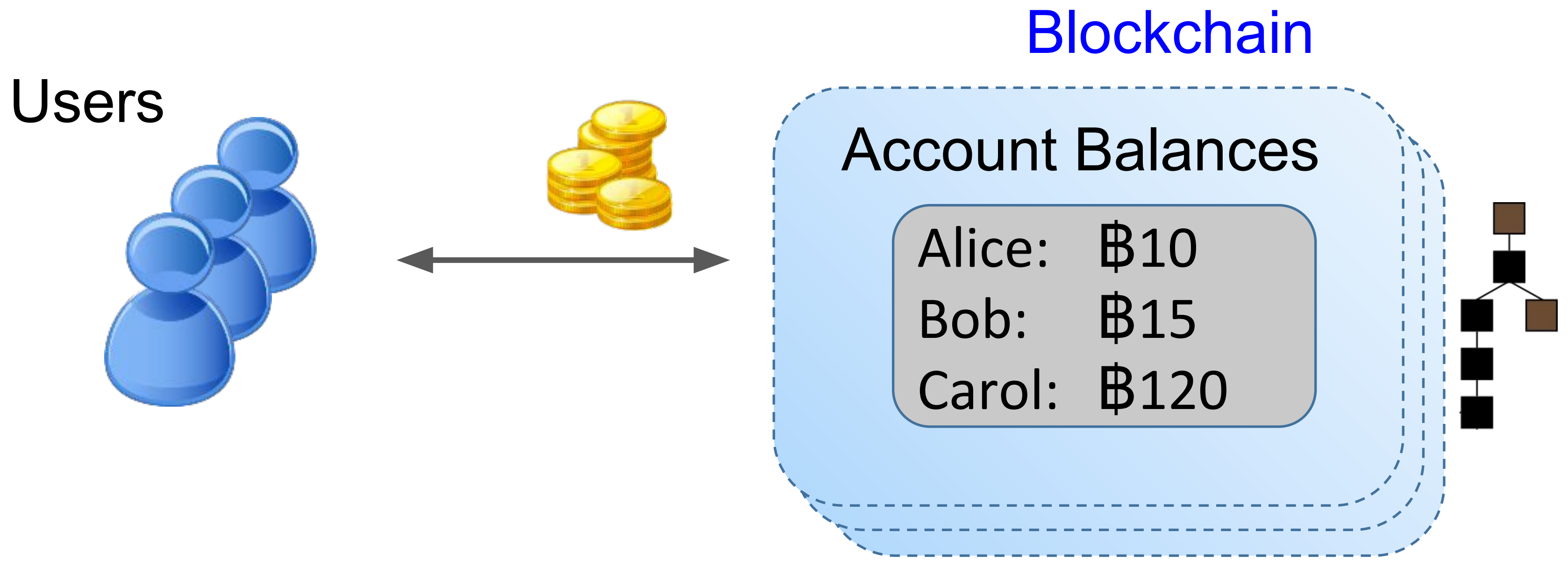
Smart Contracts and Opportunities for Formal Methods

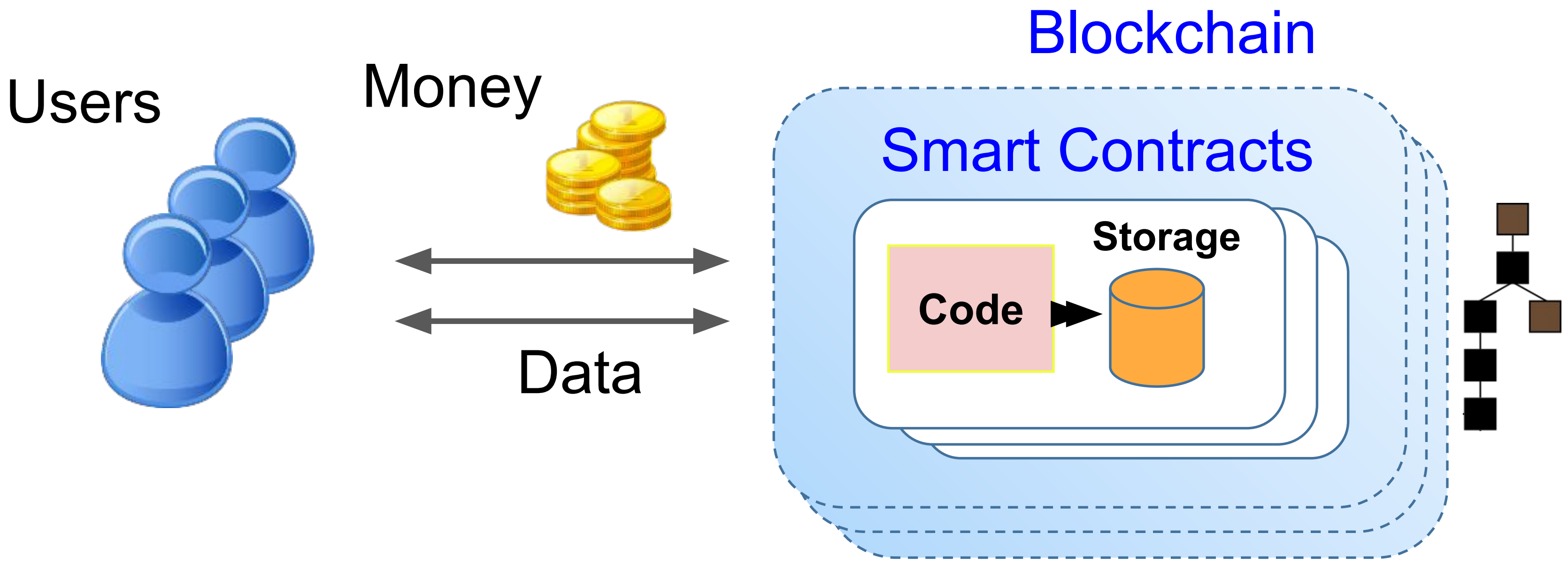
Andrew Miller[1], Zhicheng Cai[2], **Somesh Jha**[2]

[1] University of Illinois at Urbana-Champaign
[2] University of Wisconsin Madison

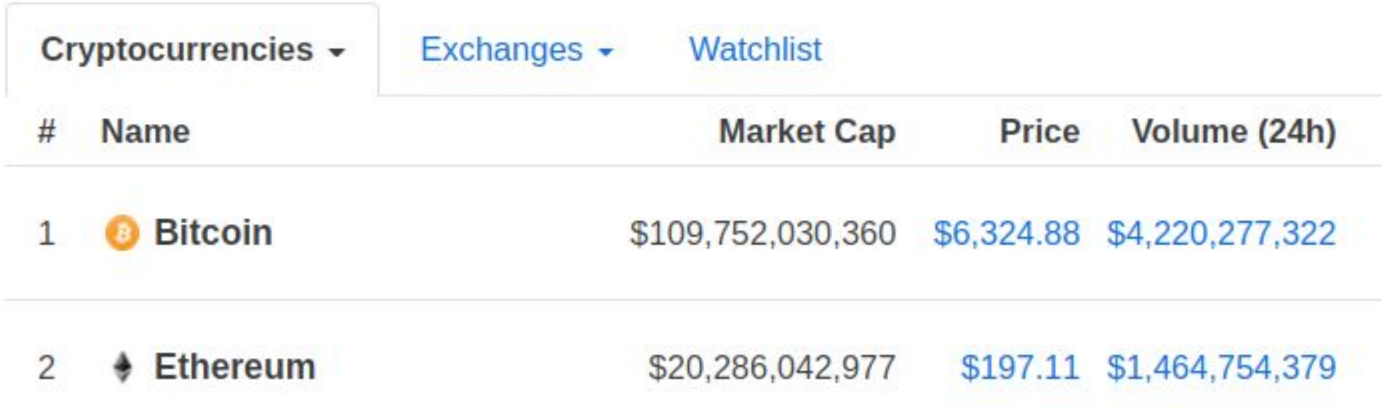
Blockchain: a shared database implemented on top of a peer-to-peer network, usually implementing a digital currency.



Digital currency is just one application
Smart contracts: programs running on a blockchain



Ethereum: the 2nd largest cryptocurrency,
features a general smart contract programing language (Solidity)



#	Name	Market Cap	Price	Volume (24h)
1	Bitcoin	$109,752,030,360	$6,324.88	$4,220,277,322
2	Ethereum	$20,286,042,977	$197.11	$1,464,754,379

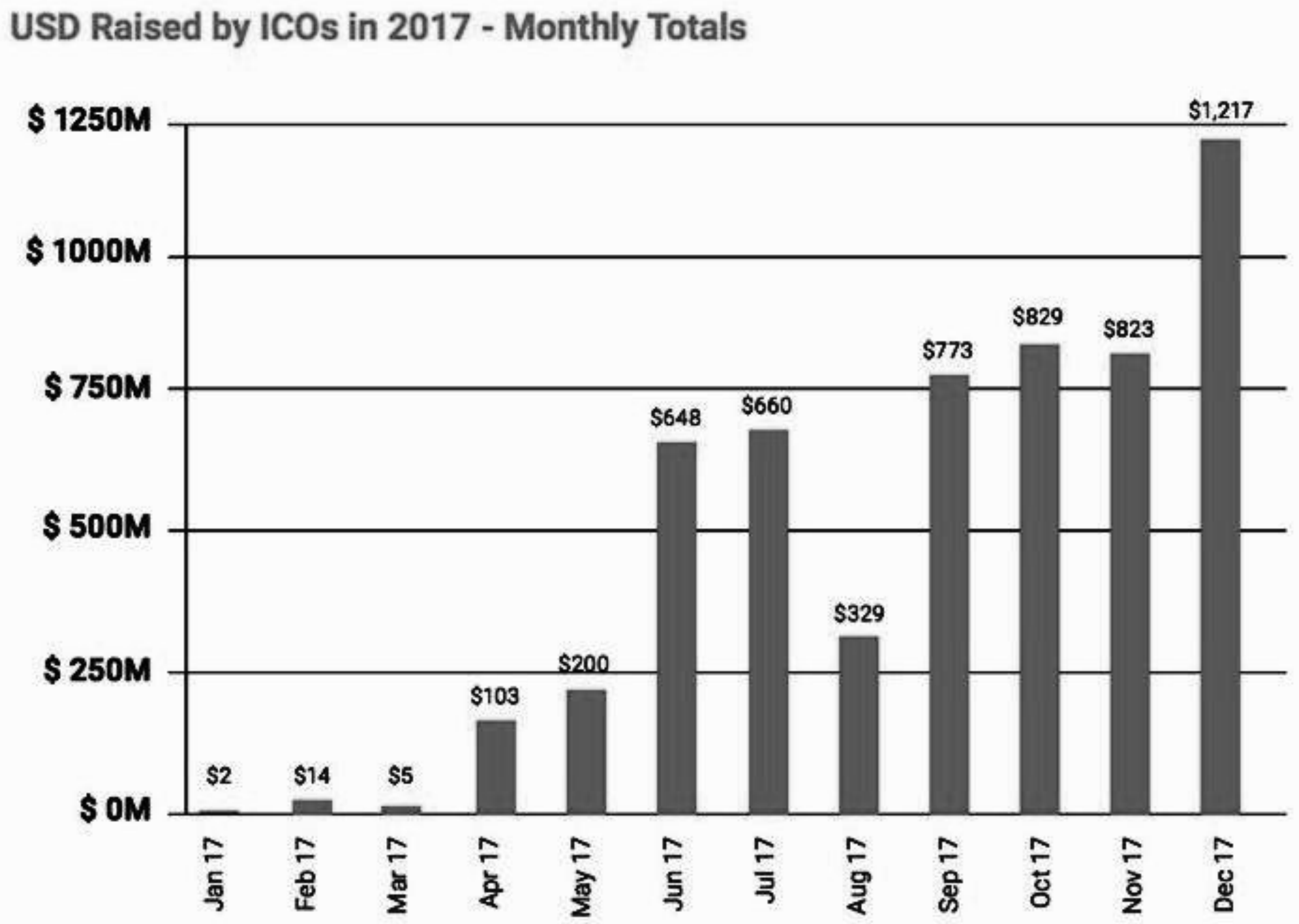
Solidity example: an ERC20 token

```
contract Token {
    mapping (address => uint) balance;

    function transfer(address to,
                      uint    amount) {
        require (balance[msg.sender] >= amount);
        balance[msg.sender] -= amount;
        balance[to] += amount;
    }
    ...
}
```

Ethereum Tokens Market Capitalization

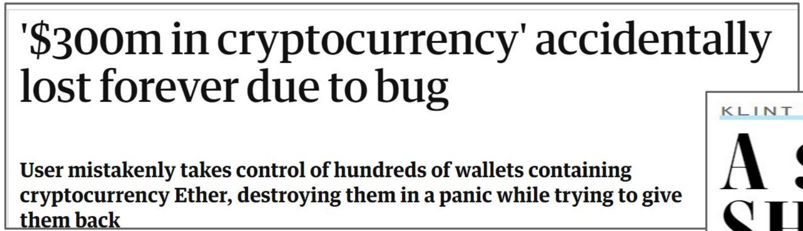
There are a total of 38562 Erc20 Token Contracts

Total Market Cap: $48,943,200,901



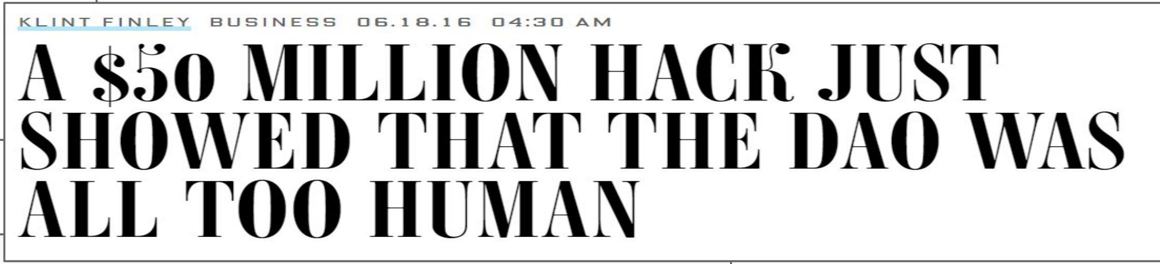
Source: Business insider

Ethereum’s brief history has been marred by high-profile disasters caused by buggy and insecure smart contract programs



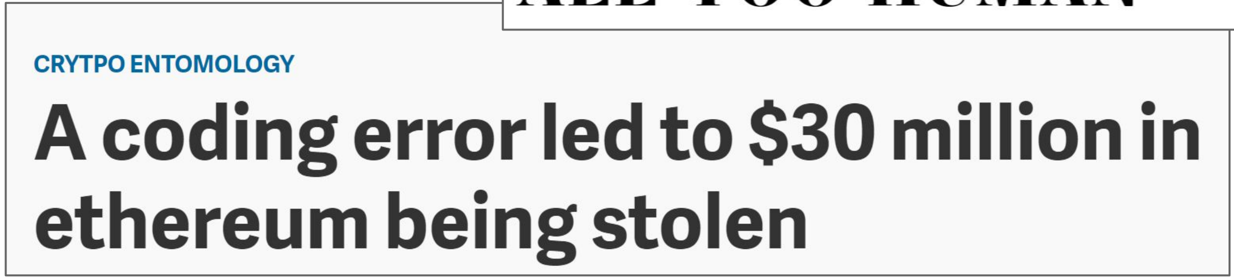
'$300m in cryptocurrency' accidentally lost forever due to bug

