FYEO

Security Code Review
Endless Bridge

Endless Labs

June 2025
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

Endless Labs engaged FYEO Inc. to perform a Security Code Review Endless Bridge.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on May 15 - May 27, 2025, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-Endless-01 – Authentication bypassed while default admin
- FYEO-Endless-02 – The verify_message function accepts empty signers as valid
- FYEO-Endless-03 – Signature recovery concern
- FYEO-Endless-04 – Validator limits and threshold
- FYEO-Endless-05 – Deduplication on 32 bytes instead of address
- FYEO-Endless-06 – Excess amount gets stuck
- FYEO-Endless-07 – Missing zero checks, one step admin transfer

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review Endless Bridge. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/endless-labs/endless-bridge-contract with the commit hash a0cf317ac33073ebb6f629c535eed35837ed0d49.

Remediations were submitted with the commit hash f8db1740ff8e3ffc89d280128e69b05d06187cab.

Files included in the code review
```
endless-bridge-contract/
├── endless/
│   ├── bridge-core/
│   │   └── sources/
│   │       ├── message.move
│   │       └── validator.move
│   └── bridge-token/
│       └── sources/
│           ├── config.move
│           ├── execute.move
│           ├── fund_manage.move
│           ├── pool.move
│           ├── pool_v2.move
│           └── token.move
└── evm/
    ├── comn/
    │   ├── ComFunUtil.sol
    │   ├── IAdmin.sol
    │   ├── ICREATE3Factory.sol
    │   ├── IExecutor.sol
    │   ├── IFundManager.sol
    │   ├── IMessager.sol
    │   ├── IPool.sol
    │   ├── IToken.sol
    │   ├── IValidator.sol
    │   ├── SafeERC20.sol
    │   ├── TokenBatch.sol
    │   ├── Types.sol
    │   └── WToken.sol
    ├── core/
    │   ├── Admin.sol
```

