F Y E O

Security Code Review Samo Airdrop

Samo

March 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

Samo engaged FYEO Inc. to perform a Security Code Review Samo Airdrop.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on March 06 - March 20, 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-SAMO-01 – Withdraw function does bad accounting
- FYEO-SAMO-02 – Manual airdrop function ignores accounting
- FYEO-SAMO-03 – New owner does not co-sign transaction
- FYEO-SAMO-04 – Project owner can withdraw to any account
- FYEO-SAMO-05 – Bank authority is constant but regenerated every request
- FYEO-SAMO-06 – Code clarity
- FYEO-SAMO-07 – Overflow concerns
- FYEO-SAMO-08 – PDA does not set space for strings
- FYEO-SAMO-09 – PDA restrictions for NFTs and FTs collections
- FYEO-SAMO-10 – Project can be disabled and enabled

- FYEO-SAMO-11 – Unclear implementation
- FYEO-SAMO-12 – Use custom errors to help users

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 Samo Airdrop. 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/solTrust/solairdrop with the commit hash be80ac41fae5cece855dc0272baabf5117ca7748.

Files included in the code review

```
solairdrop/
└── programs/
    └── solairdrop/
        └── src/
            ├── instructions/
            │   ├── airdrop.rs
            │   ├── airdrop_manual.rs
            │   ├── deposit.rs
            │   ├── disable_project.rs
            │   ├── enable_project.rs
            │   ├── initialize.rs
            │   ├── mod.rs
            │   ├── register_project.rs
            │   ├── set_collection.rs
            │   ├── set_owner.rs
            │   └── withdraw.rs
            ├── state/
            │   ├── accounts.rs
            │   ├── error_model.rs
            │   ├── mod.rs
            │   └── utils.rs
            └── lib.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review Samo Airdrop, we discovered:

- 1 finding with MEDIUM severity rating.
- 3 findings with LOW severity rating.
- 8 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

