Least Authority
PRIVACY MATTERS

Linea zkEVM (crypto-beta-v1)

Security Audit Report

Consensys Software, Inc.

Final Audit Report: 26 November 2024

Table of Contents

Overview

Background

Consensys Software, Inc. has requested that Least Authority perform a security audit of Linea's zkEVM (crypto-beta-v1) and cryptography libraries.

Project Dates

- **July 8, 2024 - September 30, 2024:** Initial Code Review *(Completed)*
- **September 30, 2024:** Delivery of Initial Audit Report *(Completed)*
- **November 18, 2024:** Verification Review *(Completed)*
- **November 26, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- George Gkitsas, Security / Cryptography Researcher and Engineer
- Sven M. Hallberg, Security Researcher and Engineer
- Jasper Hepp, 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 Linea's zkEVM (crypto-beta-v1) and accompanying libraries followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

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

- See Appendix A.

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

- zkEVM:
  - a3ac02514a2692952d31f9ea7adfb20f32a4bf98
  - f9fae838927bb16826d293367a83a2d24d7aacd2
  - 5009fb0d68506b9cfa6512dbaf2242f4e8e8e75d
  - b14a1a5fea038c9215206b971a0ef76a69b182c9

- gnark:
  - d94368b154d73449e796ebdc2665534b086114f2
  - c36ff9e8c58daf199a333c0f6c6fe6cacabc0bbb

- compress :
  - 2efdc672da09afb47b80d7dd35d79cbe9411a023

For the verification, we examined the Git revision:

- zkEVM: f462a8233375bb819bd037b3f7eba475cbeaedfb
- gnark: 7512178ac1fc1c308206c07bc04c8fd2747e981e

For the review, these repositories were cloned for use during the audit and for reference in this report:

- zkEVM:
  https://github.com/LeastAuthority/zkevm-monorepo/tree/zkevm/prover/zkevm
- gnark:
  https://github.com/LeastAuthority/Consensys-gnark
- compress:
  https://github.com/LeastAuthority/compress/tree/main

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

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Linea Prover Documentation:
  https://eprint.iacr.org/2022/1633
- Vortex:
  https://eprint.iacr.org/2024/185
- Blob Specification:
  https://github.com/LeastAuthority/zkevm-monorepo/blob/f9fae838927bb16826d293367a83a2d24d7aacd2/prover/lib/compressor/blob/v1/blob_spec.md#linea-blob-format-specification
- Keccak Specification:
  https://hackmd.io/adB-EjnKSa-P7HwI542l9g?view
- EIP-4844:
  https://www.eip4844.com
- gnark:
  https://docs.gnark.consensys.io/overview
- The zkEVM Architecture:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/zkEVM_Architecture.pdf
- State Management:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/zkEVM_Statemanagement.pdf
- Prover Completeness Public Inputs:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Prover_Completeness_Public_Inputs.pdf
- Prover Completeness ECDSA Module:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Prover_Completeness_ECDSA_Module.pdf
- Prover Completeness Statemanager Module:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Prover_Completeness_Statemanager_Module.pdf
- Documentation Hash Module:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Documentation__Hash_Module.pdf
- Proof Aggregation EIP4844 Data Compression design:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Proof_Aggregation_EIP4844_Data_Compression%20design.pdf

- Prover Component ECPAIR:
  https://github.com/LeastAuthority/Linea-Prover-Cryptography-Audit-Documentation/blob/main/Prover_Component_%20ECPAIR.pdf

In addition, this audit report references the following documents:

- M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity." *IACR Cryptology ePrint Archive*, 2016, [AGR+16]
- A. Belling, A. Soleimanian, and O. Bégassat, "Recursion over Public-Coin Interactive Proof Systems; Faster Hash Verification." *IACR Cryptology ePrint Archive*, 2022, [BSB22]
- A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge." *IACR Cryptology ePrint Archive*, 2022, [GWC22]
- Y. E. Housni and A. Guillevic, "Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition." *IACR Cryptology ePrint Archive*, 2020, [HG20]
- W. Nguyen, D. Boneh, and S. Setty, "Revisiting the Nova Proof System on a Cycle of Curves." *IACR Cryptology ePrint Archive*, 2023, [NBS23]
- G. Wood, "Ethereum: A Secure Decentralized Generalised Transaction Ledger." *Ethereum*, 2024, [Wood24]
- EIP4844:
  https://github.com/ethereum/EIPs/blob/master/EIPS/eip-4844.md
- Ethereum execution specification:
  https://github.com/ethereum/execution-specs/blob/c854868f4abf2ab0c3e8790d4c40607e0d251147/src/ethereum/paris/vm/precompiled_contracts/ecrecover.py
- Bitcoin `libsecp256k1` library:
  https://github.com/bitcoin-core/secp256k1/blob/ea2d5f0f17881031a033b0cc049230183a5826d1/src/modules/recovery/main_impl.h

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Data privacy, data leaking, and information integrity;
- Vulnerabilities in the code leading to adversarial actions and other attacks;
- Protection against malicious attacks and other methods of exploitation; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of the zkEVM implementation of Linea. The Linea team implements a zkEVM based on the Wizard protocol and the gnark library. The zkEVM aims to provide an execution environment equivalent to the Ethereum Virtual Machine (EVM), allowing Ethereum transactions and smart contract executions. Because of the underlying cryptographic tools, zkEVMs aim to provide a solution to the scalability problem of the Ethereum blockchain.

