Smart Contracts (concluded)

Instructor: Matthew Green

Fall 2020

Many slides courtesy Andrew Miller

Some slides adapted from NBFMG

Housekeeping

- A2 is out
- I owe you another lecture this week
- Midterm out 10/27, due approx 1 day later
  - Multiple choice, not terribly long
  - "Take home", not in class
  - Based on reading and problem solving

News?

Today

- A bit more on Solidity and Ethereum smart contracts, then (assuming we have time) back to consensus and mining

- Some EVM/Solidity internals

- Next time(s): mining and attacks, anonymity and privacy

Contract programming model

```
1  pragma solidity ^0.4.19;
2  contract TestContract {
3
4      // declare storage
5      int[] myStorage1;
6
7      // define functions
8      function testFunction() {
9          // Code goes here
10     }
11
12     // ...
13 }
```

- Contract class

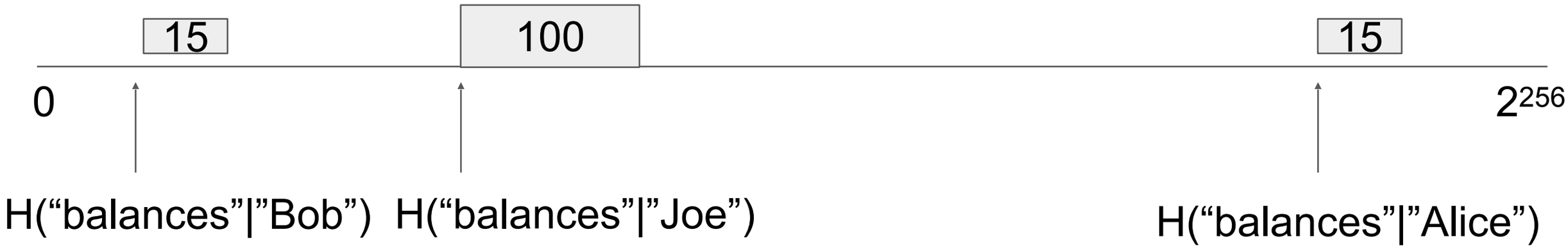
Create an object of this class by making a transaction

- Define functions you can call

Clever implementation of maps in Solidity

```
mapping(string => uint256) balances;
```

Alice	15
Bob	15
Joe	100



- every item requires at least one 256-bit word
- balances[“Andrew”] is 0 if “Andrew” doesn’t exist or if “Andrew” has 0 balance
- to delete a key, set balances[“Andrew’] = 0
- Cannot delete an entire map!

```
pragma solidity ^0.4.0;
contract TestMapping {
  mapping(uint => uint) aMap;
  function doit() { aMap[10] = 20; }
}
```

```
JUMPDEST            function doit() { aMap[10] = 2...
PUSH 14             20
PUSH 0              aMap
PUSH 0              aMap[10]
PUSH A              10
DUP2                aMap[10]
MSTORE              aMap[10]
PUSH 20             aMap[10]
ADD                 aMap[10]
SWAP1               aMap[10]
DUP2                aMap[10]
MSTORE              aMap[10]
PUSH 20             aMap[10]
ADD                 aMap[10]
PUSH 0              aMap[10]
SHA3                aMap[10]
DUP2                aMap[10] = 20
SWAP1               aMap[10] = 20
SSTORE              aMap[10] = 20
POP
```

Source: Derek Anderson, https://hackernoon.com/are-ethereum-contracts-vulnerable-to-hash-table-poisoning-attacks-a4d9241e16c4

```
pragma solidity ^0.4.0;
contract TestMapping {
  mapping(uint => uint) aMap;
  function doit() { aMap[10] = 20; }
}
```

```
JUMPDEST          function doit() { aMap[10] = 2...
PUSH 14           20
PUSH 0            aMap
PUSH 0            aMap[10]
PUSH A            10
DUP2              aMap[10]
MSTORE            aMap[10]
PUSH 20           aMap[10]
ADD               aMap[10]
SWAP1             aMap[10]
DUP2              aMap[10]
MSTORE            aMap[10]
PUSH 20           aMap[10]
ADD               aMap[10]
PUSH 0            aMap[10]
SHA3              aMap[10]
DUP2              aMap[10] = 20
SWAP1             aMap[10] = 20
SSTORE            aMap[10] = 20
POP
```

Stores at address SHA3(key)

Source: Derek Anderson, https://hackernoon.com/are-ethereum-contracts-vulnerable-to-hash-table-poisoning-attacks-a4d9241e16c4

EVM memory

- EVM uses virtual memory to instantiate a 2^256 sized memory space (each consisting of 256 bit cells)

- This doesn't really exist, it's virtualized and stored down into a "sparse" format (array)

- This is a little bit inefficient

Solidity gotchas (many more later)

- Member variables can be marked `public`
  - Getter methods automatically provided
- Functions must be marked `payable` to accept funds
- Member variables go to storage by default
  - Method variables go to memory
- Fallback `function()`
  - Called if no function specified (e.g. send)
  - Called if non-existent function called
- msg.sender vs. tx.origin

Gas in Ethereum is a necessary evil

- All miners and full nodes must evaluate all transactions
  - limit computation cost
- All miners must store all state
  - limit storage use
- Short-cut the halting problem
  - There is an upper GAS_LIMIT, so all programs will halt

Every EVM operation has a fixed gas cost

	opcodes	gas cost
Basic operations	ADD, MUL, PUSH, JUMP	2-10
Storage read	SLOAD	200
Storage write	SSTORE	5000
Storage write (from zero)	SSTORE	20000
Storage zeroize	SSTORE	-10000
Contract call	CALL, CODECALL, etc.	700
Transaction overhead	n/a	21000
Contract creation	n/a	32000
Contract destruction	SELFDESTRUCT	-19000

Callers can choose how much gas to send

100

A:

```
function a():
    Assert msg.gas ==
100;
x = B.b.gas(10)()
return x + " World!"
```

"Hello World!"

