Least Authority
PRIVACY MATTERS

Ethereum 2.0 Specifications

Security Audit Report

Ethereum Foundation

Final Report Version: 6 March 2020

Table of Contents

Overview

Background

The Ethereum Foundation has requested that Least Authority perform a security audit of the Ethereum 2.0 Consensus and Networking specifications. Ethereum 2.0, a Proof of Stake (PoS) / sharded protocol, is a major network upgrade that is set to take place in 3 distinct phases: Phase 0 - Beacon Chain, Phase 1 - Shard Chains, and Phase 2 - Execution Environments. This audit is to be performed as in preparation for the Phase 0 mainnet launch in April 2020.

Project Dates

- **January 13 - February 5**: Initial Review *(Completed)*
- **February 7**: Initial Audit Report delivered *(Completed)*
- **March 2 - 5**: Verification Review *(Completed)*
- **March 6**: Final Audit Report delivered *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer
- Meejah, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Dominc Tarr, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Ethereum 2.0 Specifications followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Core Eth2.0 specifications for Phase 0: https://github.com/ethereum/eth2.0-specs
- The Beacon Chain spec: https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/beacon-chain.md
- The Beacon Chain Fork Choice specification: https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/fork-choice.md
- Networking specification: https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md
- Honest Validator documentation: https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/validator.md
- Go Implementation of Eth2.0 specification: https://github.com/protolambda/zrnt/

Specifically, we examined version 0.10.0.

Supporting Documentation

The following documentation was available to the review team:

- Serenity Design Rationale: https://notes.ethereum.org/s/rkhCgQteN#
- Phase 0 for Humans (v0.9.3): https://notes.ethereum.org/@djrtwo/Bkn3zpwxB?type=view
- Compendium of Eth2.0 links: https://eth2.info

Areas of Concern

Our investigation focused on the following areas:

- Denial of Service (DoS) attacks;
- Attacks intending to misuse resources, cause unintended forks and create unwanted or adversarial chains;
- Any attack that impacts funds, such as draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Network attacks, including flooding of or misusing data and causing inappropriate taxing;
- Key management implementation: secure private key storage and proper management of encryption and signing keys;
- Exposure of any critical information during user interactions with the blockchain and external libraries;
- Any potential attacks with high ROI of efforts;
- General use of external libraries;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions, ambient and excess authority;
- Vulnerabilities within individual components as well as secure interaction between the network components;
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Ethereum 2.0 is one of the first PoS projects planned for production and will likely have the greatest market cap value and the largest number of users at launch. As a result, there have not been many opportunities to study the impacts of design decisions on real world uses of such blockchain implementations, and none at the same scale. Although aspects of the design can be reviewed by comparing them to similar implementations, the collective system may not behave as intended due to the complexity. Additionally, the relevant information about these system components is distributed across a variety of resources. Furthermore, research on the impacts of certain design decisions, particularly with incentive mechanisms and collective system interactions, is still in the early stages. The long term stability of PoS blockchains is an area that will need to be monitored over time and through real world uses.

Since no other large-scale implementations of a PoS system like Ethereum exist in production, auditing the Ethereum 2.0 Specifications presented challenges. Without other systems to compare and contrast it against, the Ethereum Foundation team was an indispensable resource throughout the audit. Their time and effort in answering questions and helping to identify solutions contributed to an improved review effort, as reflected in this report. We appreciate their openness to engage in discussions about the issues we raised. We encourage the Ethereum Foundation team to continue to engage in design discussions with the various stakeholders and the broader community and hope this report will be a basis for them to do so.

Given that our investigation consisted of reviewing a specification as opposed to a coded implementation, the attack vectors we identified were necessarily based on certain assumptions and theory. It should be noted that, in practice, there may be some differences between our models and a given implementation. Where possible, we tested our assumptions against the suggested Go implementation of Ethereum 2.0, created by the Ethereum Foundation team. Nonetheless, it will be key for them to continue to revisit and test past assumptions as more of the design is implemented.

