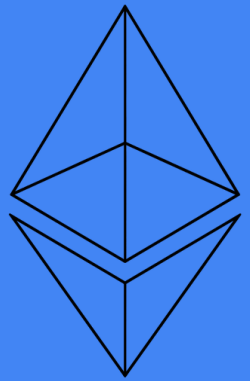
Solidity Security Best Practices

tl;dr Don’t be Parity



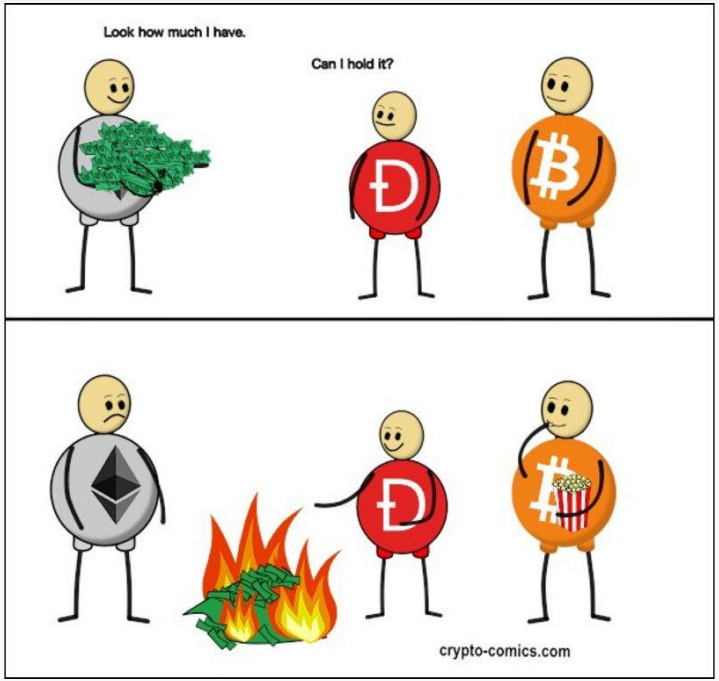
By: Hernando Castano
For the Waterloo Ethereum Developers Meetup
Nov 23, 2017

Topics

- Solidity Security
  - require() vs. assert()
  - Safe Math
  - Payments
  - Blockhashes
- Software Engineering Best Practices
  - Test Coverage
  - Standard Libraries
  - Upgradeable Smart Contracts
  - Bounties

Intro

- The actions we make have serious consequences
  - Not mission critical hardware code, but millions of dollars
- To grow the ecosystem we need trust from the general public
- How do you expect your mom to use the blockchain if it’s unsafe?



Solidity Security

Visibility Modifier: Private

```
// ANYONE CAN SEE THIS
bytes32 private privKey = "0xn0t4actua77yakey...";
```

require()

- Use to validate inputs
- Generally found at beginning of function
- Returns your gas
- Use it often!

```
function send(uint256 _amount) public {
  require(_amount < balances[msg.sender]);
  require(_amount < dailyLimit);
  // Send the funds
}
```

assert()

- Use to validate conditions that shouldn’t happen
- Use at the end of functions
- Will consume all your gas
- Doesn’t want to deal with your shit

```
// Stolen from Consensys Smart Contract Best Practices
function deposit(uint256 _amount) public payable {
  balances[msg.sender] += msg.value;
  totalSupply += msg.value;
  assert(this.balance >= totalSupply);
}
```

Safe Math

- General for max value is:
  - *uint<n>: $2^n - 1$*
- Examples:
  - *uint8: $2^8 - 1 = 255$*
  - *uint256: $2^{256} - 1$ = really big*
- It's important to watch these limits
  - assert() is good for this kind of stuff

```
function sneakyLoop(uint256 _length) public returns (uint256) {
  for (var i = 0; i < _length; i++) {
    // Do Some Action
  }
}
```

send() vs. transfer()

- Each only given a gas stipend of 2300 gas
  - Safe against reentrancy
  - Comes at the cost of failing if there are “complex” fallback functions

```
function giveToPoorFriend(address _to, uint256 _amount) public {
  require(msg.sender.balance > _to.balance);
  require(msg.sender.balance >= _amount);
  _to.transfer(_amount); // If this fails, it will throw
  // Or
  _to.send(_amount); // If this fails, it will return false
}
```