User mistakenly takes control of hundreds of wallets containing cryptocurrency Ether, destroying them in a panic while trying to give them back



KLINT FINLEY BUSINESS 06.18.16 04:30 AM

A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN



CRYTPO ENTOMOLOGY

A coding error led to $30 million in ethereum being stolen

Formal Methods are seen as a key part of the solution going forward

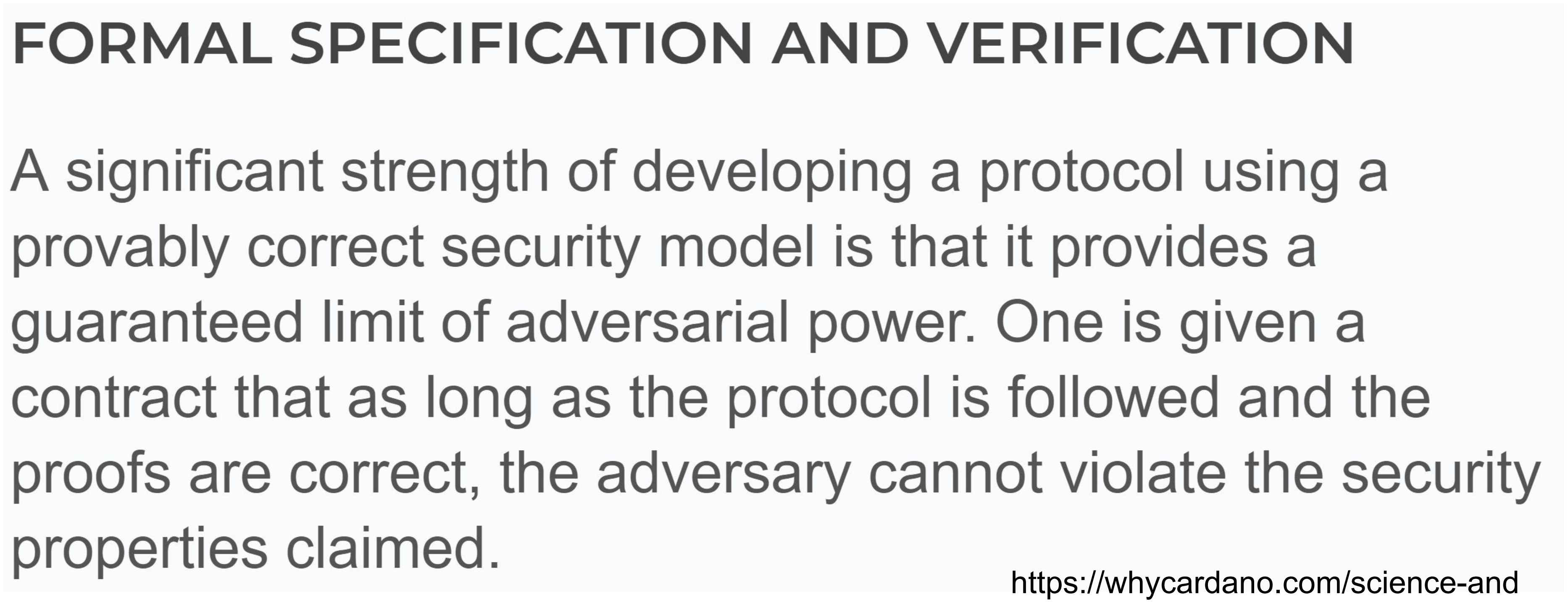
Posted by Christian Reitwiessner on September 1st, 2016.

Today, I am delighted to announce that Yoichi Hirai (pirapira on github) is joining the Ethereum project as a formal verification engineer. He holds a PhD from the University of Tokyo on the topic of formalizing communicating parallel processes and created formal verification tools for Ethereum in his spare time.

I'm joining Ethereum as a formal verification engineer. My reasoning: formal verification makes sense as a profession only in a rare situation where

- *the verification target follows short, simple rules (EVM);*
- *the target carries lots of value (Eth and other tokens);*
- *the target is tricky enough to get right (any nontrivial program);*
- *and the community is aware that it's important to get it right (maybe).*

Lead Blockchain Researcher (Facebook)



FORMAL SPECIFICATION AND VERIFICATION

A significant strength of developing a protocol using a provably correct security model is that it provides a guaranteed limit of adversarial power. One is given a contract that as long as the protocol is followed and the proofs are correct, the adversary cannot violate the security properties claimed.

https://whycardano.com/science-and-engineering/#formal-specification-and-verification

In this talk...

1. Stories of disasters from Ethereum’s first few years

2. What are the unique challenges about smart contracts that lead to research opportunities for FM?

Parity Wallet disasters

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.

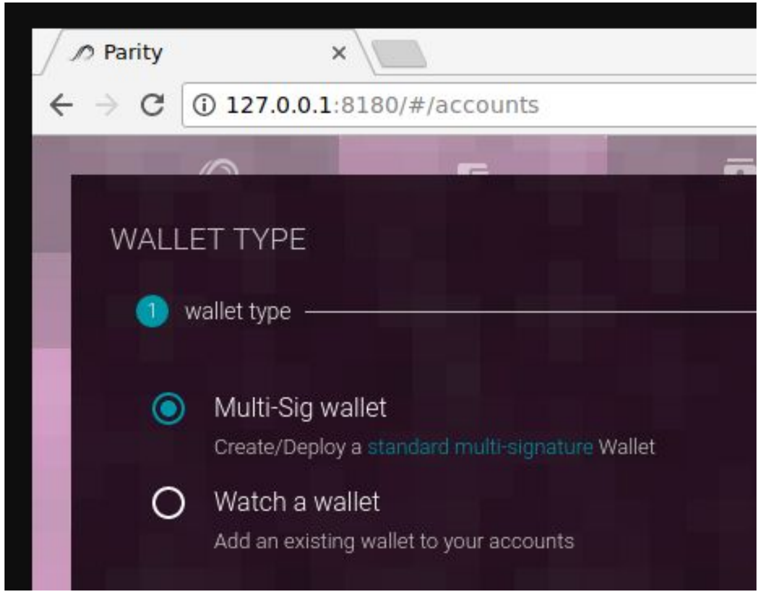
What is the Parity Wallet?

“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



[✓ Multi-Sig wallet]
Create/Deploy a standard multi-signature Wallet

Watch a wallet
Add an existing wallet to your accounts

Parity Wallet uses the “Prototype Inheritance” pattern

* Toy version of code

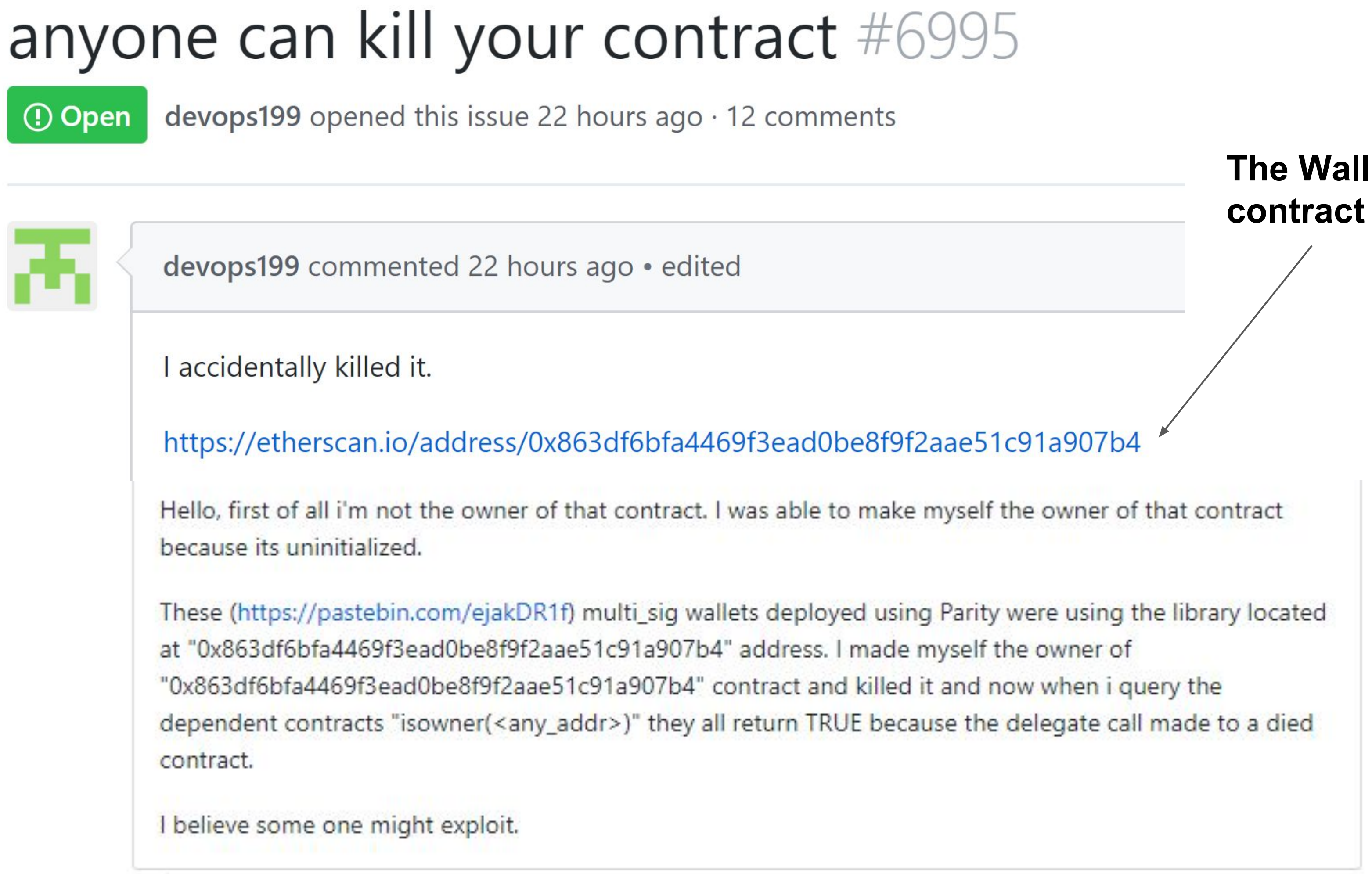
```
contract Wallet {
  address _walletLibrary;
  address owner;

  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) {
         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

Multiple Wallet contracts, all refer to Wallet Library
Why? Reduces fees from duplicate data!


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

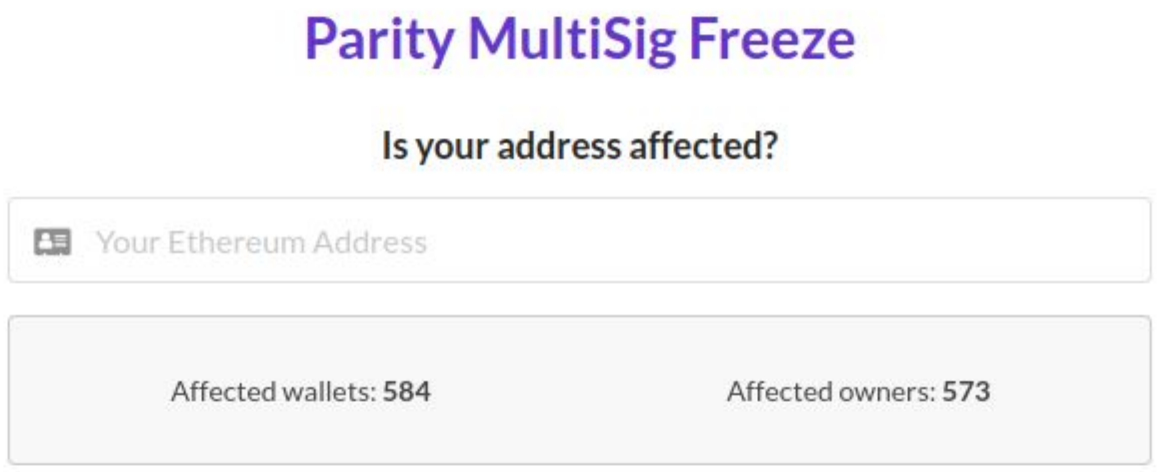
Wallet Library contract:

```
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.



Parity MultiSig Freeze

Is your address affected?

Your Ethereum Address

Affected wallets: **584**

Affected owners: **573**

All about TheDAO

slock.it   a Blockchain + IoT company

Slock Home Server

Slock Power Switch

In Progress (with partners)

- Slock Door Lock
- Slock Bike Lock
- Slock Pad Lock
- Slock Car Lock

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)

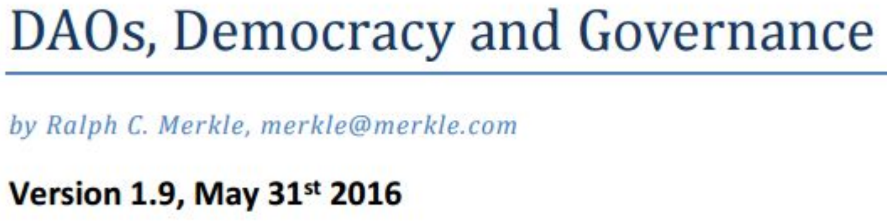
Built by slock.it to raise funds for their company

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

Ξ

Ξ	Slock Token	Reward	Vote
↑	↓	↓	↑

Tasks:

- *Fund the development*
- *Vote on major decisions*
- *Control the funds (!)*
- *Profitable*

Service provider

Reward

- % fee of every transaction
- one time deployment fee

Tasks:

- *Produce Slocks*
- *Marketing*
- *Partnerships*

Slock Home Server

supports:

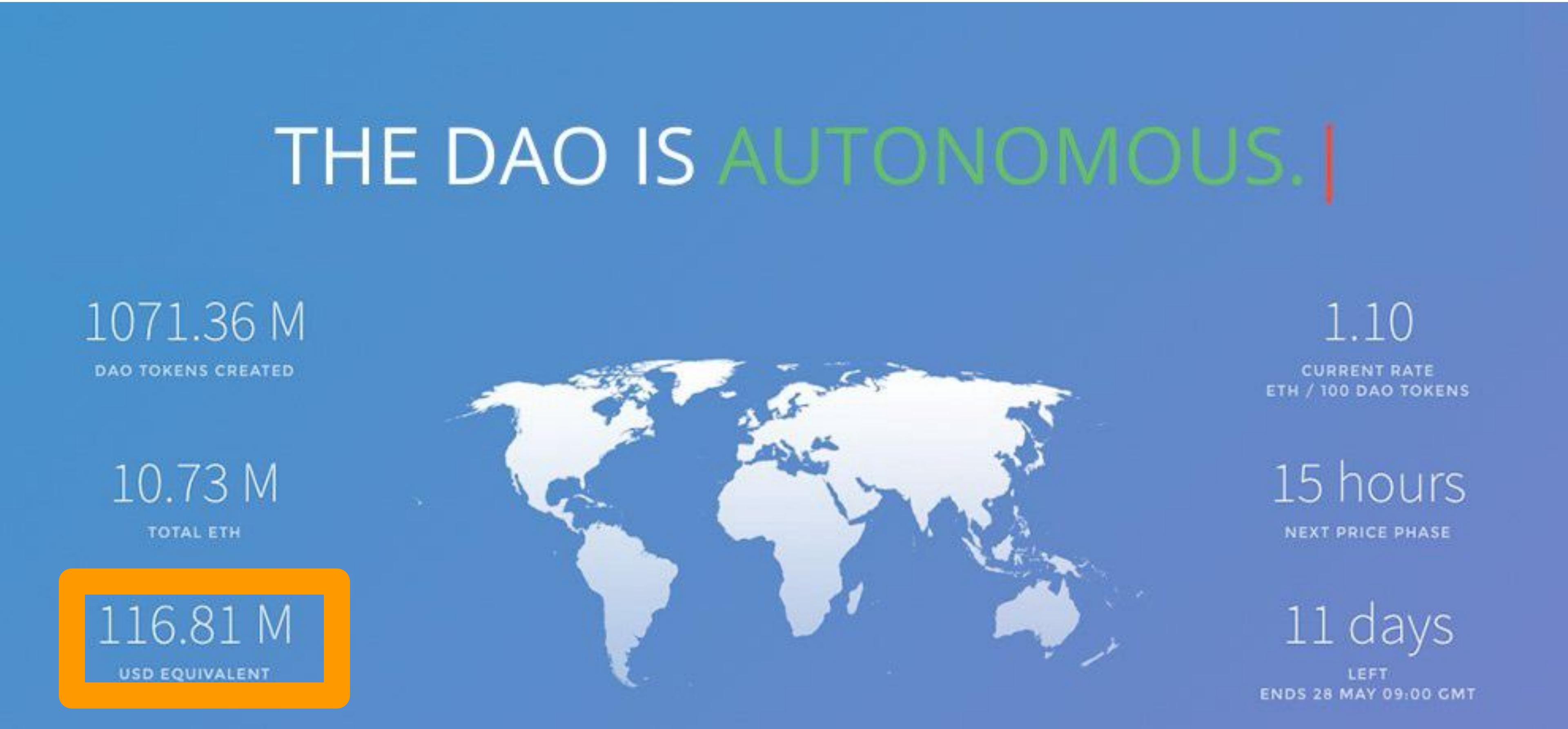
- Z-Wave
- Zigbee
- Bluetooth LE

Slock Power Switch

intel inside

In Progress (with partners)

- Slock Door Lock
- Slock Bike Lock
- Slock Pad Lock
- Slock Car Lock



THE DAO IS AUTONOMOUS.

1071.36 M
DAO TOKENS CREATED

10.73 M
TOTAL ETH

116.81 M
USD EQUIVALENT

1.10
CURRENT RATE
ETH / 100 DAO TOKENS

15 hours
NEXT PRICE PHASE

11 days
LEFT
ENDS 28 MAY 09:00 GMT

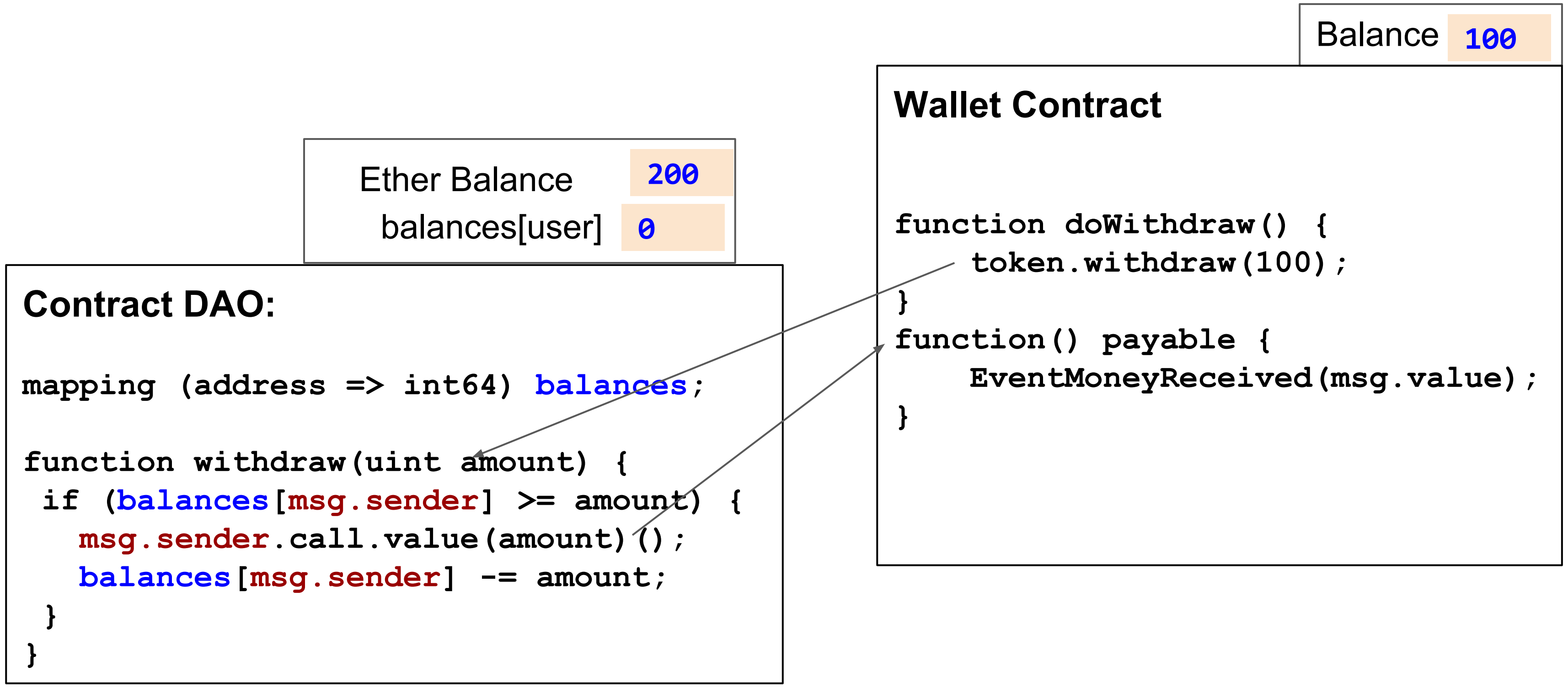
Raised ~150 million dollars in ~ 1 month

Excerpt from DAO contract (simplified)

Contract DAO:

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

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

Re-entrancy hazards in Ethereum



Ether Balance	200
balances[user]	0

Contract DAO:

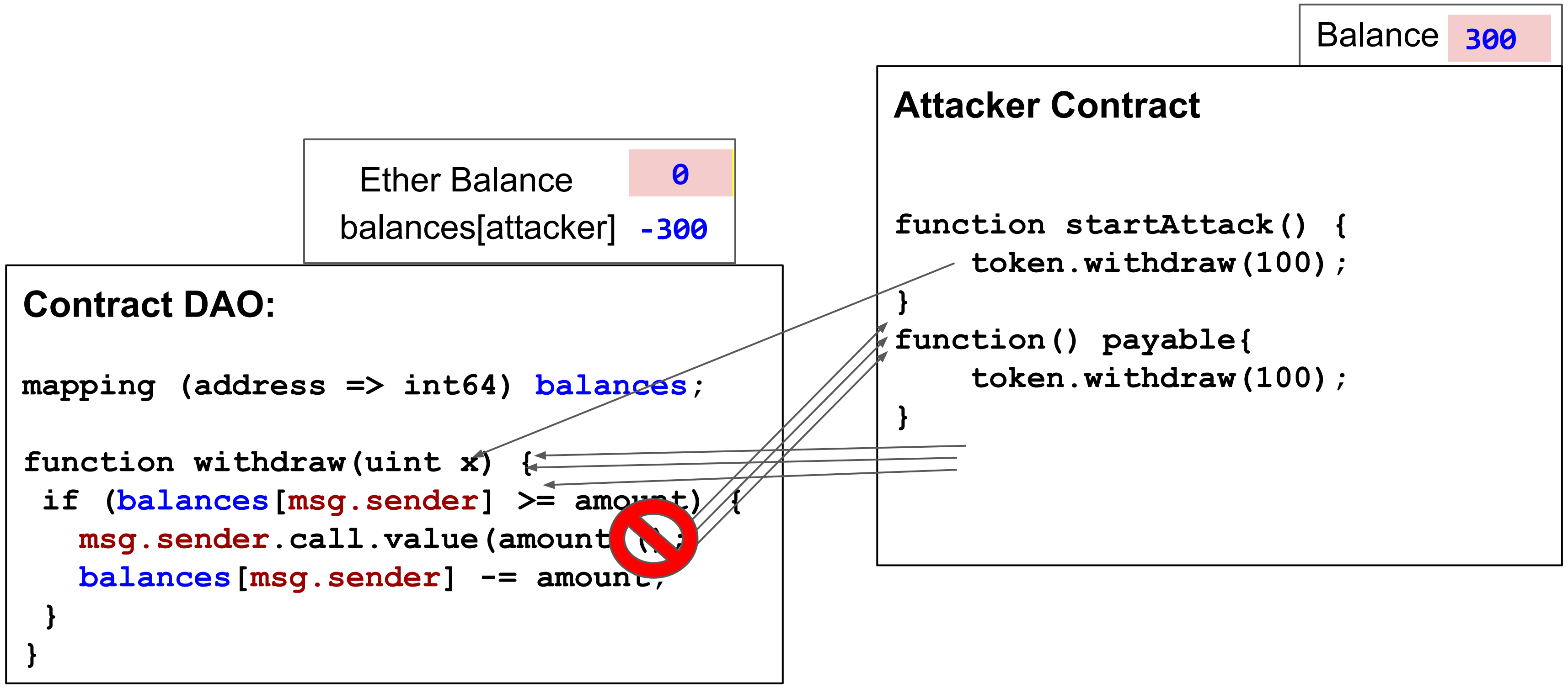
```
mapping (address => int64) balances;

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

Balance	100

Wallet Contract

```
function doWithdraw() {
    token.withdraw(100);
}
function() payable {
    EventMoneyReceived(msg.value);
}
```

Re-entrancy hazards in Ethereum



Balance 300

Attacker Contract

```
function startAttack() {
     token.withdraw(100);
}
function() payable{
     token.withdraw(100);
}
```

Ether Balance 0

balances[attacker] -300

Contract DAO:

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

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

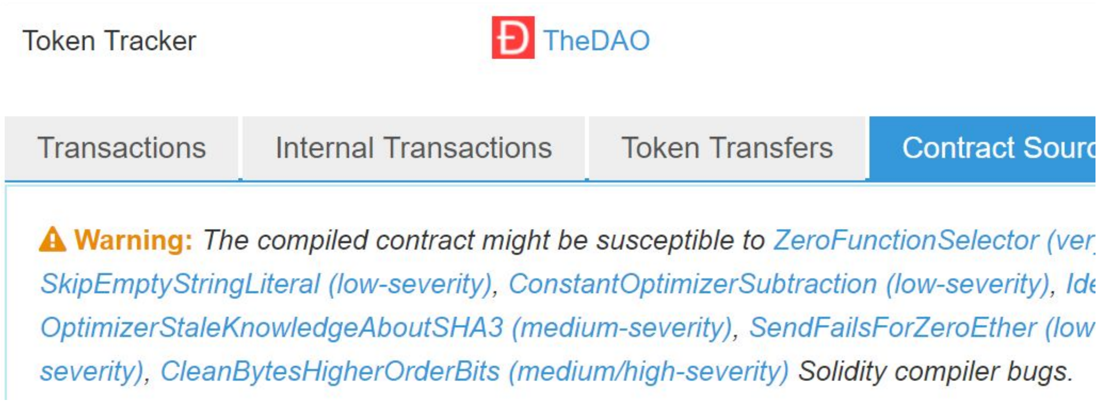
Directions for improving smart contract security

- Learn the hard way, adopt hindsight-oriented best practices

- Make iterative ergonomic improvements to the language

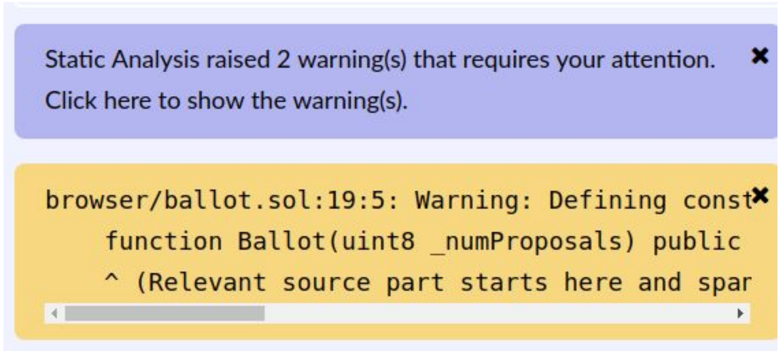
- Design new language abstractions for making secure-by-construction programs

- Design program analysis tools to exclude classes of errors

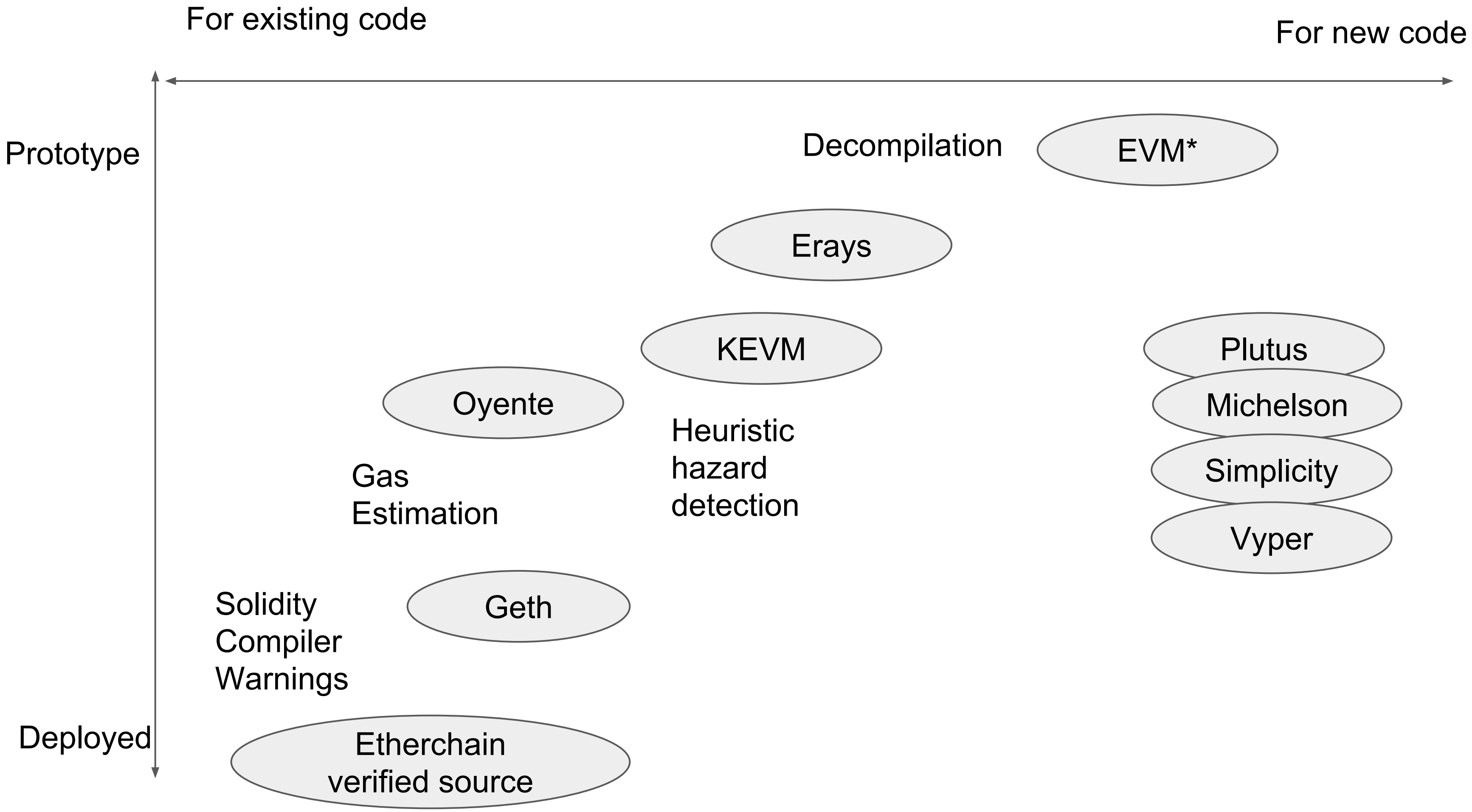

Token Tracker TheDAO

Transactions | Internal Transactions | Token Transfers | Contract Sourc

⚠ **Warning:** *The compiled contract might be susceptible to ZeroFunctionSelector (ver SkipEmptyStringLiteral (low-severity), ConstantOptimizerSubtraction (low-severity), Ide OptimizerStaleKnowledgeAboutSHA3 (medium-severity), SendFailsForZeroEther (low severity), CleanBytesHigherOrderBits (medium/high-severity) Solidity compiler bugs.*


Static Analysis raised 2 warning(s) that requires your attention.
Click here to show the warning(s).