Files included in the code review
```
│     ├── Comn.sol
│     ├── Messager.sol
│     └── Validator.sol
├── proxy/
│     ├── Proxy.sol
│     └── ProxyAdmin.sol
├── token/
│     ├── Comn.sol
│     ├── Executor.sol
│     ├── FundManager.sol
│     ├── Pool.sol
│     └── Token.sol
└── BaseComn.sol
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review Endless Bridge, we discovered:

- 2 findings with HIGH severity rating.
- 2 findings with MEDIUM severity rating.
- 3 findings with LOW severity rating.

The following chart displays the findings by severity.

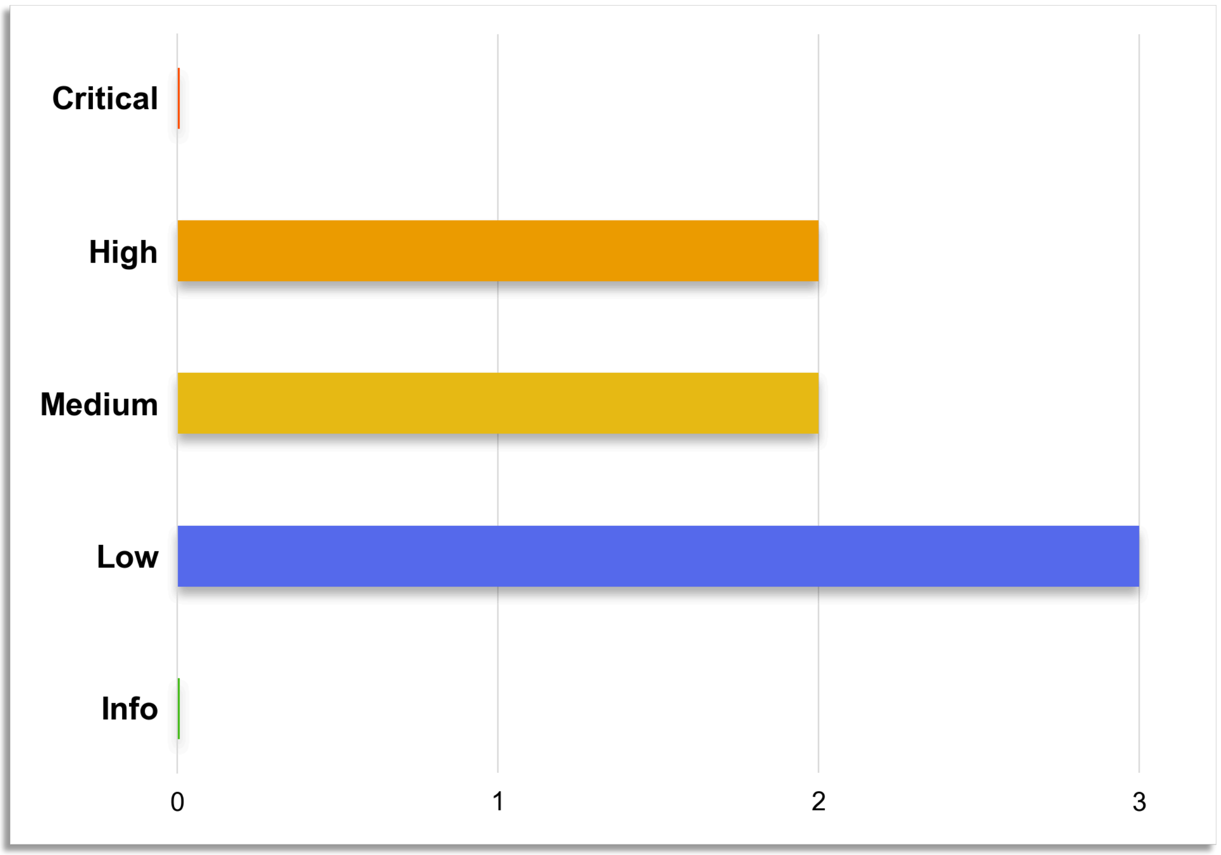

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description	Status
FYEO-Endless-01	High	Authentication bypassed while default admin	Remediated
FYEO-Endless-02	High	The verify_message function accepts empty signers as valid	Remediated
FYEO-Endless-03	Medium	Signature recovery concern	Remediated
FYEO-Endless-04	Medium	Validator limits and threshold	Acknowledged
FYEO-Endless-05	Low	Deduplication on 32 bytes instead of address	Remediated
FYEO-Endless-06	Low	Excess amount gets stuck	Remediated
FYEO-Endless-07	Low	Missing zero checks, one step admin transfer	Acknowledged

Table 2: Findings Overview

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

This Endless Bridge implements on-chain liquidity pools for cross-chain token transfers both in Move and the EVM. Internally, each token has its own Pool struct tracking total staked liquidity, per-provider balances, accumulated rewards, and timestamps for APY calculations. Liquidity providers can add or remove their stake, earn protocol fees pro-rata, and withdraw both their principal and accrued rewards; all arithmetic uses fixed-point operations to maintain precision without overflow.

As part of the broader “Endless Bridge” flow, the Pools sit downstream of the Fund Manager: once the off-chain Bridge node signs and the Relay calls collect, funds flow into the Pool module. There they’re recorded, rewards are refreshed based on incoming fees, and providers’ shares are updated. This design cleanly separates cross-chain message handling from liquidity accounting and reward distribution, enabling secure tracking of user funds and protocol earnings.

Both frameworks rely on a global OwnerConf.admin or onlyMaster pattern. Yet the current initialization and transfer logic are inconsistent (e.g. single-step ownership and no zero-address checks), leading to repeated access control vulnerabilities. A unified, multi-step transfer template should be codified across all modules.

Key actions (pool updates, refunds, collects, threshold changes) sometimes lack corresponding events, making off-chain indexers and explorers unaware of state transitions. Full event coverage - especially for deprecations, threshold updates, and fee collections - would improve transparency and debuggability.

A notable gap is the absence of end-to-end integration tests that exercise the complete cross-chain workflow - from message creation on one chain, through signature aggregation, to fund deposition and reward distribution on the other. Unit tests validate individual modules in isolation, but without orchestrated scenarios that involve both Move and Solidity components together (or multi-node setups), it’s easy for subtle mismatches - like fee-accounting discrepancies or nonce-mismatch behaviors - to slip through. Integration tests would catch issues that only emerge at the system level, inconsistent event handling, or improper state transitions when moving funds and messages between chains. These tests should include negative tests as well as fuzz testing to complete the program.

Authentication bypassed while default admin

Finding ID: FYEO-Endless-01
Severity: High
Status: Remediated

Description

Both the bridge_token::config and bridge_core::validator modules initialize their OwnerConf.admin to the module's own address (e.g. @bridge_token, @bridge_core), and then use an || chain in verify_admin that always passes when owner_conf.admin equals the module address. This effectively grants administrative rights to any caller until the admin is transferred, completely bypassing intended access controls.

Proof of Issue

File name: endless/bridge-token/sources/config.move
Line number: 148

```
move_to(account, OwnerConf { admin: @bridge_token });

...