10

B:

```
function b() {
  assert msg.gas
== 10
  y = C.c.gas(5)
()

  require(y ==
0);
  // out of gas
  return "Hello"
```

"Hello"

5

C:

```
function c():
 assert msg.gas
== 5
while (true)
    Loop
}

return "Bonjour"
```

Out of gas

(pseudocode: exact syntax used here does not work in Solidity)

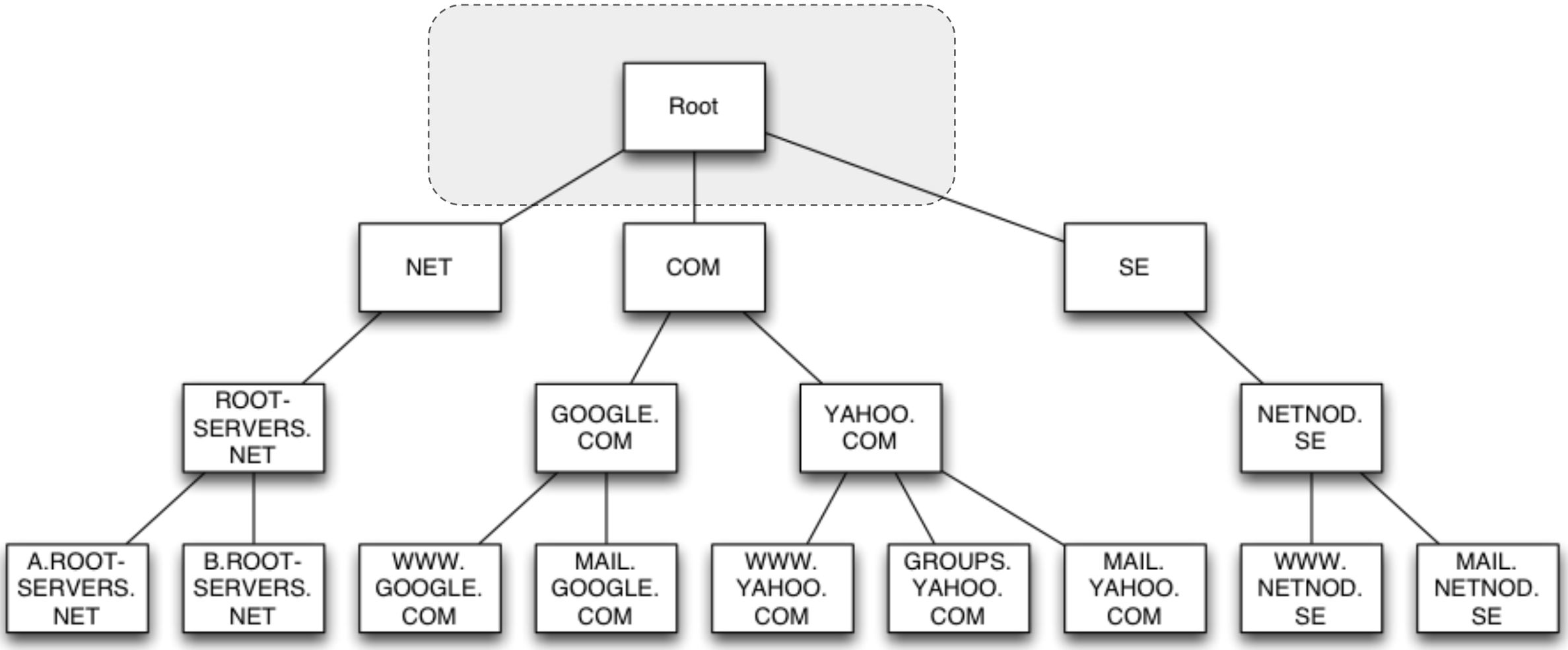
Key challenges in smart contract design:

- Smart contracts on public blockchains can be trusted for ***correctness*** and ***availability***, but not ***privacy***
- Blockchain resources are ***expensive***
- ***Uncertain delays***, and ***front running***
- On the blockchain, "***Code is law***"

Example:

Namecoin in Ethereum

What is the Domain Name System?

