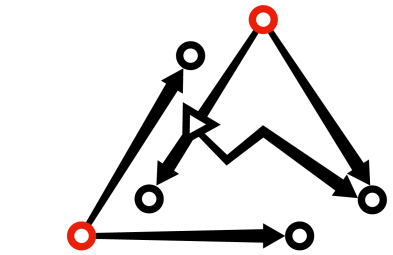

Least Authority

PRIVACY MATTERS

OPRF Noir Circuits

Security Audit Report

TACEO

Final Audit Report: 26 January 2026

Table of Contents

Overview

Background

TACEO has requested Least Authority perform a security audit of their OPRF Noir Circuits.

Project Dates

- **December 4, 2025 - December 15, 2025:** Initial Code Review *(Completed)*
- **December 17, 2025:** Delivery of Initial Audit Report *(Completed)*
- **26 January, 2026:** Verification Review *(Completed)*
- **26 January, 2026:** Delivery of Final Audit Report *(Completed)*

Review Team

- Mirco Richter, Cryptography Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Noir OPRF Circuits 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:

- https://github.com/TaceoLabs/oprf-service/tree/main
  - `noir/oprf/src/commitment.nr`
  - `noir/oprf/src/constants.nr`
  - `noir/oprf/src/dlog.nr`
  - `noir/oprf/src/main.nr`
  - `noir/oprf/src/merkle_proof.nr`
  - `noir/oprf/src/nullifier.nr`
  - `noir/oprf/src/query.nr`
  - `noir/oprf/src/types.nr`
  - `noir/eddsa_poseidon2/src/lib.nr`
  - `noir/babyjubjub/src/lib.nr`
  - `noir/babyjubjub/src/hash_to_curve.nr`
  - `noir/babyjubjub/src/montgomery.nr`
  - `noir/babyjubjub/src/window_table.nr`
  - `noir/babyjubjub/src/check_sub_group.nr`
  - `noir/babyjubjub/src/escalar_mul_fix.nr`

Specifically, we examined the following Git revision for our initial review:

- `aaf0c8886b03b26ad483f97cc5403c37b89f7fad`

For the verification, we examined the following Git revision:

- `50798b402e81ed3726674746c8c36dc7ef0931ee`

For the review, this repository was cloned for use during the audit and for reference in this report:

- TaceoLabs-nullifier-oprf-service:
  https://github.com/LeastAuthority/TaceoLabs-nullifier-oprf-service

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:

- A Nullifier Protocol based on a Verifiable, Threshold OPRF:
  https://github.com/LeastAuthority/TaceoLabs-nullifier-oprf-service/blob/audit/docs/oprf.pdf
- Website:
  https://taceo.io
- Previous Security Audit Report by Least Authority on TACEO's OPRF Circuits (pdf) *(delivered via email on 10 November 2024)*

In addition, this audit report references the following documents:

- S. S. Chow, C. Ma, and J. Weng, "Zero-Knowledge Argument for Simultaneous Discrete Logarithms." *ACM Digital Library,* 2012, [CMW12]
- L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A Faster Version of the Poseidon Hash Function." *IACR Cryptology ePrint Archive*, 2023, [GKS23]
- RFC 9380 | Hashing to Elliptic Curves:
  https://www.rfc-editor.org/rfc/rfc9380.html
- Noir language documentation:
  https://noir-lang.org/docs

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The OPRF service provides publicly verifiable, privacy-preserving nullifiers via a verifiable threshold OPRF (oblivious pseudorandom function). The `Noir` subdirectory contains `Noir` circuits for the OPRF nullifier workflow and implements prover logic for the protocol described in the OPRF Nullifier Specification. Reusable primitives, such as `Poseidon2`, `BabyJubJub` operations, a Merkle tree, encode-to-curve, and a DLEQ verifier, along with OPRF key generation circuits, support the threshold setup and proof composition.

This assessment constitutes our second audit of the same functionality. It follows our earlier audit of the circom implementation (referenced in the Supporting Documentation section) and allows a direct comparison between the `Noir` and `circom` versions.

System Design

Our team examined the design of TACEO's `Noir` circuits and found that security has generally been taken into consideration. The implementation is based on the OPRF Nullifier Specification. Below, we provide additional details on the system's design and security considerations.

Across all reviewed modules, we first compared the `Noir` circuits against the previously audited `circom` circuits and verified essential constraints such as booleanity, range bounds, subgroup membership, division-by-zero guards, and logic invariants, applying this uniformly across all files mentioned.

