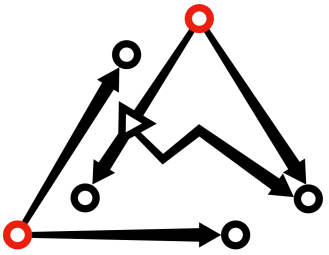

Least Authority

PRIVACY MATTERS

Gossipsub v1.1 Protocol Design + Implementation

Security Audit Report

Protocol Labs

Final Report Version: 3 June 2020

Table of Contents

This audit makes no statements or warranties and is for discussion purposes only.

Overview

Background

Protocol Labs has requested that Least Authority perform a security audit of Gossipsub, a pubsub protocol built on the libp2p library, a modular peer-to-peer networking stack. The most recent version, Gossipsub v1.1, implements a peer scoring layer so that peers are equipped with the information necessary to mitigate a series of attacks.

Project Dates

- **April 20 - May 8**: Code review (*Completed*)
- **May 13**: Delivery of Initial Audit Report (*Completed*)
- **May 25 - 28:** Verification *(Completed)*
- **May 29:** Delivery of Final Audit Report *(Completed)*
- **June 3:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Dylan Lott, 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 Gossipsub 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:

- Protocol Design: https://github.com/libp2p/specs/tree/master/pubsub/gossipsub
- Implementation: https://github.com/libp2p/go-libp2p-pubsub/
- Gossipsub Hardening: https://github.com/libp2p/gossipsub-hardening/

Specifically, we examined the Git revisions for our initial review:

`53c709a6caefa379398d95c2a828d12d9f81ddfa`

For the verification, we examined the Git revision:

`c0712c6e92cf957e35f5c78141dfcd28d5ffe062`

All file references in this document use Unix-style paths relative to the project's root directory.

Supporting Documentation

The following documentation was available to the review team:

- Gossipsub-v1.1.md: https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md
- Audit Scope Document: `Audit Scope - Gossipsub v1.1 Mar 23, 2020.pdf`

This audit makes no statements or warranties and is for discussion purposes only.

- Gossipsub-v1.1 Evaluation Report: Gossipsub v1.1 Evaluation Report - Apr 18.pdf

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and adherence of the implementation to the specification;
- Common and case-specific implementation errors;
- Vulnerabilities within individual components as well as secure interaction between the network components;
- Networking and communication with external data;
- Generic attacks on peer-to-peer networks;
- Denial of Service (DoS) attacks;
- Wire protocol level attacks, such as sybil attacks, and whether there are ways to exploit flaws in the wire protocols;
- Eclipse attacks (manipulate a target to connect only to malicious peers);
- Identifying potential patterns of uninitialized memory that may produce unexpected results;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions, ambient and excess authority of system components in a way that facilitates attacks; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

During this audit, our team reviewed the libp2p Gossipsub v1.1 protocol design and the corresponding implementation written in Go. Described as a security update to the 1.0 protocol, several new concepts were introduced, including flood publishing, peer exchange, and, most notably, peer scoring.

Since the protocol is used in multiple decentralized applications, such as Ethereum 2.0 and Filecoin, security issues could have significant and wide ranging implications for decentralized protocols. Simulations conducted by both our team and the Gossipsub team have already shown that v1.1 is much more resistant to attacks than v1.0. However, in an effort to better understand what steps can be taken towards achieving increased security for Gossipsub v1.1, in addition to what is outlined in this report resulting from our investigation and analysis, our team recommends that additional review time be considered to provide further coverage of the design protocol and coded implementation.

Code Implementation

Our team found the code quality of the protocol implementation in Go to be up to date and in line with best practices. The Gossipsub repository is well organized in a logical manner that fits with Go styles and paradigms. Test coverage was comprehensive and further enhanced by a network simulator created by the Gossipsub team for the purposes of testing the code. Given that the Gossipsub team mostly rely on their own libraries, the number of external libraries is fairly small, resulting in a smaller surface area for dependency issues and possible dependency vulnerabilities.

Specifically, the code conforms to the standard methodology of packaging smaller libraries together for a more modular and extensible codebase. To better meet Gossipsub's stated goal of being an extensible protocol, we recommend refactoring the coupling between the pubsub type and router types. This would make it possible to implement new routers outside of the Go libp2p pubsub package (see Suggestion 6).