The zkEVM aims to prove the correct Ethereum state transition via a constraint system built utilizing the Wizard protocol. The Wizard protocol is designed to handle a wide class of queries (permutation, inclusion, etc). The generated proof from the zkEVM is then compressed and modified by a set of gnark circuits over the elliptic curves BLS12-377, BW6-761 and BN254. The circuits are used for recursive verification of execution and blob decompression updates as well as to generate the final proof that can be verified on the Ethereum blockchain.

System Design

We reviewed the implementation of the Linea zkEVM as well as the underlying proving system Wizard-IOP and accompanying libraries. Our team found the system to be well-designed, with a strong emphasis on security.

Prover Protocol Wizard

We reviewed the implementation of the Wizard-IOP in the folder `prover/protocol`, which includes the compiler, prover, and verifier. Wizard-IOP has been developed by the Linea team. Although our team did not review the paper, we nevertheless recommend updating it, as it is erroneous and deviates partly from the codebase (Suggestion 1).

We reviewed the compiler and found several issues (Issue A, Issue B, and Issue C). Two of the findings are critical issues that allow a malicious prover to convince an honest verifier to accept an incorrect fixed permutation (Issue A) and an incorrect inclusion query (Issue B), effectively breaking soundness. In addition, we found a minor issue for an inner product query (Issue C) that could break completeness. After the compilation of the queries, the compiler applies a column sticking and splitting technique ending with same-sized columns. We found one critical issue (Issue D) that leads to an unsafe verifier. In this case, the compiler does not add a verification to check the correct transformation for local opening points (Issue D).

We did not find any issues in the Vortex commitment scheme. We reviewed the correct usage of the cryptographic primitives of Vortex in the compiler; however, we did not review its core functions (for example, the folder vortex), as they were out of the scope of this audit.

We reviewed the additional subprotocols based on the Wizard-IOP in the folder `dedicated`, which includes an implementation of Plonk [GWC22] and subprotocols, as described in the appendix of the Linea Prover Documentation paper. The code is implementing Plonk with one custom gate based on [BSB22] to allow for efficient usage of random coins in a circuit. We did not find any issue relating to this.

We reviewed the usage of Fiat-Shamir in the Wizard protocol and did not find any issues. The initialization of Fiat-Shamir starts from a serialized version of the compiled IOP; however, since the serialization folder is not in scope, we did not review the completeness of its initialization.

We reviewed the prover and verifier of the Wizard-IOP as well as the accompanying folders, such as the implementation of symbolic expressions, the columns, or the queries. We found a minor edge case error in the symbolic expression of a product (Issue E).

We also identified several areas of improvements in the prover protocol and recommend reducing inefficiencies, removing redundant code, as well as adding further panics (Suggestion 2).

EVM Precompiled Contracts

We reviewed the functions for the precompile contracts MODEXP, ECRECOVER and ALT_BN128_PAIRING_CHECK as well as SHA2 and MiMC. The remaining precompiled contracts are either implemented directly in the zkEVM or are not addressed by the zkEVM (`ripemd`, `blake2f`, and `point evaluation`). We found two issues – a missing constraint in ECRECOVER (Issue F) and an incorrect edge case in MODEXP (Issue G).

zkEVM

We examined the implementation of the zkEVM and its corresponding submodules. We reviewed the constraint system against the shared documentation of each of the submodules. We identified several missing constraints (Issue H, Issue I, Issue J, Issue K, Issue M, Issue N, Issue O, and Issue P), and an incorrect local opening (Issue L). The missing constraints all lead to soundness issues, allowing a malicious prover to manipulate the proof, such that an honest verifier accepts incorrect Ethereum state updates.

Compress

The codebase uses a customized implementation of the LZSS lossless compression following the principle of LZSS rather than adhering to a standard specification. Consequently, a proof of losslessness is missing. We reviewed the implementation along with the specification and did not find any issues but report a few potential improvements to enhance the code quality (Suggestion 3). However, for future audits and independent third parties, we recommend including proper references or an argument for losslessness in the documentation.

Gnark Frontend

We reviewed the implementation of the gnark frontend with a particular focus on missing constraints in all primitives and iterative circuit generation issues. We identified a missing constraint in the selector primitive (Issue Q). In addition, we found various instances with imprecise or misleading code comments that could be improved (Suggestion 6).

Linea zkEVM gnark Circuits and Backend

We reviewed the implementation of the `execution[BLS12-377]`, `aggregation[BW6-761]`, `blobdecompression[BLS12-377]`, `pi_interconnection[BLS12-377]`, `emulation[BN254]` as well as the dummy circuit, focusing on correct circuit generation as well as missing constraints. In addition, we reviewed the associated backend functionality, which generates proof witnesses for the circuit. We did not identify any issues.

Note that since core functionality, such as the emulated field arithmetics and the emulated Plonk prover systems were out of scope, our team conducted the audit assuming that the aforementioned functions operate as intended.

In addition, we analyzed the correct instantiation of the Common Reference string (Powers of Tau) for Linea's gnark circuits. Linea uses:

- `BN254`: Ignition ceremony from Aztec;
- `BLS12-377`: Ceremony from Aleo; and
- `BW6-761`: Plumo ceremony from Celo.

Our team assumed the correct deletion of at least one secret randomness in each of these ceremonies, which is a common assumption. Moreover, all of those ceremonies are publicly verifiable.
In the recursive Plonk verifier, the common reference string for the KZG commitment scheme (KZG CRS) is hard coded into the circuit specification, making it constant and determined at the time of circuit compilation. For the last proof in the emulation circuit, the `BN254` proof is verified on chain, where the KZG CRS is also hard coded (using a code-generated Solidity verifier).

We did not identify any issues in this approach; however, our team noted that in order for a third party to independently verify that the compiled circuits utilize the referenced CRS, the independent third party needs to recompile the circuit to ensure it matches the deployed version. This, however, depends on the assumption that the build process is deterministic and equivalent on all hardware.

