FYEO

Security Code Review of Spree Points

Spree

March 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

Spree engaged FYEO Inc. to perform a Security Code Review of Spree Points.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on February 17 - February 21, 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-SPREE-01 – Any signing account can update fee parameters
- FYEO-SPREE-02 – Unoptimal storage of data in whitelist
- FYEO-SPREE-03 – Absence of event emission in key state update functions
- FYEO-SPREE-04 – Duplicate accounts in whitelist
- FYEO-SPREE-05 – Logging deletion of a non-existent account from whitelist

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 Spree Points. 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/crazspree/spree-points/ with the commit hash d37a73dbdbee6580cf0b2eeca6516ff409f0fc7b. Remediations were provided with the commit hash d1a5c39f0575c137cc9075e8dc79435860c144ef.

Files included in the code review
```
spree-points/
└── programs/
    ├── spree_points/
    │   └── src/
    │       ├── instructions/
    │       │   ├── burn_tokens.rs
    │       │   ├── fees.rs
    │       │   ├── initialize_token.rs
    │       │   ├── mint_tokens.rs
    │       │   ├── mod.rs
    │       │   └── toggle_freeze.rs
    │       ├── state/
    │       │   ├── events.rs
    │       │   └── mod.rs
    │       ├── utils/
    │       │   ├── mod.rs
    │       │   └── token2022.rs
    │       ├── constants.rs
    │       ├── error.rs
    │       └── lib.rs
    └── transfer-hook/
        └── src/
            ├── instructions/
            │   ├── mod.rs
            │   ├── transfer_hook.rs
            │   └── whitelist.rs
            ├── state/
            │   ├── events.rs
            │   ├── freeze.rs
            │   └── mod.rs
            ├── constants.rs
            ├── error.rs
            └── lib.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Spree Points, we discovered:

- 1 finding with HIGH severity rating.
- 1 finding with MEDIUM severity rating.
- 3 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

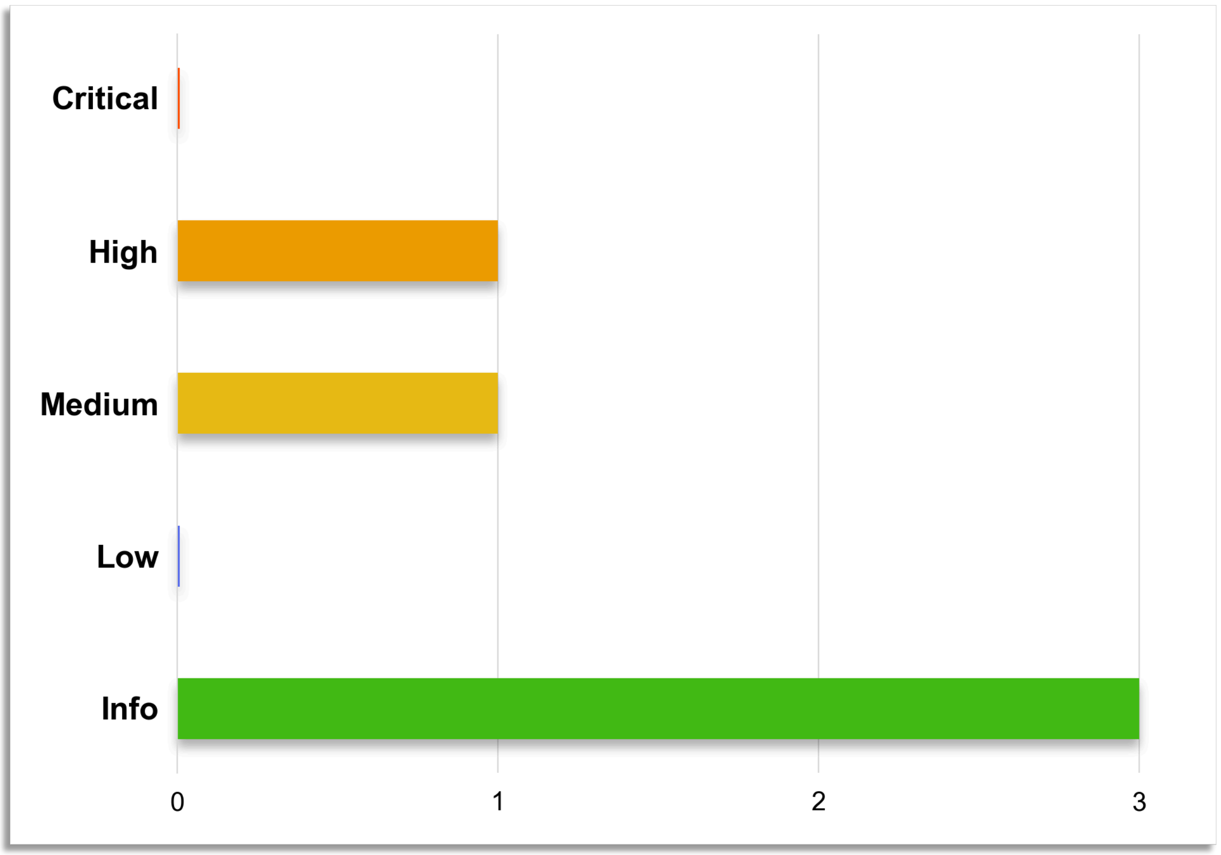

Figure 1: Findings by Severity

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-SPREE-01	High	Any signing account can update fee parameters
FYEO-SPREE-02	Medium	Unoptimal storage of data in whitelist
FYEO-SPREE-03	Informational	Absence of event emission in key state update functions
FYEO-SPREE-04	Informational	Duplicate accounts in whitelist
FYEO-SPREE-05	Informational	Logging deletion of a non-existent account from whitelist

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 Spree Points system is an ecosystem built on modular components that provide processes for mining, token burning, commission management, and whitelist compliance. This closed-loop structure demonstrates the potential for flexible and robust token management, effectively utilizing Anchor's account handling and CPI challenge capabilities.

However, storing the whitelist as a vector of PublicKeys proves to be a poor solution. As the whitelist grows, the computational overhead increases, which in extreme cases can lead to the risk of DoS attacks. In addition, the lack of event logging in key functions, such as freeze state switching (transfer-hook) or updating commission parameters (spree-points), reduces transparency and makes auditing difficult.

Additional challenges arise in the whitelist management logic: deletion logs an event even if the target account does not exist, and the lack of duplicate checking when adding an account leads to excessive space consumption. To improve scalability and optimize computation, it is recommended to move to more efficient data structures such as Merkle trees or mapping.

A critical weakness of the system lies in the commission update mechanism: any signing account can change the parameters, which poses serious economic risks in case of unauthorized access. Overall, despite its strengths in modularity and integration with Solana, the system requires refinement in terms of data efficiency, event consistency, and strict access control to ensure reliability and security.

Any signing account can update fee parameters

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

Description

The `_update_fees()` function does not check who can update the fees parameters.

Proof of Issue

File name: programs/spree_points/src/instructions/fees.rs
Line number: 19