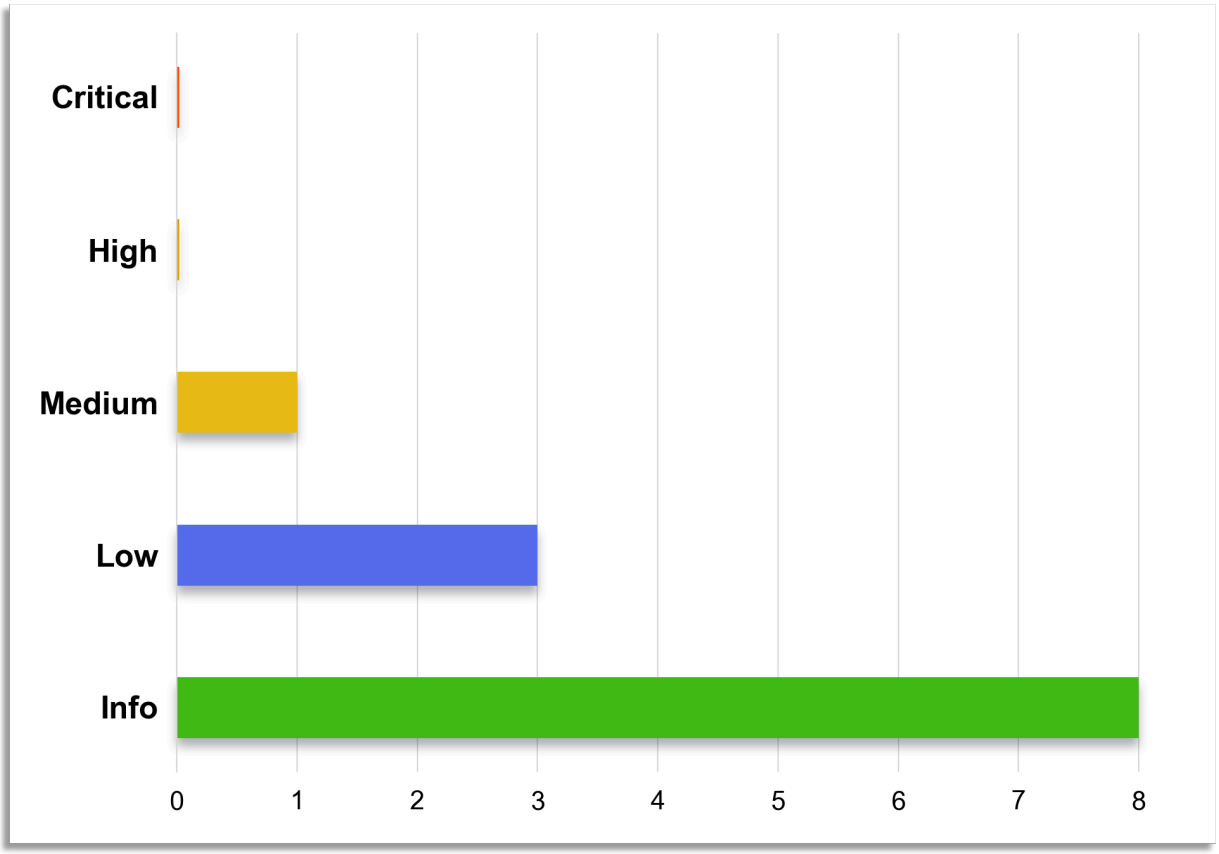

Figure 1: Findings by Severity

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-SAMO-01	**Medium**	Withdraw function does bad accounting
FYEO-SAMO-02	**Low**	Manual airdrop function ignores accounting
FYEO-SAMO-03	**Low**	New owner does not co-sign transaction
FYEO-SAMO-04	**Low**	Project owner can withdraw to any account
FYEO-SAMO-05	**Informational**	Bank authority is constant but regenerated every request
FYEO-SAMO-06	**Informational**	Code clarity
FYEO-SAMO-07	**Informational**	Overflow concerns
FYEO-SAMO-08	**Informational**	PDA does not set space for strings
FYEO-SAMO-09	**Informational**	PDA restrictions for NFTs and FTs collections
FYEO-SAMO-10	**Informational**	Project can be disabled and enabled
FYEO-SAMO-11	**Informational**	Unclear implementation
FYEO-SAMO-12	**Informational**	Use custom errors to help users

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

Upon review, it became evident that the code base exhibits exceptional structural organization. Leveraging the Anchor framework's account macros, the code is succinct and highly comprehensible. However, there remains room for enhancement, particularly in the provision of custom errors and events tailored for third-party developers. Furthermore, while the Rust code demonstrates proficiency, there is a noticeable scarcity in in-code documentation.

It is noteworthy that the team has been exceedingly responsive throughout the review process, promptly addressing raised issues and providing comprehensive answers to inquiries.

Withdraw function does bad accounting

Finding ID: FYEO-SAMO-01
Severity: **Medium**
Status: **Remediated**

Description

The withdraw function does not properly update state which could be an issue with future changes / updates to the program. The `total_supply` should be reset to 0. Also it does disable the project but it was never checked if it is already disabled.

Proof of Issue

File name: programs/solairdrop/src/instructions/withdraw.rs
Line number: 37

