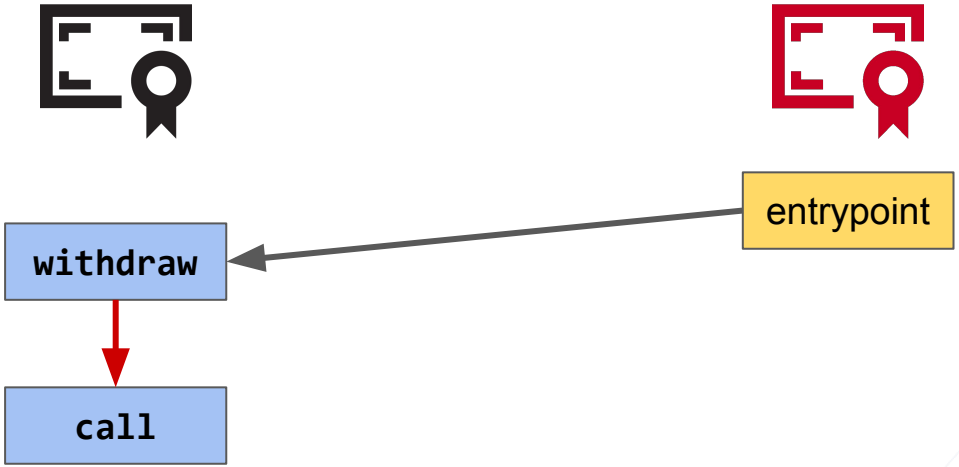
`balances` will never be updated.

```
15    function withdraw(address _recipient) returns (bool) {
16        if (balances[msg.sender] > 0){
17            if (_recipient.call.value(balances[msg.sender])()) {
18                balances[msg.sender] = 0;
19                return true;
20            }
21        }
22        throw;
23    }
```

