Imperial College London

A Layered Approach to Improving Blockchain Systems Security

Daniel Perez

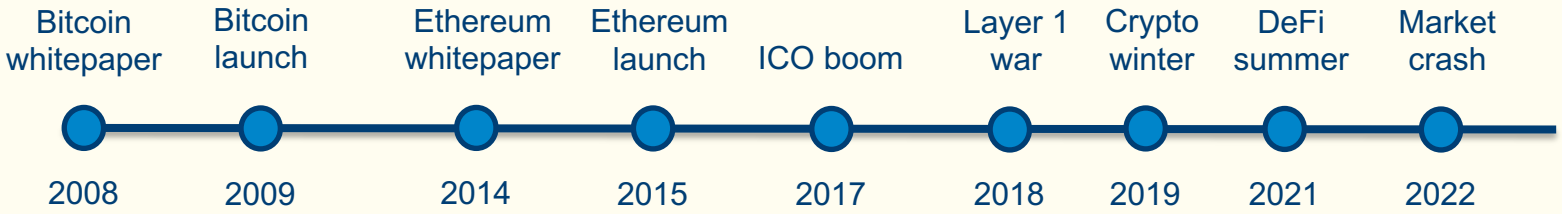
16th May 2023

Overview

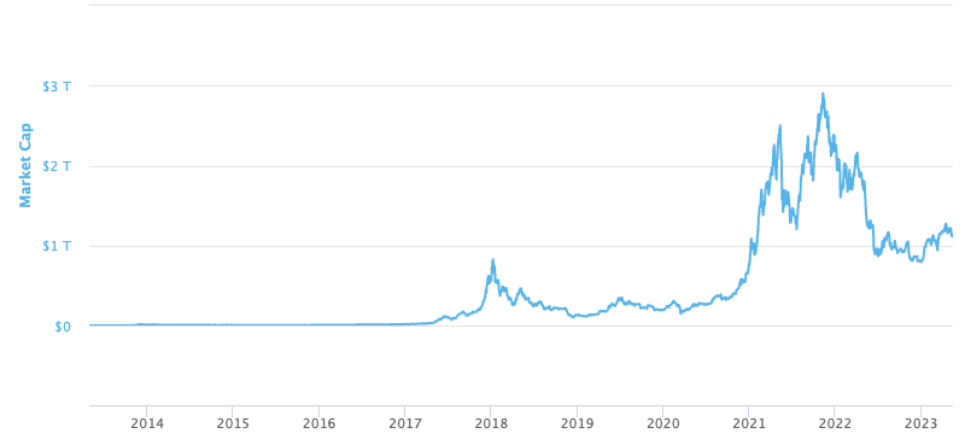
1. Introduction
2. Background
3. Execution Layer Security
4. Transactional Layer Security
5. Application Layer Security
    a) Technical Security
    b) Economic Security
6. Conclusion

Introduction

Blockchain Systems Timeline

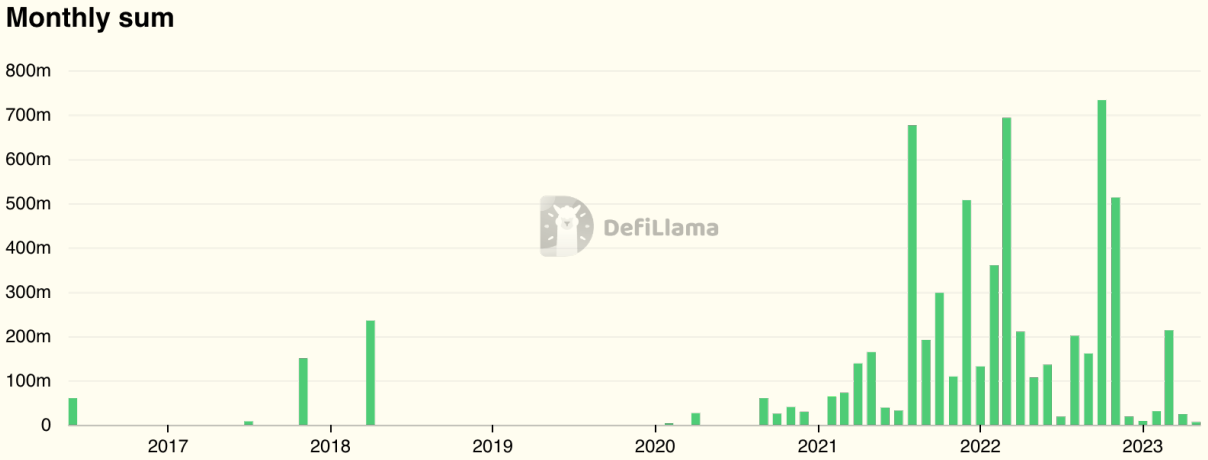


Total Market Cap of Crytpo



Source: CoinMarketCap

Total Amount of Money Hacked



Source: DefiLlama

Thesis Objective

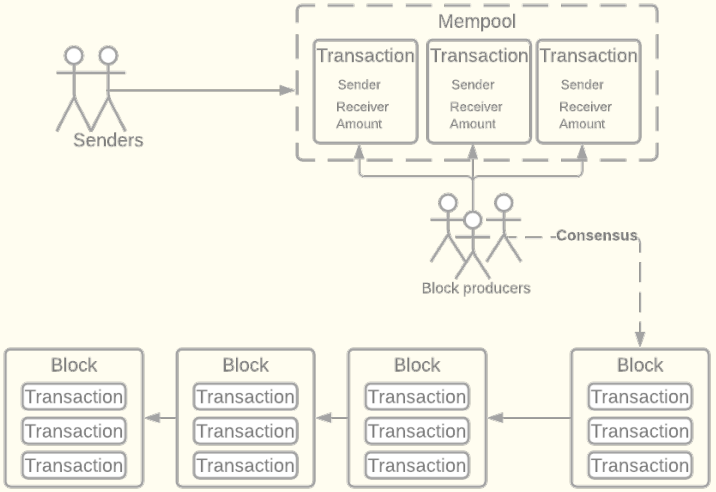
Improve the security of blockchain systems

- Blockchain systems are too large to be studied at once
- We split our study into different layers of the stack
  - Execution Layer
  - Transactional Layer
  - Application Layer