```
browser/ballot.sol:19:5: Warning: Defining const
    function Ballot(uint8 _numProposals) public
    ^ (Relevant source part starts here and spar
```


For existing code
For new code
Prototype
Deployed
Decompilation
EVM*
Erays
KEVM
Plutus
Oyente
Michelson
Heuristic hazard detection
Simplicity
Gas Estimation
Vyper
Solidity Compiler Warnings
Geth
Etherchain verified source

Safer Smart Contract Languages

- Scripting Languages
- Typed functional programming languages
- Formal logics
- Automata model

Safer Smart Contract Languages

Name	Level	Blockchain
Scilla	Intermediate-level	Zilliqa
FSolidM	Higher-level, framework	Ethereum
Rholang	Higher-level	RChain
Vyper	Higher-level	Ethereum
Type-coin		Bitcoin
Simplicity		Bitcoin

Safer Smart Contract Languages

Name	Level	Blockchain
Michelson	Lower-level, Functional	Tezos
Liquidity	Higher-level, Functional	Tezos
Plutus	Higher-level	IOHK
Plutus-Core	Lower-level	IOHK
Owlchain	Framework	BOSCoin

Table 1: Languages

Languages	Descriptions (motivation, expressivity, type system, analysis-friendly features, etc)	Reference
Scilla (intermediate-level) (Zilliqa)	- motivated by achieving expressivity and tractability - based on communicating automata [50] - provide limited translation from higher-level languages (i.e., Solidity) - provide translation into Coq for verification, along with contract protocols, semantics, safety/liveness properties and proof machinery	paper [71], code [70]
FSolidM (Ethereum, framework, higher-level)	- aims to develop more secure smart contracts - a formal, finite-state machine based model - provide several plugins (i.e., design patterns) to enhance security and functionality, targeting at vulnerabilities as reentry bugs and transaction ordering, or design patterns as time constraint and authorization. - primarily for Ethereum, but it may applied on other platforms - provide translation into Solidity	paper [55], code [54]
Rholang (higher-level, RChain)	- primitively for RChain, but could be used in other settings - focus on message-passing and formally modeled by the ρ-calculus, a reflective, higher-order extension of the π-calculus, which is good for concurrent settings [57]	code [19]
Vyper (Ethereum, higher-level)	- mainly target at security and auditability - provide the following features: bounds and overflow checking, support for signed integers and decimal fixed point numbers, decidability, strong typing, small and understandable compiler code, and limited support for pure functions - does not support the following features: modifiers, class inheritance, inline assembly, operator overloading, recursive calling, infinite-length loops and binary fixed point. - statically typed language	code [5], doc [27]
Type-coin (Bitcoin)	- a logical commitment mechanism - the logic is linear and not rich to handle complex situations	paper [34]
Simplicity (Bitcoin)	- type-safety, no unbounded loops, no named variables - no function types and thus no higher-order functions,	paper [66], blog [31]
Michelson (Tezos)(lower-level)(functional)	- a strongly-typed, stack-based language - It doesnt include many features like polymorphism, closures, or named functions. - more as a way to implement pieces of business logic than as a generic "world computer" - Programs written in Michelson can be reasonably analyzed by SMT solvers and formalized in Coq without the need for more complicated techniques like separation logic. - To provide a straightforward platform for business logic, to provide a readable bytecode, and to be introspectable. - Entirely original implementation in OCaml - Isolated economical rules, self-amendable via voting - purely PoS - Blockchain state in a git-like persistent store - Highly functional, defensive coding style for the critical parts - designed with formal certification in mind	paper [12], web [26]
Liquidity (Tezos) (higher-level)(functional)	- It uses the syntax of OCaml, and strictly complies to Michelson security restrictions	code [7], web [65]
Plutus (higher-level) and Plutus Core (lower-level) (IOHK)	- compiled to Plutus Core (lower level), Lisp-like syntax - a pure functional strictly typed programming language, with user-defined data types and polymorphism. - several issues: unbounded integers supporting, non-supporting abstract data types and data constructors.	code [17], paper [56]
Owlchain (BOSCoin)	- a decidable programming framework, which consists of the Web Ontology Language and the Timed Automata Language. - OWL is defined as W3C standard, a declarative language that provides decidability. - separate declaration from processing - TAL, Timed Automata Language, is a new language that is used to create operators. It is a finite state programming environment with two constraints: time limit and pure functions. Timed automata modeling can detect undefined areas(reachability problem) in the code that developers missed. Pure function can eliminate side effects that can occur during development.	article [32]

Program Analysis

More than 11 tools or frameworks to detect various types of vulnerabilities:

- From the scale of checking abilities
- From the view of vulnerabilities
- From the view of programming analysis techniques
- Most are developed in Python

Semantics design

Program Analysis

Tools	CertiK	Dr.Y's Analyzer	Maian
	Manticore	Mythril	Oyente
	porosity	SmartCheck	Securify
	Solgraph	ZEUS	
Semantics Design	eth-isabel	F* (2016)	TU Wien F* (2018)
	KEVM	SMAC	

Table 2: Tools and frameworks for analyzing smart contracts

Tool/Framework	Capabilities	Reference
CertiK (Demo)	- target at fully trustworthy blockchain ecosystems in the future - specifications for each function can be expressed using CertiK labels, indicating pre-condition, post-condition and invariants respectively, as comments in Solidity programs	white paper [2]
Dr.Y's Ethereum Contract Analyzer	- a symbolic execution tool, reflecting contract behavior to some point	code [44]
Maian	- check locked money - detect unchecked suicide or Ether sending - generate inputs to validate through private blockchain	paper [64], code [8]
Manticore	- detect potential overflow and underflow conditions on "ADD", "MUL" and "SUB" instructions - detect potential uses of uninitialized memory or storage - calculate code coverage - generate inputs which could trigger unique code paths (Solidity source code needed) - Other: offer a Python API for analysis of EVM bytecodes	article [11], code [9], doc [10]
Mythril	- detect reentry bugs and external calls to untrusted contracts - detect unchecked suicide or Ether sending - check mishandled exceptions (i.e., detect unchecked CALL return value) - check integer underflows - detect usage of "tx.origin" [21] - check dependence on predictable variables (e.g., coinbase, gaslimit, timestamp, number, etc.) - Other: generate control flow graph, blockchain exploration and some utilities - support on-chain contracts analysis	article [61] [60], doc [13], code [62],
Oyente	- detect reentry bugs - check mishandled exceptions (i.e., detect unchecked CALL return value) - check transaction-order-dependency (a.k.a. money concurrency, or front running) - check timestamp dependency - check possible assertion failure (Solidity source code required) - calculate code coverage	paper [52], web access [16], code [15]
porosity	- find potential reentrancy vulnerability - support decompilation and disassembly	code [3], white paper [73], article [72]
SmartCheck (target at Solidity)	- detect reentry bugs - check locked money - detect possibly infinite or impractical loops - detect unchecked low-level call - check integer overflow and underflow, and recommend to use the SafeMath library [14] - check timestamp dependence - Other: more better programming design pattern recommendation - Other: recommendations for standard ERC-20 function usages, and check style guide violation - Other: some checking for recommended Solidity programming style	code [23], web access [22]
Securify	- check reentry bugs - check mishandled exception - check transaction-order-dependency - check insecure coding patterns, e.g., unchecked transaction data length, use of ORIGIN instruction and missing input validation - check unexpected Ether flows, such as locked Ether [68] - check use of untrusted inputs in security operations, i.e., checking whether the inputs to the SHA3 depend on block information (timestamp, number, coinbase)	web access [20]
Solgraph	- highlight potential unchecked money receiver - generate function control flow of a Solidity contract	code [67]
ZEUS	- support self-defined policy verification, e.g., reentry bugs, unchecked "send", possibly vulnerable failed "send", integer overflow, transaction state dependency (i.e., usage of "tx.origin"), block state dependency (including all "block" parameters) and transaction order dependency - specification limited to quantifier-free logic with integer linear arithmetic	paper [47]

Table 3: Language design and model translation

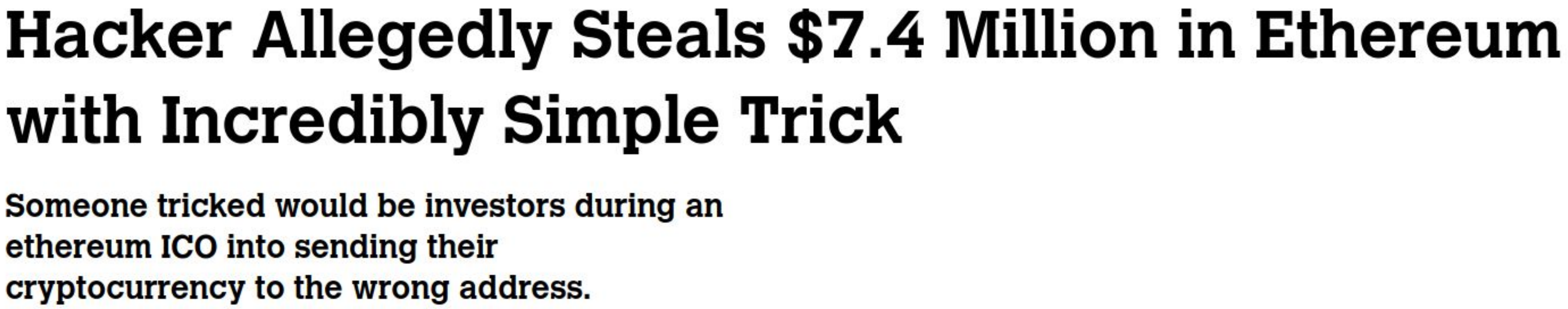
Languages or semantics	Descriptions (motivation, expressivity, type system, analysis-friendly features, etc)	Reference
SMAC (modular reasoning)	- introduce ECF (Effectively callback free) property for modular object-level analysis - develop online detection algorithm which can apply to Ethereum full node, and monitor non-ECF executions, including the infamous DAO bug	paper [40]
eth-isabelle (semantics)	- define the complete instruction set of EVM in Lem, a language that can be compiled into Coq, Isabelle/HOL and HOL4 - can prove invariants and safety properties	paper [46], code [45]
TU Wien F* (2018) (Ethereum)	- present the complete small-step semantics of EVM bytecode in the F* proof assistant - define a number of central security properties, such as call integrity, atomicity, and independence from miner controlled parameters	paper [39], code [25]
F* (2016) (Ethereum)	- motivated by formal verification - partial semantics for converting Solidity to F*, EVM to F* - show the correspondence between Solidity and EVM to some point	paper [30]
KEVM (semantics, high-level, Ethereum)	- a complete K Semantics of the Ethereum Virtual Machine (EVM)	code [6], paper [42]

Formal methods for smart contracts: Opportunity or bubble?

What’s unique about smart contracts?

1. Ease of vulnerability monetization
2. Blockchain programming specialties (gas, public, …)
3. Tendency to rely on incentive mechanisms
4. For disruptive potential, must appeal to novice developers

Not all Cryptocurrency failures are high-tech



Hacker Allegedly Steals $7.4 Million in Ethereum with Incredibly Simple Trick

Someone tricked would be investors during an ethereum ICO into sending their cryptocurrency to the wrong address.

What drives the “demand” for formal methods?

Fear-of-Missing-Out can outweigh Prudence

Externalities:

High profile failures drag the price down for everyone

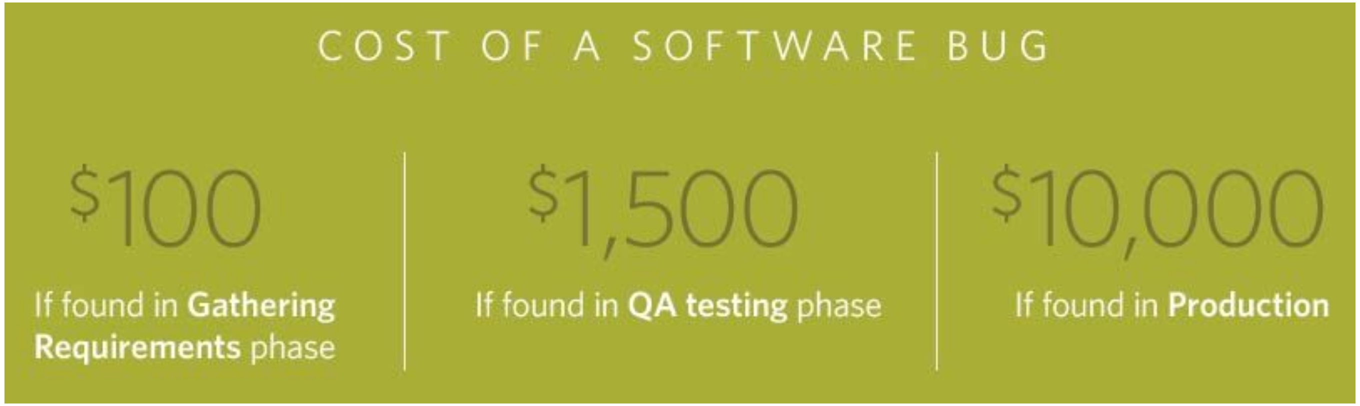
Who should take responsibility for security?

Investors?

App developers?

Protocol engineers/stewards?

In Smart Contracts, bugs are easily monetized



COST OF A SOFTWARE BUG

$100	$1,500	$10,000
If found in **Gathering Requirements** phase	If found in **QA testing** phase	If found in **Production**

A 2003 study commissioned by the Department of Commerce’s National Institute of Standards and Technology found that **software bugs cost the US economy $59.5 billion annually.**

Smart Contract challenges in the blockchain model

- Transactions are processed after a delay
- Adversaries can re-order and “frontrun” transactions
- All smart contract data is public
- Execution is expensive (gas costs, tx fees)
- Code is difficult to change (“Code is law”)

We need tools for “crypto-economic” reasoning

Incentive mechanism analysis

- Security deposits and penalties
    (based on detected misbehavior)
    Nash equilibrium (and variations...)

- Fees (charged based on market price)
    Bounding the consumption of “gas”
    apply methods for bounding space/time

We need tools for “cryptoeconomic” reasoning
Example: Sealed Bid Auction

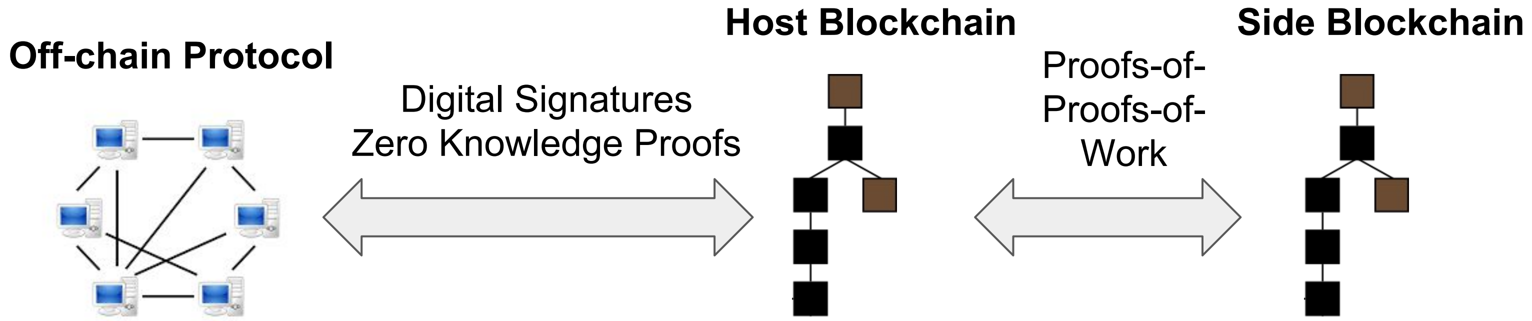
Uses `commit-and-reveal` cryptography to prevent bid front running. However, there remains a concern about misbehavior where players "abort" and go away before revealing

The "cryptoeconomics" solution: require a security deposit, that you lose if you do not reveal

Property 1: The security deposit at least covers the gas cost of the `reveal` method

Property 2: Revealing should be an equilibrium strategy. (Vickrey auctions assume all bids are committed simultaneously and then revealed, so it becomes a proof obligation to show it still holds).

Smart contracts are just one component of increasingly complex protocols



Non-exhaustive list of startups focusing on inter-blockchain interaction:

CØSMOS INTERNET OF BLOCKCHAINS

Interledger The protocol for connecting ledgers.

PARSEC LABS

Celer Network Bring Internet Scale to Every Blockchain

Example of Offchain Protocol: Payment Channels

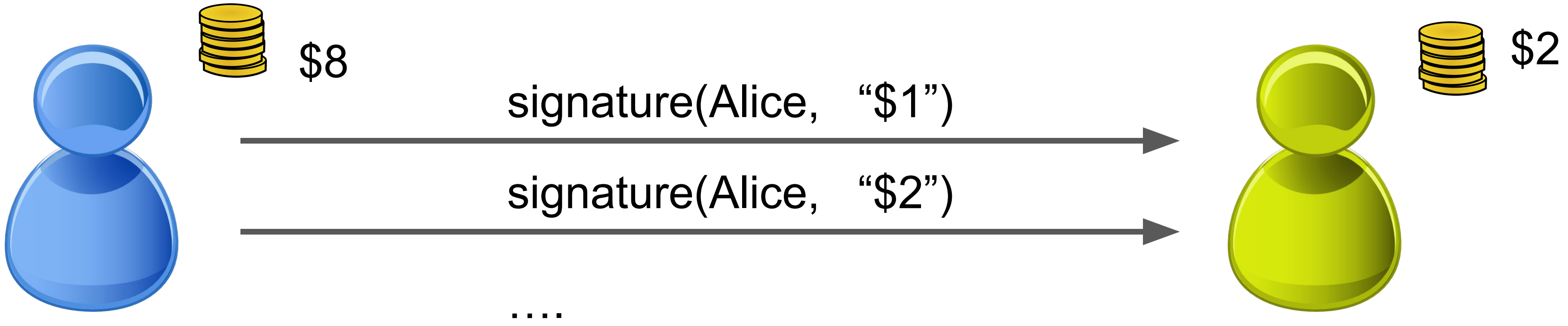
1. Alice deposits $10

2. Alice pays Bob by sending signatures on incrementally larger payments

Blockchain

3. At any point, Bob closes the channel by posting the largest signed message on the blockchain.

4. If Bob aborts, Alice can claim a refund (after a time limit)



Example of Offchain Protocol: Payment Channels (Implemented in Ethereum)