Code Quality

We performed a manual review of the repositories in scope and found the code to be generally clean, well-organized, and of high quality, in that it adheres closely to development best practices. However, we did identify some areas that can be improved in the prover protocol and compress library components to enhance the quality of the code (Suggestion 2 and Suggestion 3).

Tests

Our team did not assess whether test coverage was sufficient, as the tests were out of the scope of this audit. However, we did identify one issue in the prover protocol component that results in any permutation being accepted by the verifier, which could have been detected by performing a simple test (Issue A).

Documentation and Code Comments

The project documentation provided by the Linea team and the code comments sufficiently describe most of the intended functionality of the system. However, our team found that in some components (for example, the zkEVM in particular) the provided documentation is incomplete and scattered across many files, pdfs, documents, and research papers. We recommend improving the documentation and storing it in a single location to provide future reviewers of the code with a better understanding and ability to reason about the system design (Suggestion 8).

Scope

During our review, we found that parts of the code are still missing crucial implementation features (for example, the public input module does not have a consistency check on the block data), while other parts have been commented out, pending the resolution of some other issues. Due to this, some issues, such as (Issue M), were not identified by the Linea team.

In addition, in some parts of the code, such as the EVM precompiled contracts, important low-level functions were not included in the scope of this audit.

The core file of the arithmetization module (`define.go`) was also not in scope. It is an auto-generated file containing the complete set of constraints for the arithmetization module, including the correct execution of opcodes as well as the registration and correct formation of many columns. This made it difficult to reason about certain parts of the audited modules in the zkEVM since columns are fetched across modules, and our team had to therefore assume that they were properly constrained.

Dependencies

Running nancy for the repositories in scope yielded no issues. Hence, our team did not identify any vulnerabilities in the implementation's use of dependencies.

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: [Prover Protocol] Verifier Accepts Any Permutation Query	Resolved

Issue B: [Prover Protocol] Partially Ineffective Inclusion Query	Resolved
Issue C: [Prover Protocol] Incorrect Round for Inner Product Query	Resolved
Issue D: [Prover Protocol] Correctness of Local Opening Point Is Not Verified During Sticking	Resolved
Issue E: [Prover Protocol] Incorrect Symbolic Expression Product for Zero Edge Case	Resolved
Issue F: [Gnark EVM-Precompiles] Incomplete Constraint in ECRecover Precompiled Contract	Resolved
Issue G: [Gnark EVM-Precompiles] Incorrect Edge Case in ExpMod Precompile Contract	Resolved
Issue H: [zkEVM] Missing Activation Constraint in ECPair and ECDSA Modules	Resolved
Issue I: [zkEVM] Missing Binary Constraints in ECPair Module	Resolved
Issue J: [zkEVM] Missing Exclusive Binary Constraint in ECPair Module	Resolved
Issue K: [zkEVM] Missing Zeroization Constraint in Public Input Module	Resolved
Issue L: [zkEVM] Incorrect Value for TotalBytesCounter in the Local Opening of the Public Input Module	Resolved
Issue M: [zkEVM] Correctness of Leaf Hashing Constraint In the State Manager Can Be Turned Off	Resolved
Issue N: [zkEVM] Missing Constraint for the Hashed Key During the Update Operation	Resolved
Issue O: [zkEVM] Merkle Tree Can Be Updated Without Updating the Value of NextFreeNode Due To Missing Constraint	Resolved
Issue P: [zkEVM] Missing Consistency Constraint on the Hashed Key Value	Resolved
Issue Q: [Gnark Frontend] Missing Binary Constraint in Select Primitive	Resolved
Suggestion 1: [Prover Protocol] Update the Linea Prover Documentation	Unresolved
Suggestion 2: [Prover Protocol] Improve Code Quality	Unresolved
Suggestion 3: [Compress Library] Improve Code Quality	Unresolved
Suggestion 4: [zkEVM] Improve Code Quality	Unresolved
Suggestion 5: [zkEVM] Refactor Common Patterns in Constraints	Partially Resolved

Suggestion 6: [Gnark Frontend] Improve Code Comments	Resolved
Suggestion 7: [Gnark Frontend] Missing State Reset	Resolved
Suggestion 8: Improve Documentation	Unresolved

Issue A: [Prover Protocol] Verifier Accepts Any Permutation Query

Location

`compiler/permutation/compiler.go#L85`

Synopsis

A permutation query allows checking that two tables A and B contain the same rows up to a permutation. During compilation of this query, the code erroneously compares A with A instead of B, hence making this query trivial.

Impact

Critical. This issue would result in any permutation query being accepted by the verifier, which breaks the soundness of the protocol.

Preconditions

None.

Feasibility

High.

Technical Details

During compilation of a permutation query between tables A and B, a random linear combination over the columns is built. The Boolean `multiColumns` is set to `true` if there is more than one column for A. For this case, the code sets the columns to be collapsed (i.e., folded by a random linear combination) to the columns of A for both cases, table A and B. This sets the numerator and denominator of Z to $\sum_i \alpha_i q.A_i$ and, hence, Z=1. The verifier will accept the permutation query independently of A and B. Instead, the code should correctly assign the columns to also be collapsed for table B – that is, it should set the columns to be collapsed for the case of table B to the columns of table B.

Remediation

We recommend resolving this issue by implementing the suggestion in the Technical Details.
In addition, this issue could have been detected by performing a test checking whether the permutation fails if B is not a permutation of A. We recommend increasing test coverage for this query.

Status

The Linea team has resolved the issue in this PR. In addition, test coverage has been increased.