Background

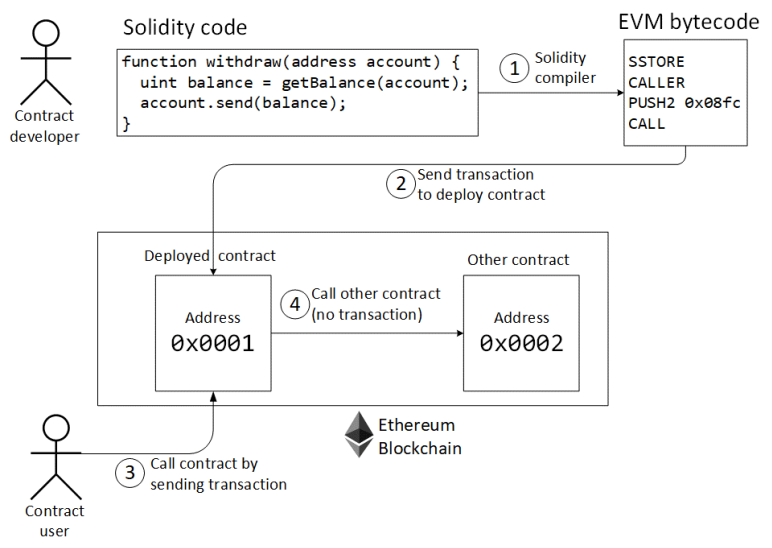
Blockchain Overview

- Append-only data structures
- Unit-of-work is a block
- Each block has many transactions
- Each transaction can contain several actions
- Block producers (miners) need to reach consensus



Ethereum Smart Contracts

- Programs deployed on the Ethereum blockchain
- Usually written in Solidity, compiled into EVM bytecode
- Can transfer money to other addresses (including contracts)
- Each instruction execution consumes gas



Solidity

- High-level language targeting the EVM
- Looks vaguely like JavaScript
- Strongly typed, with a fairly simple type-system
- Contains smart contract related primitives

```
contract Coin {
  address public minter;
  mapping (address => uint) public balances;
  constructor() public { minter = msg.sender; }
  function mint(address receiver, uint amount) public {
    require(msg.sender == minter);
    require(amount < 1e60);
    balances[receiver] += amount;
  }
  function send(address receiver, uint amount) public {
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[receiver] += amount;
  }
}
```

Contract implementing a simple coin

Ethereum Virtual Machine

- Stack-based virtual machine
- Very low-level
  - No functions/block, only jumps
  - No types
- Has regular VM instructions
  - `ADD, SUB, PUSH, POP`
- Has instructions to interact with environment
  - `SENDER, CALL, BALANCE`
- Has both ephemeral and permanent storage
- Uses 256-bits words

```
PUSH1 0x00
PUSH1 0x00
MSTORE
JUMPDEST
PUSH1 0x0a
PUSH1 0x00
MLOAD
PUSH1 0x01
ADD
DUP1
PUSH1 0x00
MSTORE
LT
PUSH1 0x05
JUMPI
```

Loop from 0 to 10 with EVM opcodes

Execution Layer Security

Broken Metre: Attacking Resource Metering in EVM (NDSS 2020)

Overview

- Ethereum's security relies on "correct" metering of code execution
- We analyse the "correctness" of Ethereum's metering mechanism
- We design an attack on Ethereum's metering and show how it can be exploited

Gas Metering

- Each instruction consumes gas to execute
- Instruction gas cost is fixed or computed depending on arguments/state
- Program gas cost = base cost + sum of instructions cost
- Program stops if it runs over its gas budget
- Transaction sender choses gas price and pays “gas cost x gas price”

Sample gas costs

- ADD: 3
- MUL: 5
- JUMPI: 10
- EXTCODESIZE: 700
- BALANCE: 700
- SSTORE
  - Allocation: 20,000
  - Modification: 5,000
  - Deallocation: -15,000

Empirical Analysis

Analysis Setup

- Fork aleth (C++ client)
- Instrument CPU
  - Record execution time/instruction
  - Aggregate over 1,000 instructions
- Instrument memory
  - Override new/delete
- Replay transactions and record stats

Arithmetic Instructions

Instruction	Gas Cost	Count	Mean Time (ns)	Throughput (gas/µs)
ADD	3	453,069	82.20	36.50
MUL	5	62,818	96.96	51.57
DIV	5	107,992	476.23	**10.50**
EXP	~51	186,004	287.93	**177.1**

Gas and Resources Correlation

- Compute correlation between gas usage and different resources
- Correlation with CPU (execution time) alone is non-existent
- Adding **CPU decreases** the **correlation** with gas

Phase	Resource	Correlation
Pre EIP-150	Memory	0.545
	CPU	0.528
	Storage	0.775
	Storage/Memory	**0.845**
	Storage/Memory/CPU	0.759
Post EIP-150	Memory	0.755
	CPU	0.507
	Storage	0.907
	Storage/Memory	**0.938**
	Storage/Memory/CPU	0.893

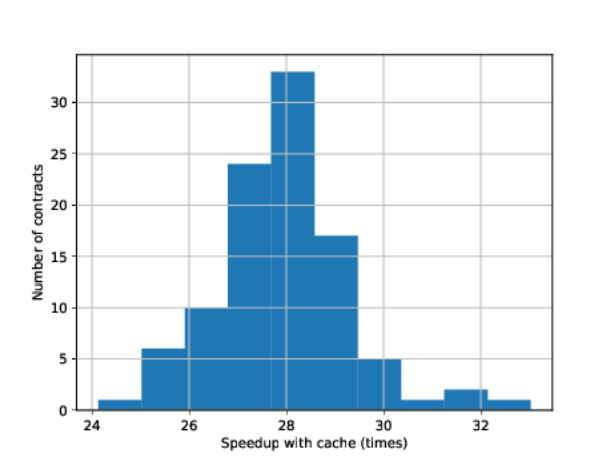
High-Variance Instructions

- Most high-variance instructions **depend on state**
- Even when aggregated over 1,000 calls, standard deviation is close to mean

Instruction	Mean (µs)	Stdev
BLOCKHASH	768	578
BALANCE	762	449
SLOAD	514	402
EXTCODECOPY	403	361
EXTCODESIZE	221	245

Effect of Cache on Execution Time

- Focus on OS page cache
- Generate random programs and measure speed with and without cache
- Programs run on average **28 times faster** with page cache



Analysis Summary

- **Gas cost**: **Many inconsistencies**
- **IO operations**: very **high execution time variance**
- **Cache**: very important effect on speed
- **Overall**: **cannot model IO operations** very well

Attacking EVM Metering

Resource Exhaustion Attack

- Goal is to find programs which minimize throughput (gas / second)
- Can be formulated as a search problem
  - Search space: Set of valid programs
  - Function to optimize: throughput
  - Constraint: gas budget
- Search space is too large to be explored entirely
  - We use a **genetic algorithm** to approximate a solution

Generated Programs

- We create programs valid by construction
  - Enough elements on stack
  - No stack overflows
  - Only access “reasonable” memory locations
- Cross-over and mutations also only create valid programs
- Generated programs do not contains loop
  - i.e. we do not include JUMP or JUMPI instructions

Initial Program Construction

- **Good initialization values** are important to converge in reasonable time
- To create initial program, we sample instructions as follow: given set of instructions *I*, we define the weight and probability of choosing an instruction with