```
def finalize(signature, amount):
  assert(msg.sender == bob)
  assert(verify_signature(alice, amount, sig)))
  send(bob, amount)
  send(alice, self.balance)


def refund():
  assert(msg.sender == alice)
  assert(block.number > deadline) // only after deadline
  send(alice, self.balance)
```

Some Thoughts

- Compositional Verification
    - Assume Guarantee Reasoning
    - *Challenge:* Various types of primitives

- Synthesis
    - *Challenge:* High-level architecture design drives the implementation
    - Roles of various primitives is clear

Concluding remarks

- Several technical challenges for FM/PL research arising from smart contracts, which are well motivated by the real world use seen already

- Smart contracts, like other hot domains, such as Internet of Things, have encountered expensive security failures early on, shaping the field

- However, the close relationship between the smart contracts industry and formal methods is a unique opportunity

Public debates are raging that amount to choosing which layer of the system must improve

Should we change the underlying EVM virtual machine?

Or should we change the high level language?

Smart contracts are small components of larger distributed applications (dApps)

A dApp includes:

- Smart contract used as a backstop / dispute resolution handler

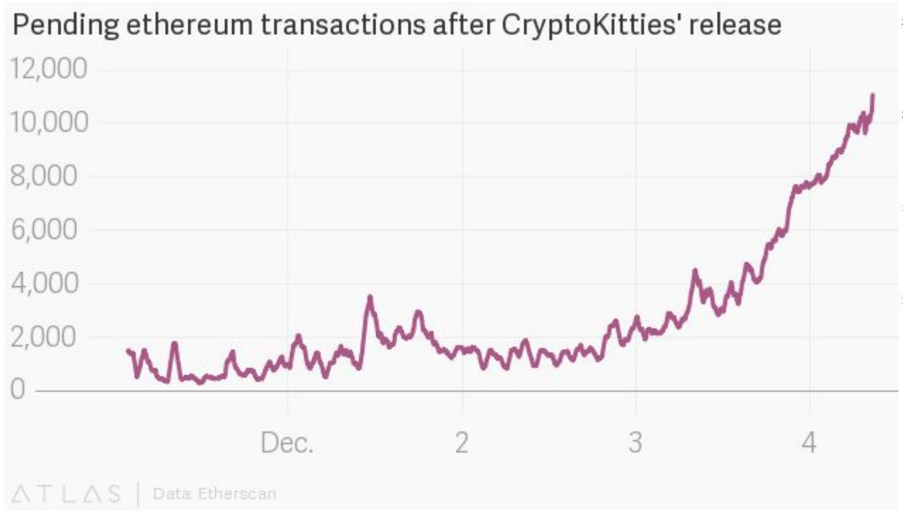
- Cryptography computed locally by clients (signatures, zero knowledge proofs)

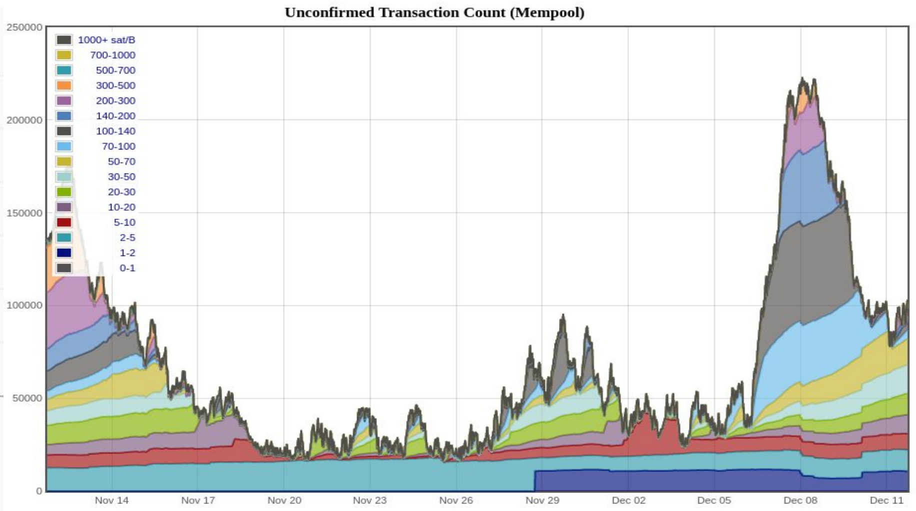
- Point-to-point messages exchanged out of band

Examples:

- Multi-player Lotteries / Coin flips https://arxiv.org/abs/1612.05390 (WTSC'17)
- Scalable payment channel networks https://arxiv.org/abs/1702.05812
- Interactive games (e.g. Poker) https://arxiv.org/abs/1701.06726 (AsiaCrypt'17)

Blockchains are powerful, but expensive



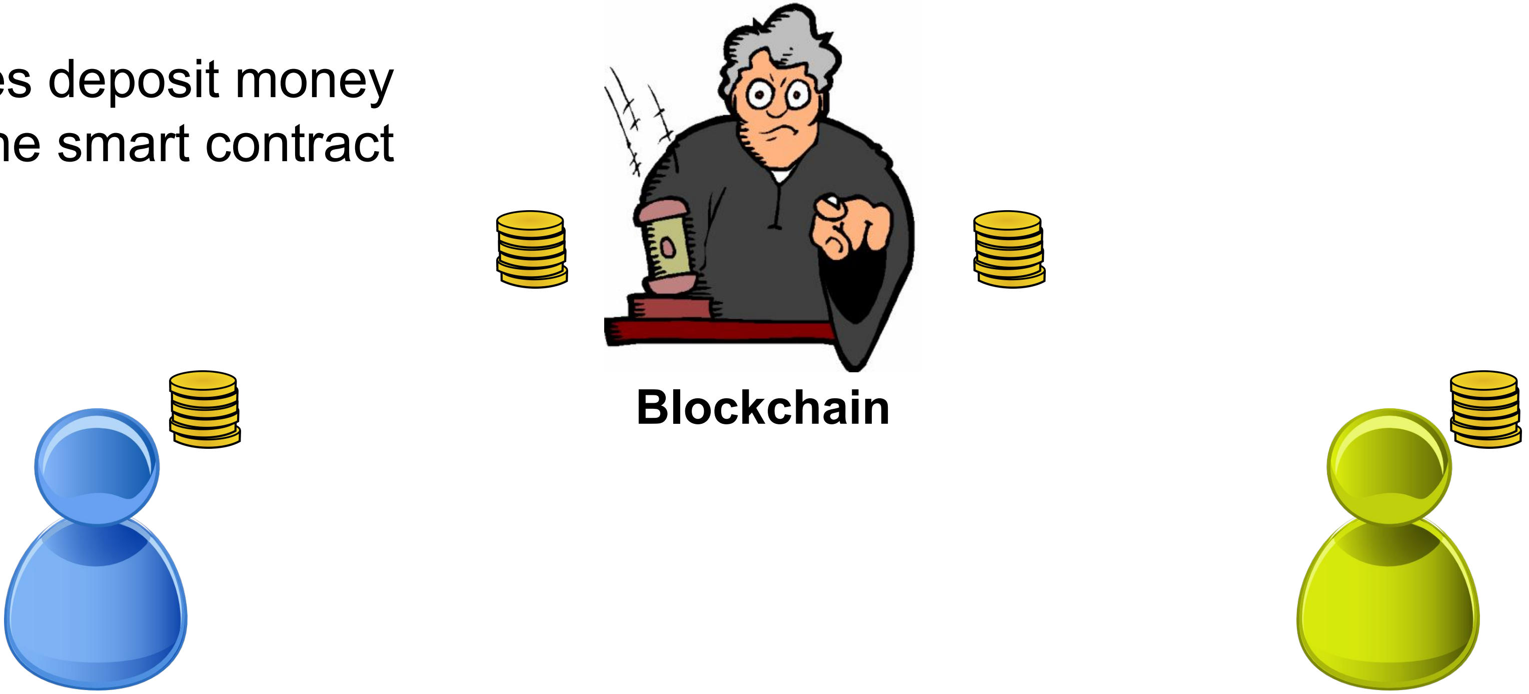


- Blockchains derive their security through wide replication

- We do not fully understand the security consequences of scalable alternatives (incentive compatibility, plausibility of assumptions, etc.)

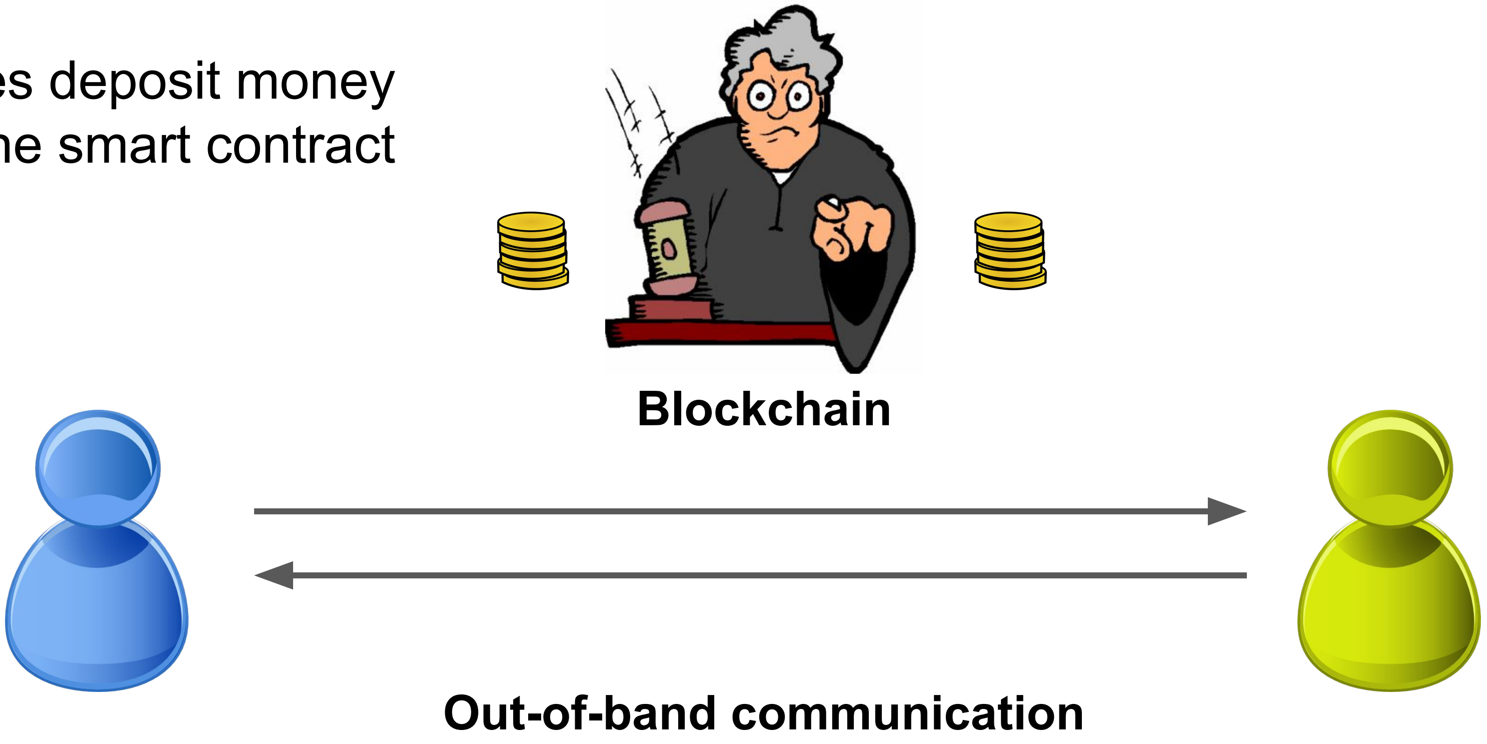
The “Off-chain” method:
Avoid blockchain transactions except in rare cases

Parties deposit money into the smart contract



The “Off-chain” method: Avoid blockchain transactions except in rare cases

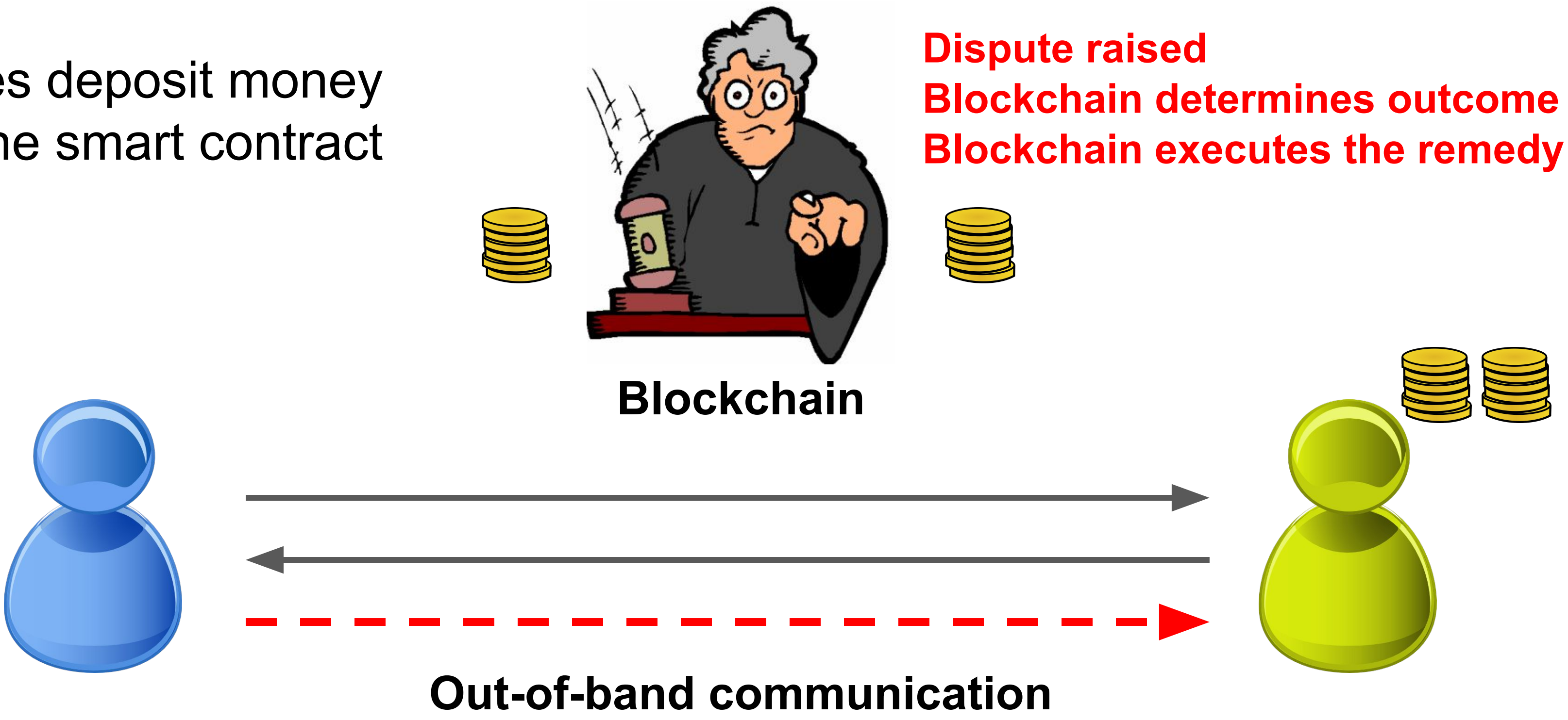
Parties deposit money into the smart contract



The “Off-chain” method: Avoid blockchain transactions except in rare cases

Parties deposit money into the smart contract

Dispute raised
Blockchain determines outcome
Blockchain executes the remedy

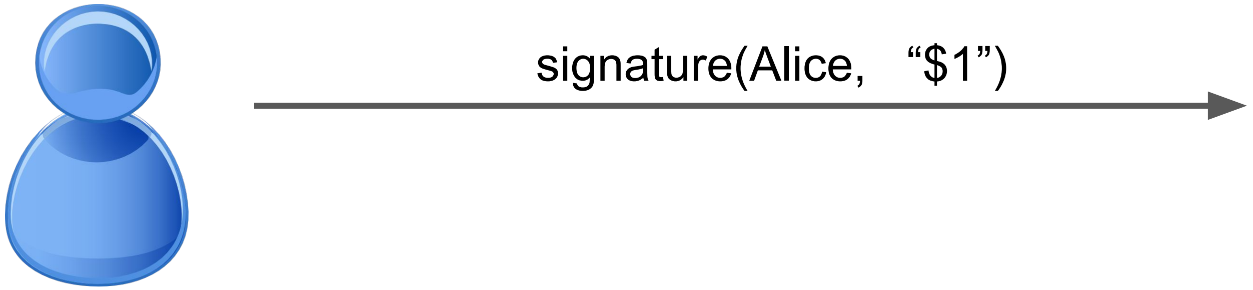


Unidirectional Payment Channels

Alice deposits $10

