Least Authority
PRIVACY MATTERS

Loopring 3.6 Design + Implementation: Smart Contracts

Security Audit Report

Loopring

Final Report Version: 16 March 2021

Table of Contents

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

Overview

Background

Loopring is a flexible layer-2 scalability solution for basic value transactions as well as a variety of exchanges such as order book and Automated Market Maker (AMM). This system uses advanced cryptography in the form of a limited one-way homomorphic encryption using bilinear pairings, popularized in the implementation of the Zcash protocol. This solution is classified as a validity proof system that ensures that state transition must be correct by the properties provided in the encryption scheme.

Loopring has requested that Least Authority perform a security audit of Loopring 3.6, a zkRollup layer-2 decentralized exchange and payment protocol implementation on the Ethereum blockchain. Loopring 3.6 is an improved version of Loopring 3.1, which is built on top of the same technical stack, and introduces Solidity smart contracts and `libsnark` and `ethsnark`-based circuits code.

Project Dates

- **October 5 - November 13**: Initial Review *(Completed)*
- **November 18**: Initial Audit Report delivered (*Complet*ed)
- **March 11 - 15**: Verification Review *(Complet*ed)
- **March 16**: Final Audit Report delivered (*Complet*ed)

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Dominc Tarr, 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 Loopring 3.6 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:

- Smart Contracts: https://github.com/Loopring/protocols/tree/master/packages/loopring_v3/contracts
  - Core: https://github.com/Loopring/protocols/tree/master/packages/loopring_v3/contracts/core
  - AddressSet.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/AddressSet.sol
  - AddressUtil.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/AddressUtil.sol
  - EIP712.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/EIP712.sol

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

  - FloatUtil.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/FloatUtil.sol
  - MathUtil.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/MathUint.sol
  - Poseidon.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/Poseidon.sol
  - SignatureUtil.sol: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/lib/SignatureUtil.sol
  - AMM: https://github.com/Loopring/protocols/tree/master/packages/loopring_v3/contracts/amm.

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

```
c918b164d30f7b9a3d948225f09b635257b06844
```

For the verification, we examined the Git revision:

```
5eb273fe76ba242c6a5f1bb3d1cd0edd57d070b4
```

These repositories were cloned for use during the audit and are linked for reference in this report:

https://github.com/LeastAuthority/loopring-protocols

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:

- Loopring 3.6 Design: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md
- Loopring 3.6 README: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/README.md
- Loopring 3.6 vs. 3.1: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/security_audit/LoopringV3_6_vs_V3_1.pdf
- Loopring 3.6 Circuit Documentation: https://github.com/Loopring/protocols/blob/d0eec91edc9bb195acbeddd38ebbdb71e6938127/packages/loopring_v3/circuit/statements.md

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation based on the changes made to the design documentation;
- Resistance to Distributed Denial of Service (DDoS), Reentrance attacks, and similar attacks;
- Common and case-specific implementation errors in the circuit code;
- Cost analysis for the Deposit, Withdrawal, Trades, Transfer, low cost and high throughput metrics;
- Overflow protection against the SNARK scalar field;

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

- Sybil attack in layer-2 account registration;
- Adversarial actions and other attacks on the smart contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Alignment of incentive mechanisms to help prevent unwanted or unexpected behavior;
- Vulnerabilities in the smart contracts code as well as secure interaction between the contracts and with related network components;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other ways to exploit contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity;
- Performance problems or other potential impacts on performance; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Loopring smart contracts investigated during this audit handle the deposit, withdrawal, and block state updates of the exchange that are verified against the SNARK proofs on each update. To address the issue of liveness, and ensure that the Loopring system operator continues to process transactions submitted offline, a staking protocol enables users to shut the exchange down and burn this stake if their withdrawals are not processed within a reasonable amount of time.

Loopring poses challenges due to the integration of zk-SNARK circuits for state updates and verification. These challenges include complexity of verifying correctness of the proof the circuit is enforcing leading to a system for swapping SNARK circuits if necessary, the cost of pairing functions in the Ethereum Virtual Machine (EVM), and the cost of needing to supply public inputs for data availability. This is a layer-2 scalability solution that handles more than simple transfers: it also includes an order book exchange, which makes state updates more complicated than most layer-2 systems. An additional challenge is the use of the Poseidon hash function, which optimizes the SNARK constraints and is relatively new and has a new Solidity implementation. Most other projects use the well-known `keccak256` function instead and, as a result, this novel approach by the Loopring team presents some risk.