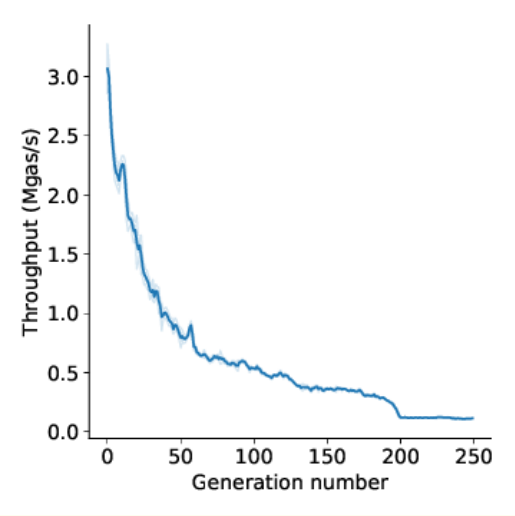
$$W(i \in I) = \log\left(1 + \frac{1}{throughput(i)}\right)$$

$$P(i \in I) = \frac{W(i)}{\sum_{i' \in I} W(i')}$$

Genetic Algorithm Results

- Initial program throughput: ~3M gas/s (compared to 20M on average)
- Decreases quickly to 500K
- Plateau at **~100K gas/s** at generation 200

200x slower than average contract



Evaluation on Different Clients

Client	Throughput (gas/s)	Time (s)	IO load (MB/s)
Aleth	107,349	93.6	9.12
Parity	210,746	47.1	10.0
Geth	131,053	75.6	6.57
Parity (bare-metal)	542,702	18.2	17.2
Geth (fixed)	3,021,038	3.33	0.72

Evaluation of different clients when executing 10M (1 block) gas worth of malicious transactions

DoS potential

- Implications
  - Nodes not being able to sync
  - Decrease in network throughput
- Probable attackers
  - Miners (selfish-mining)
  - Parties hostile to Ethereum (other chains)
  - Speculators
- Feasibility
  - Cost **only ~0.7 USD** to keep commodity hardware node out-of-sync for 1 block (~2M gas/block)
- Limitations
  - Current attack works best on commodity hardware
  - Hard to know what hardware full nodes are running

Responsible Disclosure

- 2019/10/3: Sent report to Ethereum Foundation through bounty program (thanks to Matthias Egli and Hubert Ritzdorf from PwC Switzerland)
- 2019/10/4: Reply from Ethereum Foundation
- 2019/10 – 2019/11: Tests with ongoing fixes
- 2019/11/17: Ethereum Foundation confirmed reward of 5000 USD
- 2020/1/7: Official bounty reward announcement

Improving Metering

Short term

- Increase cost of IO operations
  - Already seen in EIP 150 or EIP 2200
- Reduce number of required IO accesses
  - Flattened contracts state
  - Bloom filter to reduce search of inexistent contracts

Long term

- Stateless clients
  - Client do not need to keep track of all the state
  - Necessary data is sent with the transactions
- Sharding
  - Not a direct solution but less state needed per node

Summary

- Re-execute several months of transactions and measure gas, CPU and memory consumption
  - Find several inconsistencies
  - Show the impact of caching on execution speed
- Present a new attack targeted at metering
  - Show that the attack works on all major clients
  - Disclosed attack to Ethereum Foundation and tested fixes
- Notable outcomes:
  - 5k bug bounty from the Ethereum Foundation
  - Cited in EIP 2929 that proposes to update gas costs partly based on our findings

Transactional Layer Security

Revisiting Transactional Statistics of High-scalability Blockchain (Internet Measurement Conference 2020)

Overview

- Many blockchains have been designed and developed to improve blockchain scalability
- Most of these blockchains use different ways of pricing transactions
- We measure how such “scalable” systems end up being used in practice
- We show how cheap fees can backfire on the systems

Background

Blockchain Scalability

- Scalability often discussed in terms of transactions per second
- Average transactions per second can be computed by blocks/second x transactions/block

Platform	Block time	Txs/block	Txs/second
Visa	-	-	65,000
Bitcoin	10 minutes	3,500	6
Tezos	1 minute	2,400	40
EOS	0.5 second	2,000	4,000
XRP	?	?	65,000

Tezos

- Decentralized, open-source blockchain network for assets and applications.
- Utilizes a unique proof-of-stake consensus model and has on-chain governance.
- Self-amending protocol allows Tezos to upgrade without needing to fork.
- Focuses on smart contracts and formal verification, ensuring code correctness.

EOS

- A decentralized blockchain-based system designed to support decentralized applications (DApps).
- Aims to solve scalability issues of blockchain and provide a more user-friendly experience.
- Uses Delegated Proof-of-Stake (DPoS) system, allowing for better scalability.
- Focuses on flexibility and regulatory compliance through constitution-like set of rules.

XRP

- Digital asset and technology introduced by Ripple Labs.
- Primarily used for faster and low-cost international transactions.
- Not mined, unlike Bitcoin and Ethereum. All XRP tokens were pre-mined.
- RippleNet, the network on which XRP operates, is used by institutions like banks for efficient financial transactions.

Data Overview

Data Collected

- Data from October 1, 2019 to April 30, 2020
- Grouped in 6-hour interval for TPS computation

Platform	Number of blocks	Number of transactions	Avg. TPS	Max TPS
EOS	36,133,709	631,445,236	34	136
Tezos	301,822	7,890,133	0.43	0.57
XRP	4,753,965	271,546,797	15	56

Data and measurement framework open-sourced:
https://github.com/danhper/blockchain-analyzer

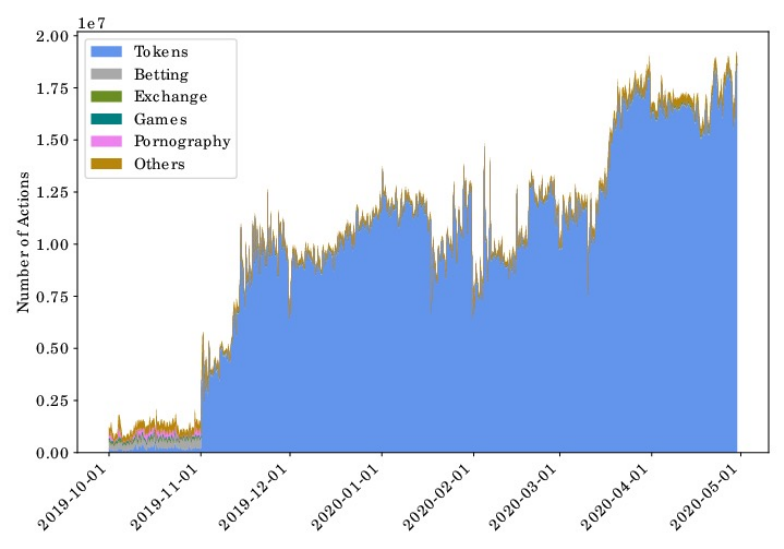
Types of Transactions

	EOS		Tezos		XRP	
Category	Action Name	%	Operation Kind	%	Transaction Type	%
P2P Transactions	Transfer	**96.2**	Transaction	21.4	Payment	**36.9**
					EscrowFinish	0.0
Account actions	newaccount	0.0	Reveal	0.0	TrustSet	1.2
	bidname	0.0	Origination	1.3	AccountSet	0.1
Other actions	delegatebw	0.0	Endorsement	**76.6**	OfferCreate	**59.1**
	undelegatebw	0.0	Delegation	0.6	OfferCancel	2.7
	buyrambytes	0.0	Reveal nonce	0.1	EscrowCreate	0.0
	Others	3.8			EnableAmendment	0.0

