Abstract

We propose expanding Ethereum's execution model to support Account Abstraction

(AA) or validation generalization. This makes contract accounts first class citizens; they may define their own validation function, submit their own transactions and pay gas on behalf of users, leading to a number of powerful usecases. In this study, we mainly evaluate the DoS vectors AA introduces and dive into its feasibility as part of inclusion in the Ethereum protocol. We define the DoS vulnerabilities in two categories: peer DoS

and block invalidation

. To evaluate the impact of these DoS vectors, we build AA into the Go Ethereum client and run a number of vulnerability tests. Through these results, we find the block invalidation vulnerability to have stronger network-wide effects than the peer DoS vulnerability. We evaluate a number of mitigation efforts and do find these to be sufficient. The block invalidation vulnerability impacts the transaction pool in varying degrees based on the maximum gas allowed for validating an AA transaction, but the slowdowns seem to be within reasonable bounds. Peer DoS vulnerabilities are already present within the network currently and do not seem to have similar network-wide effects. We do note that the increased validation time for AA transactions slows down transaction propagation through the network. However, the average transactions per second transaction pools currently receive and propagate appear to leave room for AA to scale (Table 2). To ease into the changes AA brings, we suggest a two phase release model: single-tenant

and multi-tenant

.

Introduction

Thanks to the Quilt Team (@matt, @adietrichs, @SamWilsn) and @vbuterin for their collaboration.

This post presents results from a data-driven study on the feasibility of adding AA to core protocols. AA introduces a number of DoS vectors and we evaluate the impact these DoS vectors would have on Go Ethereum. Before defnining these DoS vectors, we'll give a brief summary and background on AA.

Background

AA introduces validation generalization and contract sovereignty to blockchains. Contracts can define the rules of a transaction, including the structure and signature, and can pay gas on behalf of users who interact with the contract. Currently, validation is hardcoded to check an ECDSA signature, nonce and balance. AA puts the validation check in the power of contracts and does not require an EOA or other contract account to initiate an interaction (the contract can pay for gas). By use of AA, DEXes may institute their own signature and balance checks, mixers can maintain privacy, smart contracts & multisigs may pay for their own gas, and applications can onboard users without eth. Additionally, we see AA as a precursor to Execution Environments for eth2.

There are a number of other powerful usecases AA gives in addition to the ones listed above. This post does not go into those in depth, but for further background on the value AA may bring, review the following resources:

1. New AA EIP
2. Implementing Account Abstraction as Part of Eth1
3. Account Abstraction Link Tree

Implementation and Analysis Methods

We build AA into the Go Ethereum client with a number of interactive tools and data collection frameworks. Our specific implementation in this study is different from what we propose in the EIP. As a result of our learnings on this study and external feedback, we iterated on our design for the EIP.

Specification

: AA MVP spec originally described by Vitalik and later described in ourrationale document through step 2.

Implementation

: Our current implementation is in a fork of Go Ethereum.

Playground and Tooling

: We have provided a playground to write and deploy AA contracts that anyone can test.

Data Collection

: We built a library Bazooka to run predefined, repeatable tests across the wire to measure the impact of DoS attacks or other peer load on the system. We also extended geth to log results and load data into various databases for analysis. See data sources at the end of the document for further details.

EVM Updates

The AA MVP spec and the AA EIP thoroughly cover consensus updates required. For context in this study, we will only talk about the PAYGAS

opcode. Everything up to the PAYGAS

opcode in the contract is defined as the validation function. Transaction pools may configure a tolerance or gas cap (ie 400,000 gas). Transactions that do not call this opcode within the specified verification gas cap are treated similar to traditional transactions with invalid signatures and are dropped from the transaction pool. Before calling PAYGAS

, AA transactions cannot access external state other than precompiles.

Transaction Pool Updates

To accept and validate AA transactions, we introduce EVM execution to the transaction pool, with a configurable gas limit or gas cap. When running the validation function for a transaction, the run terminates if the gas cap has been exceeded. In our simulations, we only allow one AA transaction per AA contract in the transaction pool, however, the numbers in this writeup do not change if we accept multiple transactions up to the configured gas cap (ie 10 x 40k gas transactions vs one 400k gas transaction). Multiple transactions per account do add other complexity though, which we explore in the EIP.

Problem Statement

AA introduces and amplifies a number of DoS vectors to both the transaction pool and to the miners while populating a block with transactions. Can we provide mitigation to these DoS vectors and characterize the upper bounds for how significant these DoS vectors would be? We see two principal DoS vectors, and hence this post explores them and tests data for the DoS attack in detail:

1. Peer DoS or Transaction Propagation
2. Block Invalidation

Transaction Propagation and Peer DoS discuss slowdowns in the network a node must incur when validating new incoming AA transactions from its peers. Simply put, validation incurs a higher cost in AA.