Verification

Resolved.

Issue B: [Prover Protocol] Partially Ineffective Inclusion Query

Location

`compiler/lookup/compiler.go#L197`

Synopsis

For an inclusion query, the code erroneously switches fragments and columns, and does not fully verify queries with only one fragment and several columns.

Impact

High. A query with only one fragment but several columns effectively only checks the first column of each, the `looking` and the `looked` table. The remaining columns are excluded from the query and can hence be tampered with. An attacker can construct a false inclusion query that the verifier would accept as true, which breaks the soundness of the protocol.

Preconditions

For the attack, the query would need to have only one fragment and several columns.

Feasibility

High.

Technical Details

An inclusion query (or `lookup` query) checks for two tables S and T if S is a subset of T. Both tables are composed of a set of fragments that respectively have a set of columns. During its compilation, the code checks whether the tables have several columns. The code defines the Boolean `isMultiColumn` via the number of fragments and not the number of columns for the `lookedTable`. To obtain the number of columns, the code should perform `isMultiColumn = len(lookupTable[0]) > 1`. However, the code performs `isMultiColumn = len(lookupTable) > 1`.

Remediation

We recommend updating the code, as described in the Technical Details section above.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue C: [Prover Protocol] Incorrect Round for Inner Product Query

Location

`compiler/innerproduct/context.go#L60`

`compiler/innerproduct/compiler.go#L66`

Synopsis

For an inner product query, the prover task is registered in the incorrect round if the Boolean `hasMoreThanOnePair` is set to `true`. In addition, the Boolean `hasMoreThanOnePair` is unnecessary and increases inefficiency by adding one extra round to the protocol.

Impact
Low. The prover can run into a completeness issue.

Technical Details
After the compilation of the inner product query, the prover task is registered for round +1. This is the correct round only if `hasMoreThanOnePair` has been set to `true`. However, if `hasMoreThanOnePair` is set to `false`, this is one round too late. An honest prover should catch this incorrectness during prover runtime by throwing a panic. If the proof is still passed on to the verifier, it should be rejected.

For an inner product query, the Boolean `hasMoreThanOnePair` is set to `true` if there is more than one query. If the Boolean is set to `true`, the round number is increased by one. This step is inefficient since the protocol is made longer than necessary.

Remediation
We recommend removing the conditional increment for the round in the compilation of the inner product query.

Status
The Linea team has resolved the issue in this PR.

Verification
Resolved.

Issue D: [Prover Protocol] Correctness of Local Opening Point Is Not Verified During Sticking

Location
`splitter/sticker/sticker.go#L23`

`splitter/sticker/sticker.go#L399`

Synopsis
During the step sticker in the compiler, the compiler does not add a verification step to the verifier to check the correct transformation of a local opening query.

Impact
High. A malicious prover can assign a new opening point Y to the query, thus breaking the soundness of the protocol.

Preconditions
None.

Feasibility
High.

Technical Details
Sticking is interleaving small columns into larger ones, allowing the modification of all columns into columns with a constant size. During this process, local opening queries are replaced by equivalent queries pointing to the regrouped columns. However, a verifier check that the opening point Y has not

been modified is missing. This check should be added by the compiler to the compiled IOP as a `VerifierAction`.

Remediation

We recommend adding a verifier check on the local opening point.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue E: [Prover Protocol] Incorrect Symbolic Expression Product for Zero Edge Case

Location

`prover/symbolic/product.go#L62`

Synopsis

The symbolic expression `Product` returns a false result if one factor and its corresponding exponent are both zero.

Impact

Low.

Preconditions

To receive a false result, a factor and its corresponding exponent would need to be zero.

Feasibility

Low. In the zkEVM component, the precondition noted above seems unlikely.

Technical Details

The package `symbolic` allows generating and manipulating symbolic expressions. The generator function `NewProduct` for the symbolic expression of a product takes as inputs items and exponents and returns `\prod_i (item[i]^exponent[i])`. This function returns zero for the overall product if one of the items is zero. However, this is incorrect in the case where the corresponding exponent is zero as well since `0^0 = 1`.

Remediation

We recommend considering this edge case and checking if the exponent is non-zero when the item is zero and, only then, returning zero as a result of the overall product.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue F: [Gnark EVM-Precompiles] Incomplete Constraint in ECRecover Precompiled Contract

Location

[std/evmprecompiles/01-ecrecover.go#L52-L55]

[ethereum/execution-specs/src/ethereum/paris/vm/precompiled_contracts/ecrecover.py]

[bitcoin-core/secp256k1/src/modules/recovery/main_impl.h#L46]

[std/evmprecompiles/01-ecrecover.go#L95]

Synopsis

The `ECRECOVER` precompiled contract should constrain the parameter v to be in `{0,1}`. However, it is insufficiently constrained to be in `{0,1,2,3}` instead.

Technical Details

The function `ECRecover` implements the EVM precompile contract for `ECRECOVER` that is specified in the Ethereum Yellow Paper [Wood24] and in the execution specification. According to these sources, the value of v should be either `27` or `28`. In contrast, Bitcoin's `libsecp256k1` allows v to be in the set `{27,28,29,30}` (respectively in `{0,1,2,3}` if one subtracts `27`). The precompile contract of Linea follows the Bitcoin `libsecp256k1` library and constrains that the adjusted v is in `{0,1,2,3}`. This is incorrect since the precompiled contract should follow the Ethereum specification.

In addition, the implementation has a redundant constraint in line `95` that could be removed. More specifically, `vbits[1]` is used to select cases to compute `R.x.`, but since `vbits[1]` should be zero anyway, the first case can be directly used to compute `R.x.`

Remediation

We recommend modifying the constraint, as suggested in the Technical Details section above.