Overall Specifications

In our review and analysis of the Ethereum 2.0 Specifications, we found the specifications to be very well thought out and comprehensive. It was readily apparent that security had been a strong consideration during the design phase, particularly highlighted in the consensys layer, which the specification describes in great detail.

However, we identified two areas that would benefit from further review and additional documentation. Specifically, we found that the Peer-to-peer (P2P) networking layer and the ENR system are underrepresented. These may be elaborated on in later phases, but their significance suggests that Phase 0 would be a good starting point for laying the foundation of a strong network layer.

Areas for Further Consideration

Block Proposer Election System

Due to the nature of PoS systems and the ability of the proposer to change the outcomes of certain aspects of the chain within the block proposal mechanism, it is often a target for attacks. Our team discovered two issues within this process and recommend that, in the future, the block proposer election process be as secretive as possible and reviewed for additional attack vectors. Single Secret Leader Election surfaced as a remediation for protecting the block proposal and block leader systems in various parts of our review (Issue C, Issue D & Issue F). This is not due to a deficiency in the specification, but rather a best practice recommendation for PoS systems and a consequence of this being an area of active research.

P2P Networking Layer

Another area of potential vulnerability for attack is the P2P networking layer system and its various gossip topics. We carefully considered the P2P protocol but, given the limited documentation and the early phase of the project, it is difficult to fully analyze attack vectors and vulnerabilities. We were not able to identify a clear strategy for making these systems BAR-resilient, which should be taken under consideration. In general, BAR-resilient protocols eliminate the threat of freeriding rational peers and rational peers that rewrite the gossip protocol secretly according to their needs. In the early stages of the network, there will be mostly altruistic nodes experimenting with the new protocol, but as Ethereum 2.0 begins to grow and later phases are released, the presence of rational actors increase in the network is certain.

Small Validator Set Incentives

Although we found that the small validator set incentives do not pose a security threat, and are therefore a non issue, we have documented our findings.

Rational validators are incentivised to keep the validator set as small as possible so we searched for potential attacks along that line. For example, a general strategy would be for rational proposers to always exclude deposit transactions in their blocks and to always fork altruistic blocks that contain deposit transactions. In addition, rational validators should never attest to altruistic blocks. The former, however, is not possible as blocks that do not include pending deposit transactions are recognizable as invalid, because pending deposit transactions can be checked globally against the latest Eth1 data.

Specific Issues

We list the issues we found in the code in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Beacon Block Propagation Restrictions Are Too Loose	Resolved
Issue B: Attester and Proposer Slashing Message Propagation Attack	Resolved
Issue C: Distributed Denial of Service (DDoS) Attacks Against Block Proposer	Unresolved
Issue D: Block Proposer Eclipse Attack	Unresolved
Issue E: Misaligned Gossip Incentives	Unresolved
Issue F: Bias in Weighted Random Sampling in Election of Block Proposers	Invalid Issue
Issue G: Gossipsub Control Message DoS	Unresolved
Suggestion 1: Improve Specification for ENR and P2P Systems	Unresolved
Suggestion 2: Consider Implementing a BAR-Resilient Gossip Protocol	Unresolved
Suggestion 3: Peer Review of Consensus Papers and Proofs	Unresolved

Issue A: Block Propagation Restrictions Are Too Loose

Synopsis

Block propagation messages do not check for a range in the past, which could allow valid, yet old messages to be withheld and sent out later.

Impact

This type of attack could cause unnecessary processing by nodes, and in large enough volume, would slow down the entire network, possibly causing transaction processing to stall entirely. Additionally, the caches that track attestation aggregation messages would grow unbounded.

Preconditions

Any node participating in the gossip network can mount this attack.

Feasibility

An attacker can inexpensively start several malicious nodes for increasing the harm to the stability of the network. Although conducting this attack would be inexpensive and repeatable, it is moderately feasible due to the requirement of having some technical knowledge in order to carry it out successfully.

Technical Details

There is no lower bound on the slot for block propagation, allowing a node to use older block proposal messages.