In general, layer-2 scalability solutions are required to identify a balance between ensuring the security of the transactions and adding the value of convenience, which is a difficult undertaking. For example, some layer-2 solutions, such as payment or state channels, reduce the security requirements of a blockchain network by only requiring consensus of the parties involved, but not the network as a whole. This, in turn, requires timeouts for fraud proofs to ensure that the parties do not stall or cheat the channel.

In Loopring, the state updates are secured by a SNARK circuit that ensures that the state update is correct, making this system one of validity proofs. This addresses most of the concerns fraud proofs need to handle on-chain. In order to ensure that the operator continues to process transactions, Loopring allows the operator one to two weeks to process withdrawal transactions before users may submit a claim on-chain that shuts the Loopring system down, causing a mass exit, allowing the Loopring operator enough time to increase the chances of avoiding DoS attacks, while balancing opportunity cost losses from the users if an operator were to become defunct.

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

Scope

The scope of the audit was sufficient and encompassed the security-critical components of the system. Although the complexity of logic that states may mutate to is fairly simple, a large Solidity codebase such as Loopring can be difficult to assess for small issues that can lead to a failure where large amounts of money are at stake. In addition, there are several dependencies outside of the scope of the audit, which could introduce security issues. However, this risk is minimized by the use of standard libraries that are widely used and have been previously reviewed.

One crucial component of the system outside the scope of this audit is the operator. Since significant aspects of the security of Loopring depend on how the operator behaves, we recommend a security audit of the operator (Suggestion 4).

Code Quality + Documentation

We found the code to be well organized. Given the complexity of this codebase as a decentralized exchange, especially one using advanced validity proofs, this system logic is easy to follow. This made auditing this system less challenging and reduces the concerns that various systems states remain unexplored. As a result, we determined that fuzz testing in order to explore the system's various states was unnecessary. This simplified our audit approach and allowed opportunity to focus on other areas of concern.

We found the code comments to be minimal with the exception of some comments describing key components and feature functionality. We suggest expanding the comments coverage to be more exhaustive and ensuring that most functions adhere to the style guidelines for Solidity (Suggestion 6).

The overview documentation is sufficient in helping to understand how the system operates. However, we recommend adding more detail, particular regarding the withdrawal process (Suggestion 2).

We commend the Loopring team for providing sufficient test coverage with many quality tests. We also found that these tests anticipate the possible failure conditions that we identified during our review.

System Design

We commend the Loopring team for adhering to best practices for system design theorized by layer-2 scaling teams over the past few years. Given how complex an exchange's state is, the Loopring team has managed to reduce the logical complexity to a level that is easy to reason about. It is apparent that security is integral to the smart contract system. This is exemplified by several commendable properties of the system, including:

- The exhaustive use of non-rentrancy guards and the use of extensions to the standard ownership contracts with the new claimable ones that ensure transfer must be a two-step process;
- Careful initialization of state and thought put into the proxy pattern of updates; and
- The general use of good modifiers and Solidity patterns that ensure visibility of functions are proper.

However, we have identified areas for critical improvement: the ability for a single forced withdrawal transaction to be left out long enough for the entire exchange to be shut down (Suggestion 2) and the power that the operator has to switch the snark circuit (Issue A).

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

Specific Issues & Suggestions

We list the issues and suggestions found during the review, 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: Operator Has Total Authority Over State Verification	Partially Resolved
Issue B: Potential Limbo Exit - Withdrawal Fee Griefing	Resolved
Suggestion 1: Remove Simple Unused Code Relic	Resolved
Suggestion 2: Improve Withdrawal Processing	Unresolved
Suggestion 3: Check for Null Recipient Addresses	Resolved
Suggestion 4: Audit the Operator	Unresolved
Suggestion 5: Require a Signature as Private Key for Account Updates	Unresolved
Suggestion 6: Improve Documentation	Resolved

Issue A: Operator Has Total Authority Over State Verification

Location

/packages/loopring_v3/contracts/core/impl/BlockVerifier.sol#L27

Synopsis

In the current implementation, the Loopring contracts allow for the centrally owned operator to replace the SNARK circuit at any time. This SNARK circuit is used to verify incoming blocks generated off-chain that will subsequently update the stored merkle root that controls the account state of all users and all components of the system. If this verification circuit is changed for something malicious, it will allow state updates that are malicious.

