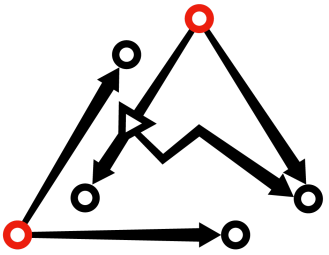

Least Authority

PRIVACY MATTERS

Nervos Blockchain

Final Security Audit Report

Nervos

Report Version: 18 October 2019

Table of Contents

Overview

Background

Nervos has requested that Least Authority perform a security audit of the Nervos Network, an open source multi-asset, Proof of Work blockchain, featuring a novel consensus scheme called NC-Max. Nervos is a decentralized application network consisting of a layered architecture, including the layer 1 protocol known as CKB (Common Knowledge Base), the foundational layer of the Nervos Network, in addition to the layer 2 protocols and scaling solutions.

The following components were considered in scope:

1. Consensus
    - NC-Max (a variation of the Nakamoto consensus)
    - PoW hash function, Eaglesong
    - Block verification logic
2. Transaction
    - Token transfer
    - Transaction fee
3. Economic Model
    - New token issuance
    - NervosDAO
4. Smart Contract
    - CKB-VM
5. Communication
    - P2P protocol / implementation
    - Serialization / deserialization
    - Eclipse attacks
    - RPC implementation

Project Dates

- **August 26 - September 25**: Code review completed
- **September 30**: Delivery of Initial Audit Report
- **October 14 - 17:** Verification completed
- **October 18:** Delivery of Final Audit Report

Review Team

- Ramakrishnan Muthukrishnan, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer
- Emery Rose Hall, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Nervos Blockchain 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:

- Nervos Network: https://github.com/nervosnetwork

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

Nervos CKB: 8a28087e38efd5efd226393193392d07912c7c39

Nervos CKB System Scripts: 7fc45c07296b8693d484d7c51ea2e1cc64b602a2

Nervos CKB VM: 03847a48b92753dfb818998ded41e89b02e8a6b5

Nervos Neuron: 4926d394c8480db11b7c5f4632d0df70be34499f

Nervos P2P: ef007b780f409edb23ab84101d508b691457d9bf

Nervos RFCs: 0316b6797482779259d12fc1b771b249ccbb4547

For the verification, we examined the Git revision:

Nervos CKB: 23904c76eb34e8845ebfcecac0f8bfc0a421bbb5

Nervos CKB System Scripts: 315e2a339a126ed7ad3d0b62527892c6d21583d0

Nervos CKB VM: cd8ef6ac061fe01a8dd9e162c222ad68052370c6

Nervos Neuron: 2eef612339fcba11f3a5b721a89d95393fc5d551

Nervos P2P: f6cf320517a35290d40f68bd1fe1badaf3c542ff

Nervos RFCs: 17c738d8f48feb53ca7a3ad36677fadc8942b033

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:

- Nervos Network RFCs: https://github.com/nervosnetwork/rfcs

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the protocol implementation;
- User funds are secure on the blockchain and cannot be transferred without user permission;
- Vulnerabilities within each component as well as secure interaction between the network components;
- Correctly passing requests to the network core;
- Data privacy, data leaking, and information integrity;
- Key management implementation: secure private key storage and proper management of encryption and signing keys;
- Handling large volumes of network traffic;
- Resistance to DDoS and similar attacks;
- Aligning incentives with the rest of the network;
- Vulnerabilities, potential misuse, and gaming of smart contracts;
- Any attack that impacts funds, such as draining or manipulating of funds;
- Mismanagement of funds via transactions;

- Inappropriate permissions and excess authority;
- Secure communication between the nodes;
- Special token issuance model; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

During our investigation, we found the code to be of good quality. The documentation provided by Nervos in the form of Request for Comments (RFCs) was helpful and served to explain and provide context regarding the design and implementation decisions made by the Nervos team. The RFCs were well formatted, logically structured, and provided deep insight into the system and design choices. Along with the documentation published and provided to us by the Nervos team, we were able to understand the blockchain at a fairly deep level using only the specifications. This was highly beneficial for the auditing process as it allowed us to separate the code from the theory when and where necessary.

However, we found that there was a general lack of detailed commenting in the code itself. In addition, several of the code repositories we reviewed exist in personal GitHub accounts. Instead, we recommend the use of the Nervos GitHub organization, a practice that makes the code easier to locate and review. Please find additional information on this in *Suggestion 2*.