We audited the OPRF-based nullifier and query logic against the OPRF Specification and analyzed `dlog.nr` as a noninteractive Chaum-Pedersen proof in accordance with [CMW12]. We found that the nullifier circuit does not constrain the public inputs `signal_hash` and `nonce` (Issue C) and that nullifiers are malleable due to missing subgroup checks of `oprf_response` and `oprf_response_blinded` (Issue A).

We also compared `merkle_proof.nr` against standard binary Merkle constructions, assuming that `depth <= N` is checked by the verifier outside of the circuit, and recommend improving the domain separator (Suggestion 1). Regarding signatures and hashing, we assessed the `eddsaposeidon2` implementation against established EdDSA-on-BabyJubJub practices and the `Poseidon2` specification [GKS23]. With respect to curve arithmetic and subgroup handling, we examined `the BabyJubJub` implementations and found that function `segment_mul_fix_generator` uses `MontgomeryPoint::add` in a manner that permits unconstrained inputs (Issue B). Additionally, in the context of hashing to the curve, we reviewed the `Elligator2` implementation in `hash_to_curve.nr` against RFC 9380, identifying that the `inverse_or_zero` function deviates from the definition of inv0 in RFC 9380 (Issue D) and that the handling of the 2-torsion exception deviates from RFC 9380 (Issue E).

Code Quality

We performed a manual review of the repositories and found the code to be well-organized, which facilitated navigation.

Tests

Our team found test coverage for the in-scope repositories to be insufficient, as it does not fully cover the generated code. We recommend improving test coverage (Suggestion 2).

Documentation and Code Comments

The project documentation provided by the TACEO team was sufficient in describing the intended functionality of the system. Additionally, the codebase includes comments that were helpful in understanding most of the components of the system.

Scope

Similar to the previous audit, we note that several important checks are intentionally omitted from the circuit to minimize the number of constraints and are instead expected to be enforced by the verifier. Because the verifier was not in scope for this review, the code was audited under the assumption that these checks are correctly performed by the verifier. In particular, the verifier must perform at least the following checks, in addition to those described in the whitepaper as part of the protocol:

- Here and in all other circuits, the public key `oprf_pk` must be a valid `BabyJubJub` point in the correct subgroup; and
- Here, the verifier should enforce `depth_u32 <= MAX_DEPTH` outside of the circuit.

Since the verifier itself was outside the scope of this review, our team could not determine a complete list of checks the verifier must perform and therefore recommends conducting a comprehensive follow-up security assessment of this component.

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	SEVERITY	STATUS
Issue A: Nullifier Malleability From Missing Subgroup Checks in Unblinding	Critical	Resolved
Issue B: Underconstrained Montgomery Addition Can Break Circuit Soundness	High	Resolved
Issue C: Noir Nullifier Circuit Leaves signal_hash and Nonce Unconstrained	High	Resolved
Issue D: Function inverse_or_zero Violates RFC 9380 inv0 by Making Zero Inputs Unsatisfiable	Informational	Resolved
Issue E: Deviation From RFC 9380 Exceptional Case Handling in Edwards–Montgomery Conversions	Informational	Determined Non-Issue
Suggestion 1: Bind the Merkle Leaf Hash Domain Separator to NUM_KEYS	Informational	Unresolved
Suggestion 2: Improve Test Coverage	Informational	Partially Resolved

Issue A: Nullifier Malleability From Missing Subgroup Checks in Unblinding

Location
noir/oprf/src/query.nr#L94

Synopsis
The Noir circuit function verify_unblinding constrains oprf_response_blinded == [β]oprf_response while only constraining oprf_response to be on the BabyJubJub curve. It does not enforce that the involved points are in the prime-order subgroup as assumed by the protocol when injecting and validating the unblinded point.

Impact
High.

For a fixed query and scope, a prover can generate multiple distinct valid nullifiers, undermining the protocol's one-nullifier-per-scope replay protection and enabling limited double use.

Feasibility
High.

The prover controls β and can select β divisible by BabyJubJub's cofactor 8, then set oprf_response to the correct unblinded result plus a nontrivial torsion point so that reblinding still matches oprf_response_blinded while the hash input to the nullifier changes.

Severity
Critical.

Preconditions
The deployment must accept proofs produced by this Noir circuit, and no other in-circuit constraint or verifier check should enforce that oprf_response lies in the prime-order subgroup.