inline fun verify_admin(admin: &signer, owner_conf: &OwnerConf) {
    assert!(
        owner_conf.admin == @bridge_token
            || owner_conf.admin == signer::address_of(admin)
            || signer::address_of(admin) == @bridge_token,
        0
    );
}
```

File name: endless/bridge-core/sources/validator.move
Line number: 124

```
fun init_module(account: &signer) {
    ...
    move_to(account, OwnerConf { admin: @bridge_core });
}

inline fun verify_admin(admin: &signer, owner_conf: &OwnerConf) {
    assert!(
        owner_conf.admin == @bridge_core
            || owner_conf.admin == signer::address_of(admin)
            || signer::address_of(admin) == @bridge_core,
        0
    );
}
```

Severity and Impact Summary

Complete bypass of access control on all administrative functions guarded by verify_admin in both modules.

Until an admin transfer occurs, any user (or arbitrary script) can call onlyAdmin or role_check–protected entrypoints.

Recommendation

Redesign verify_admin to require exact signer match.

The verify_message function accepts empty signers as valid

Finding ID: FYEO-Endless-02
Severity: High
Status: Remediated

Description

The verify_message function treats an empty multisig vector as valid, skipping all signature verification. In normal (non-estimate) flows there is no guard preventing multisig from being empty, so an attacker can confirm any message without providing any signatures completely defeating the multisignature security model.

Proof of Issue

File name: endless/bridge-core/sources/message.move
Line number: 352

```
public fun verify_message(
    multisig: vector<u8>,
    accum_pk: vector<PublicKeyWithPoP>,
    msg_header: vector<u8>,
    msg_body: vector<u8>
): bool {
    // Verify the multi-signature
    let message = reverse_header(msg_header);
    vector::append(&mut message, msg_body);
    let message = endless_hash::keccak256(message);

    if (!vector::is_empty(&multisig)) { // estimate_gas
        ...
    };
    true
}
```

This might related to gas estimation, but these only change these two entry points do is an additional assert.

```
public entry fun bridge_finish(
    sender: &signer,
    msg_header: vector<u8>,
    msg_body: vector<u8>,
    multisig: vector<u8>,
    pks: vector<u64>
) acquires ExecuteResource {
    internal_bridge_finish(
        sender,
        msg_header,
        msg_body,
        multisig,
        pks,
        false
    );
}
```

```
public entry fun bridge_finish_estimate_gas(
    sender: &signer,
    msg_header: vector<u8>,
    msg_body: vector<u8>,
    multisig: vector<u8>,
    pks: vector<u64>
) acquires ExecuteResource {
    internal_bridge_finish(
        ...
        true
    );
}
```

The additional assert - this does not check if multisig is empty for non simulation mode.

```
if (is_estimate) {
    assert!(
        signer::address_of(sender) == @bridge_token, EREVERT_FOR_GAS_ESTIMATION
    );
};
```

Severity and Impact Summary

Unauthenticated message confirmation as anyone can submit empty multisig and have messages confirmed without any cryptographic proof.

Recommendation

Require a non-empty multisig for production confirmations and separate out “gas-estimation” paths explicitly.

Signature recovery concern

Finding ID: FYEO-Endless-03
Severity: Medium
Status: Remediated

Description

The recovery ID (v) byte in a signature is interpreted in a way that allows any value other than 27 to default to 1, regardless of validity. This results in potentially accepting malformed or malicious signatures, which can lead to incorrect public key recovery or signature verification bypasses.

Proof of Issue

File name: endless/bridge-token/sources/fund_manage.move
Line number: 157

```
let recovery_id =
    if (*vector::borrow(&signature_bytes, 64) == 27) { 0 }
    else { 1 };
```

This logic assumes only v = 27 and defaults all other values to recovery_id = 1, including invalid ones like 0, 2, 37, etc., without rejecting them.

Severity and Impact Summary

This can lead to accepting malformed or non Ethereum standard signatures, potentially breaking security guarantees of the signature scheme. Specifically, an attacker could craft a signature with a non-standard v byte that still passes validation, which might result in recovering an incorrect public key or verifying a signature that should be rejected. This weakens the signature verification process and could lead to signature forgery or unauthorized access in critical transaction paths.

Recommendation

Validate that the recovery ID (v) is strictly 27 or 28. Reject any signature with a v byte outside these values to prevent malformed inputs.

Validator limits and threshold

Finding ID: FYEO-Endless-04
Severity: Medium
Status: Acknowledged

Description

All validators can be removed from the validator set, and the verification threshold will be automatically reduced to zero. This effectively disables the validator approval mechanism, allowing potentially unauthorized operations to proceed without any validator signatures. This undermines the core security assumptions of any multisig or consensus-based operation relying on the validator set. Furthermore, impossible thresholds can be set.

Proof of Issue

File name: endless/bridge-core/sources/validator.move
Line number: 93