NC-Max Consensus

The RFCs included in the scope of this audit covered the consensus algorithm at length. However, review of the "NC-Max: Breaking the Throughput Limit of Nakamoto Consensus" specification, detailing the proofs, was out of scope. As a result, we only reviewed what was described in the RFC. While our audit of the RFCs found no specific security issues, we recommend more measurement data on the performance improvement claims is gathered and reviewed.

In addition, while NC-Max incorporates orphan blocks to achieve a more accurate difficulty estimation in order to counteract selfish mining, the mining reward function still implements a competition where the winner is awarded the entire reward. This leads to an extremely high payoff variance, especially for miners with limited computational power. Such a miner often needs to wait a significant amount of time to receive its fair share. As a result, a rational miner is motivated to collaborate with others by forming mining pools, which distributes the work, lowers the variance, and makes the income more predictable as a result. However, potential issues of mining pools are known and well documented and should be considered.

Peer Discovery

The Nervos peer discovery network is well designed and accounts for known security issues in other similar blockchains, particularly issues concerning the routing table and eclipse attacks, which Bitcoin notably suffered. The Nervos team is aware of these issues and has documented their approaches to preventing them in the relevant RFCs, which was helpful in our review.

Economic Model

Our review of the NervosDAO and associated token issuance models did not uncover any specific issues. The manner in which state rent is collected and miners are compensated appears to be fair and sustainable. It is, however, worth noting that crypto-economies are an area of very active research and the long-term viability of any of the current models is unknown. It's our recommendation that the Nervos team

periodically conduct economic viability evaluations by analyzing how funds move around the network to gain a more concrete understanding of how users are participating.

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: Clear Secrets After Use	Resolved
Issue B: Combine Eaglesong with a Well-Known Hash like Keccak	Unresolved
Issue C: Use an Alternative to libp2p's secio	Resolution in Progress
Issue D: Prevent Possible VM Escape Attacks	Unresolved
Suggestion 1: Improve Code Comments and Documentation	Resolution in Progress
Suggestion 2: Move Relevant Repositories to Nervos Organization	Resolution in Progress
Suggestion 3: Cargo-audit to Analyze Dependencies	Resolved
Suggestion 4: Regularly Fuzz Critical Libraries	Partially Resolved
Suggestion 5: Increase the Test Coverage	Resolution in Progress
Suggestion 6: Eliminate Invocations of Panic()	Resolution in Progress
Suggestion 7: Define/Use Safe Language to Write VM Scripts	Unresolved

Issue A: Clear Secrets After Use

Location

Code that defines Privkey and various traits:
https://github.com/LeastAuthority/nervos-ckb/blob/develop/util/crypto/src/secp/privkey.rs#L11

Synopsis

Privkey may be leaking secrets.

Impact

Even though the Privkey struct element inner is not public, it may leave traces in the stack. Rust does not allow uninitialized values, so it is difficult to exploit and the impact of this is currently very low. However, we cannot rule out the existence of a future unrelated vulnerability that may exploit it.

Preconditions

None.

Feasibility

Rust makes it difficult to create uninitialized memory which could claim a previously freed value of type Privkey. However, a debugger can still read leftover memory which leaves the secrets vulnerable.

Technical Details

In Rust, when a value goes out of scope, the memory used by that value is claimed back and reused for another allocation. The programmer is not required to do anything special to claim that memory. If a Drop trait implement exist for that type, the `drop()` function from this trait gets called. As a result, the leftover key or other secrets represented by the value is not cleared. Another process like a debugger can read this value.

Mitigation

Use a crate like zeroize or clear_on_drop.

Remediation

While doing code reviews of pull requests, inspect for any types that represent secrets and be diligent about using one of the above crates to clear the memory when the identifier goes out of scope.

Status

Nervos has fixed this issue by disabling Debug/Display trait on Privkey and using zeroize when Privkey is dropped, outlined in the following commit: https://github.com/nervosnetwork/ckb/pull/1701.

Verification

Resolved.

Issue B: Combine Eaglesong with a Well-Known Hash like Keccak

Location

https://github.com/LeastAuthority/nervos-rfcs/issues/3

Synopsis

Eaglesong has undergone some cryptanalysis, but not nearly as much as other well-known hash functions such as SHA3. As a result, our confidence that Eaglesong is multi-target one-way is not as high as with other well established hash functions.

That being said, we recognize the motivation to introduce and use a novel hash function.