Block invalidation discusses how transactions in incoming blocks may invalidate transactions in the transaction pool. Any incoming transaction to an AA contract could alter state (say a simple nonce) and thus render the pending transactions invalid to the same contract. While these transactions will simply be dropped in the most simple versions of single-tenant AA, under advanced AA they instead trigger re-validation, causing additional computational costs.

Network Transaction Propagation and Peer DoS

To evaluate AA feasibility, we must understand how AA affects transaction propagation in the network and whether it makes nodes more susceptible to a Denial-of-Service (DoS) attack from its peers in the process.

Transaction Propagation

Current Transaction Propagation (non-AA)

[

910×639 55.7 KB

](https://ethresear.ch/uploads/default/original/2X/0/0431d427e62d437be4190c5d48bbb73f6a779c2b.png)

Figure 1

(3)

The spike in the beginning is the process of populating the transaction pool once the node is synced. We remove duplicate transactions that come in from the dataset as it is trivial to dismiss them. The following represents various averages and percentiles:

Measurement

Transactions

0

Average (tx/s)

155.93

1

20th Percentile (tx/s)

66

2

80th Percentile (tx/s)

225

Table 1

(3)

This data was collected on 08/13/20-8/14/20 during a particularly high traffic period on the ethereum network.

Transaction Propagation (AA)

Transaction Type

Transactions Per Second

Validation Gas

Slow Down

0

EOA

2993.2

N/A

N/A

1

AA

2029.14

27451

1.5x

2

AA

1156.08

100000

2.6x

3

AA

530.14

200000

5.6x

4

AA

345.4

200000

8.7x

5

AA

255.47

400000

11.7x

Table 2

(3)

Table 2 shows the maximum transactions per second we are able to propagate and process in the transaction pool. Considering the current average transaction propagation rate is 155.93 tx/s on the network during a high traffic period, the measurements suggest there is room to scale for AA transactions.

Keep in mind, these numbers represent worst case scenarios. For example, we only propagate transactions which are not cached in the memory database. Additionally, we evaluate high gas transactions (100k - 400k) which likely would not be representative of most transactions through the network. For example, we estimate 30k gas for smart contract wallet validation. Also, our implementation is not optimized for production, so these numbers could likely see further improvements.

An area we did not cover and needs further exploration, discusses the slow downs in propagating a transaction to 80% of the network. If we see anywhere from 1.5x to 11.7x slowdowns as in Table 2, we need to evaluate how much longer it will take to pass to a sufficient number of nodes and how the throughput of AA transactions will be affected.

Peer DoS Vectors

Denial-of-Service (DoS) attacks are notoriously difficult to defend. This is due to the difficulty in identifying sybils within a peer list. At any moment, one may decide (or be bribed) to initiate an attack. This is not a problem that AA introduces. It can be accomplished today by inundating a target with transactions whose signatures are invalid. However, due to increased allotment of validation work allowed to AA – it's important to understand the worst case.

Existing (Non-AA) Peer DoS

Already, nodes are susceptible to peer DoS attacks. This section tries to summarize the existing issue for context, however, its network wide impact appears to be fairly limited.

To familiarize yourself more closely with the mechanism of transaction broadcasts,EIP 2464 gives a thorough explanation. Currently, up to 10MBs of recent validated transactions are accepted from any peer. These broadcasts may come inbound without the solicitation of the receiving node. A malicious peer can currently bombard other peers with 10MB of invalid transactions with no recourse (the node will not be dropped). We ran various simulations and share one here:

Setup

: 10 x 10MB packets of invalid transactions (no balance)

Transaction Quantity

: 952,380 total, 95,238 per packet

Delay Type

Time (s)

0

Total Delay

202.83

1

Pool Lock

4.48695

Table 3

AA Peer DoS

With AA, a peer could similarly bombard its victim with transactions requiring heavy validation requirements. We evaluate the impact of this scenario.

Setup

: 1 packet of 16k invalid transactions

Validation Gas

Pool Lock Time (s)

Total Delay Time (s)

0

27451

6.4

7.92

1

100000

12.34

13.84

2

200000

28.6

30.17

3

300000

44.72

46.33

4

400000

61.03

62.63

Table 4

Setup

: 1 packet of 1k invalid transactions

Validation Gas

Pool Lock Time (s)

Total Delay Time (s)

0

27451

0.37

0.46

1

100000

0.74

0.83

2

200000

1.76

1.86

3

300000

2.78

2.88

4

400000

3.77

3.86

Table 5

Although the numbers in Table 4 may seem high, it would likely require a malicious peer and exceeds the network traffic average measured during a particularly high traffic period of 155.93 tx/s. Additionally, none of these accounts were in the memory cache during testing, so worst case scenarios are displayed.