This audit makes no statements or warranties and is for discussion purposes only.

The use of documentation and commenting in the code is also commendable. The Gossipsub specific portions of the code are well commented and provide explanatory notes with the relevant, corresponding sections in the specification. We also found that the various README.md files in the repository provided a sufficient amount of detail, allowing both reviewers and implementers to understand the intended behavior more clearly and effectively.

However, the scoring function can be difficult to follow and reason about. Consider a refactor of the scoring function to make it a more explicit and declarative style of mutating the `TopicScore` state. Additionally, to maintain separation of concerns, we recommend pulling the scoring functionality out into its own package and making an interface that the score package would fulfill. This would be more idiomatic Go and would allow for different scoring systems to be easily tested and built (see Suggestion 7).

Our team ran the gosec tool against the Gossipsub codebase and found twelve unhandled error conditions. However, since the unhandled errors are related to closing streams and are already inside of the error condition, we consider the errors adequately handled. In addition, running the code through golangci-lint produced no major security or usage issues, as a result of the Gossipsub team's use of a linter and the conforming to code quality best practices.

Protocol Design

The protocol design specification is well organized, with adequate documentation covering both the fundamentals of the network and the design choices made by the Gossipsub team. The intended behavior and intentions informing the design are explicit and clear arguments are stated for the key design choices. Overall, the specification is well structured and it is clear that the design was approached with security in mind at both the network and peer level. We commend the Gossipsub team for strongly considering and prioritizing security best practices.

However, our team identified some areas of improvement, specifically, consolidating the specification so that it covers all aspects of v1.0 and v1.1. Although the documentation uses versioning, it is required to read both versions in order to understand the protocol, given that v.1.1 is not strictly an upgrade of the v1.0 documentation. Since both are presented as the Gossipsub protocol, they would benefit from being consolidated. Furthermore, we recommend that the scoring documentation be made into a specific chapter or extension to one document that encapsulates all of the protocol design information (See Suggestion 8).

Peer Scoring

Scoring systems are difficult to perfect and the scoring system present in Gossipsub attempts to measure both desired and undesired behavior. As such, it is subject to Goodhart's Law, which states that whenever a measure becomes a target, it ceases to be a good measure. Specifically, the *First Message Delivered* metric of the peers scoring function can be improved arbitrarily by attacker nodes. It should be noted that peer scoring is not a fix preventing all attacks, but rather a set of rules that restrict the attacker. Some of these rules are only meant to discourage attackers with no real incentive. An example of this would be the IP colocation penalty, which is mostly useless in the face of abundant IPv6 addresses. Other peer scoring rules further restrict the possibilities of the adversaries. As a key component of the latest v1.1 update, the scoring function should be the subject of continuous scrutiny and any changes made to it should be very carefully considered and based on measurements in real-world deployments or realistic simulations.

In closely evaluating the peer scoring system, we discovered several ways to game this feature for favorable outcomes with peers (see Issue A, Issue B, and Issue C).

This audit makes no statements or warranties and is for discussion purposes only.

Additionally, the scoring function favors peers that can provide messages with low latency. This latency has two sources: First, the number of nodes the messages have passed through the Gossipsub mesh, and second, the latency of the individual connections between the nodes. This suggests that nodes favor peers with a low-latency internet connection, which is a force of centralization. We recommend further studying this effect in the simulations (see Suggestion 3).

Gossipsub allows applications to specify an application-specific scoring metric. This allows for great flexibility and opens the possibility for using reputation systems, where scores are not only based on observations made locally but shared between nodes (e.g. using the deposit in Proof of Stake blockchains) or data only available to the application layer (e.g. by increasing the local score of friends in a decentralized social network). However, from a security perspective, with flexibility comes the risk of introducing vulnerabilities. The feasibility and impact of such vulnerabilities can not be judged adequately without looking at the specific application. To help mitigate this risk, we encourage Protocol Labs to provide guidelines for application designers to understand how to design the scoring function, along with providing additional information on the circumstances where using an application-specific scoring function is advisable or warranted (see Suggestion 2).