Status

The Linea team has resolved the issue in this PR and also removed the redundant constraints.

Verification

Resolved.

Issue G: [Gnark EVM-Precompiles] Incorrect Edge Case in ExpMod Precompile Contract

Location

[std/evmprecompiles/05-expmod.go#L17]

Synopsis

The implementation of the precompiled contract `MODEXP` returns a false value for an edge case.

Impact

Low.

Preconditions

The values `a`, `b=0`, and `c=1` need to appear in an actual use case.

Feasibility

Low.

Technical Details

The function `ExpMod` computes `a^b mod c`. For `a=b=0` and `c=1`, the code returns `1`, while the correct value is `0`.

Remediation

We recommend considering the edge case of the modulus being one when checking for the case zero to the power of zero.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue H: [zkEVM] Missing Activation Constraint in ECPair and ECDSA Modules

Location

`prover/ecdsa/antichamber.go#L55`

`prover/ecpair/ecpair.go#L47`

`common/common_constraints/common_constraints.go#L34`

Synopsis

For both modules ECPair and ECDSA in the zkEVM, the `isActive` columns are not constrained to be prevented from transitioning from `0` to 1. A typical `isActive` column is constrained to be binary and to refrain from the aforementioned transition. Rows with `isActive` set to `0` are used for padding, such that the column reaches a length that equals a power of two.

Impact

Critical. A malicious prover can exploit this soundness issue. In particular, columns such as `UnalignedPairingData.Limb` in the ECPair module can be activated in padding rows and can hence be used to manipulate the behavior of the modules ECPair and ECDSA.

Remediation

We recommend adding the constraints.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue I: [zkEVM] Missing Binary Constraints in ECPair Module

Location

prover/ecpair/ecpair.go#L236-L237

Synopsis

The binary constraints for the columns `IsFirstLineOfPrevAccumulator` and `IsFirstLineOfCurrAccumulator` are missing in the ECPair module.

Impact

Low. Since the two columns are only used as terms in products, a malicious prover assigning arbitrary, non-binary values to the columns does not gain any advantage in manipulating the behavior of the module.

Remediation

We recommend adding the binary constraints.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue J: [zkEVM] Missing Exclusive Binary Constraint in ECPair Module

Location

prover/ecpair/ecpair.go#L232-L233

std/evmprecompiles/08-bnpairing.go

Synopsis

The module ECPair is missing a constraint on the interplay between the columns `ToMillerLoopCircuitMask` and `ToFinalExpCircuitMask`. Their addition should sum to the column `isActive`, such that only one of the two can be activated in a row. The columns are used to distinguish the two circuits defined for the ECPair precompile module.

Impact

High. A malicious prover can exploit this soundness issue by activating both columns simultaneously and manipulate the data input to the two circuits responsible for the correct verification of the ECPair precompile.

Remediation

We recommend adding the constraint.

Status

The Linea team has resolved the issue in this PR by adding the new column `IsActive` to the `UnalignedPairingData` struct and checking that the sum of the two columns mentioned above equals this column. In addition, due to this new column, several other constraints have been either modified or newly added.

Verification
Resolved.

Issue K: [zkEVM] Missing Zeroization Constraint in Public Input Module

Location
[publicInput/execution_data_collector/execution_data_collector.go#L184]

[publicInput/execution_data_collector/execution_data_collector.go#L1057]

Synopsis
The module Public Inputs does not have a zeroization constraint for the column `TotalBytesCounter`.

Impact
Critical. A malicious prover can manipulate the data during padding (i.e., when rows are inactive) and more specifically, share an incorrect value for the final value of this column with the execution circuit (assuming Issue L has been resolved).

Remediation
We recommend adding the column to the set of columns being zeroized via the `isActive` column.

Status
The Linea team has resolved the issue in this PR.

Verification
Resolved.

Issue L: [zkEVM] Incorrect Value for TotalBytesCounter in the Local Opening of the Public Input Module

Location
prover/publicInput/input_extractor.go#L50

prover/circuits/execution/pi_wizard_extraction.go#L34

Synopsis
An incorrect value of the column `TotalBytesCounter` is being propagated as a local opening to the execution circuit.

Impact
Since this issue can be detected by the Linea team as soon as the corresponding check is enabled in the execution circuit, we consider it to be a low impact issue.

Technical Details
Currently, the first entry of the column `TotalBytesCounter` is shared via a local opening with the execution circuit. This is the incorrect entry and, instead, the last non-zero value should be shared. This has not been identified by the execution circuit because the corresponding check is being disabled until another issue is resolved.

Remediation

We recommend sharing the correct entry of the column.

Status

The Linea team has resolved the issue in this PR by adding the new column `FinalTotalBytesCounter`.

Verification

Resolved.

Issue M: [zkEVM] Correctness of Leaf Hashing Constraint In the State Manager Can Be Turned Off

Location

`prover/statemanager/accumulator/define.go#L510`

Synopsis

The hash verification for the Merkle tree in the state manager can be turned off because the column `isEmptyLeaf` is not constrained properly.

Impact

Critical. The verification of leaf hashes can be turned off by a malicious prover.

Technical Details

In the zkEVM, the state manager verifies the Merkle proofs for access to the state – that is, deletion, insertion, updates, and other operations on the leaves. In the submodule `accumulator` there is a constraint that checks the correct hashing of the leaf opening `(1 - IsEmptyLeaf[i]) * (Leaves[i] - LeafHashes[i])`, which is only active for non-empty leaves.