```
public entry fun remove_validator(
    admin: &signer, pk_with_pop: vector<u8>
) acquires OwnerConf, Validator {
    // check admin permission
    let owner_conf = borrow_global<OwnerConf>(@bridge_core);
    verify_admin(admin, owner_conf);

    let node = borrow_global_mut<Validator>(@bridge_core);
    let validators = &mut node.validators;
    let pk_pop = public_key_with_pop_from_bytes(pk_with_pop);

    if (vector::contains(validators, &pk_pop)) {
        vector::remove_value(validators, &pk_pop);
    };
    node.threshold = (min((node.threshold as u64), vector::length(validators)) as u8);
}
```

This function allows the admin to remove any validator from the set. The threshold is then adjusted downwards to the smaller of the current threshold and the number of remaining validators. However, there is no check to enforce that at least one validator remains, or that the threshold stays above zero. Therefore, the following scenario is possible:

- Admin calls remove_validator() repeatedly until all validators are removed.
- The validator set becomes empty.
- The threshold becomes 0, meaning 0 signatures are required for consensus.

```
public entry fun threshold_change(admin: &signer, threshold: u8) acquires OwnerConf,
Validator {
    let owner_conf = borrow_global<OwnerConf>(@bridge_core);
    verify_admin(admin, owner_conf);

    let node = borrow_global_mut<Validator>(@bridge_core);
    assert!(
        node.threshold > 0
```

```
            && (node.threshold as u64) <= vector::length(&node.validators),
        ETHRESHOLD
    );
    node.threshold = threshold;
}
```

This function checks the current configuration instead of the new value to be set. So if the threshold was reduced to 0 this function can never be called.

```
function batch_delete_validators(
    bytes32[] memory signer_pk
) public onlyAdmin returns (uint) {
    // Iterates through the signer_pk array and removes each public key from the
validators set
    for (uint i = 0; i < signer_pk.length; i++) {
        validators.remove(signer_pk[i]);
    }

    // Gets the current length of the validator set
    uint threshold = validators.length();
    // Sets the new minimum verification threshold to the smaller value between the
current minimum verification threshold and the length of the validator set
    min_verify_threshold = Math.min(min_verify_threshold, threshold);
    return min_verify_threshold;
}
```

The above function similarly does not enforce that any validators remain in the set after deletion, or that the threshold remains above zero.

```
function batch_add_validators(
    bytes32[] memory signer_pk,
    uint threshold
) public onlyAdmin {
    // Iterates through the signer_pk array and adds each public key to the validators
set
    for (uint i = 0; i < signer_pk.length; i++) {
        validators.add(signer_pk[i]);
    }

    // Checks if the threshold is greater than 255. If so, it reverts with an error
message.
    if (threshold > 255) {
        revert(
            "The minimum verification threshold cannot be greater than 255"
        );
    }

...

function set_min_verify_threshold(uint threshold) public onlyAdmin {
    // Checks if the threshold is greater than 255. If so, it reverts with an error
message.
    if (threshold > 255) {
        revert(
            "The minimum verification threshold cannot be greater than 255"
        );
    }
    // Sets the minimum verification threshold
    min_verify_threshold = threshold;
}
```

These additional functions allow an admin to manually set the threshold but only validate that it is not too high—there is no lower bound enforced. An admin could explicitly set the threshold to 0. Or such that the threshold can never be met.

Severity and Impact Summary

This issue can completely disable the validator approval mechanism, allowing any admin to push through validatorless operations by setting the threshold to zero and removing all validators. In systems relying on validator consensus, this would mean that no validator approvals are required for critical operations—effectively removing all security from these processes and enabling arbitrary, unapproved actions.

Recommendation

Establish lower bounds for validator requirements and make sure the threshold can always be met.

Deduplication on 32 bytes instead of address

Finding ID: FYEO-Endless-05
Severity: Low
Status: Remediated

Description

The batch_add_validators and signature-verification logic allow arbitrarily many duplicate entries in the input array before de-duplication, leading to wasted gas (and potentially DoS via out-of-gas) when the same 32-byte public key is repeated up to 2^96−1 times. Although the set ultimately stores unique values, every duplicate still incurs an add call's gas cost, and during signature verification the full loop pays for each duplicate before skipping it.

Proof of Issue

File name: evm/core/Validator.sol
Line number: 37