One effect that may be difficult to assess, however, is how the stability of the network is affected when a subset of peers runs on various different scoring functions instead. The result will likely differ based on which alternative scoring functions are used and whether they perform better or worse in rating their peers in the particular setting the node is in.

Although it is reasonably commented, the scoring function and how scores are changed and updated within the codebase is disorganized and can be difficult to follow. An `internalTracer` interface delegates calls to different scoring functions as events like message delivery occur. A `score()` function sums the results of events mutating state and provides this result to the router for decision making. These events contain a mutex as they may be reading or writing to the score state in parallel. The `score.go` file contains functions that refresh time parameters, calculate score, and record events. As this interface grows to capture the needs of applications implementing it, we suggest a refactor of the score calculation code to be more explicit with its changes and mutations (see Suggestion 7).

To what extent these mitigations are effective needs to be further evaluated based on realistic simulations. While the simulations presented to us indicate that the network is stable and reliable, some improvements to the modeling are discussed in the next section.

Simulations

The Gossipsub team provided us with access to a simulator repository written in Go, in addition to a network of virtual machines to test it against, which our team found to be very useful. We commend the Gossipsub team for proactively working on simulations and encourage further exploration into various simulation possibilities.

To this end, we created and used our own internal modeling and data. During the course of the audit, we wrote two simulation scripts that map and measure distances of publishers to a node given random distances. We then measured the distribution of scoring based on first message delivery, message delivery rate, and time in mesh parameters. A simple way to examine all of the parameters in use and how the mesh is maintained over time would be to continue work on a lightweight simulator designed to quickly look at the scoring system.

The Gossipsub team also generated a separate simulation script designed to look specifically at how message delivery scores are distributed throughout the network. This script only examines the P2 scoring parameters (first message delivery), which comprise the majority of positive weights that affect score. However, there are other parameters that have yet to be simulated.

This audit makes no statements or warranties and is for discussion purposes only.

Given the time constraints faced by our team during the audit, along with the risk presented with the scoring system, we strongly recommend that further work is done and additional review time is spent evaluating possible attack strategies through simulations (Suggestion 3, Suggestion 4, and Suggestion 5).

Network Mapping

While looking for security vulnerabilities, we also investigated what observers might be able to learn about the network from observing it, either passively or actively. Gossipsub does not try to obscure this information so it is not considered a vulnerability. However, since this information may be useful in planning an attack, we have included analysis from this perspective.

Peer Score

Assuming that a peer has correctly implemented the Gossipsub v1.1 specification, has followed the recommendations for parameter weights, and that any given peer is simply using the defaults for a given application, it is possible to estimate what score another peer has given based on their behavior. In some cases, it would be necessary to observe a given peer from multiple connected peers to know that the peer in question has published a message but not sent it to all their connected peers, for example. The following thresholds are listed from lowest to highest:

- `GrayListThreshold`. Behavior: all Remote Procedure Call (RPC) messages are ignored. Must be lower than the published threshold.
- `PublishThreshold`. Behavior: self-published messages are sent to peers below the threshold. Must be lower than gossip threshold.
- `GossipThreshold`. Behavior: no gossip emitted to peers below the threshold and incoming gossip is ignored. Must be less than 0.
- `BaseLineThreshold`. Behavior: pruned from mesh without sending Peer Exchange. Always 0 and not configurable.
- `AcceptPXThreshold`. Behavior: peer accepts your peer exchange, connecting to peers you gave. Must be greater or equal to zero.

Network Topology

It is more useful however, to be able to measure where a peer is in relation to publishers. It is not possible to easily measure who a peer is connected to, but it is possible to measure a peer's distance from any publisher by observing their reaction time on meshed messages. In order to make the measurements very accurate, it is necessary to take latency into account, however, this can be measured by observing RPC response times.

On publishing a new message, a peer sends that message to all their peers that are above the publish threshold. If you have a peer that is connected directly to a publisher, you can detect when that message becomes available in the network and compare the time it is received from other meshed peers. If a peer has a fast response time to RPC (i.e. a low latency) but takes a longer time to deliver mesh messages, it can be concluded that it is likely they did not receive the message directly from the publisher but have instead received them indirectly from another peer. If the average network latency between peers is known (which, to some degree, must be an assumption or best guess) then we can estimate whether this peer is one hop, two hops, three hops, etc., from the source of that message.

