FYEO

Security Code Review
Levana Governance

Levana

May 2024
Version 1.0

Presented by:

FYEO Inc.

PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

TABLE OF CONTENTS

Executive Summary

Overview

Levana engaged FYEO Inc. to perform a Security Code Review Levana Governance.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on April 26 - May 03, 2024, 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-LEVANA-01 – Insufficient input checks

- FYEO-LEVANA-02 – TimeData is susceptible to ordering issues

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 Levana Governance. 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 private repository at
https://github.com/Levana-Protocol/levana-governance with the commit hash
d878f2713a0a58112de150cb207729fee29164b4.

Files included in the code review
| ```
levana-governance/
└── contract/
    ├── examples/
    │   └── schema.rs
    └── src/
        ├── state/
        │   ├── account.rs
        │   ├── distribution.rs
        │   ├── killswitch.rs
        │   ├── map_ext.rs
        │   ├── mod.rs
        │   ├── monotonic_id.rs
        │   ├── proposal.rs
        │   ├── time_data.rs
        │   ├── timelock.rs
        │   └── totals.rs
        ├── lib.rs
        ├── range.rs
        ├── tests.rs
        └── time.rs
``` |

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review Levana Governance, we discovered:

- 2 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
FYEO-LEVANA-01	**Low**	Insufficient input checks
FYEO-LEVANA-02	**Low**	TimeData is susceptible to ordering issues

Table 2: Findings Overview

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

The review of the Levana Governance program reveals a well-structured codebase, implemented with an emphasis on security practices. The presence of numerous tests is excellent, extending them with random amounts could further bolster resilience. It is noteworthy that the team promptly addressed all concerns raised during the review process. The absence of major issues underscores the solidity of the program, instilling confidence in its reliability and security.

Insufficient input checks

Finding ID: FYEO-LEVANA-01
Severity: **Low**
Status: **Remediated**

Description

Inputs are not properly checked against 0 amounts and other values.

Proof of Issue

File name: contract/src/state/timelock.rs
Line number: 52

```
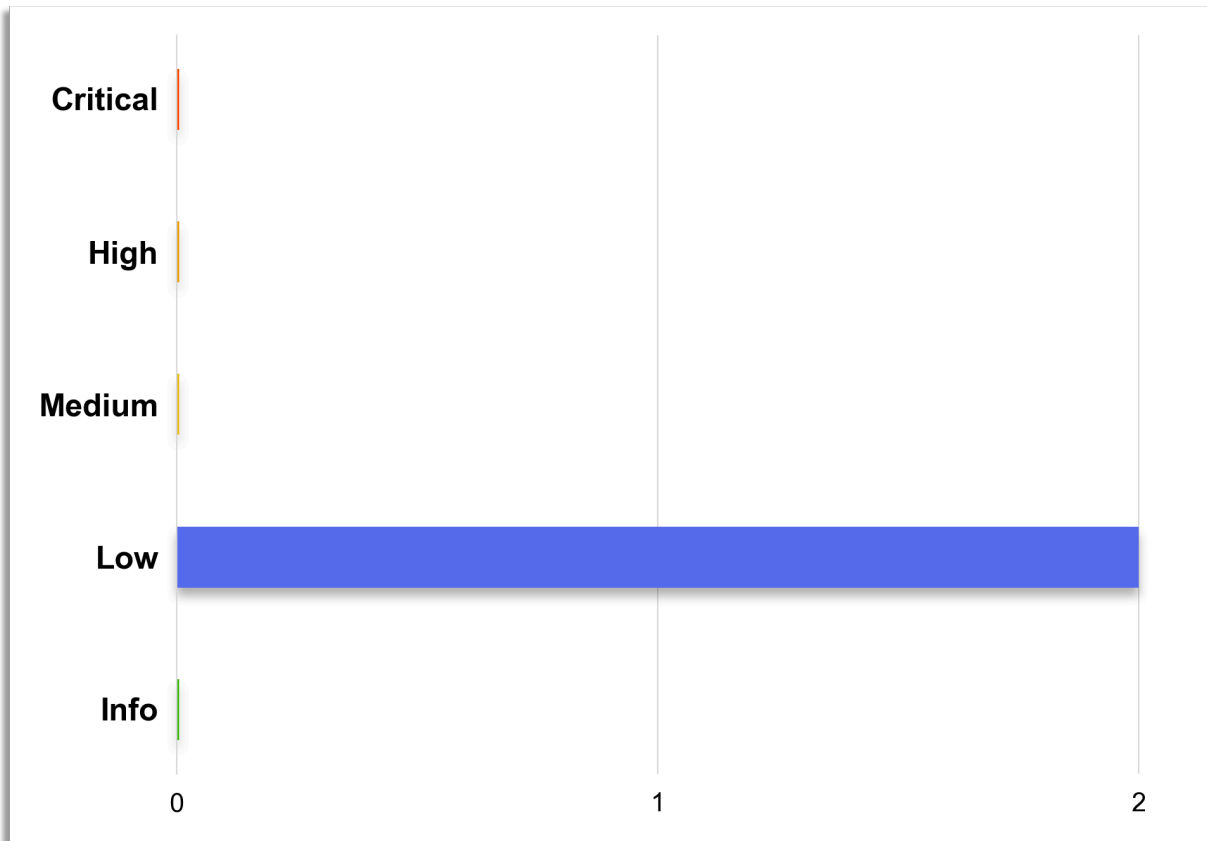let bucket = Bucket {
    multiplier: bucket.multiplier,
    duration,
};
```

Buckets can have 0 multipliers.

File name: contract/src/lib.rs
Line number: 492