DoS Mitigation and Propagation Improvements

Single node DoS attacks are a current vulnerability in the ethereum network. These attacks impact a single peer, do not broadcast through the network, and do not appear to give many benefits to the attacker. When the victim node receives a batch of invalid transactions, it will not propagate them further to its other peers.

If mitigation is ever needed (for non-AA), a node may drop a peer broadcasting invalid transactions, rate limit incoming transactions, or decrease the accepted packet size.

Mitigation

To protect against anomalies, bursts of transactions or an attacker, AA transaction propagation would likely need basic rate limiting in its handler (proportional to the decided max validation gas accepted). Additionally, the receiving node would want to have a way to prioritize transactions by gas price in order to avoid further unecessary validation since it will not know the gas price until it reaches PAYGAS

.

Introducing new messages to the eth

protocol (see EIP 2464) for AA transactions can solve this:

AATransactionMsg[[tx, block_hash, gas_price], ...]]

AAPooledTransactionMsg[tx, block_hash, gas_price], ...]]

AAGetPooledTransactionMsg[tx], ...]]

Where AATransactionMsg

and AAPooledTransactionMsg

would broadcast with an associated block hash and gas price. If the block hash is outdated, the transaction may be discarded. If the block hash is not oudated and the transaction is still invalid, then the connecting peer is malicious and may be dropped. Additionally, syncing an empty transaction pool could be accomplished efficiently via AAGetPooledTransactionMsg

as happens now with GetPooledTransactionMsg

.

Finally, the EIP we present suggests taking a single-tenant vs. multi-tenant approach. Since increasing the burden on the network presents risk, we suggest first starting off with a 100k gas cap and only allowing one transaction per account at a time. The initial single-tenant approach only allows for one transaction per AA account at a time in the transaction pool, which likely provides enough protection initially. For multi-tenant AA, we would want to adjust the eth

protocol and likely increase the validation gas limit. However, we suggest stability is first found via the single-tenant approach.

Block Invalidation

In later versions of AA, as new blocks emerge in the network, all AA transactions in the transaction pool to the same account must be re-validated in a worst case scenario. For example, a transaction in the block may set a nonce or flip a bit in its internal state which then renders the validated pending transactions in the transaction pool invalid. This introduces extra computation to nodes. If these transactions are invalidated, there is no compensation for this computation. The transactions drop from the pool. Keep in mind, this does not impact block validation, but can be considered mempool clean up after a new block is received. The section below analyzes the impact in a worst case scenario.

Test Setup

Block Size

: 12.5 million gas

Transaction Size

: 27,490 gas

Transaction Count

: 454

Transaction Invalidation

: To maximize invalidations, we minimize the size of each transaction in the block. Each transaction alters the state such that existing AA transactions to the same account in the transaction pool are rendered invalid. Given a block size of 12.5 million gas, the maximum number of accounts we would need to re-validate per any given block caps at 454 accounts (in a worst case scenario).

Results

Validation Gas Limit (gas)

Total Validation Time (ms)

Total Validation Time with Processing (ms)

Mean Validation Time (ms)

Block Gas per ms of Validation (gas)

0

100000

435.63

501.25

0.9595

28649.50

1

200000

949.85

1001.04

2.0922

13139.40

2

300000

1474.00

1529.90

3.2467

8467.09

3

400000

1999.17

2053.08

4.4035

6242.82

Table 6

(1)

[

1746×1189 96.4 KB

](https://ethresear.ch/uploads/default/original/2X/3/33f08326c915417f1377da68a9215ebe2632f74c.png)

Figure 2

(1)

[

910×368 15.4 KB

](https://ethresear.ch/uploads/default/original/2X/5/5db7693ecf6f0e07b9943042ed470f9742c582f2.png)

Figure 3

(1)

[

907×639 22.7 KB

](https://ethresear.ch/uploads/default/original/2X/b/b362be9110b25713f04063f44aaec396243fb83a.png)

Figure 4

(1)

Discussion

As can be seen in the figures and tables above, a worst case scenario incurs up to 2s of re-validation time depending on the gas cap configured in the transaction pool. This worst case scenario would likely only happen under a malicious attack and does not appear to bring many benefits to the attacker. However, this attack does

impact the entire network.

The entire network incurs processing time in the transaction pool, slowing down pool resources allocated to receiving and sending transactions. As a result, transaction propagation across the network likely throttles.

Additionally, the miner algorithm needs to adapt. Transactions undergoing re-validation are no longer immediately eligible for inclusion in the next block. The miner needs to build a block as quick as possible and cannot wait for processing to complete. Without mitigation, this may have the effect of only having one AA transaction per account included at most every other block. However, the miner algorithm may be adjusted to adaptively swap these transactions for others in the block as the miner plays the hashing game.