Impact

If miners find attacks on the multi-target one-wayness of Eaglesong, this would give them the ability to mine faster than other members of the network. As a result, this may allow them to vastly increase their hash rate, possibly to the point that it exceeds 50% of that of the entire network (with much less hardware cost). Alternatively, they make money from having a much higher probability of receiving block rewards.

Preconditions

None.

Feasibility

The attacker would require solid knowledge in cryptanalysis, and potentially need to make a hardware implementation of the algorithm.

Technical Details

An attacker first finds a way to find a nonce such that the block hashes to a low value much quicker than with brute force. They then implement that algorithm in hardware and use it for mining in order to achieve a hash rate that other miners, that are using brute force, can not keep up with. The attacker can use this for mining with low electricity or to stage 51% attacks on the network.

Remediation

Use a combiner that preserves the one-wayness property to combine Eaglesong with a more established hash function like SHA3. We specifically recommend the composition combiner `H(m) = H1(H2(m))`. What remains is the choice of the second hash function. We propose two options: a safe option and an optimal option.

Safe option: We realize that when using a composition combiner, the hash rate is not higher than that of the slower hash. SHA3 and Eaglesong can be implemented very efficiently in hardware. For SHA2, this is not the case. When looking at the structure of SHA3 and Eaglesong, we realize that by Theorem 1[1], the security of the hash function is reduced to that of the permutations. Since Eaglesong and SHA3 use very different permutations, the security of the hash hinges on different problems. Therefore, attacks on one hash function are not likely to impact the other.

Optimal option: In order to avoid the attacker from breaking both hash functions at once, attacks on one hash should not be translatable to the other hash. As a result, they should be different in structure. Eaglesong is very similar to SHA3, but different to SHA2, so SHA2 should be used.

Status

The remediation was updated to incorporate feedback by Alan Szepieniec, the author of Eaglesong. The discussion can be found in the following GitHub issue:
https://github.com/LeastAuthority/nervos-rfcs/issues/3.

Verification

Unresolved.

Issue C: Use an Alternative to libp2p's secio

Location

https://github.com/LeastAuthority/nervos-ckb/blob/develop/network/src/lib.rs#L28

Synopsis

The secio protocol is a key exchange protocol that has not yet received the amount of analysis that it deserves. As can be noted from the history of several key exchange protocols, mistakes are subtle and lead to easily exploitable attacks. The prime example is TLS where there were repeated attacks for over a decade. Alternatively, TLS 1.3 has undergone rigorous analysis and implementations in several programming languages that are readily available. The Noise protocol framework easily allows designing and formally analyzing key exchange protocols.

[1] Bertoni, Guido, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the indifferentiability of the sponge construction." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 181-197. Springer, Berlin, Heidelberg, 2008. https://www.iacr.org/archive/eurocrypt2008/49650180/49650180.pdf

Impact

In the event that an attack on secio is found, it would be possible to apply it to all network traffic encryption in Nervos. Depending on the attack, this would result in the inability to guarantee authentication of node and data, as well as traffic confidentiality.

Preconditions

secio needs to be broken in some way. There have not yet been reports that this is the case.

Feasibility

The attacker must have the ability to intercept, modify and inject network packets. This is the security model that key exchange protocols are analyzed against. While this is not feasible for everyone, one also does not necessarily need to be a state-level actor.

Remediation

Use a key exchange protocol with more analysis than secio, preferably TLS 1.3.

Status

Nervos has notified Least Authority that they plan to begin research and level of effort estimation on switching to TLS 1.3.

Verification

Resolution in Progress.

Issue D: Prevent Possible VM Escape Attacks

Location

https://github.com/LeastAuthority/nervos-ckb-vm

Synopsis

If a smart contract manages to escape the VM, it could execute arbitrary code with user privileges.

Impact

Arbitrary code execution on the VM host.

Preconditions

None.

Feasibility

The attacker gets the node of the user to execute a smart contract code that is designed to escape the VM. Most virtualized environments and runtimes of interpreted languages have experienced attacks from this class, from JVM over V8 to Qemu. Attacks are less likely in this case if the VM does not offer features like a virtual network device. However, the guest code can access values from the blockchain. The code implementation in Rust helps to protect against the vulnerability, but there may also be subtle bugs in the interpreted assembly VM or the AoT compiled VM.

Technical Details

The attacker needs to be able to get the victim to execute their malicious smart contract. The way in which this would happen likely depends on the wallet that is being used. That script could then escape the VM and execute code.

