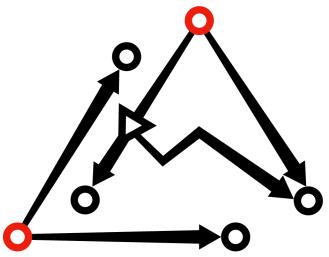


Least Authority

PRIVACY MATTERS

Overview

Background

The Ethereum Foundatio

Supporting Documentation

The following documentation was available to the review team:

- Devp2p: https://github.com/ethereum/devp2p
- EIP 778: Ethereum Node Records (ENR): https://eips.ethereum.org/EIPS/eip-778

Areas of Concern

Our investigation focused on the following areas:

- Considerations for individual nodes participating in the network
- Considerations for the collective network operations as a whole
- Potential attacks, such as:
    - Amplification attacks (e.g via spoofing make peers transmit data to a target),
    - Eclipse attacks (manipulate a target to connect only to malicious peers)
    - DoS attacks, such as:
        - With small effort, cause a peer to expend a large effort (cpu/memory etc),
        - With low bandwidth, cause a peer to utilize large bandwidth
    - Other possible attacks, as identified during the review
- The cryptography used by the protocol:
    - Suitability of the chosen algorithms
    - Correct use of the chosen algorithms

Findings

General Comments

These are overall findings that in most cases should be addressed holistically.

Design and Documentation Approach

We found the specification to be readable, well-explained and easy to understand, however, there were terms in the specification that lacked proper definition and technical description. This presented certain challenges for us and required some assumptions to be made in our audit, particularly around the Topic registration system. It is commendable that many security concerns are already noted, including a number of proposed mitigation strategies. This allowed us to identify leads early on in our investigation and expand upon some of the known shortcomings.

Identity Key Generation

The primary concern resulting from our review is the lack of a proof scheme for identity key generation, especially given that eclipse attacks have been a problem with the Ethereum network in the past. We strongly recommend that an identity proof system, like the one proposed in this report, be employed in the next iteration of the specification.

Topic Subsystem Complexity

In addition, we found that the specification did not adequately explain use-cases for the topic subsystem in order to justify its relative complexity. There were no data or test cases identified to support the need for its inclusion and, as a result, we were not able to properly evaluate the topic subsystem. We strongly recommend a follow up evaluation after this aspect of the specification is more thoroughly documented.

AES-GCM in the Wire Protocol

Overall, we found that in the wire protocol AES-GCM is used properly, and in accordance with best practices. However, this is based on certain assumptions and we were not able to assess side channel attacks due to the lack of code. Our assumption is that the implementation of `aesgcm_(de/en)crypt` is in accordance with the specification (https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf) and that the key auth-resp-key is a one time key.

Since there was no mention of the strength of the authentication tag, we were not able to reason about that particular component. However, it is worth noting that the use of short tag lengths will limit the

Our analysis found that 128bit key length is best practice for non quantum-computer attacks and, since messages are not longer than 1280byte ~=~2^6*128= bit, weak key attacks as described in https://eprint.iacr.org/2011/202.pdf, whether possible or not, can only lower the security to about 122bits, which is acceptable.

We also reviewed the following list of steps with which aes_gcm is used in the wire protocol:

1. auth-response = `aesgcm_encrypt(auth-resp-key, zero-nonce, auth-response-pt, "")`
2. message = `aesgcm_encrypt(initiator-key, auth-tag, message-pt, tag || auth-header)`
3. message = `aesgcm_encrypt(initiator-key, auth-tag, message-pt, tag)`
4. ticket = aesgcm_encrypt(ticket-key, ticket-nonce, ticket-pt, '')

For auth-response, the auth-resp-key&zero-nonce combination can only be used a single time. Since this

Specific Issues

We list the issues we found in the specification 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: Intentional Lack of Proof-of-x Scheme for Identity Generation	Unresolved: In Discussion
Issue B: Lookup Operation Does Not Include Disjoint Paths	Resolved
Issue C: Handshake Authentication is Broken	Resolved
Suggestion 1: Better Documentation of Terms and Definitions	Resolved
Suggestion 2: Handshake Tag Can Be Used to Confirm Protocol	Unresolved
Suggestion 3: Make auth-tag the Output of an Approved RBG	Unresolved

Issue A: Intentional Lack of Proof-of-x Scheme for Identity Generation

Synopsis

The specification directly acknowledges the lack of employment of a proof scheme for node identity generation, supporting this choice by citing hardware requirements posing a barrier to entry. This allows nodes to freely choose their node fingerprint, making eclipse attacks trivial.