Remediation

Introduction of a lower bound on the slot for block propagation would stop forwarding of old, but valid, messages and make an attack with this method far less effective.

Status

The specification was updated to include the requirement that received blocks need to be newer than the latest finalized block before forwarding them to other nodes in the network.

Verification

Resolved.

Issue B: Attester and Proposer Slashing Message Propagation Attack

Location

Global topics

Validations

Synopsis

Attester slashing and proposer slashing messages can be propagated with minimal punishment if they look valid. This allows nodes to spam these messages and create unnecessary traffic in the network, creating a DoS attack vector.

Impact

This type of attack would slow down or potentially halt network processing for the duration it was carried out.

Preconditions

First, a node must create a valid attester or proposer slashing message by signing. Once complete, however, they can create an unlimited amount of similar, valid-looking messages. Second, the attacker must have access to bandwidth capable of transmitting the volume of messages necessary to create a noticeable slow-down of the network. The node must also be able to join the network as a validator, meaning their 32 ETH stake must be deposited and activated before they can send these types of messages.

Feasibility

This attack is possible by an individual with experience in the language the client is written in and is able to work with the protocol. The requirements in computing resources are ordinary and satisfied by most modern laptops. However, the bandwidth required to carry this out is more substantial. With the increased availability of fiber lines, home internet connections could possibly transmit the data required for this attack. Between the required bandwidth, computing power, and skills necessary to carry this attack out, this attack is moderately feasible, now and more feasible in the near future.

Technical Details

Before a slashing message is propagated by a node in the network, a node must validate that it is legitimate. However, these validations currently only check that a signature is present and valid. A node could create a number of valid looking messages and publish them. When other nodes receive these messages, they will check the signature but nothing else, deem them valid, and forward them to the rest of their subnet. Given the right volume, this could stall the block proposal and finalization processes.

For a specific example, a node could create a slashable message (e.g. a double signed message) and then propagate it. This would result in them being slashed. Once slashed, however, they can send an unlimited amount of similar messages without punishment, since slashing is a binary state (i.e. a node is either slashed or is not slashed).

Remediation

Since slashing is a binary state, `proposer_slashing` can be limited to one message per validator and ignore any other duplicates that are received. Otherwise, a validator that was going to be slashed or was already slashed could create infinite valid `proposer_slashing` messages.

`voluntary_exit` messages can be treated similarly and can accept one valid `voluntary_exit` message per validator, since it is a one-time operation. If another `voluntary_exit` message is received for a validator, it can be forwarded if it has a more recent epoch than the one that is known. Otherwise, duplicates can be safely ignored.

The `attester_slashing` messages are a little more difficult to declare the rules for because the slashable attestations included in the message contain many different validators. We recommend updating the specification so that a node only forwards `attester_slashing` messages that contain new validators not yet seen in another `attester_slashing` message.

We recommend that stronger validations be included in the specification for clients around message forwarding for these three message types in particular.

Status

The specification was updated and applies the recommended remediation. Specifically, any `voluntary_exit` and `proposer_slashing` message is only forwarded once, ignoring duplicates. Additionally, `attester_slashing` messages that do not contain new information are also not propagated. Details on the specifics of their approach can be found in PR #1617.

Verification

Resolved.

Issue C: Distributed Denial of Service (DDoS) Attacks Against Block Proposer

Location

`compute_shuffled_index`, `compute_proposer_index`, `compute_committee`

Design decision rationale

Synopsis

The network specification makes it simple for any validator to figure out the IP addresses of any other validator quickly. In addition, specification of committees and proposers imply that all proposers are public knowledge for any slot in an epoch at the beginning of that epoch. An attacker might use this knowledge to strategically execute DDoS attacks against the proposer of a slot to stall the chain, or to keep slots empty.

Impact

A validator that is made incapable of reasonably fast network access may be unable to propagate attestations and proposed blocks in a timely manner. This would lead to targeted censorship of attestations and blocks, resulting in penalties for the victim.