Impact

Severe. If the operator of the SNARK circuit becomes malicious or compromised, they are able to control all state updates. This can lead to the users losing all funds in the system.

Feasibility

In the early stages of the exchange, it is against the self interest of the Loopring organization to commit fraud. However, this is a single point of failure that could become compromised through other attack methods.

Mitigation

In advance of delivering this audit report, the Loopring team responded that they are dedicated to resolving this issue. While the SNARK circuits are still being finalized, it is favorable to be able to switch a

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

broken circuit out quickly. Once the SNARK circuits have been finalized, the update function should be immediately switched to one controlled by a `multisig` contract.

Remediation

As discussed with the Loopring team, a democratic and transparent approach to the update of the consensus rules would be an ideal alternative to the currently centralized design. One way to do this would be to place a proposed circuit update on-chain where it could be voted on by the users of the system, or by those qualified to ensure that the new circuit is correct. SNARK circuits are complicated protocols and not easily reviewed by most people. Votes should be only placed on circuits that have been thoroughly audited by reliable and ethical sources.

Status

The Loopring team has informed us that they intend to launch with a `multisig` contract, per the suggested mitigation, in addition to delays for ownership transfer and circuit upgrade of seven days that will also allow users time to exit if they suspect that a governance action is malicious. A full remediation is currently difficult since an agreed upon democratic and transparent approach to the update of the consensus rules has not been established, however, we recommend that the Loopring team continue to investigate, research, and implement long term strategies that promote decentralization.

Verification

Partially Resolved.

Issue B: Potential Limbo Exit – Withdrawal Fee Griefing

Location

/packages/loopring_v3/contracts/core/impl/libexchange/ExchangeWithdrawals.sol#L61

Synopsis

A limbo exit is a result of a single source of truth creating off-chain state updates and a data availability problem, as was first theorized during Plasma scalability discussions. A form of it is present in Loopring as well: operators have the ability to process state but not reveal this processed state to anyone. If a user submits a transaction off-chain but does not see that their transaction is being processed, they will initiate a forced withdrawal on-chain. At a later point, the operator can reveal the block that includes the transaction that invalidates the withdrawal and block the process of forced withdrawal. These forced withdrawals require fees to be processed, and if the withdrawal is invalidated by the revealed block, the user will be griefed a small amount for the fees.

Impact

The impact of this is low because fees are not very high, and the operator is not incentivized to hurt their own business by committing these types of infractions.

Preconditions

The operator includes a transaction for a trade or a transfer of funds, but does reveal or commit the block that includes this transaction to the mainnet.

Feasibility

This is easily feasible but the incentive to do so is not aligned.

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

Mitigation

This issue results from the fact that data availability is in the control of layer-2 operators and, to the best of our knowledge, reasonable protocol level solutions for addressing this issue are currently unavailable. However, out-of-band channels such as social media will ensure that there is a reputation loss for the Loopring organization if this is attempted beyond an accidental incident. We simply suggest that this issue be made public to encourage transparent reporting of incidents that may arise.

Status

The Loopring team has incorporated additional documentation on the issue of withdrawal fee griefing, along with providing a mitigation strategy, in accordance with our suggested mitigation.

Verification

Resolved.

Suggestions

Suggestion 1: Remove Simple Unused Code Relic

Location

/packages/loopring_v3/contracts/core/impl/libexchange/ExchangeAdmins.sol#L22

Synopsis

There is an event to signify that the operator has been changed, but this event is never used. This functionality has been replaced with the more robust claimable and ownable contacts provided by OpenZeppelin and can be removed. Removing it will also slightly reduce gas costs when deploying the contract.

Mitigation

We recommend removing the unused event from the code base.

Status

The Loopring team has removed the unused code relic, thus slightly reducing gas costs during contract deployment and ensuring that unnecessary functionality is removed from the code, as suggested.

Verification

Resolved.