EOS Results

EOS Throughput over time

- Change of trend in November 2019
- Due to airdrop of EIDOS token
- Caused EOS to enter congestion mode

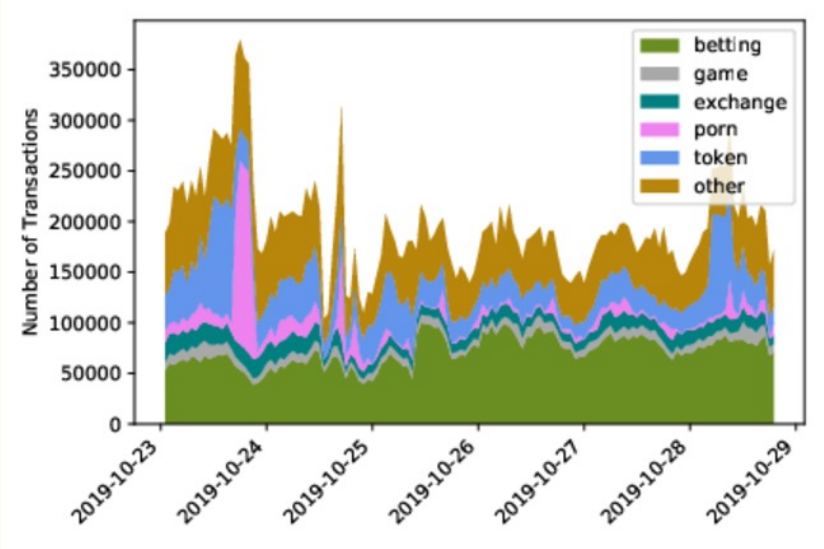


EIDOS Token Airdrop

- Very simple airdrop
  - Send EOS to a contract
  - Contract sends back the EOS and some amount of its token
- Originator of the airdrop is anonymous but tone is aggressive
- Likely a voluntary spam attack
- Caused many developers to quit EOS

EOS Throughput before spam attack

- Used a lot for betting games
- Used for some porn website payments
- Fairly stable and varied usage



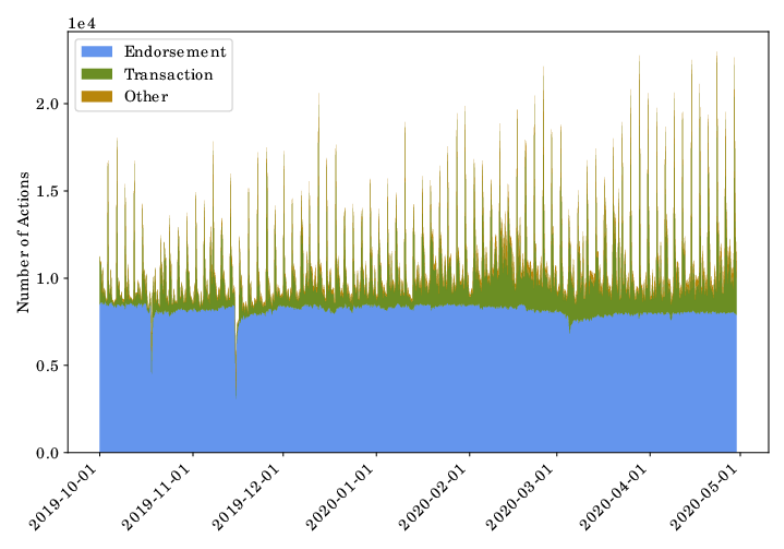
Wash-trading on EOS

- WhaleEx was most active exchange at time of analysis
- Claims to be the largest decentralized exchange in the world
- In **75%** of the trades, the buyer and seller were the same user
- The exact same amount was sent back-and-forth
- Aggregated trades for all top users resulted in 0 value transfer

Tezos Results

Tezos Throughput over time

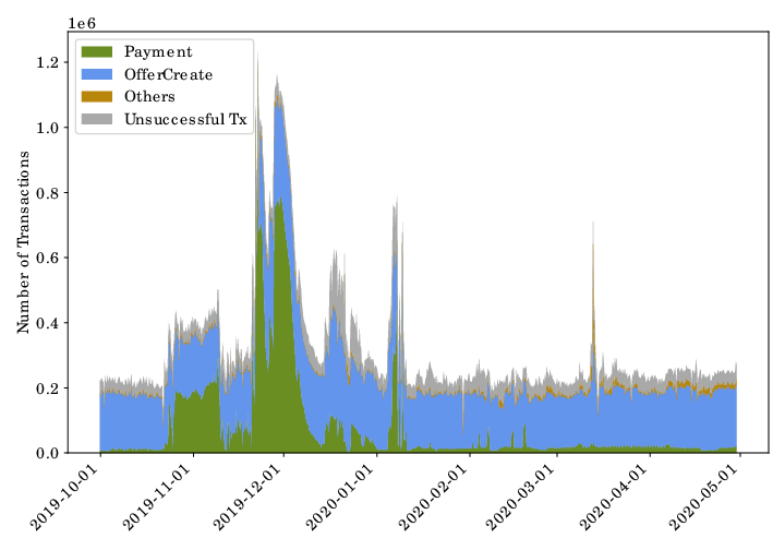
- Very stable trend
- Vastly dominated by endorsement - consensus related
- Regular spikes in number of transactions
- Transaction spikes due to block producer payouts - consensus related



XRP Results

XRP Throughput over time

- Dominated by OfferCreate
- Spike around November and December due to spam attack
- Stable throughput from January and on

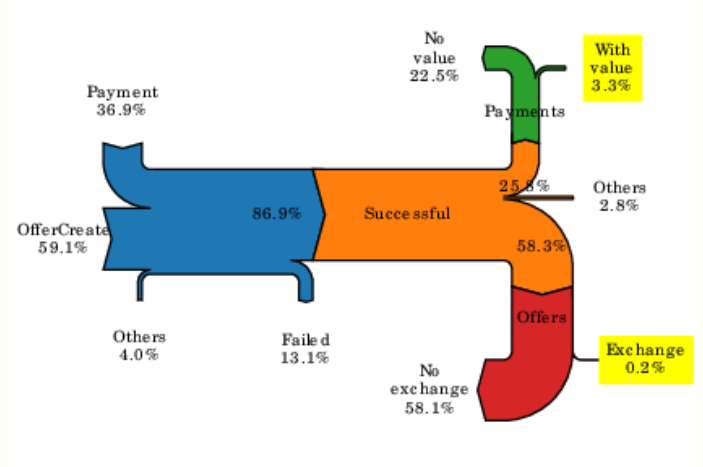


I Owe You (IOU) in XRP

- Anybody can emit an IOU
- User can establish trust line to accept IOUs from other users
- If the IOU issuer cannot pay, the IOU has no value