Furthermore, the attack is inexpensive enough to perform it over a long range of time on a series of block proposers. This means that it is a possibility that no new blocks are getting gossipped, which threatens the liveness of the chain. Once a sustained attack prevents finalization for several epochs, the validators will lose deposits due to the inactivity penalty.

Feasibility

Any attacker who is able to perform DDoS attacks can mount this attack. The cost depends on the node under attack - a home staker node with a residential internet connection is significantly less expensive to bring down than a node in a data center with a symmetric 10GBit/s uplink.

Technical Details

Committees are computed by RANDAO at the beginning of each epoch and the first member (relative to index order) of each committee is the block proposer. An attacker could then precompute IP addresses of the validator set and keep that list updated on the fly, while that set changes over time. After the RANDAO is revealed, they then select all proposer addresses and DDoS them in the usual manner, when their slot is up for execution.

Remediation

This attack can be remediated by using a proposer election system where only the elected proposer knows that they are elected, but is able prove this to others. This way, the attacker does not know which validator node to target before the elected validator proposes their block. Such schemes are called Single Secret Leader Election (SSLE), as described and constructed in [BEHG20].

Status

The team has acknowledged the issue and is working both on implementing a short term workaround and developing a long-term solution.

The long term solution will likely be using Single Secret Leader Election (SSLE) for electing block proposers, based on our recommendation. Since SSLE research is still premature and implementing SSLE is non-trivial, the Ethereum Foundation team decided to introduce this change during a later phase.

In the interim, the Ethereum Foundation team suggests that validator operators run multiple nodes, such that one node participates in network communication on behalf of the validator, and another node is used to propagate the blocks proposed by the validator. The decoupling of node identity and validator identity allows for operators to switch between nodes in order to publish their blocks.

Unfortunately, operating multiple nodes comes with a financial cost that can be prohibitive for hobbyists and home stakers. For these cases, the Ethereum Foundation team suggests introducing a community-run semi-trusted node that accepts proposed blocks and forwards them to the network.

At present, the Ethereum Foundation team has stated that they are working on formulating processes and recommendations that implement this pattern.

Verification

Unresolved.

Issue D: Block Proposer Eclipse Attack

Location

https://github.com/libp2p/go-libp2p-pubsub/blob/master/gossipsub.go#L472 `compute_shuffled_index, compute_proposer_index, compute_committee`

Synopsis

An attacker creates a large volume of nodes and uses them to connect to the block proposer. The large number of connections triggers the victim node to prune, as described in gossipsub. When the node pruning occurs, some legitimate nodes might be purged. This increases the likelihood of connections with the attacker's malicious nodes. After a number of repetitions, the target node only has connections to the attacker's node, eclipsing them from the rest of the network.

Impact

A node that has a majority of attacker nodes as peers in their topic mesh may be unable to propagate attestations and proposed blocks to the main network. This would lead to targeted censorship of attestations and blocks, resulting in penalties for the victim.

Furthermore, the attack is inexpensive enough to perform it over a long range of time on a series of block proposers. This makes it possible that no new blocks are getting gossipped, which threatens the liveness of the chain. Once a sustained attack prevents finalization for several epochs, the validators will lose deposits due to the inactivity penalty.

Preconditions

An attacker is able to control enough identities that they have a high probability of becoming the majority of the target's mesh peers.

Feasibility

This attack can be carried out by any actor with access to a reasonably modern computer and a decent internet connection.

Technical Details

Given a `D_High` of 12 and a node selecting peers on the mesh to join as a target, not at random. For each `GRAFT` message let P equal the probability of adding to the target mesh while maintaining a previously connected attack node. Assuming a target is at `D_High` with 12 connected peers, getting the first attack node already in the list with zero previous nodes is P=1. The probability of not removing the first, or any subsequent, attack nodes will be `P=11/12`, followed by `P=10/12`, `P=9/12` and so on until the final attempt to completely fill the target mesh with attack nodes being P=1/12. Summing the expected values of becoming an additional peer, an attacker expects to be all 12 peers in 37.2385 `GRAFT` messages. These probabilities assume that trials are independent and that the probability of success does not change between trials (i.e. that no other peers attempt to `GRAFT` to the target while the attack is taking place). In practice, it can be expected that more attempts to completely fill the target's mesh would be needed.