If an attacker's goal is to delay the network (as in Issue B) then this information is very useful because having this knowledge may allow for a more efficient attack on the peers closest to the publishers.

This audit makes no statements or warranties and is for discussion purposes only.

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: Man In The Middle Can Fake Colocation	Resolved
Issue B: Love Bombing Attack	Resolved
Issue C: Graft Sniping	Resolved
Issue D: Potential Vulnerability if Insecure Peer Discovery is Used	Partially Resolved
Suggestion 1: Deprecate Secio	Resolved
Suggestion 2: Provide Guidelines for Choosing Application-Specific Scoring Function	Unresolved
Suggestion 3: Model Latency with Higher Precision in Simulation	Unresolved
Suggestion 4: Analyze Messages Propagation and Asymmetries in Peer Scores During Simulation	Unresolved
Suggestion 5: Simulate an Attacker Attempting to Circumvent the Flood-Publish Mitigation Against Message Censorship and Delaying	Unresolved
Suggestion 6: Loosen Coupling Between PubSub Struct and Router Types	Unresolved
Suggestion 7: Consider Refactor of Scoring Functionality	Unresolved
Suggestion 8: Improvements to the Design Specification Document Structure	Unresolved

Issue A: Man In The Middle Can Fake Colocation

Location

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/master/score.go#L244-L249

Synopsis

The score function has an aggressive colocation penalty for running multiple peers from the same IP address, with the peers being the recipient of the penalty. This can be used by a man-in-the-middle attacker to decrease the score of other peers.

Impact

The attacker can reduce the peer score of one node at one of its peers. This can lead to them being ignored or pruned.

This audit makes no statements or warranties and is for discussion purposes only.

Preconditions

The attacker needs to convince a peer to connect to its IP address, expecting the peer ID of a target peer.

Feasibility

The attack is straightforward with unsigned peer records.

Technical Details

First, the attacker broadcasts a fake peer record with the target's ID and the attacker's IP. When a victim connects to the attacker, the attacker proxies them to the target. The target and the victim will both believe the other has the attacker's IP address. If more than one victim is proxied to the same target, the target will start to negatively score the victims. This may cause their score to fall low enough that the target refuses to mesh with them or to ignore their gossip. The target will penalize all victims that connect through the attacker, considering them sybils. If a victim falls for the same trick twice by the same attacker, the same penalty is applied to both targets.

Remediation

A more reliable way of associating IDs with IP addresses is required. Since libp2p is already in the process of switching to signed peer records, an attacker will be unable to create a signed peer record that claims an ID they do not control.

Status

The Gossipsub team has clarified that the network should be set up using signed peer records in the Gossipsub v1.1 specification. Furthermore, signed peer records are now the default and need to be explicitly disabled in order to make the protocol susceptible.

Verification

Resolved.

Issue B: Love Bombing Attack

Location

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/master/score.go#L188-L208
https://github.com/LeastAuthority/libp2p-specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md#overview-of-new-parameters

Synopsis

The most significant positive factor in the score function is first message deliveries, following recommendations in the Gossipsub 1.1 design specification for relative parameter weights, since time in mesh is capped to a small value and message delivery failure is only negative.

Sybil attackers influence a victim to mesh with the sybils by behaving in a calculated way to make their score increase. This may cause the peer to unmesh from honest peers and produce a higher proportion of sybils in the network. As a result, the mesh gets an overall higher percentage of sybil members, which, for example, could result in delaying the network.

Impact

A single peer could be influenced to unmesh from honest peers. If a peer that is delivering many messages leaves the honest mesh, one possible impact is that it could cause the network to be delayed. If that delay hits critical thresholds such as block time, that delay is equally harmful as dropping messages.

This audit makes no statements or warranties and is for discussion purposes only.

Preconditions

The attacker needs a faster route from the publishers to the victim in addition to enough unique IP addresses such that the sybils can replace the mesh of the victim, without hitting the colocation penalty.

Feasibility

It would be straightforward to apply to a single peer, however, it would be more challenging but likely possible to pull off at a larger scale.