Mitigation

Run the VM in a (para-)virtualized environment. This could go from using Linux cgroups to running it inside a Qemu VM.

Remediation

An alternative approach could be to use formal verification tools like Coq to develop a VM and show that it is not prone to escapes. We understand that this is a significant undertaking and recommend running the VM in a virtualized environment.

Status

Nervos has considered running a virtualized execution environment in the development phase, but opted for a design. Nervos provided the following reasoning:

"*1) A virtualized environment such as qemu will significantly complicate the design of CKB VM. Not only will we need to introduce a huge dependency like qemu, the design of a fast and secure communication protocol between CKB and the execution environment is also hard to get right. Even when we manage to get all of those right, a design like qemu could still suffer from VM escape problem. A simpler design, on the other hand, reduces the attack surface and is also far easier to reason about. In CKB VM's case, the whole VM could be implemented in about 8000 lines of code, which is quite pleasant to review and reason about.*"

2) While a direct implemented VM could suffer from VM escape problem, we've carefully designed and implemented the VM to reduce the risks here as much as possible. CKB VM is mainly implemented in Rust, a language known to provide memory safety while preserving high performance. Although a small portion of CKB VM is written in assembly for higher performance, the relevant code here is carefully written and reviewed to avoid security risks. In fact, a second and different team has thoroughly audited the code, they have found no security problems in the code.

In a nutshell, we are confident the current design of CKB VM layer is the best combination of simpler design, and the best security we can provide.

Note that we only talk about qemu here, there is simpler solution such as Linux cgroups, but those solutions are not portable enough to support all platforms CKB is designed to run on."

While Least Authority recognizes that this is a significant and complex undertaking, we still recommend the suggested remediation approach.

In addition, while it is true that Qemu could also be vulnerable to VM escape attacks, it would require a significant amount of additional effort. Furthermore, being aware of an unreported vulnerability in Qemu allows attacking Nervos in addition to several other services. That said, it is not clear if and how Nervos would be a worthwhile target.

Regarding the lack of portability of cgroups, most other platforms supply a similar functionality. As long as the implementation for a platform is not available, the system can fall back to using Qemu.

Finally, despite the fact that small portions of the code are written in Assembly, they are critical sections of the code base. Least Authority cannot comment on the other third-party audit results as they were not incorporated or considered in the scope of our review.

Verification

Unresolved.

Suggestions

Suggestion 1: Improve Code Comments and Documentation

Location

https://github.com/LeastAuthority/nervos-ckb

https://github.com/LeastAuthority/nervos-ckb-system-scripts

https://github.com/LeastAuthority/nervos-ckb-vm

https://github.com/LeastAuthority/nervos-neuron

https://github.com/LeastAuthority/nervos-p2p

https://github.com/LeastAuthority/nervos-rfcs

Synopsis

We found documentation lacking at a functional and interface level within the codebase.

Mitigation

We recommend adding more code comments at a functional and interface level to the code base. Additionally, documentation separate from the code including setup, development, and packages within the code would allow new contributors and reviewers to understand the entire system more easily and efficiently.

Status

Nervos has acknowledged this suggestion and has noted that they will continue to improve code comments and documentation.

Verification

Resolution in Progress.

Suggestion 2: Move Relevant Repositories to Nervos Organization

Location

https://github.com/rink1969/eaglesong/

Synopsis

Repositories that are closely related to the Nervos network reside outside of the Nervos GitHub organization.

Mitigation

We suggest moving the above repositories into the Nervos organization on GitHub.

Status

Nervos has acknowledged this suggestion and plans to move the Eaglesong and the serialization repository molecule to the Nervos Organization on Github.

Verification

Resolution in Progress.

Suggestion 3: Cargo-audit to Analyze Dependencies

Location

All Rust packages related to Nervos.

Synopsis

Cargo-audit found a few issues in the dependencies of CKB. This crate is an easy way to automatically check for known vulnerabilities. For example, running cargo-audit on the CKB repository version that is being audited gives these warnings:

```
error: Vulnerable crates found!


ID:      RUSTSEC-2019-0019

Crate:     blake2

Version: 0.8.0

Date:     2019-08-25

URL:     https://github.com/RustCrypto/MACs/issues/19

Title:     HMAC-BLAKE2 algorithms compute incorrect results

Solution: upgrade to: >= 0.8.1


ID:      RUSTSEC-2019-0013

Crate:     spin

Version: 0.5.0

Date:     2019-08-27

URL:     https://github.com/mvdnes/spin-rs/issues/65

Title:     Wrong memory orderings in RwLock potentially violates mutual exclusion

Solution: upgrade to: >= 0.5.2


error: 2 vulnerabilities found!
```

Mitigation

We recommend using cargo-audit in the build/continuous integration system or possibly as a pre-commit hook. We noticed that cargo-audit is invoked in the Makefile included in the repository, we recommend using it more stringently to identify issues in the existing and new dependencies by running it ton every commit.