For average contracts, we would assume validation gas to be well under 100k. For example, we estimate smart contract wallets to utilize 50k gas for validation. A ZK-SNARK would likely be the exception at ~400k gas. Additionally, the current eth network contains ~60% (3) of its transactions as contract calls and we would not expect most transactions in a block to be AA transactions initially. Also, there are various mitigation efforts or methods to make validation more efficient, which we discuss below. Ultimately, if a client configures the transaction pool to take a 400k gas limit, we'd likely see computation times several orders of magnitude lower on average than the worst case scenarios depicted above. First, we'll discuss the cost of producing this attack and then finally discuss mitigation efforts and other improvements.

Cost Analysis for Attack

As we reference in Table 6, an attacker must input 6242.82 gas to incur 1ms of work across the network if the transaction pool is configured to accept a max of 400k gas for validation. This attack can be carried out in two ways. We'll analyze these in the subsequent sections.

1. The miner attacks the network
2. Transactions are propagated to the network at a higher than normal gas price to ensure maximum impact

Miner Imposed Attack

Gas price is irrelevant to the miner as the miner will receive the fee regardless. If the miner is imposing the attack, the actual loss would be an opportunity cost. Between 08/09/20 - 08/24/20, block fees were between 1.1 eth to 3.49 eth.

20th % Gas Block Fees

95th % Block Fees

1.10 eth

3.49 eth

Table 7

(2)

Propagating Transactions at a High Gas Price

Unlike a miner imposed attack, the attacker must propagate transactions at a high enough gas price to be included in the next block and accepted by transaction pools in the network. Between 08/09/20 - 08/24/20 we look at the 95th % of gas price per block and then take the 20th % - 95th % of the data set to arrive at ~45 - 202.16 gwei. A sophisticated attacker would likely wait for the scenario of a lower gas price in the network near the 20th percentile.

20th %

95th%

45 gwei

202.16 gwei

Table 8

(2)

Validation Gas Limit (gas)

Gas Cost per ms of Validation(gas)

Eth Cost Per ms 20th % (eth)

Eth Cost Per ms 95th % (eth)

0

100000

28644

0.0013

0.0058

1

200000

13136.4

0.0006

0.0027

2

300000

8465.06

0.0004

0.0017

3

400000

6242.3

0.0003

0.0013

Table 9

(1) (2)

Speedups and Mitigation

If there is a desire to reduce the numbers from Table 6, we suggest a few speedups or mitigations. However, as noted in the discussion section, the numbers in Table 6 represent a worst case scenario most likely initiated by an attacker with little benefits. See also the AA EIP for further discussion.

Two-Phase Validation

AA validation can be split into two parts.

1. Pure Function
2. State dependent Function

We only need to re-run the state dependent part of the validation function. For example, a signature check does not need to re-run.

State Tracking

When validating an incoming block, we may store the SSTORE

locations for AA transactions in a map. When validating AA transactions in the transaction pool, we may also store the SLOAD

locations in a map. We only need to re-validate if they access the same location.

Adaptive Gas Cap

If an emergent block is an attack and incurring significant slowdowns on re-validation, the gas cap may dynamically adjust to a lower value.

Conclusion

We find the form of AA as specced in theAA MVP spec to be feasible with a generally positive path forward. Slowdowns in transaction propagation are within reasonable ranges and have room to scale further. Block invalidation has a network-wide effect, but the numbers shared express worst case scenarios and an attacker does not seem to have much to gain beyond griefing. For the same price, there appear to be more effective attacks against the network. We estimate typical load on the transaction pool would be orders of magnitude lower than what is captured in our results, especially if further mitigations are added such as two-phase validation.

We maintain cautioun in how many changes should be added to the protocol at once and therefore suggest a single-tenant vs. multi-tenant approach. We cover these approaches in depth within the AA EIP. The single-tenant approach initially targets contract wallets, multi-sigs and other simple usecases. It reduces load on the network by only supporting one transaction per account in the transaction pool while setting the validation gas limit to 90k. In addition, the first version does not require transaction re-validation, completely removing concerns around block invalidations. This approach gives a general transition plan to AA for the network, while maintaining caution on network wide effects or slow downs.

Data Sources

Throughout this writeup, multiple graphs and charts are shared. The data comes from three major sources and all queries, databases etc. are open:

(1) Bazooka Simulations (Google Collab Notebook)

A google collab notebook with the queries, included DB and explanations can be foundhere.

(2) Alethio Spark Hadoop Framework

Databricks is not open publicly, but the queries that are used can be foundhere

(3) Extended Geth Data Collection (Google Collab Notebook)

Custom logs were added to geth over a period of time and added to a mysql database. Queries and explanation to set up the DB can be found in a google collab notebook here.