Impact

Critical. An attacker can assert control over parts of the network identity key space, thereby allowing them to deny service, return invalid information, and poison the routing tables of other network participants.

Preconditions

Attacker must control a minimum of 16 IP addresses in order to fill one of a target's buckets entirely with nodes they control. Controlling more IP addresses increases the impact.

Feasibility

Trivial. This can be done for free using Amazon EC2 t2.micro instances.

Technical Details

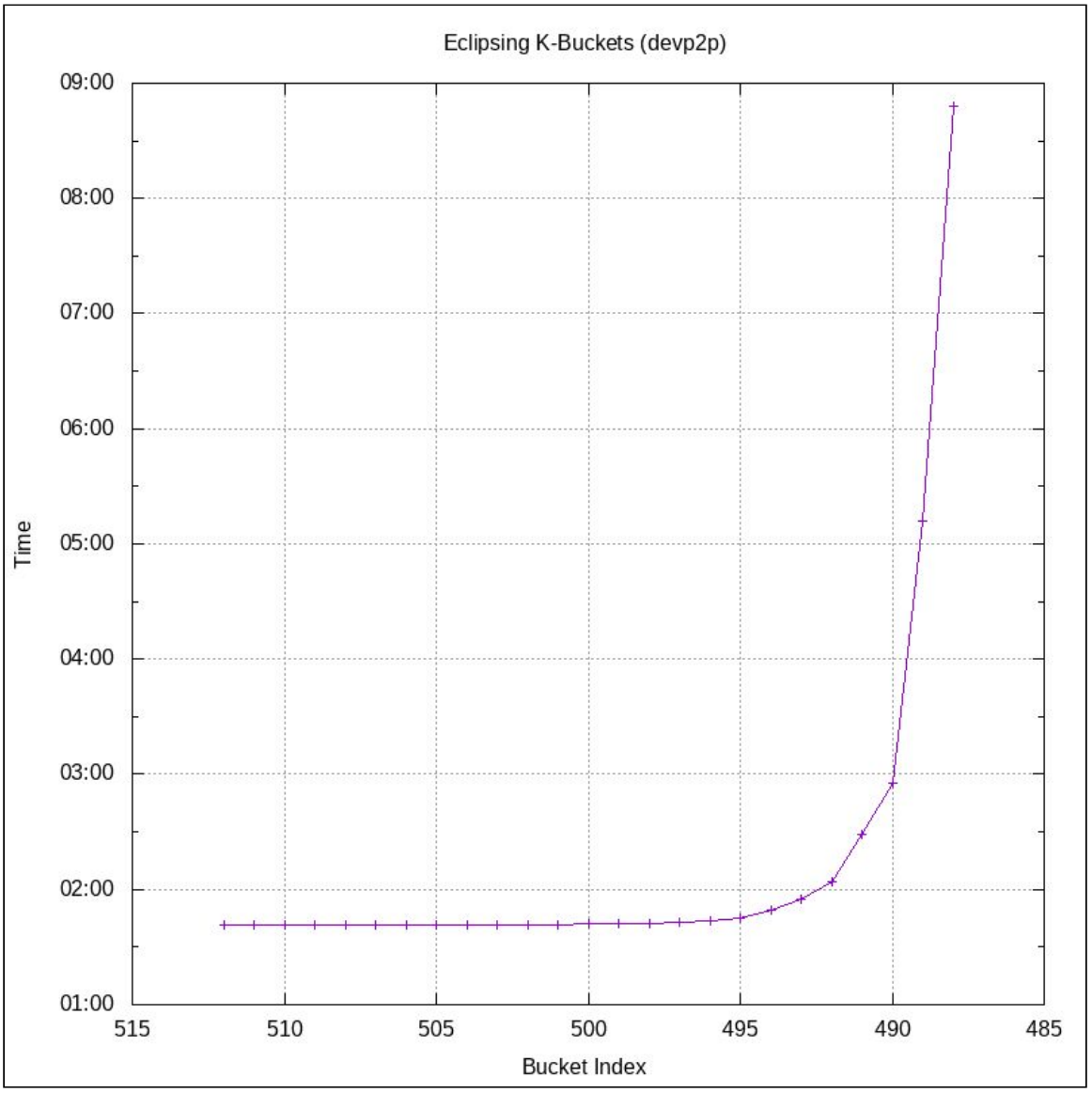
Kademlia-based overlay networks use the XOR of two identity keys as a distance metric. This distance determines the index of the K-bucket (a sorted peer list) in which a given peer will be placed. Since there are always more identities that are very distant than very close, the Kademlia routing table contains mostly peers that are very distant. The goal of an Eclipse attack is to generate identity keys that target a specific bucket in order fill a victim's peer list with nodes that are controlled by the attacker.