Status

Nervos has set up dependabot on repositories, which has similar features as cargo-audit. Makefile has a security-audit target that runs cargo-audit in which the Make target gets called from the Travis CI. Nervos has also noted that they will consider adding cargo-audit back as a cron job in Travis.

Verification

Resolved.

Suggestion 4: Regularly Fuzz Critical Libraries

Location

https://github.com/nervosnetwork

Synopsis

Regularly fuzz all the network facing libraries in addition to the VM.

Mitigation

Use cargo-fuzz to fuzz various network facing libraries. Design APIs such that it can be amenable to fuzzing.

We fuzzed parts of the VM. While we ran into some timeout issues that we were not able to investigate in detail, there were no crashes.

Status

Nervos currently uses the proptest library to do random input tests and have acknowledged that the additional fuzz testing on components related to IO would be beneficial, per Least Authority's recommendation.

Although proptests partially address this suggestion, it is recommended that smart fuzz testing (afl or similar) is done periodically on functions that parse inputs from network.

Verification

Partially Resolved.

Suggestion 5: Increase the Test Coverage

Location

https://github.com/nervosnetwork

Synopsis

Code coverage of tests is low.

Mitigation

Add code coverage reports to the CI system so that one could see the increase/decrease in code coverage caused by each change. `cargo tarpaulin` is a method to collect code coverage.

At the moment, the CKB and CKB-VM repositories have about 60% and 56% test coverage, respectively.

We understand that the strong type system eliminates many errors, however, they are not substitutes for tests. Designing the functions for fuzzing and property testing would certainly go a long way in increasing the robustness of the code.

Status

Nervos has noted that coverage should exclude some generated files and that the coverage of CKB-VM is 91% while CKB is about 60%. Nervos plans to increase the feature coverage first by adding more integration tests.

Least Authority confirms that some additional tests have been added to the CKB repository and that adding tests and increasing coverage is a time consuming process, and therefore is a long term endeavor.

Verification

Resolution in Progress.

Suggestion 6: Eliminate Invocations of `panic()`

Location

https://github.com/nervosnetwork

Synopsis

`panic!()` call aborts the program. This can result in crashes that can be confusing for the end users.

Mitigation

There are not many `panic!()` calls in the repository. It is recommended to convert these into values of type `Result<>`.

Status

Nervos acknowledged the issue and plans to reduce the usage of `panic!()` gradually by initially setting up a standard on the `panic!()` message which must provide the reason why they cannot handle the error and should let the program crash instead.

Least Authority confirms that some `panic!()` calls have been turned into values in `git log` on CKB.

Verification

Resolution in Progress.

Suggestion 7: Define/Use Safe Language to Write VM Scripts

Location

https://github.com/nervosnetwork

Synopsis

One of the features of the CKB-VM is that the VM scripts are RISC-V ELF binaries, allowing them to be written in any language for which a compiler backend for RISC-V architecture exists, which includes the vast majority of languages. Languages like C have certain undefined behaviour that a programmer should not rely upon. Languages for writing smart contracts should have well defined semantics.

Mitigation

It is recommended to restrict smart contract programs to a well defined domain specific language, which would minimize the amount of wrong smart contract programs via clever type systems.

Status

Nervos has acknowledged this suggestion and will it under consideration, as it requires a considerable amount of resources and effort.

Verification

Unresolved.

Recommendations

We recommend that there be further analysis on the unresolved and partially resolved Issues and Suggestions stated above and that they are addressed as soon as possible and followed up with verification by the auditing team.

Additionally, we recommend that the code readability continues to be improved as the codebase grows to facilitate easier code reviews and external contributions in the future.

The codebase can be further improved by properly organizing the Nervos GitHub organization, increasing test coverage report generation, and generally increasing documentation and the number of comments in the code.

Finally, we recommend that additional review time is allotted to further evaluate the VMs and the Neuron Wallet once development is complete, along with a peer-review of the theorems and proofs in the NC-Max research paper.

All of these changes would reduce the risk of code errors and therefore security vulnerabilities.

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.