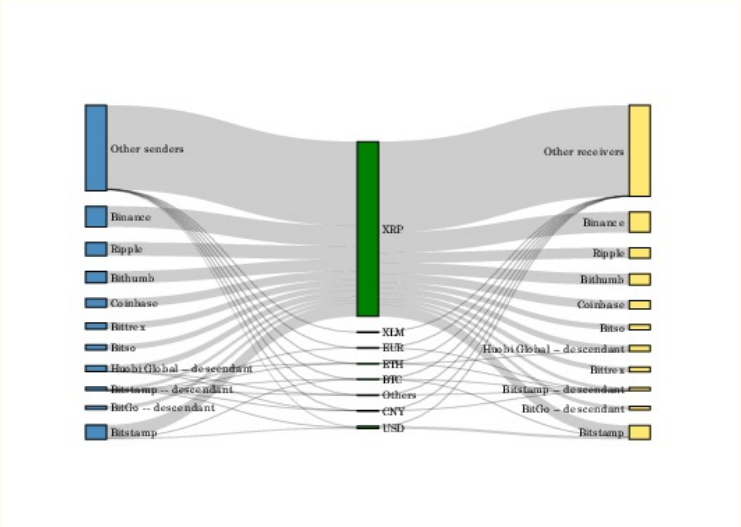
Value in XRP transactions

- Less than **6%** of the transactions had an economic value
- Only 0.2% of the transactions were successful trades
- Only 3.3% of the transactions were payments with an economic value



Valuable payments on XRP

- Vast majority of valuable payments are in XRP
- Average of 586 million XRP traded per day
- Exchanges account for a large portion of these payments



Takeaways

- All three blockchains are still very far from their potential
  - Current TPS very far away from claimed maximum
  - Most transactions have no economic value
    - **96%** of EOS transactions from valueless airdrop token
    - **76%** of Tezos transactions to maintain consensus
    - **94%** of XRP transactions have no economic value
- Low or absent fees tend to attract a lot of spam
  - XRP has dealt with spam attacks much better than EOS, likely because of the difference in fee model

Application Layer Security

SoK: Decentralized Finance (DeFi)

(Advances in Financial Technologies 2022)

Blockchain Applications

- **Decentralized Autonomous Organizations (DAOs)**: decentralized organization that uses a blockchain to facilitate the management of its funds and decision-making process.
- **Decentralized Finance (DeFi)**: peer-to-peer financial system powered by a blockchain designed to operate without the need for traditional financial intermediaries such as banks, brokerages, or exchanges.
- **Others**: non-fungible tokens (NFT), decentralized identity management, decentralized storage, decentralized social networks, etc…

Technical Security

- Technical security is about whether an on-chain system can be exploited within a **single tx** or a **bundle of txs** in a block
- Technical attacks are risk-free b/c outcomes are binary for attacker
  - Either attack is successful = profit
  - Or it reverts = only pay gas
- Examples: atomic MEV, sandwich attacks, reentrancy, logic bugs
- Best addressed: program analysis, formal models to specify protocols

Economic Security

- Economic security is about an exploiting agent who tries to manipulate the incentive structure of the protocol to profit (e.g., by stealing assets)
- Economic exploits are non-atomic
- They have upfront tangible costs and are **not risk-free**
  - The attack may fail depending what else happens in the time period
  - The attacker may mis-estimate the market response
- To address: needs economic models of how these systems and agents work

Technical Security

Smart Contract Vulnerabilities: Vulnerable Does Not Implies Exploited (USENIX Security 2021)

Overview

- Many analysis tools for smart contracts have been developed
- Due to false-positives, the amount of “exploitable” funds is vastly over-estimated
- We analyze the Ethereum blockchain to look for **actual exploitation**

Goals

- Understand better which vulnerabilities are exploited in practice
- Quantify how much has actually been exploited
- Understand why funds “at risk” are not exploited
- Get insights on where to focus to improve smart contracts security

Background

Vulnerabilities covered

- Re-entrancy
  - Can allow an attacker to drain funds
- Unhandled exceptions
  - Can result in lost funds
- Dependency on destructed contract
  - Can result in locked funds
- Transaction order dependency
  - Can allow an attacker to manipulate prices
- Integer overflow
  - Can result in locked fund

Re-entrancy

- Vulnerable contract sends money before updating state
- Attacker contract's fallback function is called
- Attacker contract makes re-entrant call to attacker

```
// vulnerable contract
function withdraw() public {
  uint256 amount = balance[msg.sender];
  // XXX: vulnerable
  if (!account.call{value: amount}())
    revert("transfer failed");
  balance[account] = 0;
}

// attacker contract
function drainVictim() external {
  victim.withdraw();
}
receive() external payable {
  if (i++ < 10) victim.withdraw();
}
```

Unhandled exceptions

- In Solidity, not all failed "calls" raise an exception
- If the failed call returns a boolean, it must be checked correctly
- Failure to do this could result in inconsistent state or even locked funds

```
// allows user to withdraw funds
function withdraw() external {
  uint256 amount = balance[msg.sender];
  balance[account] = 0;
  // could silently fail
  account.call{value: amount}();
}
```

Function lacking proper checks

Smart Contract Analysis Tools

- Usually static analysis and/or symbolic execution
- Work either on Solidity or on the EVM bytecode
- Check for known vulnerabilities/patterns

Securify web interface

Methodology

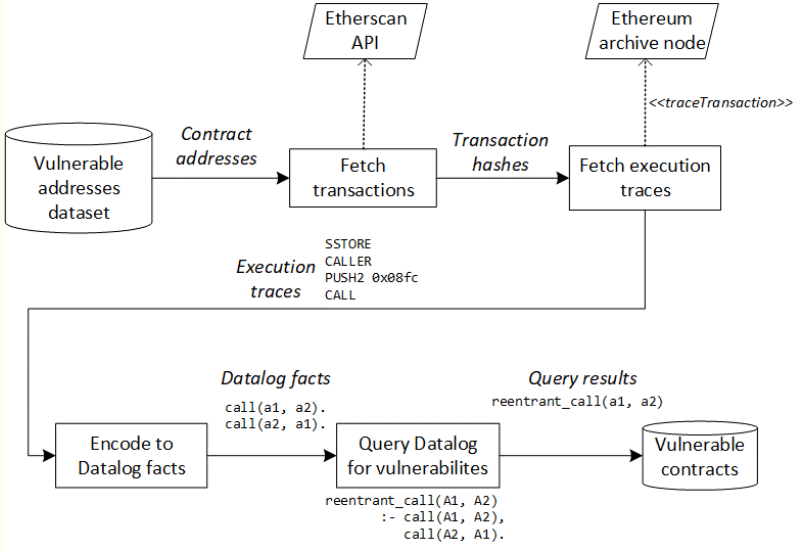
Dataset

- Data received from the paper authors
- Received replies from 5 out of 8 of the authors we contacted
- Ether at stake computed at time of the report
- Total of around **3M ETH** at stake