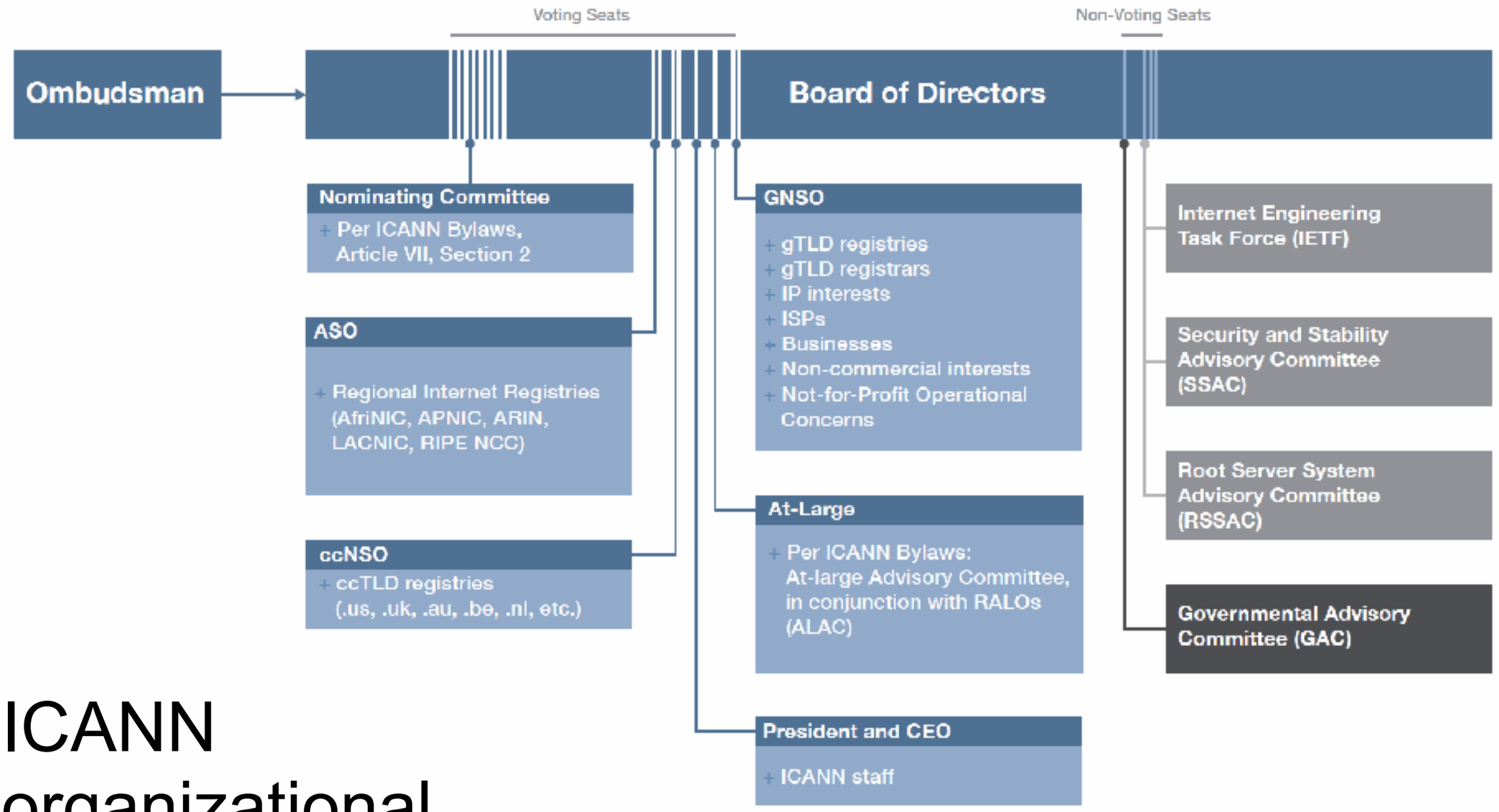


ICANN: Internet Corporation for Assigned Names and Numbers

List of Root Servers

HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

ICANN organizational structure



“Namecoin”: a simplistic DNS replacement

Initially, all names are unregistered.

Anyone can claim an unregistered name.

Once it’s registered, no one can change it.

Namecoin pseudocode

```
def register(k, v):

    if !self.storage[k]: # Is the key not yet taken?
        # Then take it!
        self.storage[k] = v

        return(1)

else:

    return(0) // Otherwise do nothing
```

```
contract Namespace {

    struct NameEntry {
        address owner;
        bytes32 value;
    }

    uint32 constant REGISTRATION_COST = 100;
    uint32 constant UPDATE_COST = 10;
    mapping(bytes32 => NameEntry) data;

    function nameNew(bytes32 hash){
        if (msg.value >= REGISTRATION_COST){
            data[hash].owner = msg.sender;
        }
    }

    function nameUpdate(bytes32 name, bytes32 newValue, address newOwner){
        bytes32 hash = sha3(name);
        if (data[hash].owner == msg.sender && msg.value >= UPDATE_COST){
            data[hash].value = newValue;
            if (newOwner != 0){
                data[hash].owner = newOwner;
            }
        }
    }

    function nameLookup(bytes32 name){
        return data[sha3(name)];
    }

}
```

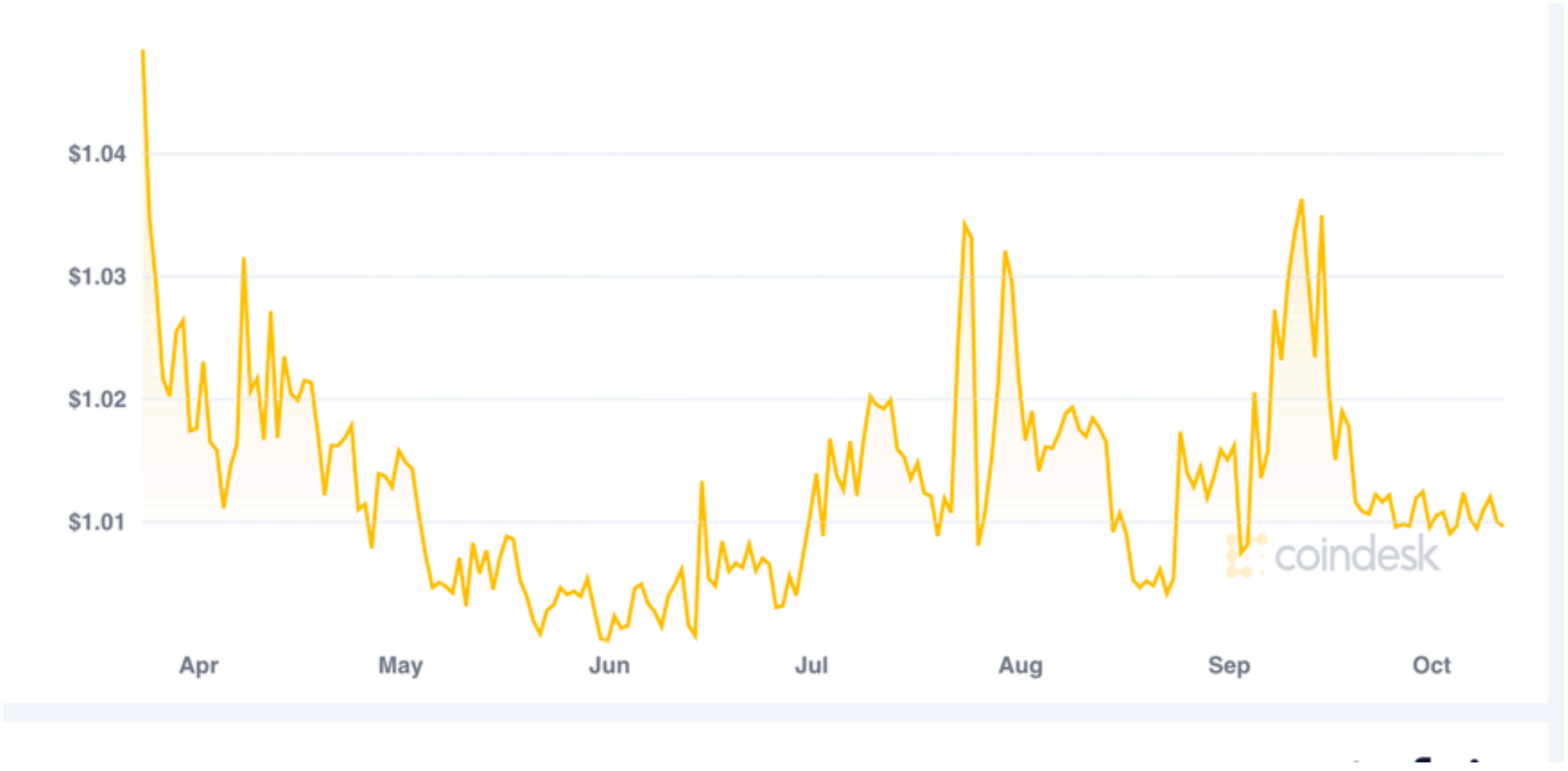
Economics of gas are similar to transaction fees