Technical Details
BabyJubJub has curve order n = 8*q, so points decompose into a prime-order component and an 8-torsion component. For β = 0 (mod 8), scalar multiplication eliminates the torsion component. Since the circuit hashes oprf_response.x and oprf_response.y into the nullifier but only checks [β]oprf_response == oprf_response_blinded, an attacker can replace oprf_response with oprf_response + T for any 8-torsion point T and produce a different nullifier that still verifies.

Remediation
We recommend adding explicit prime-order subgroup membership constraints for the points oprf_response_blinded as well as oprf_response in verify_unblinding, in line with the protocol requirement in docs/oprf.pdf.

Status
The TACEO team has added a prime-order subgroup check in verify_unblinding (query.nr), preventing torsion-based nullifier malleability.

Verification
Resolved

Issue B: Underconstrained Montgomery Addition Can Break Circuit Soundness

Location
noir/babyjubjub/src/montgomery.nr#L66

noir/babyjubjub/src/escalar_mul_fix.nr#L59

Synopsis
`MontgomeryPoint::add` constrains the slope using `lambda * (q.u - self.u) == q.v - self.v`, which becomes a tautology when `self == q`, leaving `lambda` unconstrained and allowing a malicious prover to rewrite the witness generator to steer the output point away from the group law or the curve equation. Because this addition is used inside fixed-base scalar multiplication, the underconstraint can propagate into the EdDSA verification gadget and might break soundness.

Impact
High.

A prover can satisfy the existing constraints while generating off-curve results in the fixed-base scalar multiplication.

Feasibility
Medium.

As shown below, a concrete 251-bit scalar can trigger the `self == q` case in the scalar multiplication accumulator chain, and a custom witness generator can supply an arbitrary `lambda` that satisfies the current constraints. However, computing `lambda` to break soundness requires additional steps that our team has not fully analyzed.

Severity
High.

Preconditions
The affected circuits must be used to accept proofs from potentially malicious provers, and the proof system must allow provers to supply arbitrary witnesses provided that all constraints are satisfied.

Technical Details
The root cause is the single slope constraint in `MontgomeryPoint::add` combined with unconstrained witness computation.
The code computes `lambda` using an unsafe unconstrained division and then constrains it only by the equation `lambda * (q.u - self.u) == q.v - self.v`. When the left summand equals the right summand, both differences are zero, and the constraint is satisfied for every `lambda`. Since u3 and v3 are then computed from `lambda` as

```
u3 = B*lambda^2 - A - 2*self.u
v3 = lambda*(self.u - u3) - self.v,
```

the resulting output point is no longer forced to equal the true `sum` under the `Montgomery` curve group law or to satisfy the `Montgomery` curve equation.

Because a malicious prover does not need to run the honest witness generator, they can bypass the intended witness generation in `add_lambda_unconstrained` and choose `lambda` freely to control the output value `(u3, v3)`.

The implementation documents a precondition that callers must use the function `double()` when points are equal, but `segment_mul_fix_generator` builds an accumulator chain that repeatedly evaluates `accumulator = accumulator.add(window_outputs[i])` without constraining that the two inputs differ. Functions `generator_scalar_mul` and `generator_mul_fix` then call `segment_mul_fix_generator`.

Due to this behavior, it is possible to compute a scalar S and a sufficiently large index i such that, in `segment_mul_fix_generator`, the `accumulator` at step `i-1` equals `window_outputs[i]` (because the accumulator exceeds the modulus), reaching the critical case `self.add(p)` with `self==p`. In order to find a concrete input S, our team implemented a Sage-based search to find a collision pattern at position `i=82`, resulting in

S = `0x1f58c75839a71d0ee9fe26d7ea7b2a80f16124ff9125d0f72f3d2bad481306c`,

which passes `validate_babyjubjub_field`. However, executing a test for `generator_scalar_mul(S)` fails with the expected "`division by zero`" error, implying that the circuit is underconstrained in this case.

Consequently, since `verify_eddsa_poseidon2` computes `S*G` using `generator_scalar_mul(S)` and then checks the cofactored EdDSA equation `8*(S*G-R-h*pk)=Id`, an underconstrained `generator_scalar_mul(S)` may or may not allow a prover to satisfy this equation without knowing the signing key. The analysis is subtle because many circuits in this computation assume that the inputs satisfy the BabyJubJub/`Montgomery` curve equation, an assumption that need not hold for the compromised value `S*G`.

Mitigation

We suggest refraining from relying on proofs generated with circuits that use `segment_mul_fix_generator` with the incomplete `Montgomery` addition for any authorization decision. Until a fixed circuit is deployed, any in-circuit EdDSA result should be treated as untrusted.