Name	Contracts analyzed	Vulnerabilities found	Ether at stake
Oyente	19,366	7,527	1,289,177
Zeus	1,120	855	729,376
Maian	NA	2,691	14.23
Securify	26,694	9,185	719,567
MadMax	91,800	6,039	1,114,692

Detection Overview

1. Retrieve all transactions
2. Retrieve execution traces for all transactions
3. Encode execution traces to Datalog
4. Query Datalog for vulnerabilities



Checking for re-entrancy

1. Record all calls
2. Check for mutually recursive calls between contracts

```
call_flow(A, B) :- call(A, B).
call_flow(A, B) :- call(A, C), call_flow(C, B).
reentrant_call(A, B) :- call_flow(A, B),
                        call_flow(B, A),
                        A != B.
```

Datalog rules

```
// vulnerable contract
function withdraw() public {
  uint256 amount = balance[msg.sender];
  // XXX: vulnerable
  if (!account.call{value: amount}())
    revert("transfer failed");
  balance[account] = 0;
}

// attacker contract
function drainVictim() external {
  victim.withdraw();
}
receive() external payable {
  if (i++ < 10) victim.withdraw();
}
```

Checking for unhandled exceptions

1. Record all call results: top value on the stack after a call
2. Check if the return value is used in a condition (`JUMPI`)

```
influences_condition(A) :- used_in_condition(A).
influences_condition(A) :-
  depends(B, A), used_in_condition(B).
unhandled_exception(A) :-
  failed_call(A), ~influences_condition(A).
```

Datalog rules

```
// allows user to withdraw funds
function withdraw() external {
  uint256 amount = balance[msg.sender];
  balance[account] = 0;
  // could silently fail
  account.call{value: amount}();
}
```

Function lacking proper checks

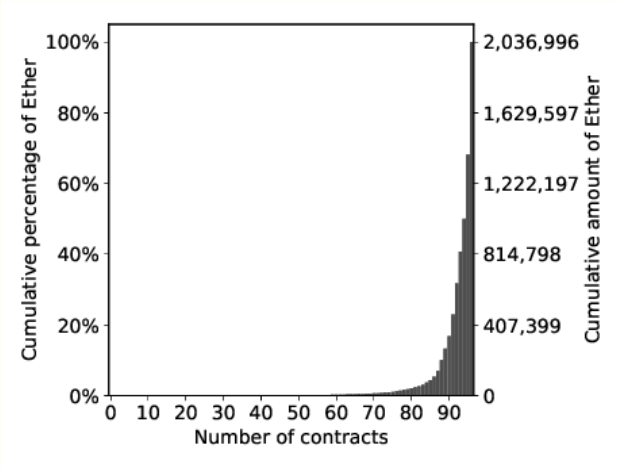
Results

Detection Results

Vulnerability	Vulnerable contracts	Total Ether at Stake	Contracts exploited (w/Ether)	Exploited Ether	% of Exploited Ether
Re-Entrancy	4,336	1,027,585	113 (6)	6,075	**0.59%**
Unhandled Exception	11,426	108,528	268 (6)	169	**0.081%**
Destructed contract dep.	7,271	1,135,313	0 (0)	0	**0%**
Transaction order dep.	1,877	107,926	57 (14)	189	**0.0091%**
Integer overflow	2,472	508,750	141 (23)	2,661	**0.52%**
Total	21,270	3,088,102	504 (49)	9,094	**0.30%**

Contracts wealth distribution

- Combined value: **~2 million ETH**
- Only **~2,000** out of 21,270 contracts **held Ether**
- Less than 100 contracts had more than 10 Ether
- The **top 6 contracts held 83%** of the total Ether



Cumulative Ether held in contracts holding more than 10 ETH

Vulnerable but not exploitable

- Many cases where "vulnerable" contracts are not exploitable
- High-value contracts flagged vulnerable are typically not exploitable
- Usually limitation due to the nature of static analysis

```
function removeOwner(address owner) onlyWallet {
  isOwner[owner] = false;
  // Could in theory run out of gas
  for (uint i=0; i<owners.length - 1; i++) {
    if (owners[i] == owner) {
      owners[i] = owners[owners.length - 1];
      break;
    }
  }
  // a bit more logic
}
```

Multisig wallet with >350K ETH at address `0x7da82C7AB4771ff031b66538D2fB9b0B047f6CF9`

Summary

- Analysed 21k contracts with **3M Ether at risk**
- **At most 0.3%** of this Ether, less than 10k ETH, was **exploited**
- Overall, **high-value contracts** seem to be quite **secure**

Economic Security

Liquidations: DeFi on a Knife-edge (Financial Cryptography 2021)

Overview

- Many DeFi protocols rely on over-collaterization to secure their system
- Over-collaterization relies on liquidations to work securely
- We liquidations analyze in-depth in one of these protocols
- We highlight the different risks associated with certain user behaviours

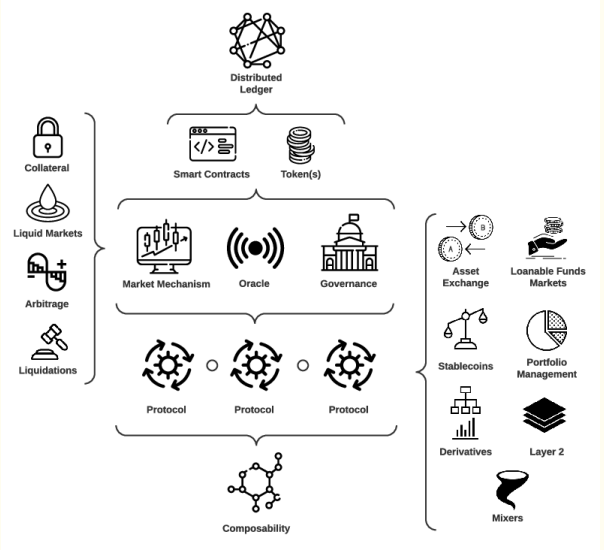
Background

What is DeFi?

Decentralized Finance is a peer-to-peer powered financial system.

