F Y E O

Security Code Review of Banger

Banger

October 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

Banger engaged FYEO Inc. to perform a Security Code Review.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on September 18 - September 24, 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-BANGER-01 – Calculation error in `sell` function.
- FYEO-BANGER-02 – The treasury account can be identical to others
- FYEO-BANGER-03 – Users can choose rarity
- FYEO-BANGER-04 – Co-signer of claim transaction must be manually verified
- FYEO-BANGER-05 – Init pool implementation not as documented
- FYEO-BANGER-06 – InitPool takes any curve account
- FYEO-BANGER-07 – Program curve is initialized on a first come first served basis
- FYEO-BANGER-08 – Unchecked Subtraction in sell Function Leading to Potential Overflows/Underflows
- FYEO-BANGER-09 – Code clarity

- FYEO-BANGER-10 – Constants are declared but not used
- FYEO-BANGER-11 – Improve test cases
- FYEO-BANGER-12 – Manual account size calculation
- FYEO-BANGER-13 – No bound checks on config values such as fees
- FYEO-BANGER-14 – Rent can not be recovered

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 of Banger. 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/BangerLol/banger_lol with the commit hash fddd03819bbce2c18c27b03d3b37397b1fddb172. Remediations were provided with the commit hash 49560795d2f1b9909a4e9b2d42074cd4ae57defe.