Technical Details

Due to the random structure of a gossip network, most peers are not connected directly to the publishers. Although peers may possibly be connected to some publishers, they are not connected to all of them. This lack of direct connection is essential to the scalability of the network since the publisher would otherwise require a significant amount of resources. However, an attacker could connect directly to publishers and then to a victim, likely having a shorter path and therefore getting the messages before the victim. By delivering the messages to the victim before the honest peers do, the victim will give them a higher score than the other peers and eventually unmesh the honest peers and only mesh with the sybils.

Remediation

This attack is possible because the attackers are able to create incoming connections at will and a Gossipsub peer treats these connections as equivalent to the outgoing connections that they control. If peers had a quota to always maintain *some* outgoing mesh connections, then the attackers would not be able to fully take over the peer's mesh, unless the peer willingly connects to them (which is somewhat more difficult for the attackers to ensure).

Status

Minimum quotas for outgoing connections chosen by the node have been established, and when pruning peers due to oversubscription, they remain connected to at least that quota of outgoing peers. As a result, it is now no longer possible for attackers to influence all a peer's connections, since that peer will always maintain at least some connections which are not controlled by the attacker.

Verification

Resolved.

Issue C: Graft Sniping

Location

Specification:
https://github.com/LeastAuthority/libp2p-specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md#opportunistic-grafting

Code: `handleGraft()` -
https://github.com/LeastAuthority/go-libp2p-pubsub/blob/master/gossipsub.go#L477

Synopsis

A Gossipsub peer accepts all graft requests and will not prune them until the next heartbeat, even if the graft puts the peer over the mesh limits. As a result, an attacker can request a graft and immediately send fresh messages which increases their score. If they increase their score enough, it is possible that honest peers will be unmeshed in a subsequent heartbeat.

This audit makes no statements or warranties and is for discussion purposes only.

Impact

This would allow for a more efficient way to carry out the love bombing attack (see Issue B).

Preconditions

The same preconditions apply as Issue B, which are further aided if the attacker knows when the victim's heartbeat takes place.

Feasibility

This attack can be performed from a cheap Virtual Private Server (VPS). To increase the chances, the VPS should be deployed in a datacenter as close to the victim as possible, which does not increase the cost or difficulty of the attack.

Technical Details

Similar to Issue B, sybils prearrange many connections to publishers. Connecting to a Gossipsub peer takes some time given that there are several layers of protocol handshakes to get through, at least one roundtrip each for multistream, secure channel, and transport upgrader. As a result, attackers would also need to prearrange a connection to each victim. Once they are ready and think they can rapidly deliver messages (faster than the peer is getting them otherwise), they send a graft request, rapidly followed by as many messages as possible. If they manage to deliver the first message, it is possible that their score becomes more than that of honest peers. If the peer is now over the mesh limits (`D_high`), then the peer should prune the lower scoring peers.

Remediation

This attack depends on the ability of a peer to request a graft and immediately be grafted. The remediation is simple and requires rejecting new grafts immediately if the peer is already at the high threshold.

Status

If a node already has `D_high` active mesh connections, incoming grafts are rejected, and a prune response is sent without delay. Outgoing grafts are still permitted, which allows recovery from eclipse events even when connected to many peers.

Verification

Resolved.

Issue D: Potential Vulnerabilities if Insecure Peer Discovery is Used

Location

Peer Discovery: https://github.com/LeastAuthority/go-libp2p-pubsub/blob/master/discovery.go

Distributed Hash Table (DHT) Peer Discovery: https://github.com/libp2p/go-libp2p-discovery/blob/e6ceacdf48dba72efae705a384a2f9f1b217db77/routing.go#L59

Synopsis

Gossipsub goes to some length to harden the protocol against attacks but recovery from an attack assumes that it is possible for honest peers to reliably connect to new random honest peers, via a peer discovery service. If that discovery service is attacked, the defense could potentially fail. The broader libp2p/IPFS ecosystem is highly modular, but some available peer discovery methods are too insecure to provide the assurances required by Gossipsub.

This audit makes no statements or warranties and is for discussion purposes only.

Impact

Depending on how severely the discovery service is attacked and how much the Gossipsub instance relies on it, it could severely impact the Gossipsub mesh's ability to recover from an attack.

Preconditions

A Gossipsub instance must depend on a vulnerable peer discovery service, such as DHT.

Feasibility

It would be easier to perform an attack on the peer discovery service than on Gossipsub directly. For the DHT discovery service, it would be fairly easy.