In a Kademlia network where nodes may freely select their identity key, this type of attack is trivial. The DevP2P specification places the requirement that identities are validated - the key is the hash of the public portion of a SECP256k1 pair. This has the side effect of preventing nodes from choosing any arbitrary identity key. Since these keys are computationally cheap to generate, we have demonstrated that eclipse attacks are still trivial.

In our proof of concept attack (which is shown in the accompanying repository), we took an arbitrary valid target key and attempted to fill that target's K-buckets (16 peers each) with valid identity keys derived in a manner consistent with the specification. The code for this attack was intentionally left poorly optimized such that it:

- Runs on a single CPU core, generating a single identity at a time
- Cracks buckets sequentially, starting at the furthest
- Discards keys found that do not belong in the bucket that is currently being cracked

This poor optimization is intended to demonstrate a sort of "best case scenario" - the attacker not having access to capable hardware. Our results showed that over the course of a 7 hour test, we were able to fill the target's furthest 24 buckets completely (384 nodes). Perhaps more importantly, the first 19 of these buckets were cracked within the first half hour or the test.



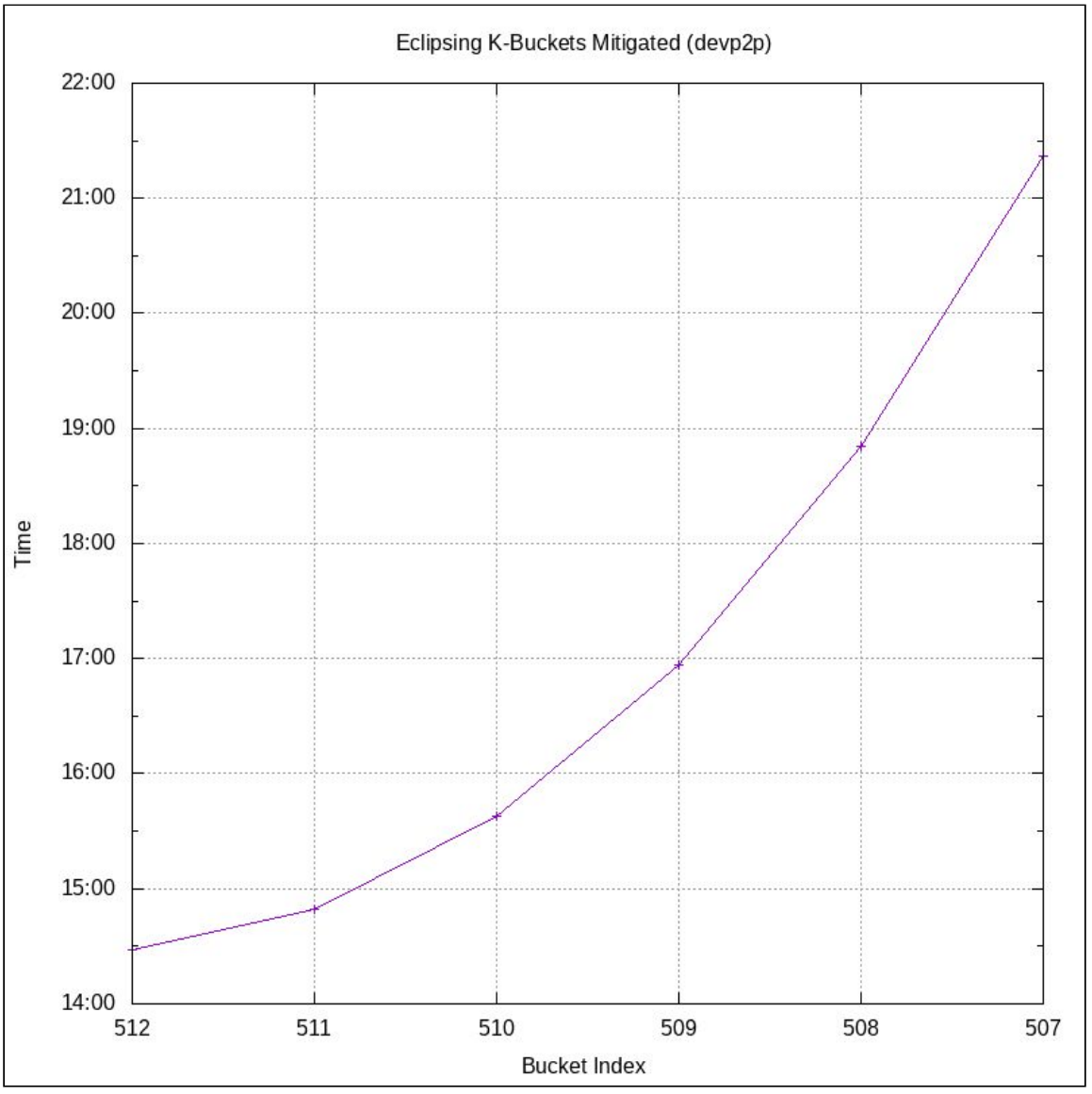
As the plot above shows, the time to crack each bucket takes roughly twice the time to crack the previous bucket, which is consistent with how we understand Kademlia's routing table. Since there are always more nodes that are distant than close, eclipsing even the first 18 buckets (as is shown to be completed within minutes) is more than sufficient to cause significant disruption to a given target(s). Given the very