Files included in the code review

```
banger_lol/
└── programs/
    └── banger-program/
        └── src/
            ├── instructions/
            │   ├── buy.rs
            │   ├── claim.rs
            │   ├── claim_collector_rewards.rs
            │   ├── init_curve.rs
            │   ├── init_pool.rs
            │   ├── mod.rs
            │   └── sell.rs
            ├── constants.rs
            ├── errors.rs
            ├── lib.rs
            └── state.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Banger, we discovered:

- 3 findings with HIGH severity rating.
- 5 findings with LOW severity rating.
- 6 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

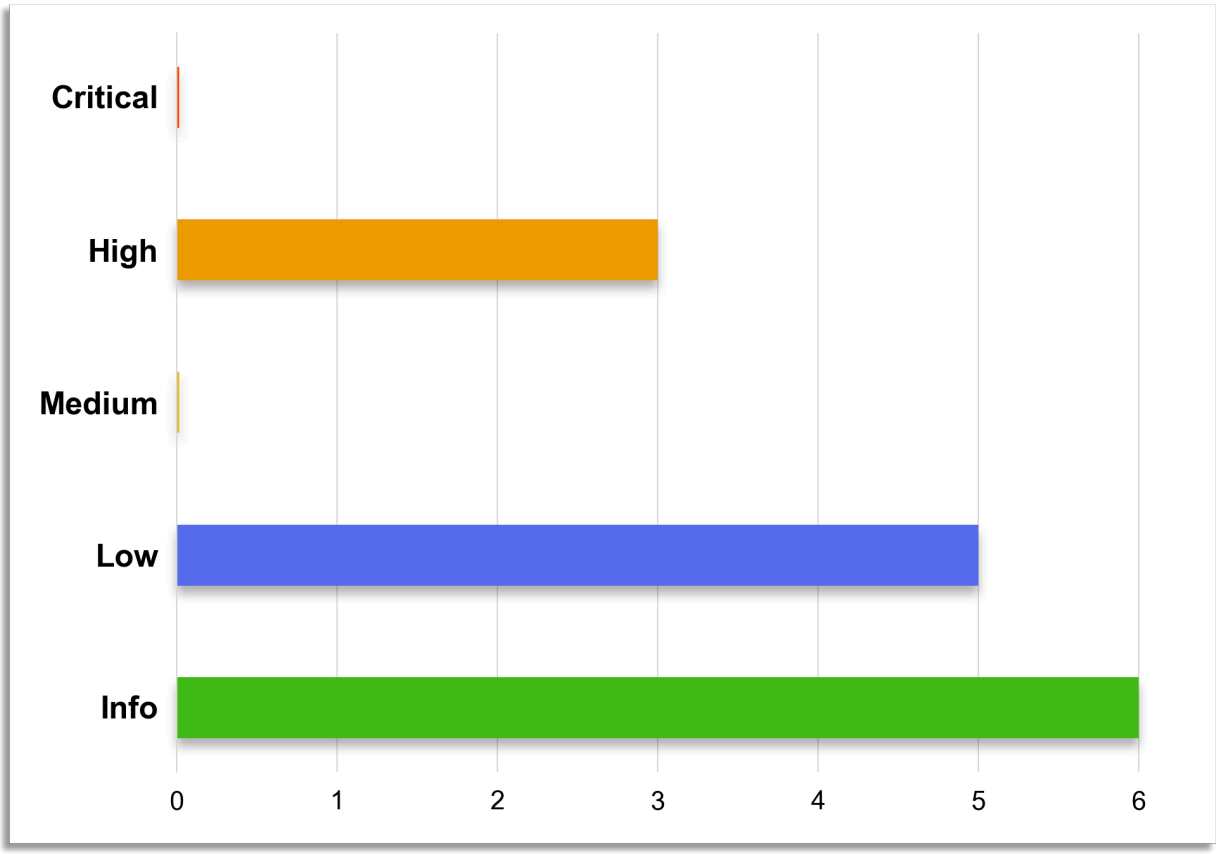

Figure 1: Findings by Severity

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-BANGER-01	**High**	Calculation error in `sell` function.
FYEO-BANGER-02	**High**	The treasury account can be identical to others
FYEO-BANGER-03	**High**	Users can choose rarity
FYEO-BANGER-04	**Low**	Co-signer of claim transaction must be manually verified
FYEO-BANGER-05	**Low**	Init pool implementation not as documented
FYEO-BANGER-06	**Low**	InitPool takes any curve account
FYEO-BANGER-07	**Low**	Program curve is initialized on a first come first served basis
FYEO-BANGER-08	**Low**	Unchecked Subtraction in sell Function Leading to Potential Overflows/Underflows
FYEO-BANGER-09	**Informational**	Code clarity
FYEO-BANGER-10	**Informational**	Constants are declared but not used
FYEO-BANGER-11	**Informational**	Improve test cases
FYEO-BANGER-12	**Informational**	Manual account size calculation
FYEO-BANGER-13	**Informational**	No bound checks on config values such as fees
FYEO-BANGER-14	**Informational**	Rent can not be recovered

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 program provides a secure and efficient way to tokenize tweets, allowing users to engage in buying and selling these tokens while ensuring that creators receive appropriate compensation. It handles token minting and burning, fee calculations, and secure fund transfers, all governed by a bonding curve to manage token pricing dynamically.

In general the code base is well-structured, demonstrating a clear and organized approach to development and the code is generally easy to follow.

- The project is well structured and has good and comprehensive test cases even if the test cases do not cover 100% of functionality.
    - Here we recommend improving the test cases to cover all of the contracts functionality.
- The use of Anchor for critical functions makes the code easy to audit and in general improves the security.
- The team has shown excellent communication throughout the review process, which has facilitated a smooth and efficient collaboration.

Calculation error in `sell` function.

Finding ID: FYEO-BANGER-01
Severity: **High**
Status: **Remediated**

Description

The sell function calculates the `subtotal` incorrectly from using the `banger_fee` twice instead of the banger_fee and the creator_fee as intended.

Proof of Issue

File name: /instructions/sell.rs
Line number: 123

```let banger_fee = total .checked_mul(self.pool.banger_fee as u64).ok_or(CurveError::Overflow)? .checked_div(10000).ok_or(CurveError::Overflow)?;
```

```
    let creator_fee = total
        .checked_mul(self.pool.creator_fee as u64).ok_or(CurveError::Overflow)?
        .checked_div(10000).ok_or(CurveError::Overflow)?;

    let subtotal = total
        .checked_sub(banger_fee).ok_or(CurveError::Overflow)?
        .checked_sub(banger_fee).ok_or(CurveError::Overflow)?;
```

````

## Severity and Impact Summary

This calculation will result in an unexpected outcome.