```
EnumerableSet.Bytes32Set private validators;
...
function batch_add_validators(
    bytes32[] memory signer_pk,
    uint threshold
) public onlyAdmin {
    // Iterates through the signer_pk array and adds each public key to the validators
set
    for (uint i = 0; i < signer_pk.length; i++) {
        validators.add(signer_pk[i]);
    }
(address[] memory validators, uint min_verify_threshold) = IValidator(
    ValidatorAddr
).fetch_all_validators();
if (signature.length < min_verify_threshold) {
    emit Types.Log("require min verify node", min_verify_threshold);
    return (false, msgDec);
}

uint256 validSignerCount = 0;
bool[] memory uniqueSigners = new bool[](validators.length);
for (uint i = 0; i < signature.length; i++) {
    address msg_signer = ECDSA.recover(msgHash, signature[i]);

    if (!isValidator(msg_signer, validators)) {
        revert("Not a valid signer node");
    }

    uint256 signerIndex = getValidatorIndex(msg_signer, validators);
    if (!uniqueSigners[signerIndex]) {
        uniqueSigners[signerIndex] = true;
        validSignerCount++;
    }
}
```

Every duplicate signature entry incurs ECDSA recovery and isValidator lookups before the boolean array deduplicates.

Severity and Impact Summary

Gas exhaustion & DoS risk: An attacker (or a misconfigured script) supplying enormous arrays of the same key/signature can burn through block gas limits, preventing legitimate transactions from succeeding. Unnecessary cost for honest users, as even benign duplicates (due to front-end bugs, …) inflate transaction costs.

Recommendation

Make this an AddressSet instead.

Excess amount gets stuck

Finding ID: FYEO-Endless-06
Severity: Low
Status: Remediated

Description

The transfer function accepts Ether when transferring ETH to multiple recipients. It checks whether the msg.value is at least the total amount to distribute, but it does not handle the case where more ETH than required is sent. Any excess Ether sent remains stuck in the contract.

Proof of Issue

File name: evm/comn/TokenBatch.sol
Line number: 33

```
if (tokenAddr == address(0)) {
    require(msg.value >= total, "Value less");
```

Severity and Impact Summary

This issue can lead to unintentional loss of funds for users who overpay.

Recommendation

Implement a refund mechanism for excess Ether or consider rejecting overpayments outright.

Missing zero checks, one step admin transfer

Finding ID: FYEO-Endless-07
Severity: Low
Status: Acknowledged

Description

The Solidity administrative role transfer functions do not validate input addresses, allowing them to be set to the zero address (0x0). This includes critical roles like master, admin, and financer. Similarly, the Move implementation lacks a two step process. Setting administrative roles to an unowned or non-signable address can result in permanent loss of control over contract functionality.

Proof of Issue

File name: evm/core/Admin.sol
Line number: 49

```
function setMaster(address addr) public onlyMaster {
    ...
    master = addr;
}

function setAdmin(address addr) public onlyMaster {
    ...
    admin = addr;
}

function setFinancer(address addr) public onlyMaster {
    ...
    financer = addr;
}
public entry fun transfer_ownership(admin: &signer, new_admin: address) acquires
OwnerConf {
    let owner_conf = borrow_global_mut<OwnerConf>(@bridge_token);
    verify_admin(admin, owner_conf);

    owner_conf.admin = new_admin;
}
```

Severity and Impact Summary

Setting critical administrative roles to the zero address can result in irreversible privilege loss, making it impossible to perform upgrades, manage validators, or perform other essential administrative tasks. This creates a serious availability and governance risk.

Recommendation

Enforce zero address checks in all administrative setter functions. For critical roles like master, implement a two-step ownership transfer mechanism: - Current master proposes a new address - Proposed new master explicitly accepts the role

This ensures the new owner can sign transactions and mitigates the risk of accidentally setting the role to an unusable address.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

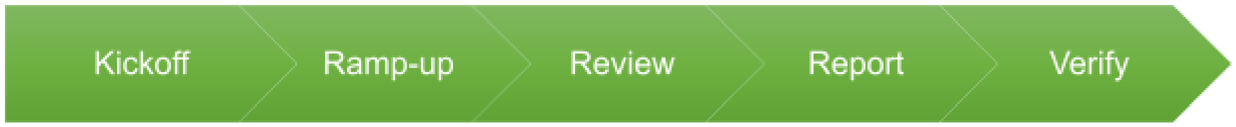

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.