Pull vs. Push Payments

```
// This is a bad idea
function refundAll() public {
  // Pretend I have a list of attendees
  for (uint i = 0; i < attendees.length; i++) {
    if (attendees[i].attended) {
      // Refund the amount they deposited
      attendees[i].send(attendees.deposit);
    }
  }
}
```

- Meetup group now taking an ETH deposit
- Will refund deposit if you show up
- Problems ‘refundAll()’ function:
  - Can run out of gas
  - Overflow of the loop counter
  - One failed transaction can ruin everything

Pull vs. Push Payments

```
// In our case, this is the better way
function widthdraw() external {
  Attendee attendee = attendees[msg.sender];

  require(!attendee.gotRefund);
  require(attendee.paidDeposit);

  attendee.gotRefund = true;
  bool success = msg.sender.send(attendee.deposit);
  if (!success) {
    attendee.gotRefund = false;
  }
}
```

- This is where *Pull* payments come in handy
- Whoever made a deposit asks for it back
- If their transaction fails, at least it is isolated from the rest

Timestamps

- There is no central atomic clock on the Ethereum network
- We rely on miners to provide us with time
- Miners can influence *block.timestamp* (a.k.a *now*) by moving it forward a bit
- Dangerous if conditions depend on it being after a certain time
- Shouldn’t use as a source of randomness either

Blockhashes

- Like timestamps, can be manipulated by malicious miners to some degree
  - This means you shouldn't use them as a source of randomness
- From Solidity Documentation

Block and Transaction Properties

- `block.blockhash(uint blockNumber) returns (bytes32)` : hash of the given block - only works for 256 most recent blocks excluding current

- Key point: Can only access last **256** blocks!
  - Will return **0** otherwise

Blockhashes: How *Not* to Use Them

- Offered a 1500 ETH ($500k USD) bounty for anyone who could hack their smart contract
- Surprise surprise, it was hacked
- A hacker got away with 400 ETH ($120k USD) before *Smart Billions* widthdrew the rest of the contract funds

Overview | Event Logs | Comments

Transaction Information

Tools & Utilities

TxHash:	0x6c28b5058aabc4a1a900a5ac6ceaaa11b033dea35e73a4cbe9aba1d4ff4627db
Block Height:	4337096 (260652 block confirmations)
TimeStamp:	48 days 4 hrs ago (Oct-04-2017 08:25:31 PM +UTC)
From:	0x6245c1804f7fceb305a60bbb5cb6e18f939edb69
To:	Contract 0x5ace17f87c7391e5792a7683069a8025b83bbd85
Value:	0.01 Ether ($3.62)
Gas Limit:	200000
Gas Used By Txn:	123205
Gas Price:	0.000000021 Ether (21 Gwei)
Actual Tx Cost/Fee:	0.002587305 Ether ($0.94)
Cumulative Gas Used:	6573385
Nonce:	0
Input Data:	Function: playSystem(uint256 _hash, address _partner) MethodID: 0x26699576 [0]:0000000000000000000000000000000000000000000000000000000000000001 [1]:0000000000000000000000006245c1804f7fceb305a60bbb5cb6e18f939edb69

Convert To Ascii

Overview | Internal Transactions | Event Logs | Comments

Transaction Information

Tools & Utilities

TxHash:	0x09cd170f4f33497b91e8a29ad2da115acfbf09586099a4b56b5564473e8c7e01
Block Height:	4337369 (260379 block confirmations)
TimeStamp:	48 days 2 hrs ago (Oct-04-2017 10:57:00 PM +UTC)
From:	0x6245c1804f7fceb305a60bbb5cb6e18f939edb69
To:	Contract 0x5ace17f87c7391e5792a7683069a8025b83bbd85 TRANSFER 200 Ether to 0x6245c1804f7fceb305a60...
Value:	0 Ether ($0.00)
Gas Limit:	200000
Gas Used By Txn:	59233
Gas Price:	0.000000021 Ether (21 Gwei)
Actual Tx Cost/Fee:	0.001243893 Ether ($0.45)
Cumulative Gas Used:	6114959
Nonce:	1
Input Data:	Function: won() MethodID: 0x12c8052f Convert To Ascii

Software Engineering

Programmer

- Develops and Writes Code
- Build Products
- One Aspect of Software Development
- Force Pushes to *master*

Software Engineer

- Manages Project Requirements
- Designs System Architecture
- Tests Software
- Implements Development and Deployment Processes
- Makes Pull Requests