## Recommendation

Correct the calculation.
````

The treasury account can be identical to others

Finding ID: FYEO-BANGER-02
Severity: **High**
Status: **Remediated**

Description

The `treasury` account is not checked to be any particular account.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 46

```
pub treasury: SystemAccount<'info>,
```

The account key could for instance be identical to either of these accounts:

```
#[account(
    mut,
    seeds = [REWARDS_POOL_DATA.as_bytes(), collection.key().as_ref()],
    bump
)]
pub rewards_pool: SystemAccount<'info>,

#[account(
    mut,
    seeds = [CREATOR_VAULT.as_bytes(), creator_id.as_bytes()],
    bump
)]
pub creator_vault: SystemAccount<'info>,
```

Severity and Impact Summary

There is no check on what the treasury is. This means any account will do.

Recommendation

Make sure to check this implementation against the requirements. It seems more likely that the treasury is supposed to be an account owned by an admin or similar.

Users can choose rarity

Finding ID: FYEO-BANGER-03
Severity: **High**
Status: **Remediated**

Description

The implementation does not check if the given account belongs to the switchboard program. Therefore any account that has the correct length and can be de-serialized to match the expected format can be used.

Proof of Issue

File name: programs/banger-program/src/instructions/buy.rs
Line number: 80

```
pub randomness_account: UncheckedAccount<'info>,

...

let randomness_data =
    RandomnessAccountData::parse(ctx.accounts.randomness_account.data.borrow())
        .map_err(|_| RandomnessError::RandomnessAccountParsingFailed)?;
```

Severity and Impact Summary

The rarity is determined by the random data. The data is however not random. Therefore users can choose the rarity as they please.

Recommendation

Check this account is in fact owned by the switchboard program.

Co-signer of claim transaction must be manually verified

Finding ID: FYEO-BANGER-04
Severity: **Low**
Status: **Acknowledged**

Description

The claim functions are co-signed by `creator` / `collector`. These accounts do not have relations to any other accounts and must be manually verified by the admin. Essentially any account can co-sign these instructions. The `creator` or `collector` accounts can also be identical to the admin account.

Proof of Issue

File name: programs/banger-program/src/instructions/claim.rs
Line number: 32

```
#[account(mut)]
pub creator: Signer<'info>,

...

#[account(
    mut,
    seeds = [b"creator_vault", creator_id.as_bytes()],
    bump
)]
pub creator_vault: SystemAccount<'info>,
```

This also applies to `ClaimCollectorRewards`.

Severity and Impact Summary

Any account can co-sign.

Recommendation

Make sure this implementation is correct.

Init pool implementation not as documented

Finding ID: FYEO-BANGER-05
Severity: **Low**
Status: **Remediated**

Description

The documentation indicates that the `InitPool` function should only be run by an authorized user. This has not been implemented however.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 17

```
// todo: once banger team gives a constant admin address, it will be added as
constraint
#[account(mut)]
pub admin: Signer<'info>,
```

Severity and Impact Summary

This is currently a public function despite being indicated as an admin account.

Recommendation

Review this code and make sure to implement it according to requirements.

InitPool takes any curve account

Finding ID: FYEO-BANGER-06
Severity: **Low**
Status: **Remediated**

Description

In `InitPool` any curve can be given while there currently is only one possible curve due to the PDA setup.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 28

```
pub curve: Account<'info, Curve>,
```

Severity and Impact Summary

This checks that the account is of the right type and owned by the program. Currently there can be only one curve account, but this is also stored in the pool accounts as if it could change. The intention here remains a bit unclear.

Recommendation

Make sure this code is implemented as required.

Program curve is initialized on a first come first served basis

Finding ID: FYEO-BANGER-07
Severity: **Low**
Status: **Remediated**

Description

The initialization is done using a public function. There is a risk someone else can call it first.

Proof of Issue

File name: programs/banger-program/src/instructions/init_curve.rs
Line number: 5

```
pub struct InitCurve<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    #[account(
        init,
        payer = admin,
        space = Curve::INIT_SPACE,
        seeds = [CURVE.as_bytes()],
        bump
    )]
    pub curve: Account<'info, Curve>,

    pub system_program: Program<'info, System>,
}
```

Severity and Impact Summary

This is a public function.

Recommendation

For upgradeable programs, it is recommended to use the following pattern to initialize the program.