Mitigation

An effort could be made to make a variant of gossipsub less susceptible to eclipse attacks and clients could deviate from regular gossipsub behavior and try to detect and counteract this attack. For example, instead of randomly pruning peers, use a conservative approach and keep old connections or connections that provide low latency pubsub (i.e. connections that send `IHAVE` first for many messages). There are likely other suitable metrics to heuristically determine the value of a connection.

Remediation

Instead of making the pubsub mechanism less susceptible to eclipse attacks, we recommend making the beacon chain rely on a protocol that is eclipse-resistant. See Issue C.

Status

See the status explanation in Issue C.

Verification

Unresolved.

Issue E: Misaligned Gossip Incentives

Synopsis

Rational, or non-altruistic Beacon Nodes (BN) in the gossip network are only incentivised to relay information to other BNs if it provides their messages with a higher probability of being seen by the network. Where it is expected that relaying this information increases their profitability (i.e. increase the likelihood that their block proposal and attestation messages are propagated). If another BN requests blocks, or asks to relay a message, then it would be a cost burden of some degree on the bandwidth of the rational BN.

Impact

This could lead to a tragedy of the commons where BNs are incentivized into using the gossip network only in their favor. This would require altruistic nodes to take on more network traffic and increase gossip latency. However, this may not be of any immediate concern and more research is needed to fully understand this concern.

Preconditions

This is only likely if the network is providing a stressful load to a rational BN and there is a clear price to behaving altruistically that many will not take on.

Feasibility

There are many examples of altruistic actions and we are unsure of the cost that will be placed on these nodes at this time. It is likely that altruistic nodes will take on rational node costs.

Technical Details

We define altruistism as a BN performing an action initiated by another BN on the network without compensation (i.e if BN A forwards attestations from BN B, then A expects that B will forward their attestations messages). Another example is provided in Issue G where control messages meant to keep nodes in sync are abused to always request more information than is produced. The common resource is a healthy network that can deliver messages that all nodes require as a whole, and a tragedy being a situation where nodes behave in a self interested way that damages the network.

A rational node we define as one that would attempt to minimize short term costs or maximize profit (i.e. saving bandwidth costs by not gossiping messages that a node has no interest in, or selectively reducing the chance of others to maintain validator good standing such that the attackers profits are increased).

Remediation

Methods to deal with rational actors in gossip networks are known, for example under models like Byzantine, Altruistic, Rational (BAR) tolerant Gossip described in [LCWNRAD06] and first introduced in [AACDMP05]. Similar strategies might succeed here, too. These strategies come with tradeoffs and it is not clear what the best approach for Eth2.0 traffic will be at this time.

Status

This is an area of ongoing research as the network unfolds and activity is monitored and analyzed.

Proof of Work (PoW) networks have a limited amount of gossip communication, primarily due to block and transaction propagation. Traditionally in PoW systems, there are tendencies towards more centralized solutions that are deployed to ensure timely block delivery and healthy propagation in general. Ethereum is working hard to avoid these strategies if possible by taking into consideration distributed reputation and scoring systems. PoS networks will require slightly more complicated messaging which could make this challenging.

Verification

Unresolved.

(Invalid) Issue F: Bias in Weighted Random Sampling in Election of Block Proposers

Location

compute_proposer_index

Synopsis

The probability for an active validator to be elected block proposer should be their deposit, divided by the deposit of all active validators. Measurements have shown that this is not the case.

Impact

Based on an initial review, it seems like the measured bias in favour of validators with a higher effective balance further incentivizes not provoking to get slashed. However, it is possible that there are edge cases where this unexpected behavior produces results that are not in the interest of the network.