Suggestion 2: Improve Withdrawal Processing

Location

/packages/loopring_v3/contracts/core/impl/ExchangeV3.sol#L512

Synopsis

There is a safety mechanism for users of the exchange to enforce that they are able to withdraw their funds. This is to prevent the case that an operator becomes malicious or offline for too long and will not process transactions off-chain for some deposited amount of funds. If the withdrawal is valid, as witnessed in the merkle tree, and `MAX_AGE_FORCED_REQUEST_UNTIL_WITHDRAW_MODE` reached, the exchange will enter withdrawal mode and be shut down entirely. This presents a danger for the operator

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

that they may miss a valid withdrawal by software failure and lose their stake, while forcing the exchange to shut down for all users.

Mitigation

Given that this is a severe penalty, some added robustness in the way that withdrawals are processed could be useful. Mitigating against this accidental closure of the exchange and stake burning is generally out of the scope of the contracts, and requires the server infrastructure to ensure that forced withdrawal requests are processed in a timely manner. However, as discussed with the Loopring team before the writing of this report, they have ideas for allowing multiple operator servers or organizations to handle simple requests like this. Having added a quorum of withdrawal operators to the contracts that are in some way incentivized, potentially by the withdrawal fee, could add an extra layer of security such that the entire system does not shut down by a single server missing a single transaction. We suggest further research be done on this topic and documentation be added that will describe this potential process in more detail.

Status

The Loopring team has acknowledged this suggestion and have noted that, while they can allow anyone to submit blocks, open source software that third party operators can use to process withdrawals is currently unavailable. The Loopring has also noted that their plans for addressing this suggestion are still to be determined and the suggestion remains unresolved at this time.

Verification

Unresolved.

Suggestion 3: Check for Null Recipient Addresses

Location

/packages/loopring_v3/contracts/core/impl/libexchange/ExchangeWithdrawals.sol#L242

Synopsis

If a user does not supply a `transfer _to` address to a transaction, the funds will automatically be transferred to the Loopring protocol fee vault. In general, for ERC-20 contracts, there is a check `require(_to != address(0));` to capture this potentially common fail case. This check is provided and conceived by the OpenZepplin team as an important modification to the ERC-20 standard to prevent bad software that causes empty arguments or improper address parsing from sending the funds to an uncontrollable address. This is reasoned to be a more common failure than supplying a properly formatted but incorrect address.

Mitigation

We do not consider this a security issue because the funds are not lost entirely and there may be a centralized method for correcting mistaken transfers. If left unchanged, however, it could be an inconvenience or potential loss of funds if there is no verifiable way to correct a transaction's intentions. We suggest separating the intention of sending to the operator from leaving the `_to` field blank and making those kinds of transactions more explicit.

Status

The Loopring team has included additional checks that prevent users from withdrawing to address (0), as suggested. In addition, they have noted that when withdrawing from the protocol fee account (`owner == address(0)`) the `to` address will still be 0 (which will be translated on-the-fly to the current fee vault address), which is done so they are able to update the protocol fee vault address at any time while making sure that previously withdrawn funds are not already assigned to the old protocol fee vault address.

Verification

Resolved.

Suggestion 4: Audit the Operator

Synopsis

Significant aspects of the security of the total Loopring system depend on what the operator chooses to do, as most user interactions go through the operator, except some that use contracts directly. This means that much of the attack surface is in the operator code. Indeed, DoS attacks against crypto markets are quite common. However, the operator is out of scope for this audit.

Mitigation

We suggest an audit of the operator so that the potential attack vectors are assessed, clarified, and resolved.

Status

The Loopring team acknowledges this suggestion, however, they have noted that they do not consider the operator to be security critical. The Loopring team has stated the following:

“The operator needs to do things on-chain and off-chain.

On-chain an extra operator contract is used that has some special privileges on the exchange contract, but once the exchange contract is fully set up the operator contract can't do anything that is critical for the security. The security is enforced strictly by the main protocol contracts.

Though as far as they know the only really critical part is that the operator (or any other UI hooking into the Loopring ecosystem) can help the user create and manage the specific EdDSA keys necessary to do rollup transactions (though most transactions can also be approved using just standard Ethereum signatures or transactions). This will be mitigated for MetaMask users when plugin support is enabled in production on their end (https://github.com/Loopring/eddsa-metamask-plugin). With the plugin the keypair is generated automatically and the private key never leaves MetaMask.