low barrier to this attack, we believe that it could be carried out trivially using free cloud infrastructure such as Amazon t2.micro and that attackers with access to even modest hardware could cause critical disruption in the network.

Mitigation

The specification acknowledges that a proof scheme is a well-accepted mitigation strategy for eclipse attacks, yet dismisses the use of one citing that mobile devices would not be able to participate in the network. This may be true for CPU intensive proof-of-work schemes, but is not necessarily the case for RAM intensive schemes.

Requiring that identity keys are derived from an asymmetric proof like Equihash could prove to be a strong mitigation strategy for these types of attacks. Memory hard proofs make for an excellent trade off. Even lower end semi-modern mobile devices often have at least 3GB of RAM and can generate a network identity without much burden or time. However, this baseline memory requirement excludes an entire class of virtual private server offerings, such as free Amazon t2.micro instances. We ran our same attack, this time doing so using our mitigation strategy for comparison.



Given the same test case, but with our mitigation strategy implemented, we were only able to fill the furthest 5 buckets within the same 8 hour period. Even this was only achievable due to availability of

~4GB RAM, making this attack much harder to parallelize than the previous test. The critical difference here is the difference between the initial time to crack the first bucket. On the first bucket, it took about 1 minute per identity, because most generated keys are valid for that bucket. We feel this is a reasonable barrier for users upon first connecting (or rather first generating an identity key), but poses significant burden on would-be attackers.