- Miners choose transactions based on GAS_PRICE
- In theory, they should not care which opcodes are used
  - In practice, some “overpriced” opcodes may be preferred
  - Remember that miners can do anything!
- Maximum gas limit per block
  - Miners can slowly raise it, each block votes

Applications

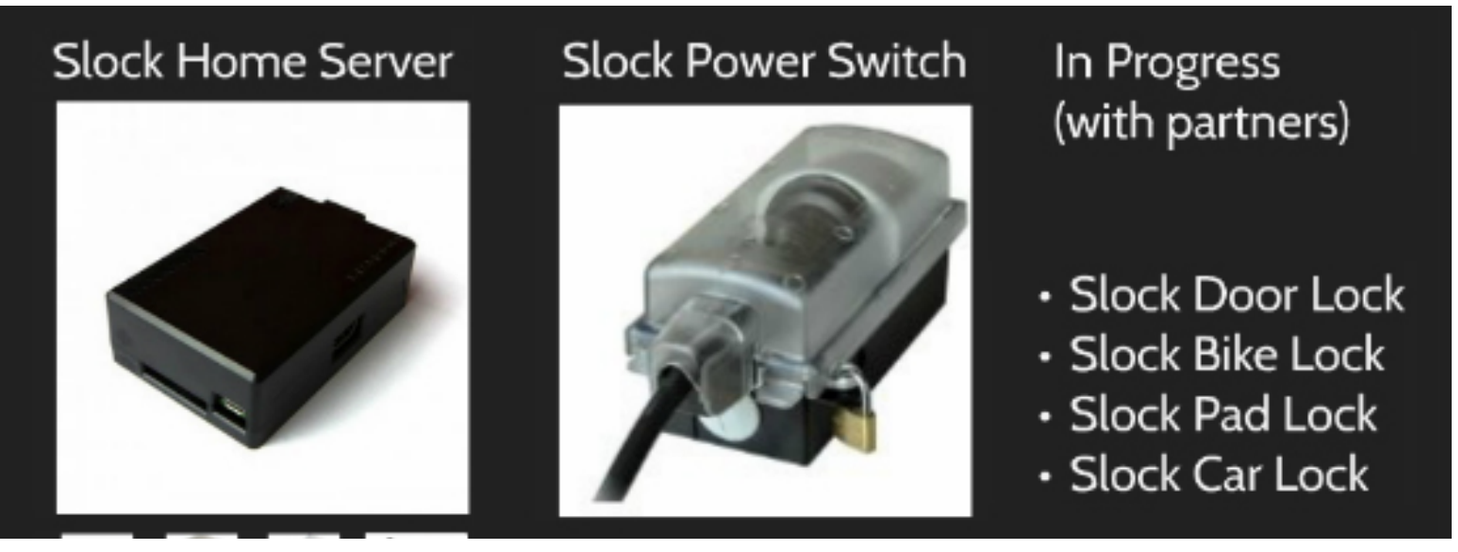
- Stablecoins (e.g., DAI)
- Decentralized Exchanges
- DAOs
- Multisig wallets

DAI



All about TheDAO

slock.it a Blockchain + IoT company



Example use case:

1. AirBnB user submits payment to the Ethereum blockchain

2. Slock Home Server (Ethereum client) receives the transaction

3. Power switch connected to Home Server receives “unlock” command, unlocks the door

slock.it built The DAO as a custom fundraising tool

“DAO”: Decentralized Autonomous Organization (coined by Vitalik in 2013)

Built by slock.it to raise funds for their company

Main idea: A decentralized hedge fund

- Investors contribute funds, receive ownership “tokens”
- Investors jointly decide how to spend funds, by voting in proportion to tokens

Many additional mechanisms:

- “Splitting” to prevent hostile takeover
- Reward disbursing


DAOs, Democracy and Governance

by Ralph C. Merkle, merkle@merkle.com

Version 1.9, May 31st 2016


DAO
Smart Contract
tasks & orders
Service provider
Slock Token
Reward
Vote
Tasks:
• Fund the development
• Vote on major decisions
• Control the funds (!)
• Profitable
Reward
• % fee of every transaction
• one time deployment fee
Tasks:
• Produce Slocks
• Marketing
• Partnerships
Slock Home Server
Slock Power Switch
In Progress
(with partners)
• Slock Door Lock
• Slock Bike Lock
• Slock Pad Lock
• Slock Car Lock
supports:
• Z-Wave
• Zigbee
• Bluetooth LE
intel inside

Raised ~150 million dollars in ~ 1 month

A Call for a Temporary Moratorium on The DAO

ethereum dao smart contracts

Dino Mark, Vlad Zamfir, and Emin Gün Sirer

May 27, 2016 at 01:35 PM

Understanding The DAO Attack

David Siegel

Jun 25, 2016 at 16:00 UTC | Updated Jun 27, 2016 at 17:52 UTC

The Hack

Unfortunately, while programmers were working on fixing this and other problems, an unknown attacker began using this approach to start draining The DAO of ether collected from the sale of its tokens.