Technical Details

It is somewhat difficult to secure a DHT since they are open by design and assume that peers will interact helpfully with random strangers. The existing literature contains a number of attacks on them, such as the Eclipse attack [SNDW06].

In the Eclipse attack, attackers create fake nodes with keys nearby the target. When honest peers request that target, they are likely to get a response from an attacker. Thus, the attackers can censor a node or provide false information.

The libp2p Kademlia (DHT) implementation could be potentially be vulnerable to Eclipse attacks. Bucket assignment is based simply on peer ID, a value the attacker controls, so it is easy and feasible to generate new peers to Eclipse a given topic. In addition, DHT peers will recommend other peers that have responded to a RPC request, which is a somewhat lower bar than Gossipsub (since there is no peer score function). Therefore, it is probably easier to attack the DHT than to attack Gossipsub. However, under the assumption that a DHT is used for peer discovery, an attack on Gossipsub could begin with attacking the DHT.

Mitigation

Node operators can configure explicit peering arrangements instead of using the DHT.

Remediation

It would be very difficult to prevent this entirely. The libp2p DHT could be hardened by any of the following:

- The discovery service could use other less vulnerable services;
- Peers could detect if a given discovery service becomes unreliable (low rate of successful connections or a high rate of connecting to bad peers); and
- Rebalance to other services or alert the operator.

Our team recommends additional research into hardening the DHT and other discovery services.

It has also been suggested to use bootstrappers and peer exchange. New peers would connect to a bootstrapper and, when they are unmeshed, they become recommended peers to mesh with. It seems possible that this could be vulnerable to attacks (such as love bombing the bootstrappers), but it has not yet been as well studied or researched as the attacks on DHTs.

Status

The Gossipsub v1.1 specification now includes a recommendation that network operators may use peer exchange via bootstrapping instead of a peer discovery service. The Least Authority team has requested

This audit makes no statements or warranties and is for discussion purposes only.

that it be more explicitly stated by emphasizing that Gossipsub is vulnerable if the peer discovery can be attacked.

Verification

Partially Resolved.

Suggestions

Suggestion 1: Deprecate Secio

Location

https://github.com/LeastAuthority/libp2p-specs/blob/master/secio/README.md

Synopsis

Although this is not a vulnerability in Gossipsub or libp2p, it is possible for an attacker to create a valid connection between two Secio peers and both peers will believe the other is the server. This is possible because Secio is too symmetrical, with the client and server both act simultaneously.

However, both libp2p and Gossipsub are not vulnerable to this attack because they happen to be asymmetric, with the client acting first.

Technical Details

An attacker connects to two libp2p peers. First, they will use the multistream1.0 protocol to select which protocol to use, since the attack is not possible in multistream2.0. The attacker sends each peer a multistream1.0 message requesting the use of Secio. First, '\x13/multistream/1.0.0\n' to signify that it is a multistream connection, followed by '\x0d/secio/1.0.0\n' in order to request Secio.

Secio is currently in the process of being deprecated, but it is still supported in the default IPFS release, thus peers still support it. Because of multistream, the attacker can easily request a downgrade, resulting in both victim's use of Secio. This then causes both the connections to each peer being connected together. Although the peers will send each other authentication handshakes, which the attacker will not be able to decrypt, a valid connection has already been created.

Due to Gossipsub's colocation penalty, this could have been used to reduce the score of any two peers. However, even though Secio let the connection through, the next layer required a client initiated action, which prevented the attack from having any effect.

Mitigation

Avoid using symmetric secure channel protocols. It should also be noted as a security policy that protocols should not be too symmetric and an error should result if both sides act first.

Since Secio is already in process of being deprecated and the protocol is already sufficiently asymmetric, no immediate action is required. However, care should be taken not to remove the symmetry in connection initiation in Gossipsub.

Status

Secio is already in process of being deprecated and the protocol is already sufficiently asymmetric, no immediate action is required.

Verification

Resolved.

This audit makes no statements or warranties and is for discussion purposes only.