Alice pays Bob by sending signatures on incrementally larger payments signature

Blockchain

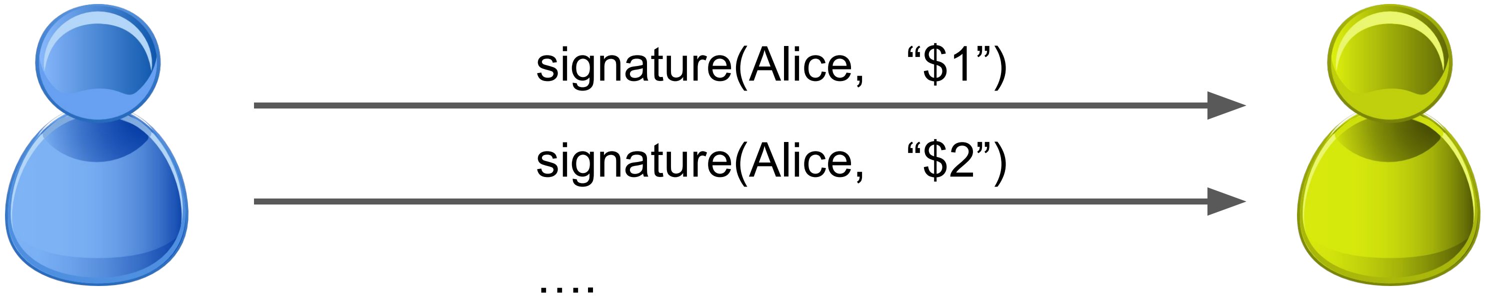


Unidirectional Payment Channels

Alice deposits $10

Alice pays Bob by sending signatures on incrementally larger payments signature

Blockchain



Unidirectional Payment Channels

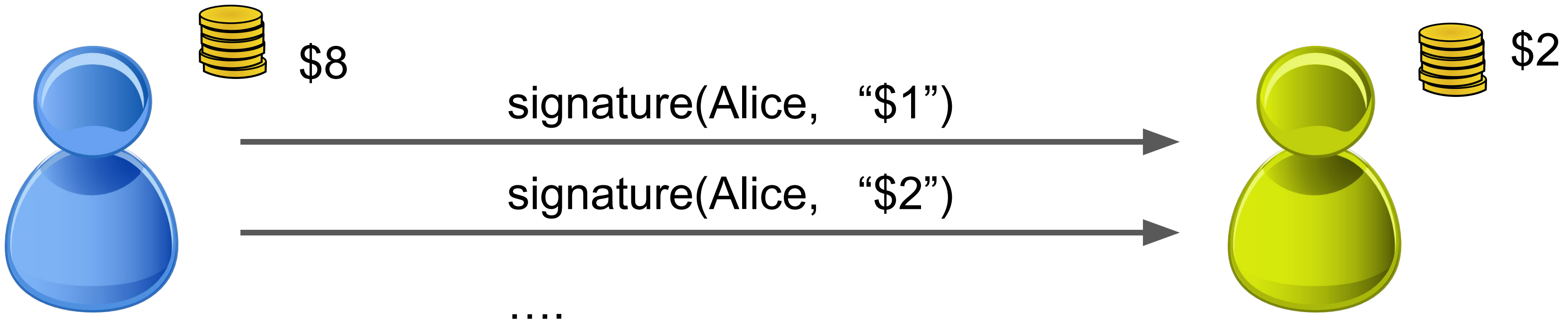
Alice deposits $10

Alice pays Bob by sending signatures on incrementally larger payments signature

Blockchain

At any point, Bob closes the channel by posting the largest signed message on the blockchain.

If Bob aborts, Alice can claim a refund (after a time limit)



Unidirectional Payment Channels (Implemented in Ethereum)

```
def finalize(signature, amount):
  assert(msg.sender == bob)
  assert(verify_signature(alice, amount, sig)))
  send(bob, amount)
  send(alice, self.balance)


def refund():
  assert(msg.sender == alice)
  assert(block.number > deadline) // only after deadline
  send(alice, self.balance)
```

Blockchain

Off-chain protocols for many applications

Application 1: Multi-player Lotteries / Coin flips https://arxiv.org/abs/1612.05390

Outcome: Bitcoin requires either $O(N^2)$ more collateral, or else $O(2^N)$ more computation vs. Eth

Application 2: Payment channels https://arxiv.org/abs/1702.05812

Outcome: Bitcoin requires O(L) more collateral for path of length L vs. Eth

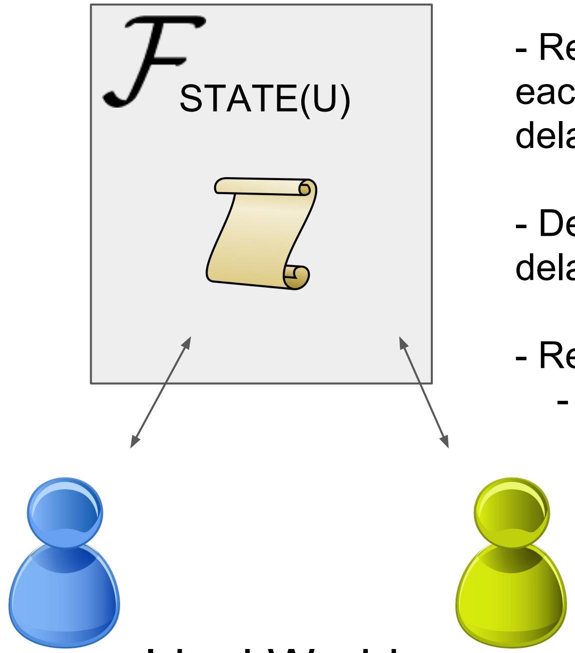
Application 3: Amortized games (e.g. Poker) https://arxiv.org/abs/1701.06726

Outcome: Both amortization and cash distribution in Ethereum only

Model complex protocols with an Ideal Functionality

In the real world: multiple views of the current state

Real World



Ideal World

- Models 1 consistent view of sequential state

- Receives inputs from each party (possibly delayed)

- Delivers output (possibly delayed)

- Real time guarantees:
    - Fast O(1) if all honest
    - At most O(D) always

- Always applies update U correctly

- Has side effects ($)

Payment Channels (spec)

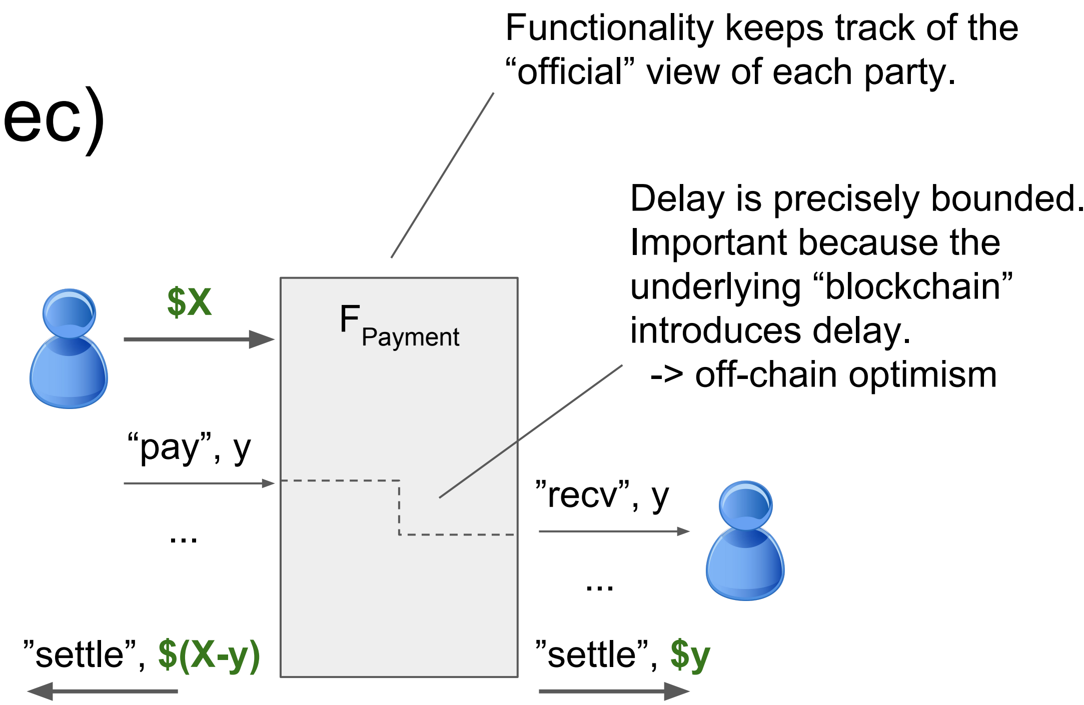
Security Properties modeled:

- Balance / no overpay

(side effect of moving money)

- Bounded delays

- Economic guarantee


Functionality keeps track of the “official” view of each party.
Delay is precisely bounded.
Important because the underlying “blockchain” introduces delay.
-> off-chain optimism
$X
F_{Payment}
“pay”, y
...
”recv”, y
...
”settle”, $(X-y)
”settle”, $y

Implementation: exchange off-chain signatures on increasing “y” (see next slide)

Future impact: can be linked into networks of collateral, e.g. Poker (AsiaCrypt’17)

Payment Channels (implementation)

$X $\longrightarrow$

paid := 0
…
paid += y
sig ← sign(paid)
send (sig, paid), to Alice
…

after $T_{Deadline}$,
call Contract.refund()