By Saturday, 18th June, the attacker managed to drain more than 3.6m ether into a "child DAO" that has the same structure as The DAO. The price of ether dropped from over $20 to under $13.

Excerpt from DAO contract (simplified)

```
contract DAO:

mapping (address => int64)
balances;

function withdraw(uint x) {
 if (balances[msg.sender] >=
x) {
msg.sender.call.value(balanc
e)();
   balances[msg.sender] -=
x;
 }
}
```

Re-entrancy hazards in Ethereum

Ether Balance balances[user] 200 0

Balance 100

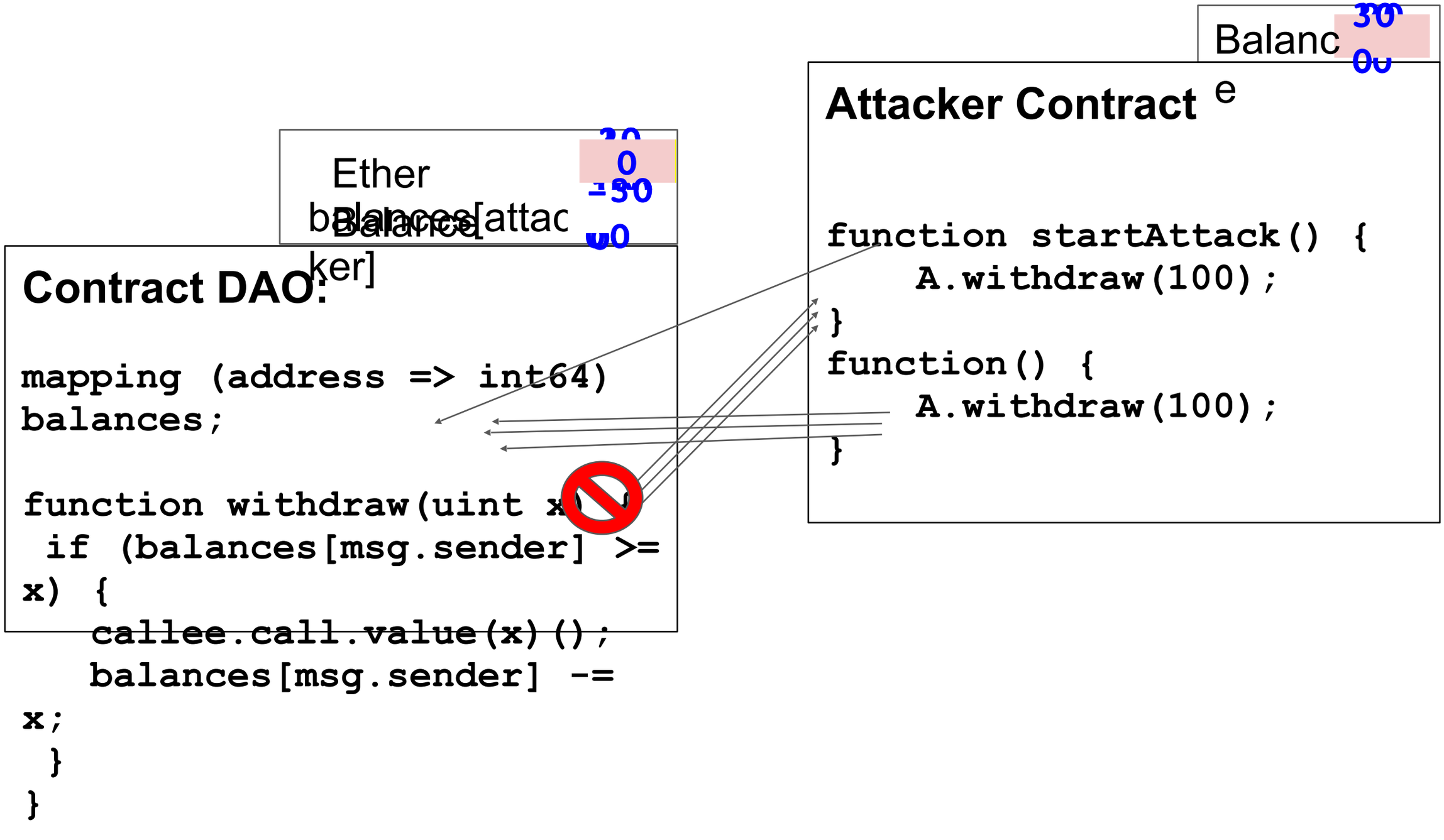
Contract DAO:

```
mapping (address => int64)
balances;

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

Wallet Contract