Of course, this strategy is not perfect and an attacker could generate the identities offline on other hardware for use in a similar manner, acknowledging that doing so would prove to be significantly more expensive. Furthering this scheme such that nodes are required to continually improve their proof over time is a worthwhile research endeavour.

Remediation

Complete remediation of this vulnerability is an area of active research and there are no known strategies that completely eliminate the possibility of a determined and capable attacker carrying out this type of attack.

Status

The specification acknowledges that proof schemes for identity generation are well researched, yet still opposes their use due to what the authors consider to be resource constraints. A ticket has been opened for public comment on how to proceed.

Verification

Unresolved: In Discussion.

Issue B: Lookup Operation Does Not Include Disjoint Paths

Synopsis

The specification does not specify the use of disjoint lookup paths, which can make various identity and malicious routing attacks far more effective

Impact

High. If an attacker manages to control multiple nodes in a lookup path, they can manipulate the information delivered back to the lookup originator.

Preconditions

Attacker must control multiple nodes along a particular lookup path.

Feasibility

Trivial. This can be done very inexpensively using Amazon EC2 t3.nano instances or a similar offering from any virtual private server provider.

Technical Details

The specification outlines a solution where Kademlia table buckets are limited to two nodes from every /24 IP subnetwork and limits the entire table to 10 nodes per /24 IP subnetwork. This is susceptible to IP spoofing and anonymizing protocols where IP addresses aren't scarce or reliably related to a single node. The handshake would not prevent this because according to the provided specification, the handshake does not verify the IP address of the requesting node or the recipient node.

The specification does not provide an exact lookup algorithm for nodes. Instead, they simply define their lookup as an "iterative lookup", which is consistent with Kademlia. However, particularly in the case of a

moderate number of adversarial nodes, the lookup algorithm benefits significantly from implementing a lookup that utilizes multiple disjoint paths to find nodes in the network.

Mitigation

Instead of limiting by IP address and subnets, we can rely on the mitigation previously outlined in Issue A in combination with an improved lookup algorithm that utilizes multiple disjoint lookup paths. In the original Kademlia specification, lookup requests iteratively query nodes, however the lookup fails if an adversarial node is encountered.

To improve the lookup algorithm, we can implement multiple disjoint lookup paths. First, we break up the `k` closest nodes into `d` independent buckets, and then we kick off `d` parallel `FIND_NODE` requests as usual, with one difference: we ensure we only query each node once during all of these lookups. This ensures that the lookup paths are truly disjoint and that encountering an adversarial node does not result in a failure or in the adversarial node becoming a contact and possibly becoming a bucket entry of ours.

Remediation

Implement multiple disjoint lookup paths.

Status

The specification has been updated to include details around session state storage, which resolves the issue where IP addresses are not verified in the handshake. In addition, the specification includes that multiple disjoint paths should be used during lookups.

Verification

Resolved.

Issue C: Handshake Authentication is Broken

Location

https://github.com/ethereum/devp2p/blob/master/discv5/discv5-wire.md#handshake

Synopsis

The handshake protocol allows an active attacker to undermine authentication.

Impact

High. Allows active attackers to impersonate other nodes.

Technical Details

The specification states

```
auth-response-pt = [version, id-nonce-sig, node-record]
version          = 5
id-nonce-sig     = sign(static-node-key, sha256("discovery-id-nonce" ||
id-nonce))
...
```

The `id-nonce-sig` proves liveness to the receiver of the packet, but since it does not sign the ephemeral key of the initiator, an active attacker can simply replace the ephemeral key with one that they know the secret to. Then, the responder will accept, and use a key that is known to the attacker (but not the initiator).

Remediation

This class of protocols is very difficult to design and implement correctly. As a result, there have been numerous attacks on TLS 1.2 in the last ten to twenty years.

We recommend learning from history by using a well-understood key exchange protocol such as DTLS, preferably DTLS 1.3, even in its current draft version.