Test Coverage

- Aim for as close to 100% test coverage
- Suggestions of things to test
  - Return values of send() or call()
  - require() and assert()
  - Loops: are they throwing out of gas errors?
  - Fallback function that consumes more than 2300 gas

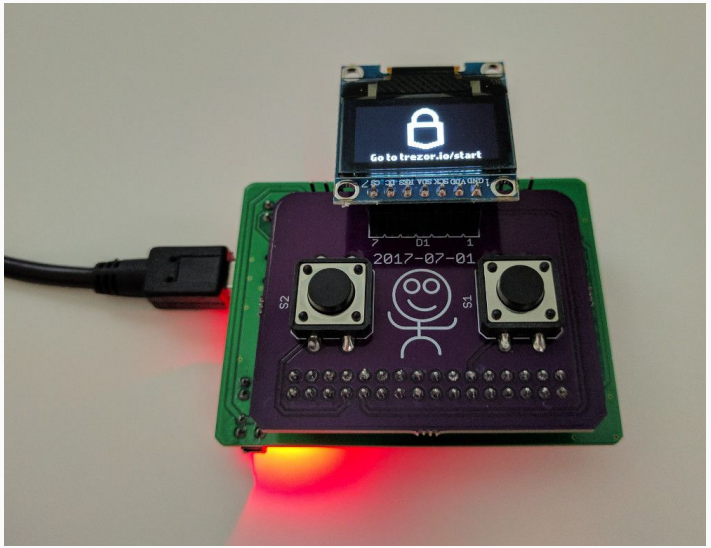
Use Well Audited Code

- Something something OpenZeppelin Libraries
- Includes things like:
  - SafeMath.sol
  - StandardToken.sol
  - Crowdsale.sol
  - Pausable.sol

```
pragma solidity ^0.4.18;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

Upgradeable Smart Contracts

- Once a contract is deployed on the network, it's up there forever
  - Can kinda think about it like a hardware device
- Very hard, if not impossible, to address bugs if precautions aren’t taken beforehand



Parity Multisig: July Hack

```
// constructor - just pass on the owner array to the multiowned and
// the limit to daylimit
function initWallet(address[] _owners, uint _required, uint _daylimit) {
  initDaylimit(_daylimit);
  initMultiowned(_owners, _required);
}

  // gets called when no other function matches
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);
}
```

- There was a hack in July, in which a hacker got away with ~$30M USD
- Whitehat group managed to save rest of vulnerable wallets
- tl;dr
  - initWallet() function in the WalletLibrary contract was called via fallback function
  - Hacker made themselves the sole owner of Multisig
  - Ran away with funds

Parity Multisig: devops199 Edition

anyone can kill your contract #6995

Open **devops199** opened this issue a day ago · 12 comments

devops199 commented a day ago • edited

I accidentally killed it.

https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

Overview | Comments

Transaction Information

TxHash:	0x05f71e1b2cb4f03e547739db15d080fd30c989eda04d37ce6264c5686e0722c9
TxReceipt Status:	Success
Block Height:	4501736 (102689 block confirmations)
TimeStamp:	16 days 12 hrs ago (Nov-06-2017 02:33:47 PM +UTC)
From:	0xae7168deb525862f4fee37d987a971b385b96952
To:	Contract 0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4
Value:	0 Ether ($0.00)
Gas Limit:	140701
Gas Used By Txn:	140701
Gas Price:	0.0000000136 Ether (13.6 Gwei)
Actual Tx Cost/Fee:	0.0019135336 Ether ($0.73)
Cumulative Gas Used:	3444021
Nonce:	88
Input Data:	Function: initWallet(address[] _owners, uint256 _required, uint256 _daylimit) MethodID: 0xe46dcfeb [0]:0000000000000000000000000000000000000000000000000000000000000060 [1]:0000000000000000000000000000000000000000000000000000000000000000 [2]:0000000000000000000000000000000000000000000000000000000000000000 [3]:0000000000000000000000000000000000000000000000000000000000000001 [4]:000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952 Convert To Ascii

Overview | Internal Transactions | Event Logs | Comments

Transaction Information

TxHash:	0x47f7cff7a5e671884629c93b368cb18f58a993f4b19c2a53a8662e3f1482f690
TxReceipt Status:	Success
Block Height:	4501969 (102456 block confirmations)
TimeStamp:	16 days 11 hrs ago (Nov-06-2017 03:25:21 PM +UTC)
From:	0xae7168deb525862f4fee37d987a971b385b96952
To:	Contract 0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4
Value:	0 Ether ($0.00)
Gas Limit:	108082
Gas Used By Txn:	69082
Gas Price:	0.0000000136 Ether (13.6 Gwei)
Actual Tx Cost/Fee:	0.0009395152 Ether ($0.36)
Cumulative Gas Used:	4750547
Nonce:	89
Input Data:	Function: kill(address _to) MethodID: 0xcbf0b0c0 [0]:000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952

Convert To Ascii

WHO WOULD WIN?

A well funded development team with over a dozen employees and years of development experience.

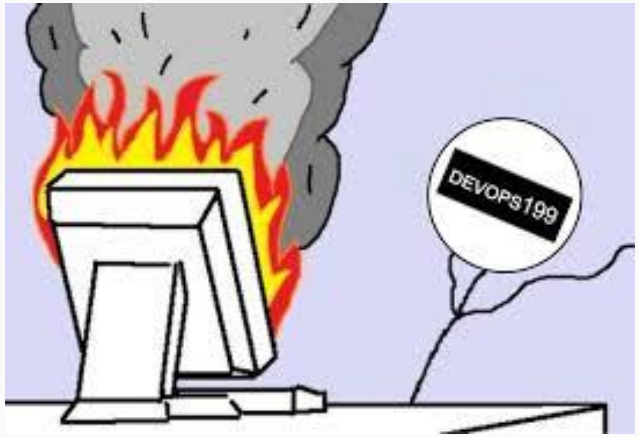
Gas Price:	0.0000000136 Ether (13.6 Gwei)
Actual Tx Cost/Fee:	0.0009395152 Ether ($0.27)
Cumulative Gas Used:	4750547
TxReceipt Status:	Success
Nonce:	89

Input Data:

```
Function: kill(address _to)

MethodID: 0xcbf0b0c0
[0]:000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

One 27 cent boi

Takeaways from devops199's Exploit



- Have a deployment checklist
  - Comprehensive list of steps to follow before deploying a piece of code
  - Great place to have things like "Remember to initialize contract"
- Have a way to mitigate damage after the fact
  - Upgradable smart contracts anyone?

Upgradeable Smart Contracts

This was line 451 of the smart contract:

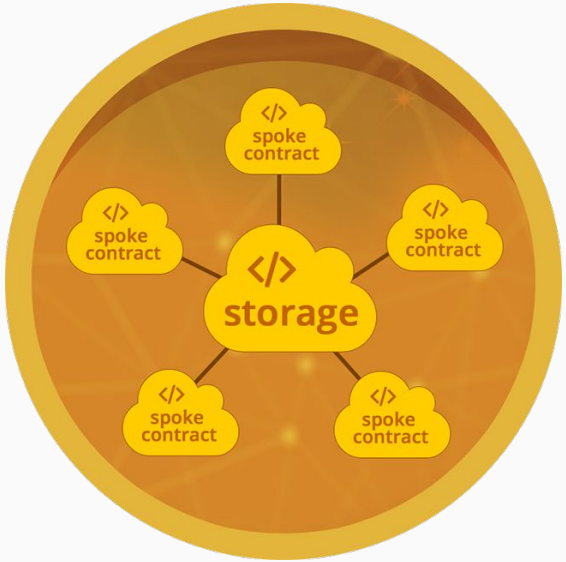
```
// FIELDS
address constant _walletLibrary = 0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4;
```

The address of the, now useless, contract was hardcoded in

```
// A Rocket Pool Suggested Solution
address _walletLibrary = 0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4;
/// @dev Set a new wallet library contract address
function setLibraryAddress(address _newAddress) external onlyOwner {
  _walletLibrary = _newAddress;
}
```

Rocket Pool's Approach: Hub and Spoke

- If you upgrade a contract, you MUST have a way to recover the data from the contract
- This is where the Hub comes in useful
  - Simple contract, only job is to handle information
  - This can include setters/getters, deletions, etc
  - Make sure to set up some sort of permission system
- Contracts can ask Hub for address of other contracts they interact with
  - Contracts can be changed, only Hub needs to know this happened



Bug Bounty Programs

Wrap Up

Wrap Up

- Always consider worst case scenarios
- Write simple smart contracts
- Be careful with timestamps and blockhashes
- Write meaningful tests
- Don’t roll your own Solidity
- Plan ahead
- Most importantly: Don’t be Parity

</presentation>