```
Contract_Payment

init: $X deposited from Alice at T

function settle(Y, sig) from Bob:
        check( 0 <= Y <= X )
        checkSig(pk_Alice, sig, h(Y))
        send($(X-Y) to Alice
        send($(Y) to Bob

function refund() from Alice:
        check( now >= T_Deadline )
        send($X) to Alice
```

receive (sig,paid)
Check signature
Discard unless “paid” is larger
oncheck , $y

”settle”, $y $\longrightarrow$

Backup slides

More detail about Parity Wallet disasters

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%)

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

```
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`**

```
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

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

```
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);
}
```

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:

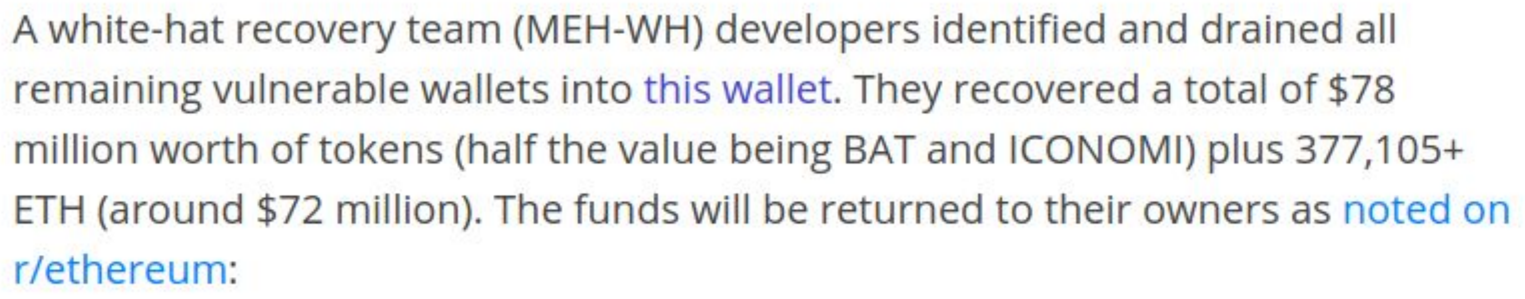
```
contract WalletLibrary {
        address owner;
        // called by constructor
        function initWallet(address _owner) {
            require(initialized == false);
            initialized = true;
            owner = _owner;
            // ... more setup ...
        }
        function changeOwner(address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

0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

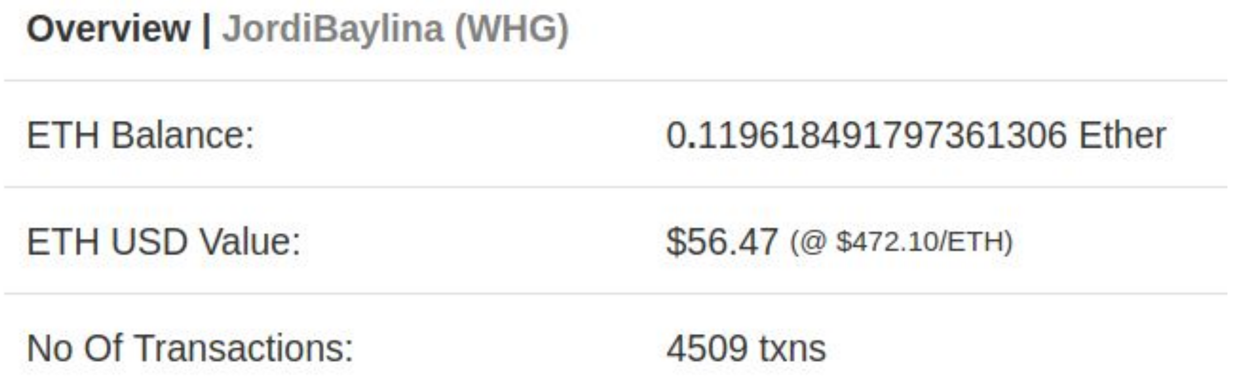
devcon three
Ethereum Foundation Developers Conference
Nov 1-4, 2017 Cancún, Mexico
status
Microsoft
Bitcoin.com
omisego
WANXIANG BLOCKCHAIN LABS
santiment
FUNFAIR

DEVCON3
CANCUN

The White Hat Recovery Team

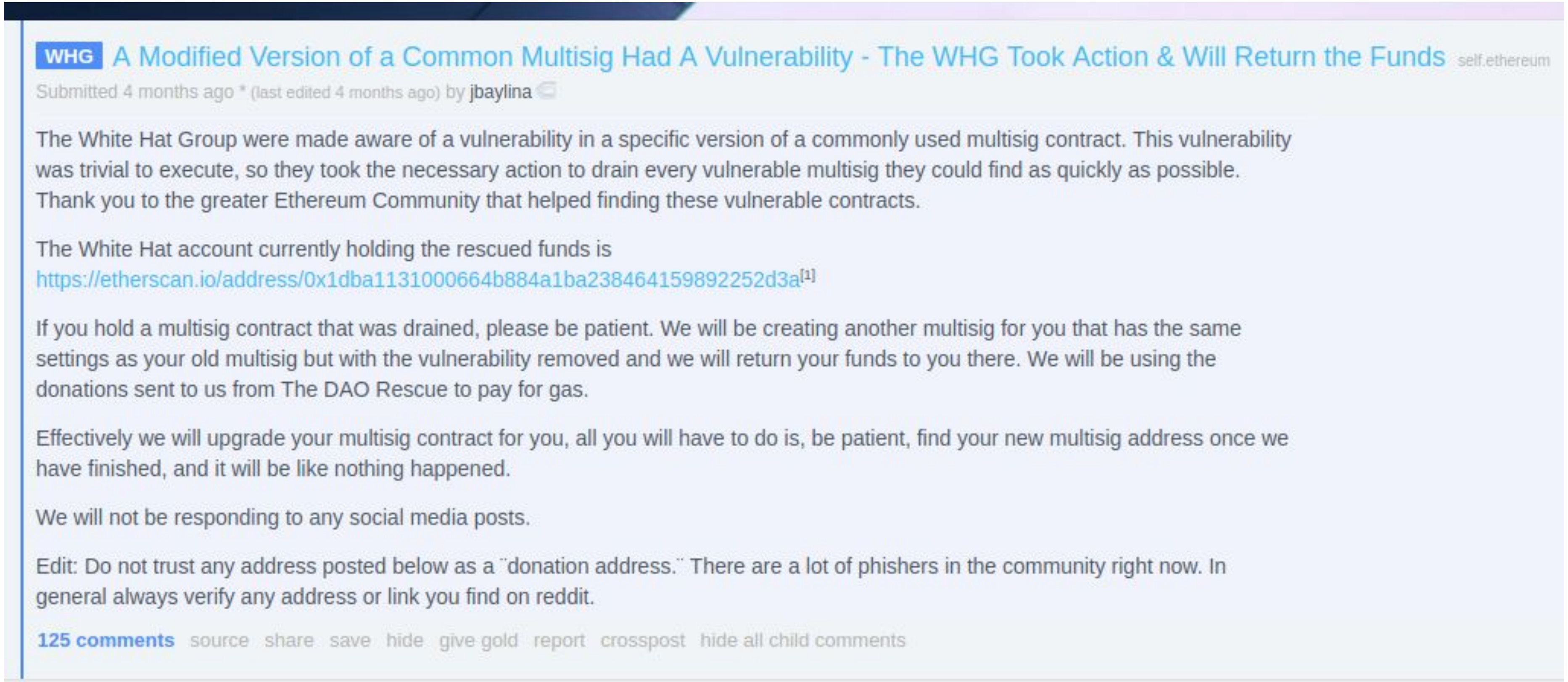


A white-hat recovery team (MEH-WH) developers identified and drained all remaining vulnerable wallets into this wallet. They recovered a total of $78 million worth of tokens (half the value being BAT and ICONOMI) plus 377,105+ ETH (around $72 million). The funds will be returned to their owners as noted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



Overview | JordiBaylina (WHG)

ETH Balance:	0.119618491797361306 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 jbaylina

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.

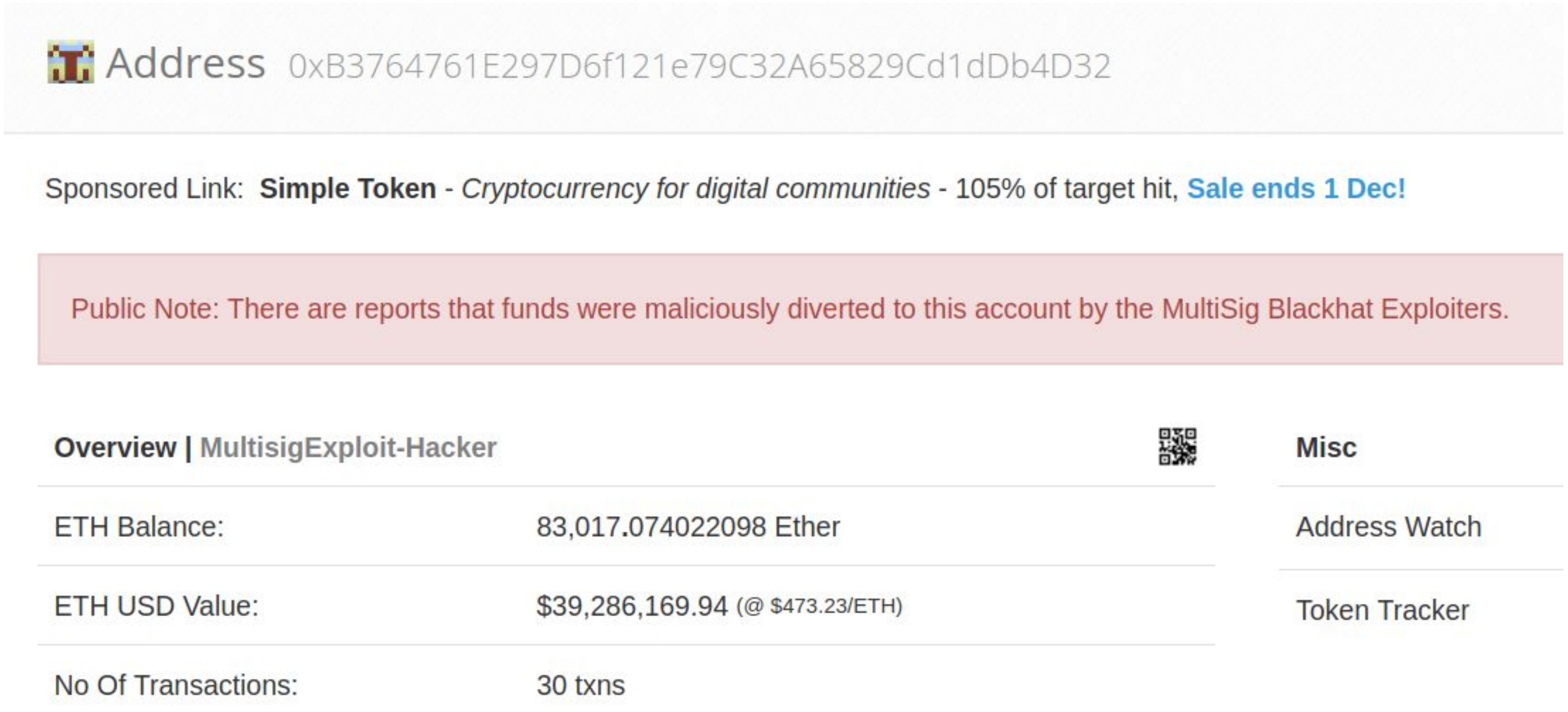
Effectively we will upgrade your multisig contract for you, all you will have to do is, be patient, find your new multisig address once we have finished, and it will be like nothing happened.