Mitigation

Another option is to include the ephemeral key in the computation of the signature:

```
id-nonce-sig = sign(static-node-key,
                    sha256("discovery-id-nonce" || id-nonce || init-eph))
```

This way, the responder will notice that the adversary replaced the ephemeral key and the attack is prevented.

Status

The mitigation strategy recommended has been included in the updated specification.

Verification

Resolved.

Suggestions

Suggestion 1: Better Documentation of Terms and Definitions

Location

https://github.com/ethereum/devp2p

Synopsis

There were several places in the protocol where terms were simply never defined or were insufficiently detailed in their definition. This required us to make some assumptions around the basic workings of some parts of the system.

For example, the Topic Registration system lacked any meaningful documentation which forced us to make some assumptions around its purpose and mechanics in the protocol.

Mitigation

More thorough documentation and definition of terms in the protocol.

Status

The topic index and registration system details has been expanded in the updated specification.

Verification

Resolved.

Suggestion 2: Handshake Tag can be used to Confirm Protocol

Location

https://github.com/ethereum/devp2p/blob/master/discv5/discv5-wire.md#handshake

Synopsis

The `tag` part of handshake messages can be used to confirm that the node protocol is used. Consider three nodes A, B and C. If all three nodes communicate with each other, the xor of their tags will result in 0:

```
tag-A-to-B              = xor(sha256(B-node-id), A-node-id)
tag-A-to-C              = xor(sha256(C-node-id), A-node-id)
tag-B-to-A              = xor(sha256(A-node-id), B-node-id)
tag-B-to-C              = xor(sha256(C-node-id), B-node-id)
tag-C-to-A              = xor(sha256(A-node-id), C-node-id)
tag-C-to-B              = xor(sha256(B-node-id), C-node-id)

xor1 = xor(tag-B-to-A, tag-C-to-A) = xor(B-node-id, C-node-id)
xor2 = xor(tag-A-to-B, tag-C-to-B) = xor(A-node-id, C-node-id)
xor3 = xor(tag-A-to-C, tag-B-to-C) = xor(A-node-id, B-node-id)

xor12 = xor(xor1, xor2) = xor(A-node-id, B-node-id) = xor3
xor(xor12, xor3) = 0
```

Since the attack is technically just out of scope, it is only added as a suggestion. Mostly, it shows that key exchange protocols are complex and it is very easy to miss subtle issues.

Mitigation

Use a well-understood key exchange protocol.

Alternatively, this is preventable by including some randomness, such as the nonce, in the computation:

```
tag = xor(sha256(dst-node-id || id-nonce), src-node-id)
```

Status

The specification authors have not decided if or how to proceed with the mitigation of this issue.

Verification

Unresolved.

Suggestion 3: Make auth-tag the Output of an Approved RBG

Location

https://github.com/ethereum/devp2p/blob/master/discv5/discv5-wire.md#handshake

Synopsis

Since RBG mode is used, there are constraints on the number of invocations, i.e the total number of invocations shall not exceed 2^32, w.r.t. to any fixed initiator-key. Also, auth-tag is a critical security parameter as defined in FIPS Pub. 140-2 and the RGB generator must properly handle edge cases like loss of power, etc.

Mitigation

The auth-tag must be the output of an approved RBG, which must be independently initiated on any device.

Status

Mitigation strategy has not been included in the updated specification, but is planned for a future version.

Verification

Unresolved.

Recommendations

We strongly recommend that an identity proof system, like the one proposed in this report, be employed in the next iteration of the specification.

We also recommend that the Ethereum team continue to provide well-documented and easy to understand resources about the design and implementation of the node discovery protocol. Doing so will best facilitate the continued assessment of their security by both the community and future security auditors.

Lastly, we recommend that the *Issues (A)* and *Suggestions (2 & 3)* stated above are addressed as soon as possible and followed up with verification by the auditing team. In addressing the items in this report, the Least Authority team is available for discussion and consultation, as needed.

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 Review

In manually reviewing all of the documentation and code, we look for any potential issues with design and code logic, including 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 documentation and 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 interip

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.