```
pub(crate) fn handler(ctx: Context<Withdraw>, _project_id: String) -> Result<()> {
    ...
    let project_pda = &mut ctx.accounts.project_pda;
    project_pda.is_enabled = false;
    project_pda.airdrop_amount = 0;
    Ok(())
}
```

Severity and Impact Summary

It is likely that this implementation will cause issues with future updates.
The team states that this is implemented as desired - the data is to be kept as is and setting `airdrop_amount` to 0 and `is_enabled` to false is good enough.

Recommendation

The `total_supply` should be reset to 0. Make sure to check the intended functionality of `is_enabled`, check whether it is already disabled.

Manual airdrop function ignores accounting

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

Description

Any amount specified is written to the claim account without any actual token transfer.

Proof of Issue

File name: programs/solairdrop/src/instructions/airdrop_manual.rs
Line number: 30

```
pub fn handler(ctx: Context<AirdropManual>, project_id: String, amount: u64) ->
anchor_lang::Result<()> {
    let claim_pda = &mut ctx.accounts.claim_pda;
    claim_pda.project_id = project_id;
    claim_pda.amount = amount;
    Ok(())
}
```

Severity and Impact Summary

In Solana it is easy to scan all PDAs for a contract and do analysis based on that data. The data stored here is fictional in that no token transfer happens and may cause issues for 3rd party developers.
The team states this is implemented correctly but will consider removing the function.

Recommendation

Consider why this function is needed and what it has to accomplish. Is this supposed to be a blacklist? Adjust it accordingly. Also consider adding events for tokens claimed.

New owner does not co-sign transaction

Finding ID: FYEO-SAMO-03
Severity: **Low**
Status: **Open**

Description

The owner of the contract can be modified. The new owner is however not a co-signer of the transaction.

Proof of Issue

File name: programs/solairdrop/src/instructions/set_owner.rs
Line number: 5

```
#[derive(Accounts)]
pub struct SetOwner<'info> {
    #[account(mut)]
    pub owner: Signer<'info>,
    #[account(
    seeds = [b"config"],
    mut,
    bump = config.bump,
    constraint = config.owner_account == *owner.key,
    )]
    pub config: Box<Account<'info, Config>>,
    pub system_program: Program<'info, System>,
}
pub(crate) fn handler(ctx: Context<SetOwner>, owner: Pubkey) -> Result<()> {
    let config = &mut ctx.accounts.config;
    config.owner_account = owner;
    Ok(())
}
```

Severity and Impact Summary

There is some risk that the contract is bricked if any address can be specified.

Recommendation

Require the new owner to be a `Signer` as well.

Project owner can withdraw to any account

Finding ID: FYEO-SAMO-04
Severity: **Low**
Status: **Remediated**

Description

The withdraw function does not check that `project_owner_ata` is owned by `project_owner`. This may be intentional. The naming however suggests it isn't.

Proof of Issue

File name: programs/solairdrop/src/instructions/withdraw.rs
Line number: 26

```
#[account(mut)]
pub project_owner_ata: Box<Account<'info, TokenAccount>>,
```

Severity and Impact Summary

The tokens can be withdrawn to accounts owned by others.

Recommendation

Make sure this is implemented as intended.

Bank authority is constant but regenerated every request

Finding ID: FYEO-SAMO-05
Severity: **Informational**
Status: **Remediated**

Description

The bank authority is used across all projects and therefore constant. It is however being generated for every request.

Proof of Issue

File name: programs/solairdrop/src/instructions/register_project.rs
Line number: 43

```
let (bank_authority, _bank_authority_bump) =
    anchor_lang::prelude::Pubkey::find_program_address(&[&crate::state::AUTHORITY_SEED],
ctx.program_id);
```

Severity and Impact Summary

This wastes gas, however the team states that they feel most comfortable with the solution as it is.

Recommendation

Consider storing this key in the global config or generate individual keys for each project and store them in the projects pda.

Code clarity

Finding ID: FYEO-SAMO-06
Severity: **Informational**
Status: **Remediated**

Description

Some improvements could be made to the code clarity in order to improve maintainability.

Proof of Issue

File name: programs/solairdrop/src/instructions/airdrop.rs
Line number: 39

```
constraint = recipient_mint_ata.mint == recipient_mint.key(),
constraint = recipient_mint_ata.owner == recipient.key(),
```

Use concise account macros when available.

File name: programs/solairdrop/src/instructions/register_project.rs
Line number: 56

```
project_pda.project_vault = *ctx.accounts.project_vault.to_account_info().key;
...
project_pda.token_address = *ctx.accounts.project_token.to_account_info().key;
```

There is a function `key()` on the account itself, so there is no need to get `account_info` first.

Severity and Impact Summary

These constraints are not as concise as the ones provided by Anchor. Also the `account.owner` and token authority (`account.data.owner`) are easily confused.

Recommendation

Use `token::mint` and `token::authority` for the constraints. Keep the code concise.

Overflow concerns

Finding ID: FYEO-SAMO-07
Severity: **Informational**
Status: **Remediated**

Description

While the overflow should not be possible as it is directly related to token instructions working, it is still recommended to use checked math mostly because other parts of the code base do not do strict accounting. Therefore it is best to use checked math to be safe with future updates.

Proof of Issue

File name: programs/solairdrop/src/instructions/deposit.rs
Line number: 44