```
let id = proposal::new(
    &mut ctx,
    &config,
    title,
    description,
    duration,
    resolution.into_inner(deps.api)?,
    info.sender.clone(),
    deposit,
)?;
```

ProposalResolution can have 0 amounts.

File name: contract/src/state/mod.rs
Line number: 124

```
pub fn validate(&self) -> StdResult<()> {
    if self.constraints.duration_range.is_invalid() {
        return Err(StdError::generic_err(
            "Minimum proposal duration cannot be bigger than the maximum.",
        ));
    }

    if self.title_range.is_invalid() {
        return Err(StdError::generic_err(
            "Minimum title length cannot be bigger than the maximum.",
        ));
    }

    if self.constraints.description_range.is_invalid() {
```

```
        return Err(StdError::generic_err(
            "Minimum proposal description length cannot be bigger than the maximum.",
        ));
    }

    if self.constraints.text_range.is_invalid() {
        return Err(StdError::generic_err(
            "Minimum proposal text length cannot be bigger than the maximum.",
        ));
    }

    if self.constraints.quorum > HUNDRED_PERCENT {
        return Err(StdError::generic_err("Quorum cannot exceed 100%."));
    }

    if self.constraints.threshold > HUNDRED_PERCENT {
        return Err(StdError::generic_err("Threshold cannot exceed 100%."));
    }

    Ok(())
}
```

The ranges, quorum, threshold, unstaking_period, execution_window and min_deposit can be 0.

File name: contract/src/lib.rs
Line number: 576

```
let id = distribution::new_vesting_unchecked(
    &mut ctx,
    &config,
    amount,
    VestingOptions::new(can_vote, can_receive_rewards),
    duration.into(),
    &title,
)?;
```

The duration can be 0.

File name: contract/src/lib.rs
Line number: 588

```
let config = CONFIG.load(ctx.storage)?;
distribution::release(&mut ctx, &config, limit)?;
```

ReleaseDistributions can be called with limit 0.

File name: contract/src/state/account.rs
Line number: 341

```
pub fn begin_unstake(
    &mut self,
    ctx: &mut Context,
    config: &Config,
    amount: Uint128,
) -> Result<TimeAction> {
    let lvn = Self::LVN
        .latest(ctx.storage, self.addr)?
        .unwrap_or_default();
```

```
    if lvn.staked < amount {
        return Err(Error::InsufficientAmount {
            requested: amount,
            available: lvn.staked,
        });
    }

    self.update_lvn(ctx.storage, |lvn| {
        lvn.staked -= amount;
        lvn.unstaking += amount;
    })?;
```

Will accept a 0 amount and store on-chain data.

File name: contract/src/state/account.rs
Line number: 378

```
pub fn begin_vesting(
    &mut self,
    ctx: &mut Context,
    duration: Duration,
    amount: Uint128,
    clawback_addr: Option<Addr>,
    title: String,
    options: VestingOptions,
) -> Result<TimeAction> {
    killswitch::assert_is_enabled(ctx.storage, Operation::Vest)?;

    let end = self.at + duration;
```

Will accept a 0 vesting duration.

Severity and Impact Summary

While it may not be likely that these values will be misconfigured, accidents may occur and some may lead to DoS situations and prevent the program from working properly. For example, having a 0 execution window makes it impossible to execute proposals.

Recommendation

Check all user input to avoid unintended behavior.

TimeData is susceptible to ordering issues

Finding ID: FYEO-LEVANA-02
Severity: **Low**
Status: **Remediated**

Description

Some data relies on the TimeData concept to store data with historic values.

Proof of Issue

```
gov.create_proposal(
    USER_B,
    MIN_PROPOSAL_DEPOSIT,
    "Proposal title",
    PROPOSAL_DURATION_RANGE.min,
    ProposalResolution::Text {
        desc: String::from("proposal description"),
    },
)
.unwrap();

let id = ProposalId::new(0);
gov.vote(USER_A, id, VoteOption::No).unwrap();
gov.vote(USER_B, id, VoteOption::No).unwrap();

gov.stake(USER_A, stake_amount).unwrap();
gov.vote(USER_A, id, VoteOption::Yes).unwrap();
```

If a user casts a vote on some proposal, updates their power and changes their vote all in the same block as the propoal is created, the value used for subtracting the old vote will be based on their new power. I.e. in this case user A votes 100 for No, then changes their power to 200. This would then subtract 200 from No and add 200 to Yes. Therefore user B's vote will be 'erased'. Note that if user B had not voted No, this would cause an underflow instead.

Severity and Impact Summary

Depending on the order in which transactions are processed within a block, it might lead to invalid results. Note that block creators could craft the block in whatever way they please.

Recommendation

Make sure to address timing issues for transactions processed in the same block.

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.

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