So while the operator is responsible for some desired properties like liveness and censorship resistance, the operator is not critical for user security because of the censorship resistant forced withdrawal + withdrawal mode system. Those are still important properties of course, but for now not something they want to focus on.”

At the time of this verification, the operator has not undergone a separate security review and we recommend that the Loopring team continue to consider a follow up audit of the operator.

Verification

Unresolved.

Suggestion 5: Require A Signature as Private Key for Account Updates

Location

/packages/loopring_v3/circuit/Circuits/AccountUpdateCircuit.h#L81-L93

Synopsis

When updating a client key, Loopring recommends using a signature as the private key. This has an interesting property that it creates a key linked to the root private key, but if an attacker knows the account key, they cannot derive the root private key. However, the actual update transaction does not depend on this property: it's simply a statement, signed by the root key, that an arbitrary key is now the account key. That means a randomly generated key could be used. A key manager usually does not consider a signature to be a private output – the assumption is that a signature is provided to a third party who will check it. Using a signature as a private secret may therefore be outside the security model of that key manager. However, a random number generator is already expected to produce private secrets. As Loopring does not directly depend on the fact that account keys are linked to root keys (other than the root key claims it), so it is recommended to simply use a new random key.

Mitigation

Clients should generate a key with a random number generator and then sign the Account Update transaction with their root key. This does not require any changes to the protocol, only to the client.

Status

The Loopring team has responded indicating that their current approach is sufficient. In particular, they state:

“Our current approach was explained here and was well received by the Ethereum community: https://medium.com/loopring-protocol/looprings-new-approach-to-generating-layer-2-account-keys-4a16cc334906

The downside of generating the keys some additional information (be it a password or some random random number) is that security is compromised in other ways. With a password the user is required to pick a strong password and then remember it. With a random number either the random number or the private key directly needs to be stored somewhere in a secure way. Both are also pretty bad from a UX perspective because we now put a lot of responsibility on the user to keep his own account safe while making sure the private key can be recovered (or even transferred to a different device) in a safe way. Loopring itself could create specialized tools to do these things, but this is suboptimal because we'd need to create and maintain this critical part of the system, while other solutions already have widespread use and have been battle-tested the last couple of years. Often users already have their preferred device or wallet.

The additional benefit of using the signature is that we seamlessly support a wide range of different wallets, not only MetaMask but also mobile wallets and hardware wallets.”

We agree that this approach is sufficient. The benefit of this approach is that it is possible to regenerate the account keys if they are lost, as long as the root key has been protected. This is a pragmatic choice that balances user experience against security. Deriving the account key from a hash of the private key (via a hash based KDF function) would have the same effect, except that wallet applications already generally have a signature feature. As a result, we consider this issue to be resolved.

Verification

Resolved.

Suggestion 6: Improve Documentation

Location

/contracts/core/impl/libexchange/ExchangeAdmins.sol#L33

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

Synopsis

Code comments were insufficient in many areas and most functions do not follow the style guidelines for Solidity. Additionally, the Loopring whitepaper describes an old version of the protocol.

Mitigation

We recommend that all outdated documentation be removed or updated, code comment coverage be expanded and that all functions be updated to follow Solidity style guidelines.

Status

The Loopring team has replaced the previous whitepaper with updated design documentation. In addition, they have documented all public functions and improved their code comments. We acknowledge that improvements to documentation is an ongoing effort and encourage the Loopring team to continue updating the project documentation and adhering to documentation best practices as the code base continues to mature.

Verification

Resolved.

Recommendations

We recommend that the unresolved and partially resolved *Issues* and *Suggestions* stated above are addressed as soon as possible and followed up with verification by the auditing team.

We recommend that the Loorping team continue to research and investigate the possibility of implementing a democratic and transparent approach to the update of the consensus rules. This would reduce the security risk of having an overly authoritative operator and prevent potential malicious updates when the operator is compromised.

In addition, we suggest the Loopring team explore an Operator Audit to identify any further attack vectors that may compromise the system

We commend the Loopring team for updating the documentation and removing unused code, thus reducing confusion for users and reviewers of the code and the potential for risks that stem from human error. The Loopring team's consideration for security throughout the system is notable as is demonstrated through their adherence to best practices for system design theorized by layer-2 scaling teams over the past few years.

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 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.