Suggestion 2: Provide Guidelines for Choosing Application-Specific Scoring Function

Location

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/53c709a6caefa379398d95c2a828d12d9f81ddfa/score.go#L230

Synopsis

The application score is weighted positively but the value of it is left up to the client implementation. This creates a disparity in scoring between different implementations that the specification is unable to reason about or protect against. For example, the Filecoin Lotus implementation has application score controls that protect against sybil attacks via malicious peer exchange by raising the score floor above what a node can achieve in normal gossip behavior. However, if a client implementation does not handle the application score correctly, or if a node is able to artificially increase its score, it would open up a route for sybil attacks via peer exchange.

Mitigation

The documentation should be updated to include guidelines for the application score, reasonable expectations of the score, and the dangers of mishandling it.

Status

No changes to the documentation had been made at the time of the verification review.

Verification

Unresolved.

Suggestion 3: Model Latency with Higher Precision in Simulation

Location

Simulation Code: https://github.com/libp2p/gossipsub-hardening

Synopsis

The Gossipsub team's current simulation uses a single latency in addition to some jitter. While this works as a first approximation, the real latency between computers on the internet depends on the distances between them and the quality of their uplinks. In order to better estimate the time it takes for messages to be received by all peers subscribing to a topic, a more realistic model of latencies should be used. This is especially important for selecting parameters for applications like Ethereum 2.0, which have hard deadlines for messages to be received by all parties.

We recommend modeling the latency by assigning each node a region and a boolean value that describes whether the node is in a data center or running on a residential connection. The distributions of the latency between data centers in the individual regions can be easily measured and then used in the simulations for the nodes running in data centers. For nodes running on residential connections, the latency needs to be additionally increased by about 15±5 ms. Such a model should give better insight on the impact of nodes with higher latency.

Status

The Least Authority team has recommended that additional simulation testing and analysis for Gossipsub v1.1 be performed, in order to further evaluate possible attack strategies through simulations in order to

This audit makes no statements or warranties and is for discussion purposes only.

test the hardening extensions of the protocol. This has not been done at the time of the verification review.

Verification

Unresolved.

Suggestion 4: Analyze Messages Propagation and Asymmetries in Peer Scores During Simulation

Location

Simulation Code: https://github.com/libp2p/gossipsub-hardening

Synopsis

Intuitively, given a pair of connected nodes A and B where node A keeps a very high score for node B, node A learns most new messages from node B, which suggests that node B does not learn about many messages from node A. This would result in a strong scoring asymmetry.

It is likely that this effect is influenced by a number of parameters. Understanding the extent requires simulation in a realistic model. We recommend recording the order in which nodes learn messages during simulation, as well as the peer scores during simulation, especially when performing the simulation in a more realistic network model, as described in Suggestion 3.

Status

The Least Authority team has recommended that additional simulation testing and analysis for Gossipsub v1.1 be performed, in order to further evaluate possible attack strategies through simulations in order to test the hardening extensions of the protocol. This has not been done at the time of the verification review.

Verification

Unresolved.

Suggestion 5: Simulate an Attacker Attempting to Circumvent the Flood-Publish Mitigation Against Message Censorship and Delaying

Location

Simulation Code: https://github.com/libp2p/gossipsub-hardening

Synopsis

The idea of the flood-publish mitigation is to prevent delaying and censoring messages by sending messages published. To do this, a node does not only send the message to the peers subscribed to the destination topic, but to all its peers. This helps protect against Eclipse attacks.

A sybil attacker that is connected to many nodes in the network is able to find the peers of the victim node by measuring from which nodes they receive messages authored by the victim first. The attacker can perform an Eclipse attack both on the victim node and its immediate peers to inhibit their messages. A simulation would show to what degree the protocol is able to prevent such attacks.

Status

The Least Authority team has recommended that additional simulation testing and analysis for Gossipsub v1.1 be performed, in order to further evaluate possible attack strategies through simulations in order to

This audit makes no statements or warranties and is for discussion purposes only.

test the hardening extensions of the protocol. This has not been done at the time of the verification review.

Verification

Unresolved.

Suggestion 6: Loosen Coupling Between PubSub Struct and Router Types

Location

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/53c709a6caefa379398d95c2a828d12d9f81ddfa/pubsub.go#L36

Synopsis

Currently, the router types and the PubSub type have each other as a struct field. This is a sign of very strong coupling, while a more loose coupling would be favourable for being able to flexibly exchange routers. One effect of this strong coupling is that it is not possible to use a new router that lives outside the go-libp2p-pubsub package.