Technical Details

The following test setup describes an extreme, but possible case in which the proposer selection function should behave as expected. Sets of validators of different sizes are prepared. In each of these sets, all validators have an effective balance of 16 ETH, except one, that has an expected balance of 32 ETH. Then run the `compute_proposer_index` function over these validators. Each test is performed 32 times and the average each of the resulting values approximates typical behaviour. The expected outcome is that in all cases, all 16 ETH validators get elected roughly the same amount of time, and the 32 ETH validator gets elected roughly twice as often.

However, test results show that this is not the behavior of the function. In order to learn about the statistical properties of the function, a large number of samples is required. Since the python executable specification is not very fast, the tests were performed using zrnt.

Two irregularities are identified with this test. The first validator in the set consistently gets elected more often than the rest. The extent of this advantage varies, depending on the number of validators and the ratio of the own stake and that of the rest of the network. Second, as the validator set grows larger, the validator with more stake is elected disproportionately more often than the others.

A possible cause of these issues is the `compute_proposer_index` function, which performs a variant of the rejection sampling method. Instead of sampling a new validator that is tested in every round, it performs a random permutation once and then iterates over that.

The fact that the first validator gets elected more than the others suggests that there is an issue in the permutation function, as the first validator in the shuffled list will have a much higher chance of getting elected. The shuffle function described in the specification appears to match the description in [HMR12].

Remediation

We advise performing tests to verify whether the shuffle really behaves randomly.

We do not have a concrete explanation for why validators with large deposits get elected more often as the validator set grows. We did identify two possible sources of error, though.

First, the rejection sampling variant used in the specification iterates over a random permutation instead of sampling fresh on every iteration. This discrepancy is measurable with very small sets of validators, but unlikely to have an impact with many validators. The probability for a validator to get tested for rejection twice within one function invocation is 0 for the permutation (ignoring the case where all validators have been rejected), but 1/n^2 for re-sampling with n validators. While we did measure small irregularities for very small validator sets (up to 16 validators), the data we measured especially shows deviations from the expected result for large validator sets. This suggests that there is another issue.

The other possible source of error we identified is that the resolution of the rejection test is not high enough. In each rejection test round, only a single byte is used to sample. The probability of the validator to be sampled has over 32 bits of resolution. Reading four bytes from the result of the hash function instead of one should reduce sampling error.

The tests we performed are in the `dist_test` branch of our fork of the `zrnt` repository, as well as PR #9 in our fork of the `eth2.0-specs` repository.

We consider the issue resolved when the provided tests indicate that proposer election rates are in fact proportional to the effective balance of the validators.

Status

Invalid Issue. We initially ran the tests with the `zrnt` testnet presets. These configured the shuffle algorithm to only perform 10 instead of 90 rounds, which resulted in statistically significant biases in the election of block proposers.

Rerunning the test with the mainnet presets did not show these biases. Therefore, the issue does not apply with the mainnet parameter selection of Ethereum 2.0.

Verification

Not Applicable.

Issue G: Gossipsub Control Message DoS

Location

Libp2p spec control messages

Libp2p gossipsub implementation

Synopsis

The Ethereum 2.0 P2P networking specification relies on libp2p gossipsub. Gossipsub uses an optimistic push to handle gossiping messages with what are called Control Messages. The control messages in questions are `IHAVE`, `IWANT`, and `GRAFT` that could cause network traffic overhead on a target node.

Impact

This overhead could come at sensitive times when the node needs bandwidth to communicate and sync with other nodes., which may result in reduced rewards (because their attestation is only added to later blocks) or penalties (if the attack is strong enough to interrupt the service of the validator).

Preconditions

There must be no limits placed on the optimistic pushing of messages, and the message cache of the target needs to be large enough to be required to send enough data that it hinders the targets connection.

Feasibility

This would only require a single peer and internet connection to cause traffic overhead in the target, therefore the costs are inexpensive on the attacker side. However, we haven't fully investigated the disparity in relative resource consumption. There is also a tight bound in libp2p on the window of the cache. A node will request messages that are not in the seen cache, meaning the attacker needs to generate valid messages not in this cache. The cache is said to hold two minutes of message ID data. A node will also only respond with data held within the `GossipHistoryGossip` slot length, which is defaulted to three slots and assumed to be a minimal amount of message history stored, making this a moderate to low feasibility attack at current understanding.