```
project_pda.token_supply += amount;
```

Severity and Impact Summary

There are some accounting concerns in this code base which, with updates, could potentially lead to overflows.

Recommendation

Use `checked_add()` to make sure no overflow happens.

PDA does not set space for strings

Finding ID: FYEO-SAMO-08
Severity: **Informational**
Status: **Remediated**

Description

The `ProjectPda` and `ClaimPda` both store a string, but neither of the macro `space` declarations make any room for strings. The size_of a `String` is fixed to 24 in Rust.

Proof of Issue

File name: programs/solairdrop/src/instructions/register_project.rs
Line number: 26

```
#[account(
..
space = 8 + std::mem::size_of::<ProjectPda>(),
..
)]
pub project_pda: Box<Account<'info, ProjectPda>>,
```

Severity and Impact Summary

The strings that can be stored will be limited to 24 bytes.
The team states that only 20 bytes will be required for storing strings. The issue was downgraded to informational.

Recommendation

Make sure this limit is acceptable for the intended functionality. Consider that utf8 is multibyte and some characters may be 4 bytes long.

PDA restrictions for NFTs and FTs collections

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

Description

The solution does not work for fungible tokens but only for NFTs. Also users with more than one NFT can claim the airdrop multiple times.

Proof of Issue

File name: programs/solairdrop/src/instructions/airdrop.rs
Line number: 47

```
#[account(
init,
seeds = [b"claim", recipient_mint.key().as_ref(), project_id.as_bytes()],
bump,
payer = recipient,
space = 8 + std::mem::size_of::< ClaimPda > (),
)]
pub claim_pda: Box<Account<'info, ClaimPda>>
```

Severity and Impact Summary

The team states that fungible tokens do not need to be supported. Thus, this issue was downgraded to informational.

Recommendation

Make sure these limitations are acceptable.

Project can be disabled and enabled

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

Description

There are two functions to enable and disable a project. It could just be one function. Neither of the current functions check that the project isn't already in the state that is being set. The code does not check the state in most functions.

Proof of Issue

File name: programs/solairdrop/src/instructions/disable_project.rs
Line number: 26

```
pub(crate) fn handler(_ctx: Context<DisableProject>, _project_id: String) -> Result<()>
{
    let project_pda = &mut _ctx.accounts.project_pda;
    project_pda.is_enabled = false;
    Ok(())
}
```

File name: programs/solairdrop/src/instructions/enable_project.rs
Line number: 27

```
pub(crate) fn handler(ctx: Context<EnableProject>, _project_id: String) -> Result<()> {
    let project_pda = &mut ctx.accounts.project_pda;
    project_pda.is_enabled = true;
    Ok(())
}
```

Severity and Impact Summary

The function may perform no operation. The code can be simplified to improve maintainability.
The team states that this functionality is as designed. The issue was therefore downgraded to informational.

Recommendation

Combine these two functions into one and make sure the function will actually update state. It is probably also advisable to implement events for this functionality. Also the state is not checked in most functions, make sure functionality that should be disabled if the project is disabled will check the state.
Also check to make sure this simple flag is sufficient for providing the necessary functionality for the airdrops.

Unclear implementation

Finding ID: FYEO-SAMO-11
Severity: **Informational**
Status: **Remediated**

Description

The program relies on a global collection being configured by the admin for users to claim different projects airdrops.

Severity and Impact Summary

This collection can be modified by the admin at any time affecting all projects in any state. It is also unclear what the end criteria for an airdrop are supposed to be. The project admin can withdraw funds at any point. The only flag preventing people from claiming is the `is_enabled` flag which can be set by the global admin or by withdrawing funds from the project - this could be meant as a pause or the end of the airdrops claim window. There is no time constraint or anything that would communicate constraints to the users.

The team has stated that this functionality is implemented as required and therefore this issue was downgraded to informational.

Recommendation

Make sure that this functionality is implemented correctly.

Use custom errors to help users

Finding ID: FYEO-SAMO-12
Severity: **Informational**
Status: **Remediated**

Description

The macro constraints used do not return intelligible errors. It would be helpful for users to receive more insightful errors.

Proof of Issue

This issue is project wide.

Severity and Impact Summary

This may make it harder for users to understand what the problem is.

Recommendation

Use some custom error codes in the account macros. Especially for authentication and project state changes.

References

https://docs.rs/anchor-lang/latest/anchor_lang/derive.Accounts.html#normal-constraints

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