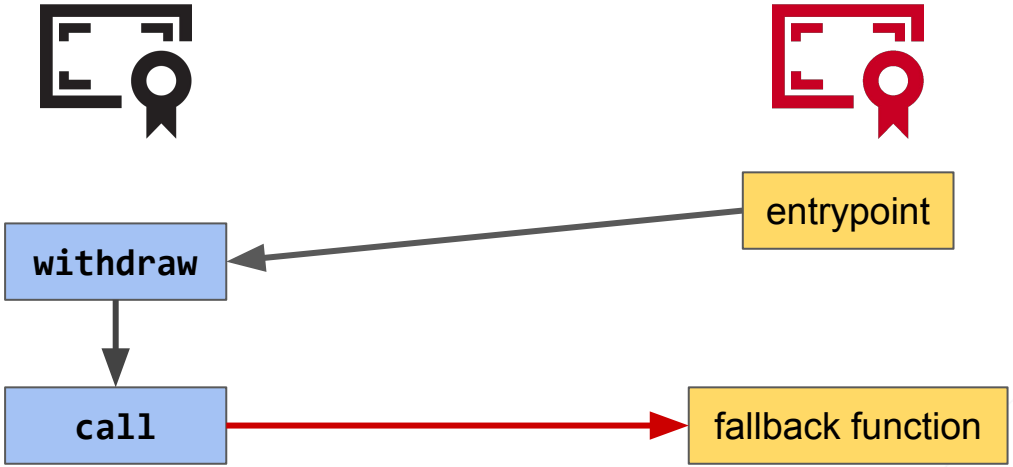
DAO Hack



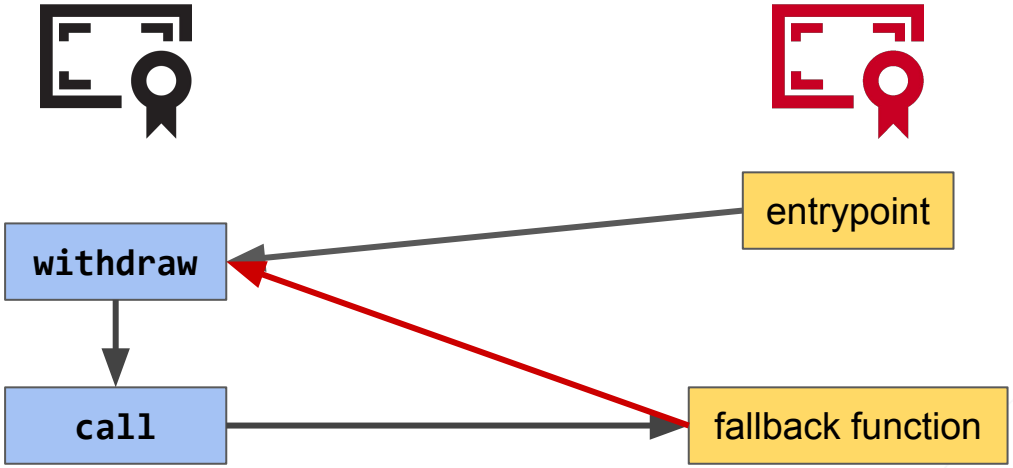
DAO Hack



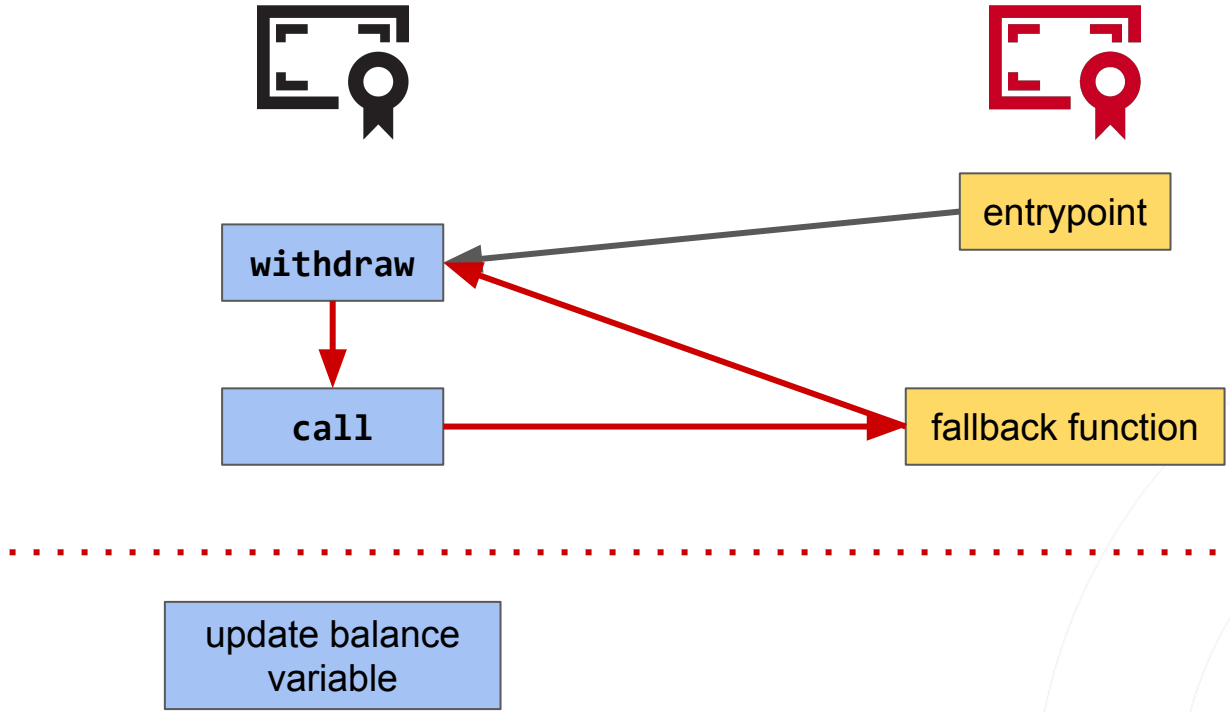
DAO Hack



DAO Hack

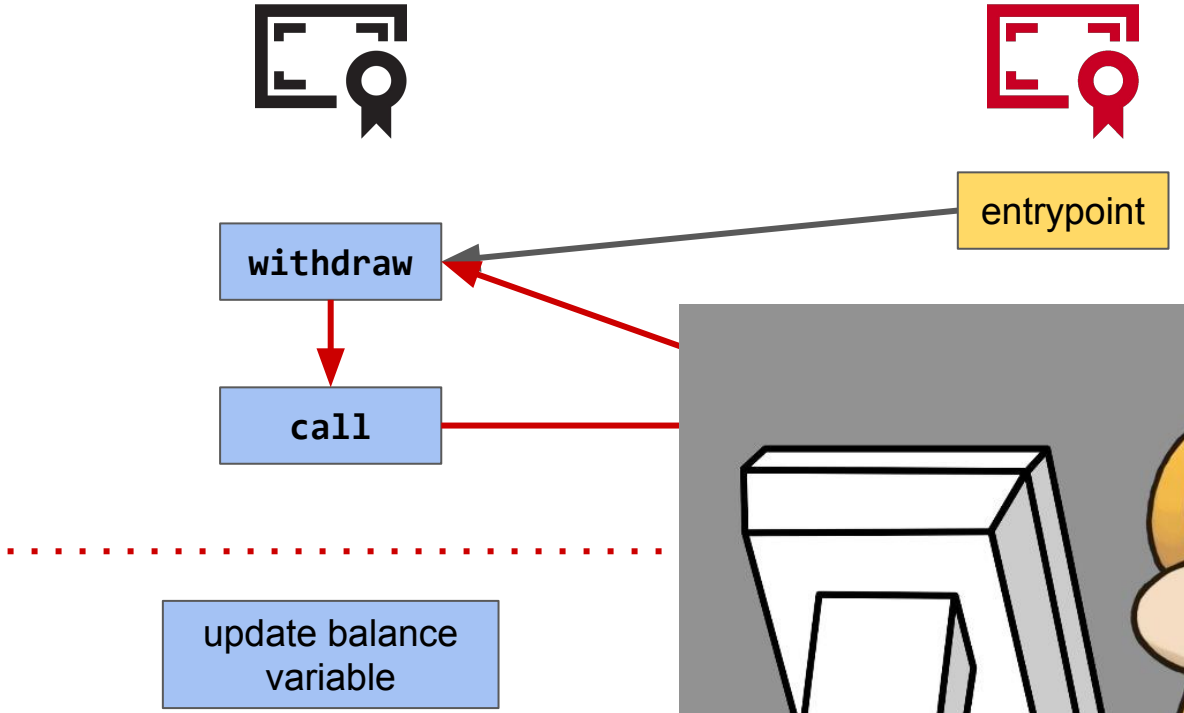


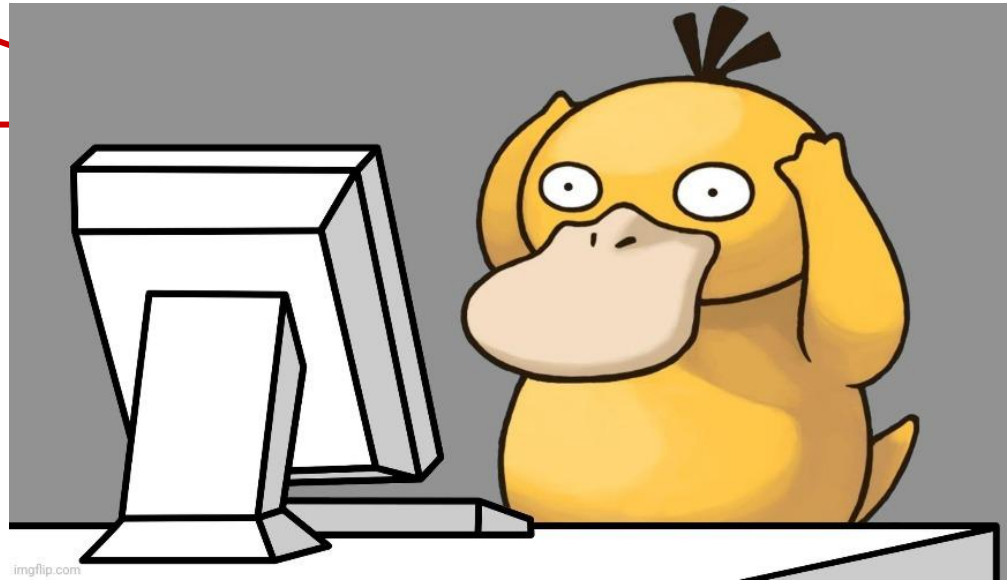
DAO Hack


entrypoint
withdraw
call
fallback function
update balance variable

DAO Hack

Root Cause Analysis





DAO Hack

"The DAO Hack - A $55 million heist that split Ethereum in two"

Controversial **hard-fork** reverted the hacker's transaction.

Fomo3D Hack

Fomo3D is a **gambling game** where players buy keys from a contract and their money goes into a pot.

Fomo3D Hack

Fomo3D is a **gambling game** where players buy keys from a contract and their money goes into a pot.

- A **time counter** is initiated, counting back from 24 hours.
- Buying a key **adds 30 seconds** to the counter.
- Key price increases every time a key is bought.
- When the **counter hits 0**, the last player wins the majority of the pot.

Fomo3D Hack

Find a way to "delay" other's transactions.