Technical Details

An example of optimistic pushing that forces network traffic overhead on a target is a sender S initiates a message to a receiver R with the control message `IHAVE`. R then ignores if they have already seen those `IDs` and responds to S with `IWANT(IHAVE.ids.length())`. The S client will then process every `IWANT ID` in the control message and begin sending all messages in their cache that match the provided IDs.

`IHAVE` control messages will trigger a peer to search to see if they have seen this list of message `IDs`, within a default value of two minutes, from the sender of the `IHAVE` message. The attacking client could be modified here to ignore if the messages have been seen and load an `IWANT` control message with all of the `IHAVE` `IDs` where `IDs` are in the targets range of [0..`CacheWindow.length`]. The attacker only needs to send a list of message IDs and the target will reply with up to 1MB message bodies depending on the message for each ID, potentially inundating the target machine. Alternatively it may be possible to originate an `IWANT(suspected_ids_target_has)` control message without seeing what the peer already has and target a peer that isn't actively gossiping with the attacker. If the target node has been filled with a large cache of messages from the following attack, this may cause a large amount of messages to be sent.

`IWANT` control messages could be forged by an attacker to contain messages `IDs` for invalid messages by first sending to the target an `IHAVE` message for a topic the target is listening to with many invalid message `IDs`. This would cause the target to download and verify the invalid messages sent by the attacker. If message validation is a slow process, this could consume resources on the targets node. Also a node may attempt to store old valid messages and wait for the `SEEN cache` timed message cache to lapse after two minutes and attempt to retransmit old messages that are no longer in the `SEEN cache`, causing the target to load their `mcache` with a large amount of old messages. If there is no bound checking on future valid messages, these may also get stored in the `mcache`.

Repeated `GRAFT` messages from an attacker to a target may also cause some unwanted network activity on the target by forcing a `PRUNE` message response. This should cost the attacker as much as the target.

Mitigation

Regulate the amount of messages a peer is willing to store in the `mcache` and respond with in `IWANT` control messages.

Remediation

Implementing something more sophisticated than simple regulation like tit-for-tat may be possible. Using the same or similar research linked to in the Remediation section of Issue D could help alleviate this traffic overhead in an even more sophisticated way. If S only responds to R with an equivalent amount of messages that S has not already seen from R, then it makes it harder for R to endlessly request data. The tradeoff for implementing such a strategy is that lagging nodes or new nodes will have a harder time syncing with the network.

Verification could require simulating game theory strategies for how nodes serve each other data. When it is not possible or in the best interest of an attacker to perform this attack, it is considered resolved.

Status

According to the Ethereum Foundation team, the following mitigation strategies are in progress:

- Protocol Labs libp2p team is working on hardening gossipsub; and
- The Ethereum Foundation is working on publishing the details on the protections that will be put in place, such as peer scoring and blacklisting.

Verification

Unresolved.

Suggestions

Suggestion 1: Improve Specification for ENR and P2P Systems

Synopsis

We found the documentation around the P2P and ENR systems of Ethereum 2.0 to be insufficient. Specifically, we were unable to conclude how the P2P system incorporates the ENR system. We recommend a more thorough and detailed document around how the ENR system should work with the P2P interface and the gossipsub system.

Status

The Ethereum Foundation team states that they are currently working on a bidirectional conversion function between multiaddr and ENR fields to allow for a canonical construction of a multiaddr from a discovered ENR, which they intend to submit to the Ethereum 2.0 specifications repository as a PR for review and integration into the specification.

Verification

Unresolved.

Suggestion 2: Consider Implementing a BAR-Resilient Gossip Protocol

Location

The specification states that gossipsub will be the protocol used for communication between nodes.

Synopsis

We recommend implementing a BAR-resilient gossip protocol as a way to provide additional guarantees around node failure and throughput for the network given different conditions of partitions and possible bad actors.