A malicious prover can set the Boolean column `IsEmptyLeaf` to `true` for all rows. By skipping this check, the prover does not need to provide a correct opening of the leaf. This can be prevented by adding a constraint, such that the Boolean is set to `true` if and only if the row is the third for an `insert` operation or the fourth for a `delete` operation.

Remediation

We recommend adding the constraint as described in the Technical Details section.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue N: [zkEVM] Missing Constraint for the Hashed Key During the Update Operation

Location

`prover/statemanager/accumulator/define.go`

Synopsis

There is no constraint preventing the `HKey` from being changed during an update operation.

Impact

Low. This issue could lead to undefined behavior.

Technical Details

An `update` operation is replacing a value in the state manager Merkle tree. It should change the value but not the `Hkey` of a leaf. In the accumulator module, there is no constraint verifying that the Hkey has not been modified.

Remediation

We recommend adding a constraint to enforce that the `Hkey` remains unchanged during an `update` operation.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue O: [zkEVM] Merkle Tree Can Be Updated Without Updating the Value of NextFreeNode Due To Missing Constraint

Location

`prover/statemanager/accumulator/define.go#L521`

Synopsis

During an insert operation for the state manager Merkle tree, the value `NextFreeNode` does not need to be updated due to a missing constraint for `IsInsertRow3`.

Impact

Medium. A malicious prover can update the Merkle tree without updating the value `NextFreeNode`. This breaks the deterministic structure of the Merkle tree and hence leads to undefined behavior.

Technical Details

In the accumulator module, the column `IsInsertRow3` indicates whether the current row is the third row. However, the column is not constrained properly and, hence, a malicious prover can set the value to `false` in any row. This can be used to turn off the below constraint. Consequently, the malicious prover would not have to update the `NextFreeNode` field.

```
IsActive[i] * (1 - IsFirst[i]) * (IsInsertRow3[i] * (NextFreeNode[i] -
NextFreeNode[i-1] - 1) + (1- IsInsertRow3[i]) * (NextFreeNode[i] -
NextFreeNode[i-1]) )
```

This can be prevented by adding the following constraint:

```
IsActive[i] * IsInsert[i] * IsEmptyLeaf[i] * (1-IsInsertRow3[i])
```

The Merkle tree structure should be deterministic – that is, the position of an insertion of a new leaf is deterministic. This requirement allows anyone to verify the Merkle tree hash with only the transaction history. Manipulating the value `NextFreeNode` by not increasing it properly breaks this assumption.

In addition, after the insertion of a new leaf with an incorrect `NextFreeNode` value, two leaves would have the same value for `NextFreeNode`. Consequently, inserting a new leaf would lead to undefined behavior.

Remediation

We recommend adding the constraint described in the Technical Details section.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue P: [zkEVM] Missing Consistency Constraint on the Hashed Key Value

Location

`statemanager/accumulator/define.go#L110`

Synopsis

The accumulator module is missing a constraint to check the consistency of the hashed key value between the column `Hkey` and the column `leafOpening.Hkey`.

Impact

High. A malicious prover could use this missing check to assign false values to `Hkey` and hence skip crucial constraints.

Technical Details

A leaf in the Merkle tree is the hash of several values, including the `Hkey` value. `Hkey` represents the hash of the 256-bits key. The accumulator module has two columns (`Hkey` and `leafOpening.Hkey`) that store this value. There is no consistency check across these two values. This allows a malicious prover to operate with two distinct values. In particular, an attacker could assign a false value to the `Hkey` and hence skip the accompanying checks on the value.

Remediation

We recommend adding a constraint to enforce the equality of the two values.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Issue Q: [Gnark Frontend] Missing Binary Constraint in Select Primitive

Location

`frontend/cs/scs/api.go#L405`

Synopsis

The `selector` constraint `Select(b,i1,i2)` does not have a constraint enforcing the selector bit b to be Boolean.

Impact

Critical. If b is not Boolean, the function does not behave like a selector.

Remediation

We recommend adding `builder.AssertIsBoolean(b)`.

Status

The Linea team has resolved the issue in this PR.

Verification

Resolved.

Suggestions

Suggestion 1: [Prover Protocol] Update the Linea Prover Documentation

Location

`https://eprint.iacr.org/2022/1633`

`zkevm-monorepo/prover/protocol`

Synopsis

We identified several deviations between the paper and its implementation, including several inconsistencies and errors, such as incorrect constraints or typographical mistakes in the definitions.

Below, we list a few findings (non-exhaustive):

- In contrast to the code here, in Appendix `A.8`, the constraint 2.) is missing the multiplication of `IsActive[i]`.
- The code computes here, `2*NotEndOfProof[i]*PosAcc[`**`i-1`**`]`, while the paper, in Appendix `A.8` for the constraint, 4.) incorrectly computes `2*NotEndOfProof[i]*PosAcc[`**`i+1`**`]`.
- In contrast to the code here, the formula for `P(r)` is missing the term `a^n-1` in Appendix `A.2`.
- The random coin `r_merge` in section `9.2`, bullet point 4, has been removed from the protocol and replaced with `r_collapse^nbOpencol` (see the corresponding code segment here).
- In the definition of `Z_i(x)` in Figure 4, the index `j` should index both `A'` and `B'`.
- The definition of the `Z_v` product in Figure 5 has no specification of the index `k`.
- In the definition of 'local constraints' in section `4.3`, it is the *oracle* that does the computation, not the verifier.

Mitigation

We recommend updating the paper. Writing a correct and in-depth specification can lead to the finding of further issues that may have been missed so far (see, for example, the soundness issue in Nova found during the writing of a proper security proof [NBS23]).