Block stuffing vulnerability!

Fomo3D Hack

Find a way to "delay" other's transactions.
Block stuffing vulnerability!

Submit transactions until the block gas limit is reached (~8,000,000 units).
Miners will happily prioritize your transactions.

Block	Date Time (UTC)	Txn	Uncles	Fee Recipient	Gas Used
6191909	2018-08-22 6:51:17	166	0	BitClubPool	7,971,592 (99.79%, +100%)
6191908	2018-08-22 6:50:57	5	0	PandaMiner	7,991,000 (99.94%, +100%)
6191907	2018-08-22 6:50:51	4	0	BitClubPool	7,979,000 (99.83%, +100%)
6191906	2018-08-22 6:50:45	3	0	Nanopool	8,000,000 (100.00%, +100%)
6191905	2018-08-22 6:50:36	7	0	MiningPoolHub: Old Add...	7,984,000 (99.80%, +100%)
6191904	2018-08-22 6:49:57	3	0	Nanopool	8,000,000 (100.00%, +100%)
6191903	2018-08-22 6:49:33	6	0	Ethermine	7,984,000 (99.80%, +100%)
6191902	2018-08-22 6:49:24	46	0	Ethermine	7,978,342 (99.83%, +100%)
6191901	2018-08-22 6:49:16	15	0	Spark Pool	7,979,663 (99.94%, +100%)
6191900	2018-08-22 6:49:07	10	0	Nanopool	7,979,192 (99.84%, +100%)
6191899	2018-08-22 6:48:58	34	0	0xd9580260be45c3c0c2...	7,975,461 (99.89%, +100%)
6191898	2018-08-22 6:48:43	25	0	Spark Pool	7,980,081 (99.99%, +100%)
6191897	2018-08-22 6:48:28	103	0	Bw Pool	3,648,328 (45.67%, -9%)

Fomo3D Hack

Was it worth it?

The attacker spent more than 80k units of gas worth of money.

Fomo3D Hack

Was it worth it?

The attacker spent more than 80k units of gas worth of money.
But gained much, much more Ether.

Closing Remarks

Closing Remarks

Dev

Sec

Closing Remarks

Dev

Sec

Avoid rapid prototyping

Closing Remarks

Dev

Avoid rapid prototyping
Careful design

Sec

Closing Remarks

Dev

Avoid rapid prototyping
Careful design
Testing before deploy!!!

Sec

Closing Remarks

Dev

Avoid rapid prototyping
Careful design
Testing before deploy!!!

Sec

Jump on the web3 🚂!

Closing Remarks

Dev

Avoid rapid prototyping
Careful design
Testing before deploy!!!

Sec

Jump on the web3 🚂!
Learn new stuff

Closing Remarks

Dev

Avoid rapid prototyping
Careful design
Testing before deploy!!!

Sec

Jump on the web3 🚂!
Learn new stuff
Find million $ bugs and 🍻