The easiest way to achieve this would be to make a public interface implemented by PubSub that the routers could use to interact with the network. However, that interface may turn out to be quite large (another sign of strong coupling) and it is possible that a more comprehensive refactor would yield more satisfactory results.

Note that the PubSubRouter interface also is considerably large, which speaks in favor of a larger refactor. What speaks against it is that the expected complexity the code is about to gain in the medium term future is manageable, so the reward in increased maintainability may not be worth the cost.

Mitigation

The easiest way to achieve this would be to make a public interface implemented by PubSub that the routers could use to interact with the network. However, that interface may turn out to be quite large (another sign of strong coupling) and it is possible that a more comprehensive refactor would yield more satisfactory results.

Note that the PubSubRouter interface also is considerably large, which speaks in favor of a larger refactor. What speaks against it is that the expected complexity the code is about to gain in the medium term future is manageable, so the reward in increased maintainability may not be worth the cost.

Status

No changes were implemented at the time of the verification review.

Verification

Unresolved.

Suggestion 7: Consider Refactor of Scoring Functionality

Location

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/53c709a6caefa379398d95c2a828d12d9f81ddfa/score.go

https://github.com/LeastAuthority/go-libp2p-pubsub/blob/53c709a6caefa379398d95c2a828d12d9f81ddfa/score_params.go

This audit makes no statements or warranties and is for discussion purposes only.

Synopsis

The Scoring function can be difficult to follow and reason about. Consider a refactor of the Scoring function to make it a more explicit and declarative style of mutating the `TopicScore` state. Additionally, to maintain separation of concerns, we recommend pulling the Scoring functionality out into its own package and making an interface that the Score package would fulfill. This would be more Go-idiomatic and would allow for different scoring systems to be easily tested and built.

Mitigation

Consider a refactor of the Scoring function to make it a more explicit and declarative style of mutating the `TopicScore` state. Additionally, to maintain separation of concerns, we recommend pulling the Scoring functionality out into its own package and making an interface that the Score package would fulfill. This would be more Go-idiomatic and would allow for different scoring systems to be easily tested and built.

Status

No changes were made to the scoring function at the time of the verification review.

Verification

Unresolved.

Suggestion 8: Improvements to the Design Specification Document Structure

Location

Design Specification v1.0:
https://github.com/LeastAuthority/libp2p-specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md

Design Specification v1.1:
https://github.com/LeastAuthority/libp2p-specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md

Synopsis

Although the documentation uses versioning, it is required to read both versions in order to understand the protocol, given that v.1.1 is not solely an upgrade of the v1.0 documentation. For example, while the v1.0 documentation provides an introduction to the design motivations, history, and implementations of the flood and gossip router extensions, the v.1.1 specification focuses specifically on the security hardening efforts made by the introduction of the scoring system, providing only a short overview and link to the other operations of Gossipsub in the v1.0 documentation, such as control messages and heartbeats.

Mitigation

Since both are presented as the Gossipsub protocol, it would benefit the reader if both versions were consolidated. Furthermore, we recommend that the scoring documentation be made into a specific chapter or extension to one document that encapsulates all of the protocol design information.

Status

No changes were made to the design specification document structure at the time of the verification review.

Verification

Unresolved.

This audit makes no statements or warranties and is for discussion purposes only.

Recommendations

Since the protocol is used in multiple decentralized applications, such as Ethereum 2.0 and Filecoin, Gossipsub is key to the development of decentralized protocols. We recommend that the unresolved *Suggestions* and partially resolved *Issue* stated above are addressed as soon as possible and followed up with a second verification review by the auditing team. Also, we recommend the team consider investing in further analysis of simulations, internally, and allocating additional independent auditing time for this type of effort.

Our review finds that the Gossipsub team has already shown that v1.1 is much more resistant to attacks than v1.0. We encourage the Gossipsub team to continue making security a priority for the project in the various ways noted in this report: thoughtful design, including documented decisions, and a comprehensive and flexible peer scoring system, along with the use of simulators.

This audit makes no statements or warranties and is for discussion purposes only.

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

This audit makes no statements or warranties and is for discussion purposes only.

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.