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

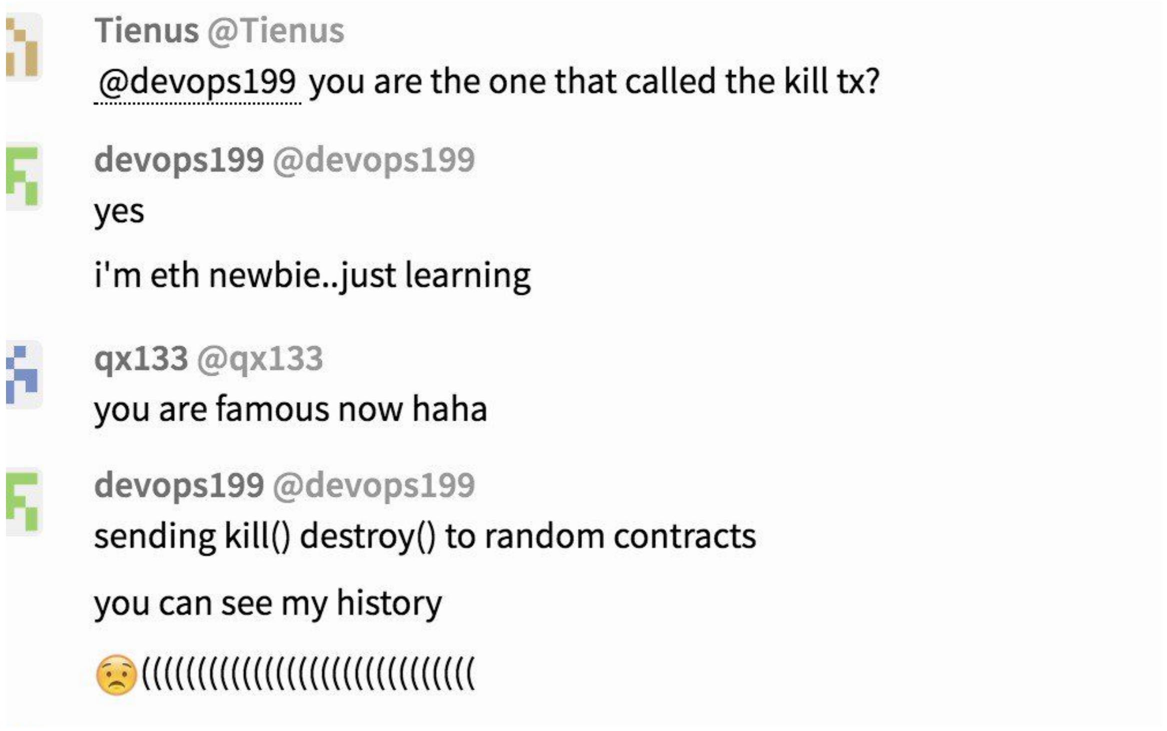
- An “initialize()” function to set the owner of the contract, was missing the “onlyOnce” modifier

- 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/

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

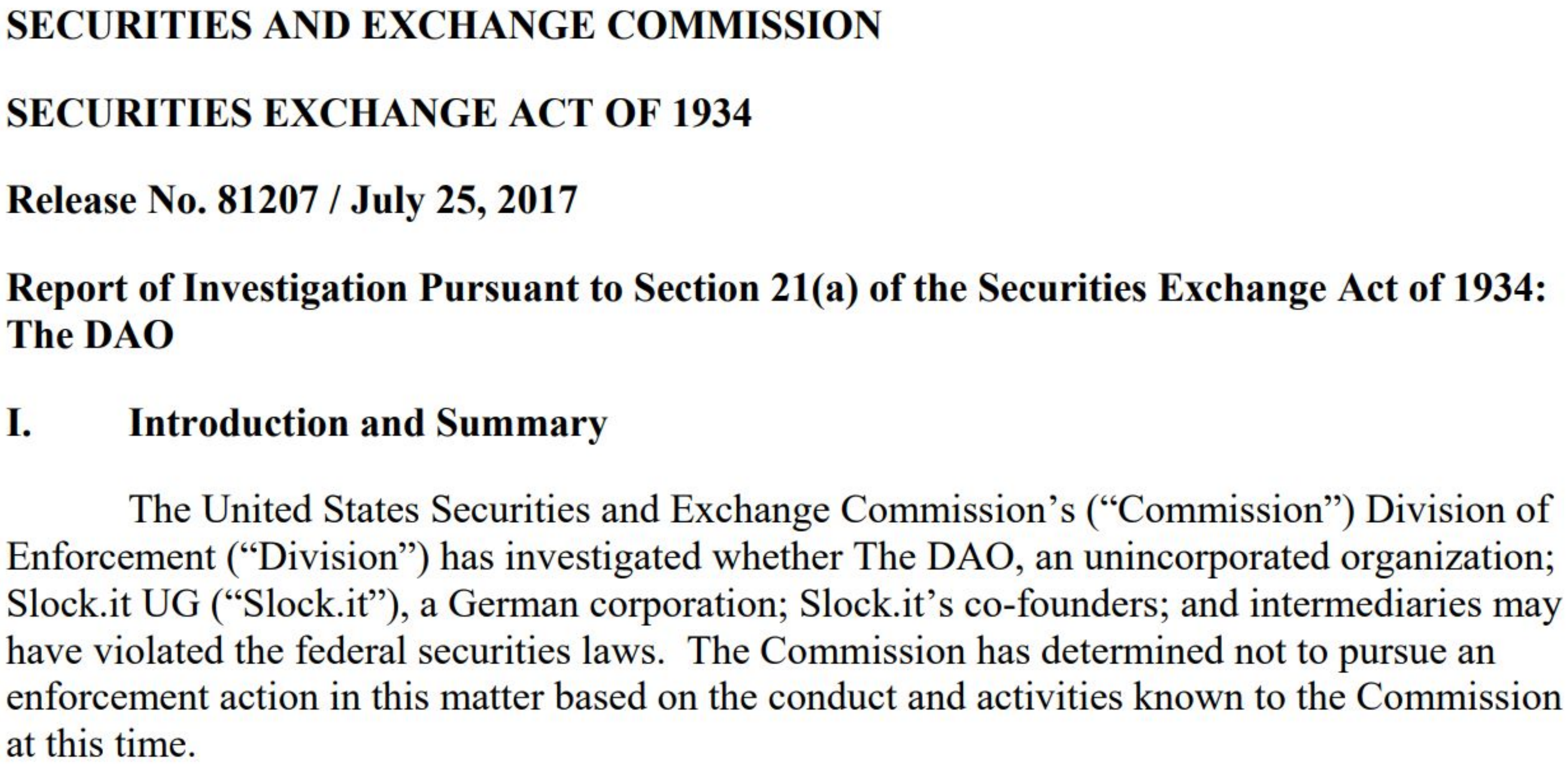
Timeline and Aftermath of The DAO

- June 12: slock.it developers announce that the 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:   Cornell freshmen identify 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: The 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 time.

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

How has the Ethereum community approached security improvements?

More resources on Smart Contract Security

- ConsenSys's “Smart Contract Best Practices”
- K EVM Semantics http://hdl.handle.net/2142/97207
- Oyente tool
    Symbolic engine catches some errors “Making Smart Contracts Smarter”
- Formal Verification of Smart Contracts - compiler written/checked in F*
    Includes a verified Decompiler and proofs of gas bounds
- A survey of attacks on Ethereum smart contracts
- An empirical analysis of smart contracts: platforms, applications, and design patterns
- Step by Step Towards a Safe Smart Contract
- Open Zeppelin https://openzeppelin.org/
- Under-optimized contracts devour your money https://arxiv.org/pdf/1703.03994.pdf

The “Checks / Effects / Interactions” paradigm

A Best Practice guideline for safe smart contract behavior

When receiving a message, do the following in order:

1. Perform all input validation and checks on current state. Discard the message if validation fails.
2. Update local state.
3. Finally, pass on interactions to trigger other contracts.

Contract A:

```
public address callee;
public int balance = 100;
…


function withdraw()
only(callee) {
 if (balance <= 0) return;
 var toSend = balance;
 balance = 0;
 callee.recv.value(toSend)();
}
```

```
contract MyBank {
  mapping (address ⇒ uint) balances;

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

  function Withdraw(uint amount) {
    if(balances[msg.sender] ≥ amount) {
      msg.sender.send(amount);
      balances[msg.sender] −= amount;
    }
  }

  function Balance() constant returns(uint) {
    return balances[msg.sender];
  }
}
```

```
module MyBank
open Solidity

type state = { balances: mapping address uint; }
val store : state = {balances = ref empty_map}

let deposit () : Eth unit =
  update_map store.balances msg.sender
      (add (lookup store.balances msg.sender) msg.value)

let withdraw (amount:uint) : Eth unit =
  if (ge (lookup store.balances msg.sender) amount) then
    send msg.sender amount;
    update_map store.balances msg.sender
       (sub (lookup store.balances msg.sender) amount)

let balance () : Eth uint =
  lookup store.balances msg.sender
```

Figure 3. A simple bank contract in Solidity translated to F*

```
let x_29 = pow [0x02uy] [0xA0uy] in
let x_30 = sub x_29 [0x01uy] in
let x_31 = get_caller () in
let x_32 = land x_31 x_30 in
burn 17 (* opcodes: SUB, CALLER, AND, PUSH1 00, SWAP1, DUP2 *);
mstore [0x00uy] x_32;
burn 9 (* opcodes: PUSH1 20, DUP2, DUP2 *);
mstore [0x20uy] [0x00uy];
burn 9 (* opcodes: PUSH1 40, SWAP1, SWAP2 *);
let x_33 = sha3 [0x00uy] [0x40uy] in
let x_34 = sload x_33 in
burn 9 (* opcodes: PUSH1 60, SWAP1, DUP2 *);
mstore [0x60uy] x_34;
loadLocal [0x60uy] [0x20uy] (* returned value *)
```

Figure 4. Decompiled version of the `Balance` method of the `MyBank` contract, instrumented with gas consumption.

Iterative improvements to Solidity

- “Modifier” macros

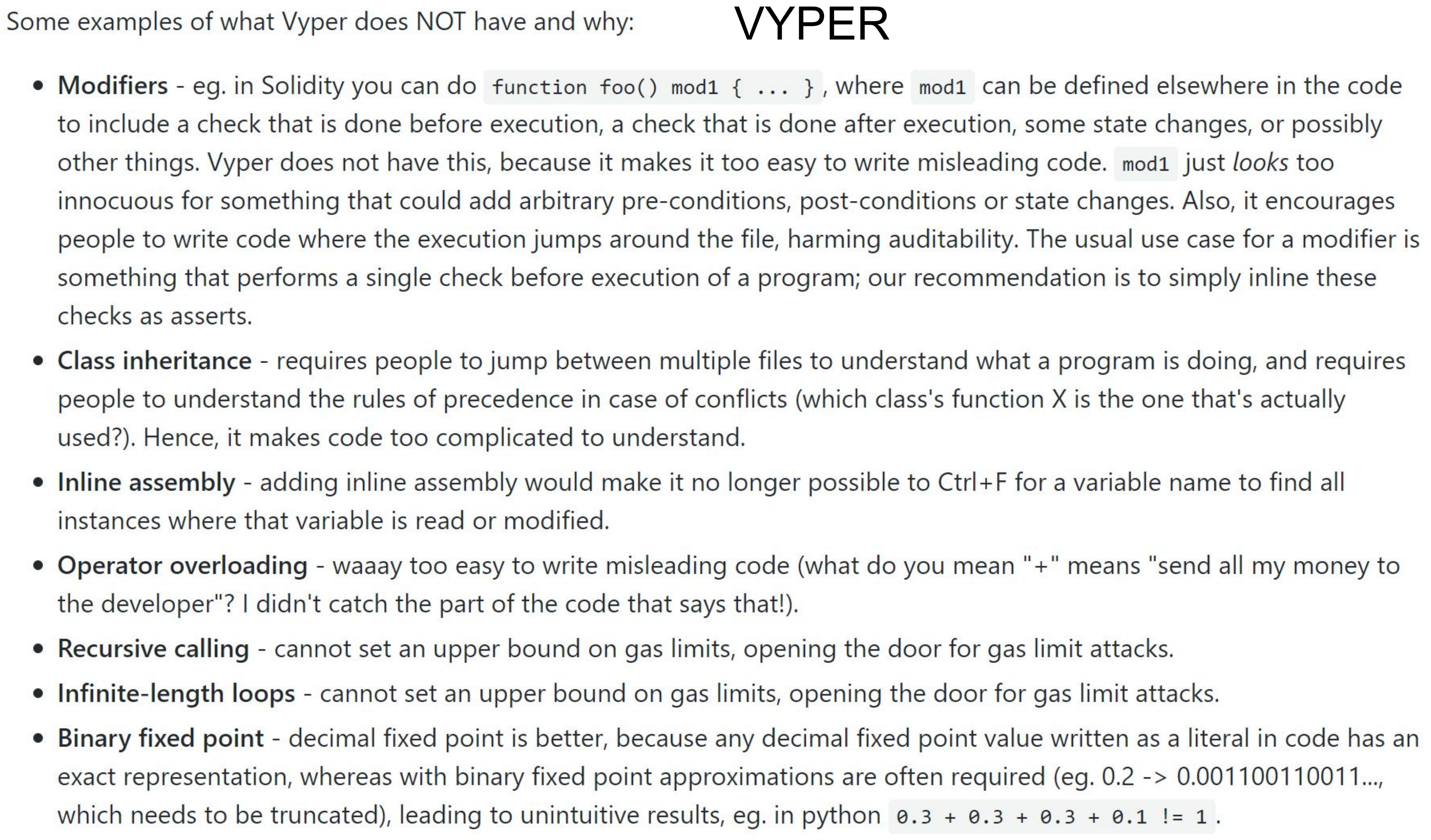
  Language syntax that explicitly indicates checked preconditions

```
function refundInvestors() _only(admins) {  … }
```

- “payable” modifier

  In modern Solidity, functions by default refuse payments, return to sender

VYPER



Some examples of what Vyper does NOT have and why:

- **Modifiers** - eg. in Solidity you can do `function foo() mod1 { ... }`, where `mod1` can be defined elsewhere in the code to include a check that is done before execution, a check that is done after execution, some state changes, or possibly other things. Vyper does not have this, because it makes it too easy to write misleading code. `mod1` just *looks* too innocuous for something that could add arbitrary pre-conditions, post-conditions or state changes. Also, it encourages people to write code where the execution jumps around the file, harming auditability. The usual use case for a modifier is something that performs a single check before execution of a program; our recommendation is to simply inline these checks as asserts.
- **Class inheritance** - requires people to jump between multiple files to understand what a program is doing, and requires people to understand the rules of precedence in case of conflicts (which class's function X is the one that's actually used?). Hence, it makes code too complicated to understand.
- **Inline assembly** - adding inline assembly would make it no longer possible to Ctrl+F for a variable name to find all instances where that variable is read or modified.
- **Operator overloading** - waaay too easy to write misleading code (what do you mean "+" means "send all my money to the developer"? I didn't catch the part of the code that says that!).
- **Recursive calling** - cannot set an upper bound on gas limits, opening the door for gas limit attacks.
- **Infinite-length loops** - cannot set an upper bound on gas limits, opening the door for gas limit attacks.
- **Binary fixed point** - decimal fixed point is better, because any decimal fixed point value written as a literal in code has an exact representation, whereas with binary fixed point approximations are often required (eg. 0.2 -> 0.001100110011..., which needs to be truncated), leading to unintuitive results, eg. in python `0.3 + 0.3 + 0.3 + 0.1 != 1`.