Status

The Linea team stated that they plan on implementing the suggested mitigation in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 2: [Prover Protocol] Improve Code Quality

Location

`zkevm-monorepo/prover/protocol`

Synopsis

Overall, the code quality in the prover protocol is very high. The wizard protocol has many defensive checks and panics to improve usability. Nevertheless, we identified several opportunities for improving the codebase by removing redundant code or inefficiencies and adding missing panics.

Mitigation

Below, we list a few opportunities for improvements in the compiler:

- In the vortex compiler here, the code excludes the case where the compiled IOP contains `Precomputed` columns but not `committed` columns. Even though this particular use case seems unlikely, we recommend also checking this for the `Precomputed` columns.
- For the code here, the function returns instead of panicking. As a result, all univariate queries are removed and, in turn, the vortex compiler runs into a panic.
- In the compilation of MiMC here, in the case of only one query, the code does not return and, instead, inefficiently adds further constraints. We recommend returning for this case in the `if-loop`.
- This code segment here, in the splitter, cannot be reached and can therefore be removed.
- The `sortingMap` here is not used and can therefore be removed.
- The columns with the status `Precomputed` are erroneously excluded from the compilation step sticking here. This is inefficient since they are verified manually by the verifier instead.
- The columns with the status `VerifyingKey` are excluded from the compiler step splitting here. Consequently, their size is not reduced into the target size. We suggest including the splitting of `VerifyingKey` columns as well.

Below, we list a few opportunities for improvements in the folder `wizard`:

- There is a redundant check in `compiledIOP` here, in `InsertColumn` (checked also in function `AddToRound`).
- The function `InsertColumn` does not check if the `size == 0` here, while other, similar functions, such as `InsertProof` here, do check this.
- There is no round consistency check in `InsertFixedPermutation` here.
- Here, the function `GetColumnAt` does not have any sanity checks, while `GetColumn`, for example, checks if the column is public.

- In `gnark_verifier.go` here, `GetColum` does not check if the column is public, while the counterpart in `verifier.go` does.
- In `gnark_verifier.go` here, `GetColumnAt` does not check if the position falls into the range, while it is properly checked in `verifier.go`.
- Some constraint constructors check whether the query ID is greater than zero (see here, for example, in global constraints). This check is only there for `Local`, `Global`, and `MiMC` constraint constructors and is missing for all other constraint constructors, such as `newLocalOpening`, `Univariate`, and `Range`.

We identified several improvements in the folder `dedicated`:

- Here, the `if-loop` should be accessed for `numCol > 1` and not `numCol > 0`. This increases efficiency since it avoids adding the merging coin.
- One line below, here, the increment of the round is not necessary and is hence inefficient.
- Here, for the `isZero` column, the `invOrZero` column name is misleading: it is constrained to be either the inverse of c or any other value. Since it is not used further in the code, it is not necessary to constrain it. However, we recommend adding a comment that this column is only constrained to be the inverse of c and that it can otherwise have any other value, even though the name suggests otherwise.
- At this code segment here, logging occurs instead of panicking. We recommend adding a panic for this case.
- The collapsing step of the columns L, R, O of the Plonk implementation here is not used. According to the Linea team, they only use Plonk with single columns L, R, O. We recommend removing it to avoid confusion.

We recommend addressing the findings listed above.

Status

The Linea team stated that they plan on implementing the suggested mitigation in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 3: [Compress Library] Improve Code Quality

Location

See the Mitigation section.

Synopsis

Within the `Compress` library, we found a few opportunities for improving the code by using hard coded values or returning error messages.

Mitigation

Below, we list a few improvements:

- The value `1 << 19` for the input buffer here should be hard coded as a constant.
- The function `Write` here does not return proper error messages in various cases.
- The input `addressableBytes` to the function `NewDynamicBackrefType` is not used anymore and should be removed here.
- In the same function, the bound that is set to 21 here should be hard coded as a constant.

Status
The Linea team stated that they plan on implementing the suggested mitigation in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification
Unresolved.

Suggestion 4: [zkEVM] Improve Code Quality

Location
See the Mitigation section.

Synopsis
Within the zkEVM codebase, we found a few opportunities for improving the code.

Mitigation
Below, we list a few improvements:

- Here and in various other places in the zkEVM folder, the option `addGateForRangeCheck` is set to `true` in the function `WithRangecheck`. This is not necessary and can be changed to `false` instead.
- Constraints are duplicated in the state manager accumulator for `IsFirst`, `IsInsert`, `IsDelete`, `IsUpdate`, `IsReadZero`, and `IsReadNonZero`. Their Booleanity constraints (e.g., for `IsInsert`) also imply that they will be zero when the accumulator is inactive. This is checked again here. Similar remarks also apply to `IsNewHash` and `IsHashEnd` in the `mimcCodeHash` module.
- Comments are incorrect and incomplete in the following locations:
  - `statemanager/accumulator/settings.go#L23-L24`
  - `statemanager/accumulatorsummary/accumulator_summary.go#L40-L42`
  - `statemanager/statesummary/state_summary.go#L553-L556`

Status
The Linea team stated that they plan on implementing the suggested mitigation in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification
Unresolved.

Suggestion 5: [zkEVM] Refactor Common Patterns in Constraints

Location
Examples (non-exhaustive):

`prover/statemanager/mimcCodeHash/define.go`

`prover/common/common_constraints/common_constraints.go`

Synopsis
Many common patterns exist in the algebraic constraints that make up the various zkEVM modules. Some abstractions already exist but are not always applied consistently (see the `common_constraints.go` file). Refactoring the code to leverage the abstraction facilities provided by

the Go programming language would make it easier to read and reason about. In addition to helping with bug prevention, it could also benefit future audits.