Four properties:

- Non-custodial
- Permissionless
- Auditable
- Composable



On-chain building blocks

Oracles

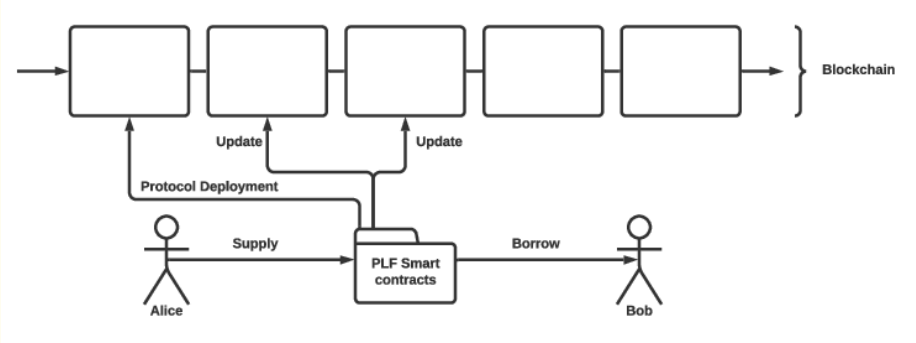
- Facilitate on-chain access to external information
- Implemented as smart contracts being regularly updated
- Often require some level of trust

Stablecoins

- Assets of which the price is pegged to a currency (e.g. USD)
- Can be implemented in very different ways (e.g. custodial vs non-custodial)
- Movements above/below target are not uncommon

Protocols for Loanable Funds (PLF)

- Protocol that intermediates funds between users
- Unlike peer-to-peer lending, funds are pooled
- Requires users to deposit collateral



PLF building blocks

- **Interest rate models**: some function(s) taking liquidity as an argument and returning an interest rate
- **Collateralization**: deposit that can be sold off to recover the debt of a defaulted position
- **Liquidations**: the process of selling a borrower's collateral to recover the debt value upon default
- **Governance mechanism**: decentralized governance typically achieved through an ERC-20 governance token, where token holders' votes are in proportion to their stake

PLF Use Cases

- **Earning interest**: Liquidity providers of funds are incentivized by accrued interest
- **Leveraged short position**: Borrowing funds of an asset expected to depreciate in value
- **Leveraged long position**: Increasing exposure to an asset expected to appreciate in value
- **Liquidity mining**: PLFs may distribute governance tokens to their users to incentivize liquidity providers and/or borrowers

Model and Methodology

PLF definitions

- **Market**: A smart contract acting as the intermediary of loanable funds for a particular crypto-asset, where users supply and borrow funds.
- **Supply Funds**: Deposited to a market that can be loaned out to other users and used as collateral against depositors' own borrow positions.
- **Borrow Funds**: Loaned out to users of a market.
- **Collateral Funds**: Available to back a user's aggregate borrow positions.
- **Locked funds**: Funds remaining in the PLF smart contracts, equal to the difference between supplied and borrowed funds.

Agents in the System

- **Supplier**: A user who deposits funds to a market.
- **Borrower**: A user who borrows funds from a market. Since a borrow position must be collateralized by deposited funds, a borrower must also be a supplier.
- **Liquidator**: A user who purchases a borrower's supply in a market when the borrower's collateral to borrow ratio falls below some threshold.

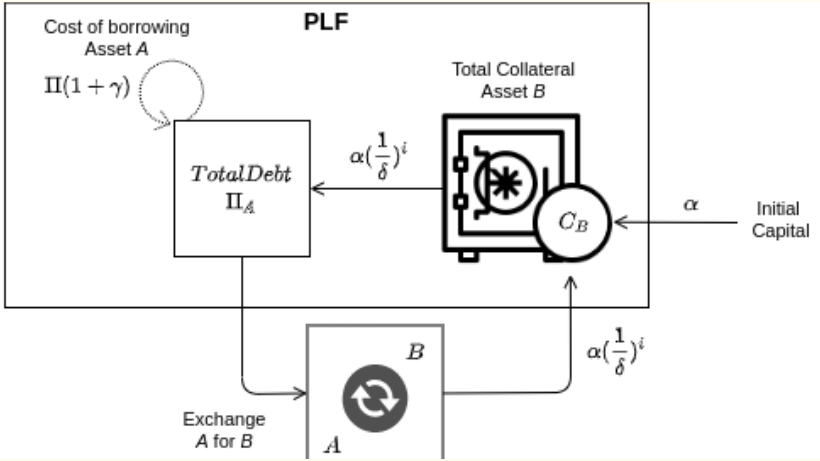
Conditions for liquidation

- All markets have a **collateral factor**, the ratio between supply and collateral funds
- When computing collateral and borrowed funds across markets, amounts are converted to a common currency, e.g. USD or ETH
- A user is liquidatable if the **sum of his borrows exceeds the sum of his collateral** across all markets

Leveraging on PLFs

Example steps for leveraging

1. Supply ETH on a PLF.
2. Leverage the deposited ETH to borrow DAI.
3. Sell the purchased DAI for ETH.
4. Repeat steps 1 to 3 as desired.



Steps of leveraging using a PLF

Analyzing Compound

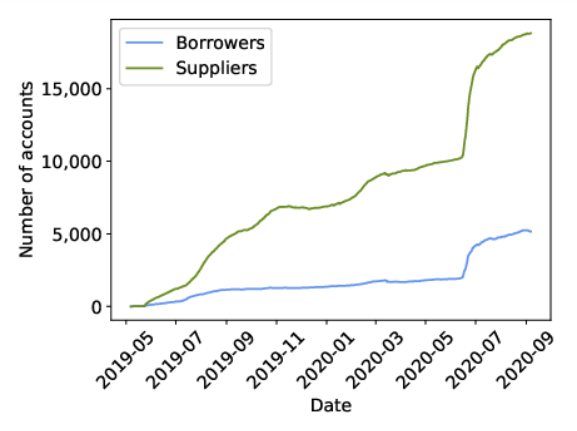
Event	Description	State change
Borrow	New borrow is created	Borrow
Mint	cTokens minted for deposit	Supply
RepayBorrow	Borrow is partially/fully repaid	Borrow
LiquidateBorrow	Borrow is liquidated	Supply & Borrow
Redeem	cTokens are used to redeem deposit of asset	Supply