```
pub owner: Signer<'info>,

#[account(constraint = program.programdata_address()? == Some(program_data.key()))]
pub program: Program<'info, MyProgram>,

#[account(constraint = program_data.upgrade_authority_address == Some(owner.key()))]
pub program_data: Account<'info, ProgramData>,
```

Unchecked Subtraction in sell Function Leading to Potential Overflows/Underflows

Finding ID: FYEO-BANGER-08
Severity: **Low**
Status: **Remediated**

Description

The subtraction current_supply - (i + 1) is not using checked arithmetic, potentially leading to underflows. If current_supply is less than i + 1, this subtraction will underflow, causing unexpected behavior or panics.

Proof of Issue

```
let supply = current_supply - (i + 1);
```

Severity and Impact Summary

Calculations may panic.

Recommendation

Use Checked Arithmetic Ensure that num_burn does not exceed current_supply. Verify that the seller has enough tokens to burn.

Code clarity

Finding ID: FYEO-BANGER-09
Severity: **Informational**
Status: **Remediated**

Description

There is a bit of duplicate and inefficient code.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 91, 110

```
if creator_vault_lamports < rent_exempt_lamports {
    let rent_exempt_accounts = Transfer {
        from: self.admin.to_account_info(),
        to: self.creator_vault.to_account_info(),
    };
    let cpi_ctx =
        CpiContext::new(self.system_program.to_account_info(), rent_exempt_accounts);
    transfer(cpi_ctx, rent_exempt_lamports - creator_vault_lamports)?;
}
```

This code fragment is duplicated.

File name: programs/banger-program/src/instructions/buy.rs
Line number: 107

```
for cur_num_mint in 0..num_mint {
    require!(cur_num_mint < RANDOM_MAX_MINT, ClaimError::ClaimMoreThan32);
```

This can be checked outside the loop.

There are various duplicated account checks. For instance:

```
#[account(
    mut,
    constraint = treasury.key() == pool.treasury,
)]
pub treasury: SystemAccount<'info>,

...

#[account(
    mut,
    seeds = [POOL_DATA.as_bytes(), collection.key().as_ref()],
    bump = pool.pool_bump,
    has_one = curve,
    has_one = treasury
)]
pub pool: Box<Account<'info, Pool>>,

#[account(
    mut,
```

```
    constraint = admin.key() == pool.admin @ PoolError::WrongAdminPassed,
)]
pub admin: Signer<'info>,

...

#[account(
    mut,
    seeds = [POOL_DATA.as_bytes(), collection.key().as_ref()],
    bump = pool.pool_bump,
    has_one = admin,
)]
pub pool: Box<Account<'info, Pool>>,
```

Severity and Impact Summary

No impact on security.

Recommendation

Keep the code base clean and concise to aid maintainability.

Constants are declared but not used

Finding ID: FYEO-BANGER-10
Severity: **Informational**
Status: **Remediated**

Description

There are constants declared for several of the PDA seeds but they are not consistently being used.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 77

```
let seeds = &[&b"authority"[..], &[bumps.authority]];

pub const AUTHORITY: &str = "authority";
```

Also for:

```
seeds = [b"creator_vault", creator_id.as_bytes()],

&b"creator_vault"[..]

pub const CREATOR_VAULT: &str = "creator_vault";
```

And for:

```
&b"reward"[..],

pub const REWARDS_POOL_DATA: &str = "reward";
```

Severity and Impact Summary

This has no direct security implications but it does impact the maintainability of the codebase.

Recommendation

Use the constants so the maintainability of the codebase can be improved.

Improve test cases

Finding ID: FYEO-BANGER-11
Severity: **Informational**
Status: **Acknowledged**

Description

The test suite may not cover all scenarios, such as concurrent transactions, maximum and minimum values for inputs, and potential overflows or underflows.

Proof of Issue

tests/*

Severity and Impact Summary

Thorough testing is recommended to avoid potential issues.

Recommendation

Expand your test cases to include:

- Simulations of concurrent buys and sells.
- Testing with extreme values for inputs to check for overflows/underflows.
- Edge cases where transactions should fail due to validations.

Example Test Scenario:

- Attempt to buy tokens with max_lamports_in slightly less than the required amount to ensure the slippage protection works correctly.

The asset transfers asset_amount is not checked to be any specific value.

Manual account size calculation

Finding ID: FYEO-BANGER-12
Severity: **Informational**
Status: **Acknowledged**

Description

The account size is calculated in various manners but always manually.

Proof of Issue

File name: programs/banger-program/src/state.rs
Line number: 21

```
impl Space for Pool {
    const INIT_SPACE: usize = 8 + 32 * 3 + (4 + 32) + 2 * 2 + 2 * 3 + 1 + 8 + 16;
}

impl Pool {
    pub fn space() -> usize {
        8 + // Discriminator
        32*3 + // admin, curve and treasury
        (4+32) + // creator_id
        2*2 + // creator_fee and banger fee
        2*3 + // bump, authority_bump & minimum bid
        1 + // is_presale_over
        8 + // total_amount_from_presale
        8 // end_unix_timestamp
    }
}

...

impl Space for Curve {
    const INIT_SPACE: usize = 8 + 1 + 8 + 8;
}

...

pub struct Market {
    pub end_unix_timestamp: u64,
}

impl Space for Market {
    const INIT_SPACE: usize = 8 + 8; // end_unix_timestamp
}
```

The second version for `Pool` is unused. The `Market` struct is unused as well.

For structs deriving `Default`, the size can be figured as such:

```
8 + std::mem::size_of::<StructType>()
```

Note this will default to 24 bytes for strings, any extra bytes or padding can be added like the discriminator.

Severity and Impact Summary

No security impact.

Recommendation

The code may be easier to maintain if cleaner.

No bound checks on config values such as fees

Finding ID: FYEO-BANGER-13
Severity: **Informational**
Status: **Acknowledged**

Description

Some values such as the fees used in a pool have no bounds. This might become an issue if the frontend doesn't properly indicate these or users do not pay attention.

Proof of Issue

File name: programs/banger-program/src/instructions/init_pool.rs
Line number: 66

```
pub fn init_pool(
    &mut self,
    creator_id: String,
    creator_fee: u16,
    banger_fee: u16,
    rewards_pool_fee: u16,
    token_name: String,
    token_metadata_uri: String,
    minimum_bid: u8,
    bumps: &InitPoolBumps,
) -> Result<()> {


...

self.pool.set_inner(Pool {
    admin: self.admin.key(),
    curve: self.curve.key(),
    treasury: self.treasury.key(),
    creator_id,
    creator_fee,
    banger_fee,
    rewards_pool_fee,
    pool_bump: bumps.pool,
    minimum_bid,
    authority_bump: bumps.authority,
    total_amount_from_presale: 0,
    end_unix_timestamp,
    rewards_pool_bump: bumps.rewards_pool,
    supply: 0,
});
```

Severity and Impact Summary

Huge fees may become a problem for the user experience.

Recommendation

Consider implementing bounds for these values.

Rent can not be recovered

Finding ID: FYEO-BANGER-14
Severity: **Informational**
Status: **Acknowledged**

Description

The program does not allow certain accounts to ever be closed which means the rent can not be recovered.

Proof of Issue

File name: programs/banger-program/src/instructions/claim.rs
Line number: 56

```
let rent_exempt_lamports =
(Rent::get()?).minimum_balance(self.creator_vault.data_len());
let rewards = self.creator_vault.lamports() - rent_exempt_lamports;
```

Severity and Impact Summary

There are no particular functions to close certain accounts and recover the rent.

Recommendation

Consider if such functionality is required.

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