Mitigation

Below, we list a few candidates for refactoring (non-exhaustive):

- Enforcing the ‘Booleanity’ of a column;
- Representing logical connectives such as negation, implication, etc; and
- Asserting equality and inequality.

In addition, we recommend using the functions in the common constraints file throughout the codebase.

Status

The Linea team has partially addressed this suggestion in some modules, such as the ECPair and ECDSA module, as demonstrated in the remediation of some of the issues reported above.

Verification

Partially Resolved.

Suggestion 6: [Gnark Frontend] Improve Code Comments

Location

`Consensys-gnark/frontend`

Synopsis

There are insufficient code comments explaining the rationale behind certain critical lines of code. This reduces the readability of the code and, as a result, makes reasoning about the security of the system more difficult. Comprehensive in-line documentation helps provide reviewers of the code with a better understanding and ability to reason about the system design.

Additionally writing pre- and post-conditions as comments on a user interface helps prevent bugs and security issues in third-party applications.

Below we list some of our findings:

- In `cs/scs/api.go#L160`, code comments should specify that in case `0/0`, the return value is unconstrained.
- In `frontend/api.go#L57`, it is not specified what should happen in case division is not defined.
- In `frontend/api.go#L60`, it is not specified what should happen in case inversion is not defined.
- In `frontend/api.go#L74`, `frontend/api.go#L92`, and `frontend/api.go#L92`, it is not specified if the b's are assumed to be Boolean-constrained already or if the function has to constrain them.
- In `frontend/api.go#L85`, the comment should reference the AND operator.
- In `frontend/api.go#L122`, the comment should be `v != 0` OR `v != 1`.
- The comment in `cs/scs/builder.go#L313` seems to suggest a panic; however, no panic was implemented.
- The comment in `cs/scs/api.go#L391` should be about the AND operator.

Our team also found that the functions `AssertIsBoolean` and `MarkBoolean` ensure that a frontend variable that is considered Boolean will have a corresponding Boolean constraint. These functions consult the underlying table `mtBooleans`. However, the guarantee that a Boolean constraint will be implemented relies on the fact that only `MarkBoolean` can alter the `mtBooleans` table. Although we did not identify any issues in the current code version, the practice of using non-modular and non-local logic can be difficult to audit and adds the risk that future code changes might not follow the needed code pattern. We suggest making this behavior clear in the comment to prevent future code updates from breaking this behavior.

Mitigation

We recommend referring to the notes above to expand and improve the code comments within the codebase in order to better describe the intended functionality of the components, thereby facilitating reasoning about the security properties of the system.

Status

The Linea team has addressed the findings listed above (see this PR).

Verification

Resolved.

Suggestion 7: [Gnark Frontend] Missing State Reset

Location

`blob/v1/blob_maker.go#L168`

Synopsis

There is no state reset when the compressor fails to write a block to the `BlobMaker`.

Mitigation

We recommend implementing the state reset, as is done in `blob/v1/blob_maker.go#L187-L190`.

Status

The Linea team has included the recommended state reset in PR #114.

Verification

Resolved.

Suggestion 8: Improve Documentation

Synopsis

Our team noted that the documentation was incomplete and scattered across many files, pdfs, documents, and research papers.

Mitigation

We recommend updating the available documentation to include clear and precise specifications, as well as providing additional, detailed documentation to allow both future developers and auditors to easily understand the different components of the system. We further recommend collecting the documentation in a single place, to facilitate future audits.

Status

The Linea team stated that they plan on implementing the suggested mitigation in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Appendix

Appendix A: In-scope Components

Review 1:
Circuit + outer circuit (aka Wizard Verifier)

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/execution
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/internal

Circuit blob decompression + out of circuit compression

- github.com/consensys/compress
- https://github.com/Consensys/gnark/tree/master/std/compress
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/blobdecompression/ (except blobdecomppression/v0)
- https://github.com/Consensys/zkevm-monorepo/blob/main/prover/lib/compressor/

Circuit EVM Precompiles

- https://github.com/Consensys/gnark/blob/master/std/evmprecompiles/01-ecrecover.go
- https://github.com/Consensys/gnark/blob/master/std/evmprecompiles/05-expmod.go
- https://github.com/Consensys/gnark/blob/master/std/evmprecompiles/08-bnpairing.go
- https://github.com/Consensys/gnark/tree/master/std/hash/mimc
- https://github.com/Consensys/gnark/tree/master/std/permutation/sha2

Circuit aggregation (Caller side / integration)

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/aggregation
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/emulation
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/circuits/pi-interconnection

Review 2:
gnark frontend

- https://github.com/Consensys/gnark/tree/master/frontend

Review 3:
Protocol

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/protocol (except /protocol/serialization)
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/symbolic

Review 4:
zkEVM

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/zkevm/
  - Except : https://github.com/Consensys/zkevm-monorepo/blob/main/prover/zkevm/arithmetization/define/define.go
  - https://github.com/Consensys/zkevm-monorepo/tree/main/prover/zkevm/prover/hash/datatransfer

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/zkevm/prover/statemanager/mock

Backend bits

- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/aggregation
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/blobdecompression
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/blobsubmission
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/ethereum
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/execution/bridge
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/backend/execution/statemanager
- https://github.com/Consensys/zkevm-monorepo/tree/main/prover/lib/shnarf_calculator

The above in-scope audit target was provided by the Linea team to Least Authority and assessed for the purposes of this report.

In addition, any dependency and third-party code, unless specifically included above, were considered out of the scope of this audit.

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 multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. 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. We are an international team that believes 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 include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website 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. As 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 and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and 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 before having 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 and comprehensive 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 Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, 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 a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.