Remediation

We recommend mirroring `circomlib`'s segmented `EscalarMulFix`. Specifically, the scalar should be split into 249-bit segments, the `Montgomery` window accumulator and compensation should be run per segment, the result should be converted to Edwards to apply the correction (`acc - comp`), and the next segment should be rebased using the segment's `dbl` output. This avoids modular wrap-around within a segment, so the `Montgomery` adder never hits the critical `self == other` condition.

Alternatively, a full `Montogomery` addition circuit should be used in `segment_mul_fix_generator`. That is, `if self==p then self.double() else self.add(p)`.

Status

The TACEO team has implemented a mitigation that avoids the `self == q` case in fixed-base scalar multiplication by enforcing a max segment size and splitting the 251-bit scalar into two segments.

Verification

Resolved.

Issue C: Noir Nullifier Circuit Leaves signal_hash and Nonce Unconstrained

Location
noir/oprf/src/nullifier.nr#L18

noir/oprf/src/nullifier.nr#L19

Synopsis
The `Noir` nullifier circuit takes `signal_hash` and `nonce` as public inputs but never constrains them, so a proof does not attest to the provided values and can be reinterpreted with attacker-chosen `signal_hash` and `nonce`.

Impact
Medium.

Depending on the application, an attacker can supply arbitrary values while still presenting a valid proof for the same underlying witness.

Feasibility
High.

Exploitation only requires choosing different public inputs that are not referenced by any constraint.

Severity
High.

Preconditions
`Noir` proofs produced from this circuit must be accepted and `signal_hash` or `nonce` must be used as meaningful public inputs in verification logic or in application policy decisions.

Technical Details
In `oprf_nullifier`, the parameters `nonce` and `signal_hash` are unused in the function body, so they do not influence any constraints. The `circom` implementation explicitly adds dummy squaring constraints to bind `signal_hash` and `nonce`, and the protocol specification describes squaring the message to prevent tampering, which matches the Semaphore V4 specification Dummy Square rationale for unused message inputs.

Remediation
We recommend binding `signal_hash` and `nonce` in the `Noir` constraint system, for example by adding dummy squaring constraints as in the `circom` circuits.

Status
The TACEO team has added dummy squaring constraints for both `nonce` and `signal_hash`.

Verification
Resolved.

Issue D: Function inverse_or_zero Violates RFC 9380 inv0 by Making Zero Inputs Unsatisfiable

Location
noir/babyjubjub/src/hash_to_curve.nr#L25

Synopsis
RFC 9380 defines `inv0` with `inv0(0) = 0`, but `inverse_or_zero` evaluates `1/x` unconditionally. As a result, `Noir` cannot construct a witness when `x = 0`, and the circuit becomes unsatisfiable instead of constraining `0`.

Impact
Low.

An attacker can at most cause proof generation to fail for inputs that hit this edge case, which is an availability issue rather than a means to forge outputs.

Feasibility
Low.

In the current implementation, `inverse_or_zero` is used in `map_to_curve_elligator2` as well as in `rational_map_mont_to_twisted_edwards`. `map_to_curve_elligator2` inverts `tv1 + 1`, which is guarded away from zero, while `rational_map_mont_to_twisted_edwards` inverts `(s + 1) * t`, which is zero when `t = 0` or `s = -1`. In this context, `t = 0` occurs when `hash_to_field` outputs `u = 0`, which occurs with negligible probability.

Severity
Informational.

Preconditions
The prover input must lead to an internal call to `inverse_or_zero` with `x = 0`.

Technical Details
The implementation computes `(1 / x) * (1 - IsZero(x))`, and `Noir` encodes division with constraints that require a valid multiplicative inverse witness for `x`, so `x = 0` renders the constraint system unsatisfiable even though the result is later multiplied by zero.

Mitigation
Since the probability of finding a `hash_to_field` output with `u=0` is negligible, this is purely informational in the current application.

Remediation
We recommend replacing `inverse_or_zero` with a witness-safe `inv0`, such as `if x == 0 { 0 } else { 1 / x }`, in case the function `inverse_or_zero` is used in contexts other than `map_to_curve_elligator2` or `rational_map_mont_to_twisted_edwards`.

Status
The TACEO team has replaced the unsafe division with a witness-safe `inv0` pattern that allows `x == 0`.

Verification
Resolved.

Issue E: Deviation From RFC 9380 Exceptional Case Handling in Edwards–Montgomery Conversions

Location
noir/babyjubjub/src/montgomery.nr#L23

noir/babyjubjub/src/montgomery.nr#L46

Synopsis
The `to_montgomery` and `from_montgomery` conversion circuits compute the rational maps in underconstrained code and enforce only the cross-multiplication identities, so exceptional inputs with zero denominators can satisfy the constraints while leaving the converted coordinates unconstrained.