```
function doWithdraw() {
     A.withdraw(100);
}
function() {

EventMoneyReceived(msg.val
ue);
}
```

Re-entrancy hazards in Ethereum



Ethical Theories

Utilitarianism: bring the most benefit to the most people

Weigh the tradeoffs, try to achieve the best outcome

Challenges: incomparable outcomes (trolley problem)? Uncertain outcomes?

Deontological: having an ethical code, and a duty to follow it

Do unto others as like you wish to be treated

Challenges: how do you choose your code?

Ethereum Developers Launch White Hat Counter-Attack on The DAO

Stan Higgins

Jun 21, 2016 at 20:14 UTC | Updated Jun 22, 2016 at 14:21 UTC

https://etherscan.io/address/0xb136707642a4ea12fb4bae820f03d2562ebff487

NEWS

Reports are emerging that members of the ethereum development community are moving funds from The DAO in attempt to stifle a new alleged attack.

Developer Alex Van de Sande, lead designer for the Ethereum Foundation, took to Twitter to announce the move, which came shortly after word emerged on social media that more funds were being siphoned from contracts associated with The DAO. He said that the action is a response to a new exploitation of The DAO's smart contract code, which comes days after millions of dollars worth of ether were taken from contracts associated with project.

The address being used by Ethereum's developers can be found here, and at press time, it had amassed more than 4m ethers, worth approximately $48m.

Timeline and Aftermath of The DAO Attack

- June 12: slock.it developers announce that a bug is found, but no funds at risk
- June 17 (Morning): attacker drains ⅓ of the DAO's Ether ($50M) over 24 hrs

Attacker's funds were trapped in a subcontract for ***40 days*** (July 27)

- June 17 (Evening): Eth Foundation proposes a "Soft Fork" to freeze the funds
- June 28: researchers publish a flaw in the Soft Fork Proposal
- July 15 (Morning): Eth Foundation proposes a "Hard Fork" to recover funds
- July 15 (Evening): "Ethereum Classic" manifesto published on github
- July 19: "Hard Fork" moves funds from attacker's contract to recovery contract

Ethereum Classic blockchain survives and is traded on exchanges

Both Ethereum and Ethereum Classic are both around, reached new peaks

SECURITIES AND EXCHANGE COMMISSION

SECURITIES EXCHANGE ACT OF 1934

Release No. 81207 / July 25, 2017

Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: Thc DAO

I. Introduction and Summary

The United States Securities and Exchange Commission's ("Commission") Division of Enforcement ("Division") has investigated whether The DAO, an unincorporated organization; Slock.it UG ("Slock.it"), a German corporation; Slock.it's co-founders; and intermediaries may have violated the federal securities laws. The Commission has determined not to pursue an enforcement action in this matter based on the conduct and activities known to the Commission at this timc.

https://www.sec.gov/litigation/investreport/34-81207.pdf

The Parity wallet: Two massive failures in a row

- In July 2017, an exploitable vulnerability was exploited and $30M

153,037 Ether (worth $30+ million) was stolen from three wallet contracts

A remaining $78 million worth of tokens (half the value being BAT and ICONOMI) plus 377,105+ ETH (around $72 million) were recovered by a daring whitehat rescue, and returned to their *rightful* owners

- In November 2017, a second bug was triggered, leading to $150M frozen funds.

Parity - The Rust Ethereum node

Parity is the Rust-Ethereum client.

Separate core development team to Ethereum Foundation.

Currently around 20% of the nodes (we think actually 6%)

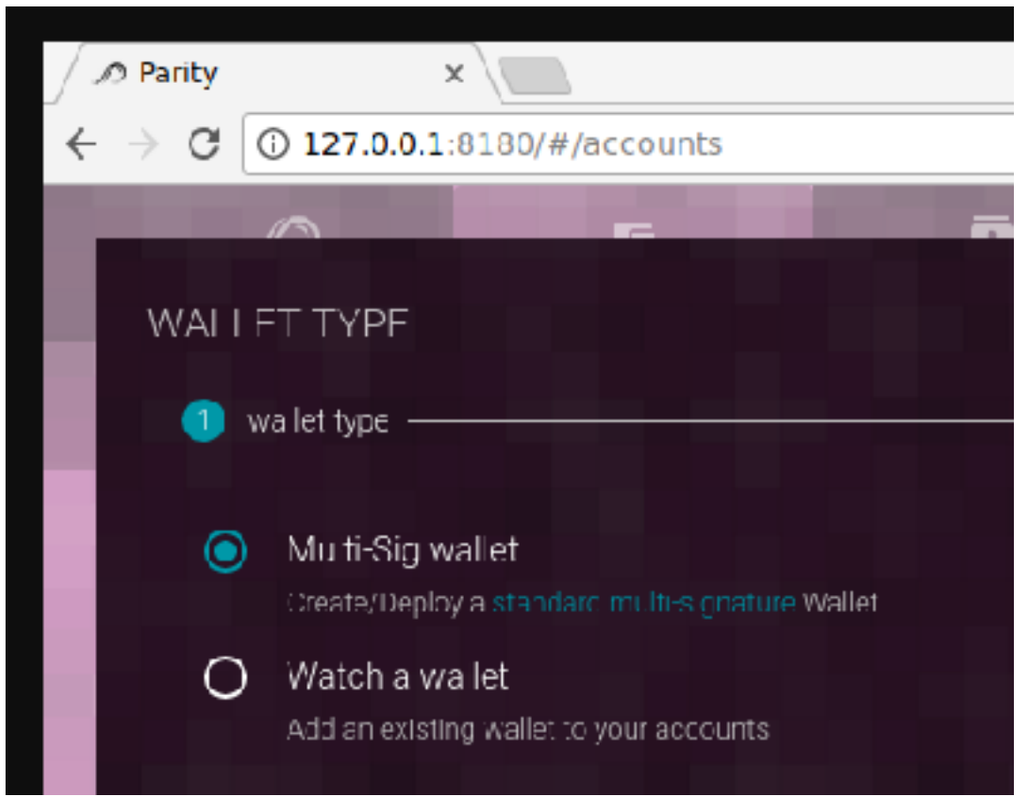
What is the Parity Wallet?

“Parity Wallet” is a Solidity smart contract that Parity can create as part of its wallet management feature.

It is a popular feature of Parity, though separate its “full node” function

https://github.com/paritytech/parity/wiki/Parity-Wallet

https://paritytech.io/blog/security-alert.html



Parity Wallet uses the “Prototype Inheritance” pattern

```
contract Wallet {
   address _walletLibrary;
   address owner;

   function Wallet(address _owner) {
       _walletLibrary = ${hardcoded
address};
        _walletLibrary.delegatecall(
          "initWallet(address)",
          _owner);
   }
   function withdraw(uint amount)
      returns (bool success) {
        return _walletLibrary.delegatecall(
           "withdraw(uint)",
            amount);
   }
    // fallback function gets called if no
    // other function matches
   function () payable {

_walletLibrary.delegatecall(msg.data);
   }
}
```

Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

[0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3](0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3)

Multiple Wallet contracts, all refer to Wallet Library

Why? Reduces fees from duplicate data!

* Toy version of code

1. Constructor is called, invokes **`initWallet`**

Wallet contract:

```
contract Wallet {
  address _walletLibrary;
  address owner;

  function Wallet(address _owner) {
      _walletLibrary = ${hardcoded address};
       _walletLibrary.delegatecall(
         "initWallet(address)",
         _owner);
  }
  function withdraw(uint amount)
     returns (bool success) {
       return _walletLibrary.delegatecall(
          "withdraw(uint)",
           amount);
  }
   // fallback function gets called if no
   // other function matches
  function () payable {

_walletLibrary.delegatecall(msg.data);
  }
}
```

Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3

2. The withdraw function invokes **`withdraw`**

Wallet contract:

```
contract Wallet {
    address _walletLibrary;
    address owner;

    function Wallet(address _owner) {
        _walletLibrary = ${hardcoded
address};

 _walletLibrary.delegatecall(
          "initWallet(address)",
          _owner);
    }
    function withdraw(uint amount)
returns (bool success) {
        return
_walletLibrary.delegatecall(
          "withdraw(uint)",
          amount);
    }
}
```

Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address _owner) {
         owner = _owner;
         // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
         // ... receive money, log events,
...
     }
     function withdraw(uint amount);
}
```

[0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3]

3. The fallback function can invoke ***any method***

Wallet contract:

```
contract Wallet {
   address _walletLibrary;
   address owner;

   function Wallet(address _owner) {
       _walletLibrary = ${hardcoded
address};
        _walletLibrary.delegatecall(
          "initWallet(address)",
          _owner);
   }
   function withdraw(uint amount)
      returns (bool success) {
        return _walletLibrary.delegatecall(
           "withdraw(uint)",
            amount);
   }
    // fallback function gets called if no
    // other function matches
   function () payable {

_walletLibrary.delegatecall(msg.data);
   }
}
```

Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3

Attacker calls:

victim.call("initWallet(address)", attacker)

Fixing the July 2017 Parity wallet bug

Old Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3

New Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          require(initialized == false);
          initialized = true;
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

The White Hat Recovery Team

A white-hat recovery team (MEH-WH) developers identified and drained all remaining vulnerable wallets into this wallet. They recovered a total of $78 million worth of tokens (half the value being BAT and ICONOMI) plus 377,105+ ETH (around $72 million). The funds will be returned to their owners as noted on r/ethereum:

Jordi Baylina
@jbaylina

Blockchain developer, Ethereum fan, white hat hacker and Catalonia Freedom fighter.

Overview | JordiBaylina (WHG)

ETH Balance:	0.11961849179736130 6 Ether
ETH USD Value:	$56.47 (@ $472.10/ETH)
No Of Transactions:	4509 txns

WHG **A Modified Version of a Common Multisig Had A Vulnerability - The WHG Took Action & Will Return the Funds** self.ethereum

Submitted 4 months ago * (last edited 4 months ago) by baylina

The White Hat Group were made aware of a vulnerability in a specific version of a commonly used multisig contract. This vulnerability was trivial to execute, so they took the necessary action to drain every vulnerable multisig they could find as quickly as possible.
Thank you to the greater Ethereum Community that helped finding these vulnerable contracts.

The White Hat account currently holding the rescued funds is
https://etherscan.io/address/0x1dba1131000664b884a1ba238464159892252d3a[1]

If you hold a multisig contract that was drained, please be patient. We will be creating another multisig for you that has the same settings as your old multisig but with the vulnerability removed and we will return your funds to you there. We will be using the donations sent to us from The DAO Rescue to pay for gas.

Effectively we will upgrade your multisig contract for you, all you will have to do is, be patient, find your new multisig address once we have finished, and it will be like nothing happened.

We will not be responding to any social media posts.

Edit: Do not trust any address posted below as a "donation address." There are a lot of phishers in the community right now. In general always verify any address or link you find on reddit.

125 comments source share save hide give gold report crosspost hide all child comments

The Parity Wallet July ‘17 stolen coins are still idle

Address 0xB3764761E297D6f121e79C32A65829Cd1dDb4D32

Sponsored Link: **Simple Token** - *Cryptocurrency for digital communities* - 105% of target hit, **Sale ends 1 Dec!**

Public Note: There are reports that funds were maliciously diverted to this account by the MultiSig Blackhat Exploiters.

Overview | MultisigExploit-Hacker	
ETH Balance:	83,017.074022098 Ether
ETH USD Value:	$39,286,169.94 (@ $473.23/ETH)
No Of Transactions:	30 txns

Misc

Address Watch

Token Tracker

Root Cause and Post Mortem from Parity

- An “initialize()” function to set the owner of the contract, was missing the “onlyOnce” modifier

- Process error more than anything -> mistriaged as “UX upgrade,” skipped review

In depth analysis of the event from IC3:

http://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/


devcon three
Ethereum Foundation Developers Conference
Nov 1-4, 2017 Cancún, Mexico
status
Microsoft
Bitcoin.com
omisego
WANXIANG BLOCKCHAIN LABS
santiment
FUNFAIR

DEVCON3
CANCUN

anyone can kill your contract #6995

Open **devops199** opened this issue 22 hours ago · 12 comments

devops199 commented 22 hours ago • edited

I accidentally killed it.

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

Hello, first of all i'm not the owner of that contract. I was able to make myself the owner of that contract because its uninitialized.

These (https://pastebin.com/ejakDR1f) multi_sig wallets deployed using Parity were using the library located at "0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4" address. I made myself the owner of "0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4" contract and killed it and now when i query the dependent contracts "isowner(<any_addr>)" they all return TRUE because the delegate call made to a died contract.

I believe some one might exploit.

The WalletLibrary contract

anyone can kill your contract #6995

Parity wallet uses “Prototype Inheritance”
The WalletLibrary should only act as a template.
- It should not have any “state”
- Its constructor has never been called

But, the WalletLibrary is actually just a contract!