Main events on compound

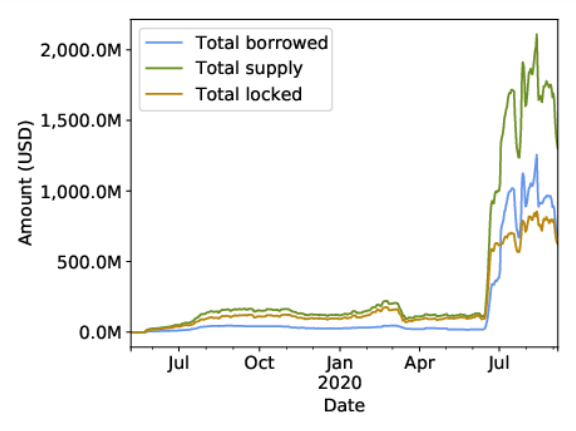
Analysis

Borrowers and Suppliers

Sharp increase when COMP rewards started to be distributed



Number of suppliers and borrowers

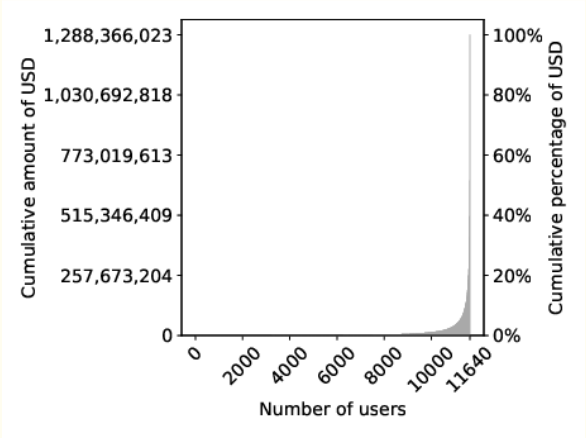


Amount of funds supplied, borrowed and locked

Distribution of funds

Top user accounts for **27.4%**

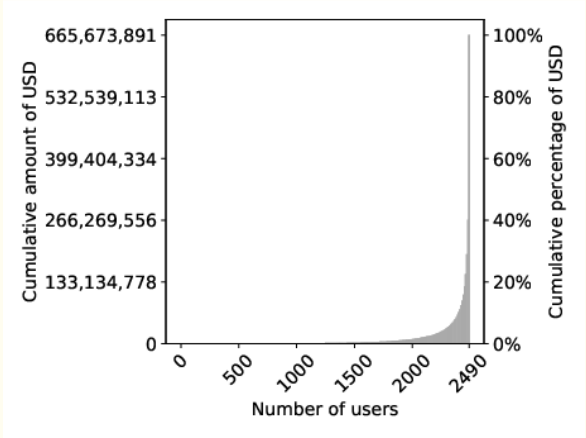
Top 10 users account for 49%



Distribution of supplied funds

Top user accounts for **37.1%**

Top 10 users account for 59.9%



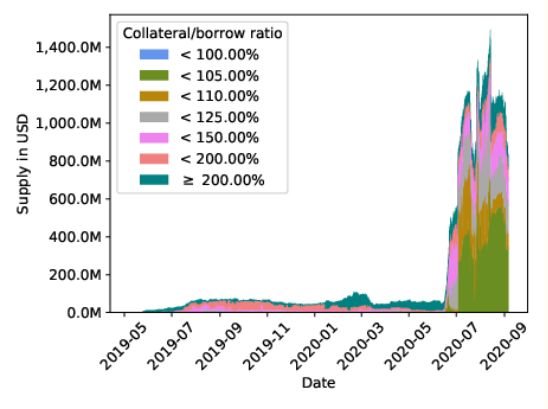
Distribution of borrowed funds

Leveraging Spirals

- Leveraging spirals is an important reason for concentration in top borrowers/suppliers
- Analysis of the top account
  - **Provided** in total **55M USD** to the protocol
  - Used spirals to **supply 342M USD** and **borrowed 247M**
- Analysis of all accounts
  - Over 2,100 accounts (40% of total number borrowers) use leveraging spirals
  - Over 600M USD (~50% of the total supply) is supplied using spirals

Liquidation Risk

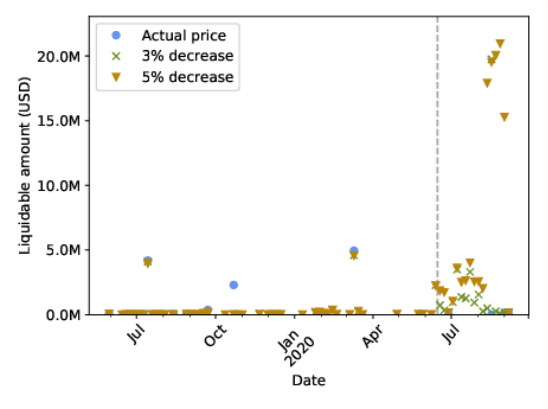
- COMP launch has changed users' behaviour
- Before launch, almost all users were at least 25% over their minimum collateral threshold
- After launch, more than 40% of the users were within 5%



Collateral locked over time, showing how close the amounts are from being liquidated

Price fluctuation and liquidation risk

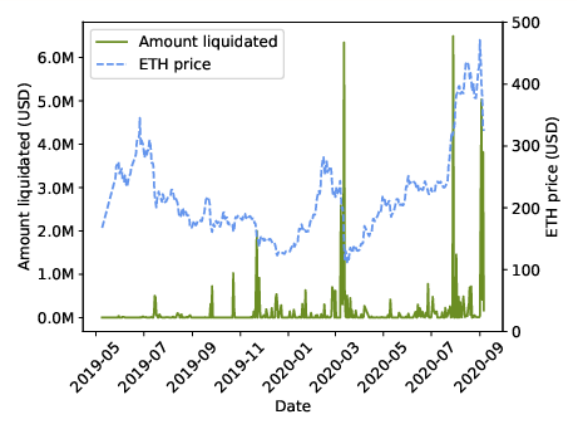
- Users rely on DAI being stable
- Small variations in DAI price can create large liquidations: 3% price change would have made more than 10M USD liquidatable
- This happened shortly after publication: over 88M USD liquidated



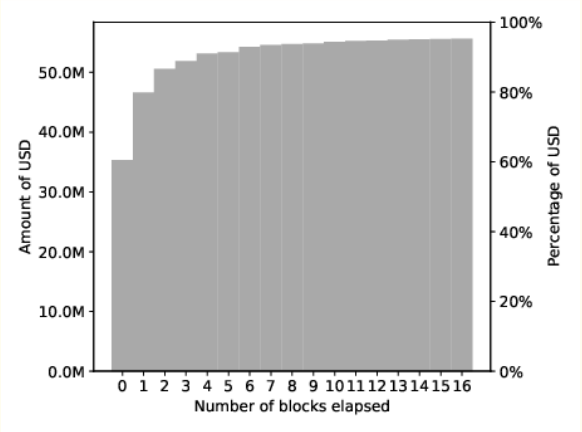
Sensitivity analysis of the liquidatable collateral amount given DAI price movement

Liquidations and Liquidators

Both liquidated amount and liquidation efficiency has increased with time



Liquidations over time



Number of blocks elapsed from the time a position can be liquidated to actual liquidation

Takeaways

- Governance token changed PLF users' behaviour significantly
  - A lot of liquidity was attracted but users took more and more risks
- Users tend to underestimate the volatility of stable coins
  - Small price deviations can lead to large liquidations
  - Large amounts were liquidated last November
- Liquidators are becoming more efficient
  - More than 70% of the liquidations happen in the block where positions became liquidatable

Conclusion

Conclusion

We made the following contributions, enhancing our understanding of the security of blockchain systems:

- **Execution Layer**: Revealed potential security issues in the EVM's resource metering mechanism, which subsequently led to significant updates by the Ethereum team.
- **Transactional Layer**: Identified the direct impacts of fee mechanisms on system performance and security, highlighting the double-edged sword of low fees and their tendency to induce spam.
- **Application Layer**
  - Developed an automated tool for on-chain exploit detection, providing insights into the underutilization of identified exploits due to feasibility and economic constraints.
  - Presented a comprehensive evaluation of DeFi's liquidation mechanisms, exposing their strengths and potential vulnerabilities, particularly during stablecoin depegging events.