Impact
High.

If the `to_montgomery` circuit accepts input `(0,-1)` or the `from_montgomery` circuit accepts input `(0,0)`, an attacker can introduce arbitrary converted points that need not be actual curve points, which can undermine any higher-level logic that assumes conversion outputs are valid.

Feasibility
Low.

In this codebase, every nontest use of these maps occurs inside subgroup checks or scalar multiplication paths that operate on prime-order points by construction or on public keys that are separately validated. As a result, the exceptional torsion cases are unreachable in executions.

Severity
Informational.

Preconditions
An attacker must be able to provide a point that reaches `to_montgomery` or `from_montgomery` without prior prime-order subgroup enforcement or explicit exceptional-case handling.

Technical Details
In `to_montgomery`, the constraints `u*(1-y)=1+y` and `v*x=u` are satisfied by the on-curve nonidentity 2-torsion input `(x,y)=(0,-1)` with `u=0`, leaving `v` free because `(u,v)` is not constrained to satisfy the `Montgomery` equation.
In `from_montgomery`, the exceptional `Montgomery` input `(u,v)=(0,0)` causes the constraints `x*v=u` and `y*(u+1)=u-1` to hold for any `x` with `y=-1` because `(x,-1)` is not constrained to satisfy the Edwards equation.
Appendix D.1 of RFC 9380 defines these maps as undefined when Edwards denominators vanish, namely `v==0` or `w==1`, and when `Montgomery` denominators vanish, namely `t==0` or `s==-1`. Accordingly, it requires returning the identity in these cases.

Mitigation
Current call sites prevent the exceptional inputs, so this is purely informational for the present application.

Remediation
We recommend adding explicit exceptional-case handling or nonzero denominator constraints in both conversions in accordance with RFC 9380, or adding a code comment explaining the undefined inputs to protect future uses of `to_montgomery` and `from_montgomery`.

Status
Our team determined that this finding is informational in nature and a non-issue in the current codebase, and the TACEO team has since improved the comments by clarifying the preconditions.

Verification
Determined Non-Issue.

Suggestions

Suggestion 1: Bind the Merkle Leaf Hash Domain Separator to NUM_KEYS

Location
noir/oprf/src/merkle_proof.nr#L6

Synopsis
The Merkle leaf hash uses a fixed domain separator and a fixed 16-element `Poseidon2` input that is zero-padded, so a leaf and therefore a Merkle root from a circuit compiled with a smaller `NUM_KEYS` can collide with one compiled with a larger `NUM_KEYS` when the extra keys are zero.

This is not exploitable in the current deployment because `NUM_KEYS` is fixed, but it may pose a concern if roots are reused across deployments or configurations. RFC 9380 requires domain separation between independent instantiations and recommends constructing domain-separation tags that include identifying information such as an application string and version, which in this context corresponds to binding the leaf hash to `NUM_KEYS`.

Mitigation
We recommend changing the leaf-hash domain separator to encode `NUM_KEYS`, for example `SEPARATOR = MERKLE_LEAF_DS || NUM_KEYS`, then using `SEPARATOR` as the capacity element while treating existing roots as configuration-specific.

Status
The TACEO team has not modified the implementation to bind the leaf hash to `NUM_KEYS`. The team opted to retain the fixed domain separator because `NUM_KEYS` is currently fixed, and plans to add the binding only if that configuration changes.

Verification
Unresolved.

Suggestion 2: Improve Test Coverage

Location
TaceoLabs-nullifier-oprf-service/tree/audit/noir

Synopsis

There is insufficient test coverage implemented to test the correctness of the implementation and that the system behaves as expected. Tests help identify implementation errors, which could lead to security vulnerabilities.

Sufficient test coverage should include tests for success and failure cases (all possible branches), which helps identify potential edge cases, and protect against errors and bugs that may lead to vulnerabilities. A test suite that includes sufficient coverage of unit tests and integration tests adheres to development best practices. In addition, end-to-end testing is also recommended to assess if the implementation behaves as intended.

Mitigation

We recommend that comprehensive unit test coverage be implemented in order to identify any implementation errors and to verify that the implementation behaves as expected.

Status

The TACEO team has either added or strengthened several Noir unit tests (for example, subgroup checks, `scalar-mul` tests, and a corrected negative EdDSA test), improving coverage.

Verification

Partially Resolved.

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.