Status

The Ethereum Foundation team has made note of the ongoing efforts with Protocol Labs, the maintainers of libp2p / gossipsub, to investigate BAR-resilient peer-sampling techniques to harden the protocol. They also raised questions on how to best achieve Sybil attack resistance in the context of blockchain networks, as it is suggested in the paper on BAR Gossip, [LCWNRAD06], that each participant is limited to a single identity. In addition, the Ethereum Foundation also made known that a design goal of theirs has been to avoid the use of validator pubkeys to enhance the P2P layer, in order to avoid coupling validator identity with node identity and therefore allowing for flexibility for additional potential and sophisticated validation/network setup designs (Issue C).

To note, the abstract principle of BAR-resilient gossip often goes beyond peer-sampling techniques and also ensures that freeriding off the gossip platform is not possible. At present and to the best of our knowledge, there is no proof of whether or not such BAR-resilience is possible in an entirely permissionless system.

Our team proposed looking into BAR-resilient gossiping in order to help prevent greedy participants freeriding on the gossip system. However, we do not consider Byzantine participants a significant issue, given that they can only send too few or too many messages. In general, gossip networks do well with handling participants that forward too few messages, for example when a few nodes completely fail. As a result, we believe that this does not require further action. Sending too many messages is equivalent to attempting to overwhelm the recipient with network traffic. While this problem may be resolved on the gossip layer, it is not on the network layer. Thus, a publicly dialable participant can always be subject to a DDoS attack.

Since the Ethereum P2P layer only forwards valuable messages, this type of attack does not pose an application-layer threat. The attacks a Sybil node could launch on the gossip layer are not significantly more powerful than network layer DDoS attacks (e.g. simple UDP floods). As a result, we consider Sybil attacks tolerable and mitigating against them to be a lower-priority target.

Verification

Unresolved.

Suggestion 3: Peer Review of Consensus Papers and Proofs

Synopsis

Consensus systems are notoriously difficult to audit. Peer review processes are the de facto standard to ensure quality research. Since consensus is at the heart of Ethereum 2.0, we recommend a peer review of the theorems and proofs in the ongoing research on the consensus protocol, described in [BHKPQRSWZ19], once finalized.

Status

The Ethereum Foundations team notes that the ongoing research paper combining Casper FFG and LMD-GHOST, [BHKPQRSWZ19], is under final review and near completion. They have stated their intent to publish the first draft on arXiv, with the intention of soliciting a community review, before seeking to publish and submit the paper for peer review in a journal and conference.

Additionally, the Ethereum Foundation team has stated that they are working with Runtime Verification to formally model the Beacon Chain Phase 0 Specification in K and are currently working on formally proving that the Phase 0 specification correctly implements the theorems and proofs outlined in [BHKPQRSWZ19]. They have stated their intent to also publish and submit this work to a conference and journal for peer review.

Verification

Unresolved.

Recommendations

Since Ethereum 2.0 will be one of the first and largest PoS systems in production, we also recommend that expected system outcomes and desired participants behavior are clearly documented in advance of the launch. These should be used to actively benchmark whether the blockchain is performing as expected. We commend the Ethereum Foundation for stating their intent to document expected system outcomes and desired participants behavior in conjunction with their upcoming testnets.

In addition, we recommend that existing research and resources distributed across multiple platforms be gathered and distilled, in order to inform future decision making and facilitate better communication of Ethereum 2.0's goals, making them more comprehensible and accessible by the community of users and engineers. We acknowledge that the specifications have been heavily optimized for those writing and consuming them for implementation, however, there is great benefit from a single point of access to the various active research efforts and educational resources.

We recommend that additional security audits be conducted in preparation for the Phase 1 and Phase 2 releases, to ensure that any potential issues and vulnerabilities are identified, addressed and verified.

Finally, we commend the Ethereum Foundation team for putting together a thoroughly comprehensive short term and long term strategy to address the issues and suggestions raised in this report. We recommend that follow up verification of the unresolved *Issues* and *Suggestions* stated above be conducted once the intended changes have been implemented by the Ethereum Foundation team.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.