Attacker calls:

```
library.call(“initWallet(address)”,
             attacker);
library.kill();
```

Thereafter, any method call to any instance “Wallet” will fail, since the target of delegatecall is destroyed

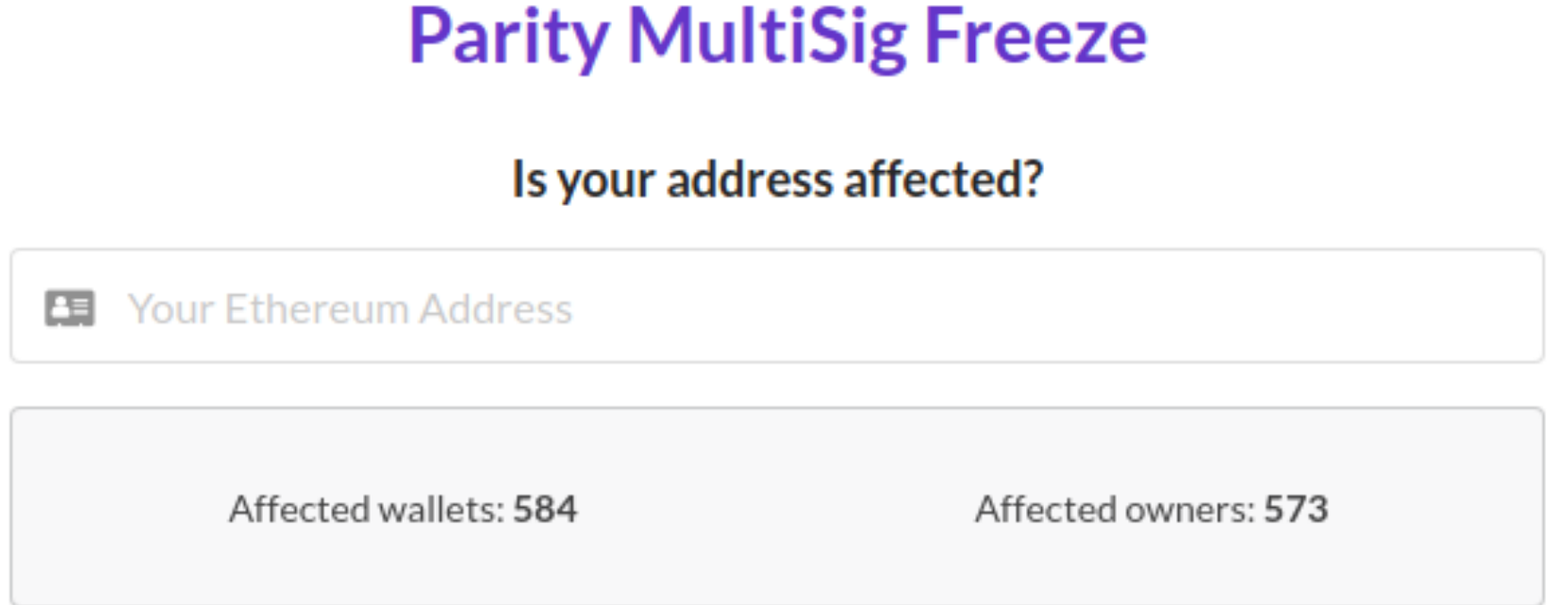
Wallet Library contract:

```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address
_owner) {
          require(initialized == false);
          initialized = true;
          owner = _owner;
          // ... more setup ...
     }
     function changeOwner(address
_new_owner)
     function () payable {
          // ... receive money, log
events, ...
     }
     function withdraw(uint amount);
}
```

0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

After a developer stumbled upon the bug – "accidentally" deleting a code library for the Parity wallet and freezing more than $152 million-worth of ether – several startups and open-source projects that recently launched initial coin offerings (ICOs) have come forward, stating that theirs are among the 151 addresses impacted by the software failure.

> According to crypto eli5, 151 wallets have been frozen, with their balances being 513,743 ETH or $152 million in total. Parity Technologies announce that 573 wallets have been affected and their total balance is unknown.



No one knows who devops199 was!

Tienus @Tienus
@devops199 you are the one that called the kill tx?

devops199 @devops199
yes

i'm eth newbie..just learning

qx133 @qx133
you are famous now haha

devops199 @devops199
sending kill() destroy() to random contracts

you can see my history

😟((((((((((((((((((((((((((((

Deleted user
ghost

Thinking like an attacker, “The Security Mindset”

What’s the worst thing that can happen? Can you make it break?

“The lack of a security mindset explains a lot of bad security out there”

https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html

Ethereum is a Dark Forest

By Dan Robinson and Georgios Konstantopoulos

Vulnerability Disclosure process

- What do you do when you find a bug?

- Give advance notice to those in position (or with a duty) to help

Why? Asymmetric advantage

(a standard amount is 90 days)

- How should companies react to researchers?

- Be proactive: build bug bounties, define disclosure processes

- Retaliate? (e.g., by banning or suing)

Bug Bounty Programs

ETHEREUM Bounty Program

So, what should a good vulnerability submission look like?

Here is an example of a real issue which was previously present in the Go client:

> **Description:** Remote Denial-of-service using non-validated blocks
>
> **Attack scenario:** An attacker can send blocks that may require a high amount of computation (the maximum gasLimit) but has no proof-of-work. If the attacker sends blocks continuously, the attacker may force the victim node to 100% CPU utilization.
>
> **Impact:** An attacker can abuse CPU utilization on remote nodes, possibly causing full DoS.
>
> **Components:** Go client version v0.6.8
>
> **Reproduction:** Send a block to a Go node that contains many txs but no valid PoW.
>
> **Details:** Blocks are validated in the method `Process(Block, dontReact)`. This method performs expensive CPU-intensive tasks, such as executing transactions (`sm.ApplyDiff`) and afterward it verifies the proof-of-work (`sm.ValidateBlock()`). This allows an attacker to send blocks that may require a high amount of computation (the maximum `gasLimit`) but has no proof-of-work. If the attacker sends blocks continuously, the attacker may force the victim node to 100% CPU utilization.
>
> **Fix:** Invert the order of the checks.

Key challenges in smart contract design:

- Smart contracts on public blockchains can be trusted for ***correctness*** and ***availability***, but not ***privacy***
- Blockchain resources are ***expensive***
- ***Uncertain delays***, and ***front running***
- On the blockchain, "***Code is law***"