```
pub fn _update_fees(ctx: Context<UpdateFees>, args: UpdateFeesArgs) -> Result<()> {
    let fees = &mut ctx.accounts.fees;

    if (args.mint_fee_bps.is_some() && args.mint_fee_bps.unwrap() > 10000) ||
        (args.transfer_fee_bps.is_some() && args.transfer_fee_bps.unwrap() > 10000) ||
        (args.redemption_fee_bps.is_some() && args.redemption_fee_bps.unwrap() > 10000)
{
        return Err(SpreeTokenError::BpsOutOfRange.into());
    }
    if let Some(mint_fee_bps) = args.mint_fee_bps {
        fees.mint_fee_bps = mint_fee_bps;
    }
    if let Some(transfer_fee_bps) = args.transfer_fee_bps {
        fees.transfer_fee_bps = transfer_fee_bps;
    }
    if let Some(redemption_fee_bps) = args.redemption_fee_bps {
        fees.redemption_fee_bps = redemption_fee_bps;
    }
    if let Some(fee_collector) = args.fee_collector {
        fees.fee_collector = fee_collector;
    }

    Ok(())
}
```

During fees initialization, fees are not set by Admin as Fees Authority, allowing any user to override the fees parameters.

Proof of Issue

File name: programs/spree_points/src/instructions/fees.rs
Line number: 5

```
pub fn _initialize_fees(ctx: Context<InitializeFees>, args: InitFeesArgs) -> Result<()>
{
    let fees = &mut ctx.accounts.fees;

    if args.mint_fee_bps > 10000 || args.transfer_fee_bps > 10000 ||
args.redemption_fee_bps > 10000 {
        return Err(SpreeTokenError::BpsOutOfRange.into());
    }
    fees.mint_fee_bps = args.mint_fee_bps;
    fees.transfer_fee_bps = args.transfer_fee_bps;
    fees.redemption_fee_bps = args.redemption_fee_bps;
    fees.fee_collector = args.fee_collector;

    Ok(())
}
```

Severity and Impact Summary

If signer verification is missing, updating commissions can lead to an attacker being able to change the fees parameters, which critically affects the economics of the token.

Recommendation

The `Fees` structure needs to be extended.

```
#[account]
#[derive(InitSpace)]
pub struct Fees {
    pub mint_fee_bps: u16,
    pub transfer_fee_bps: u16,
    pub redemption_fee_bps: u16,
    pub fee_collector: Pubkey,
    pub authority: Pubkey, // Add administrator field
}
```

In the `_initialize_fees()` function, you can add a check if the authority field is not already set (e.g., equal to `Pubkey::default()`), set it equal to `ctx.accounts.signer.key()`.

Before performing a parameter update, verify that the signer is an administrator:

```
pub fn _update_fees(ctx: Context<UpdateFees>, args: UpdateFeesArgs) -> Result<()> {
    let fees = &mut ctx.accounts.fees;

    // Add administrator permission check

    if fees.authority != ctx.accounts.signer.key() {
        return Err(SpreeTokenError::Unauthorized.into());
    }
   ...
}
```

Unoptimal storage of data in whitelist

Finding ID: FYEO-SPREE-02
Severity: Medium
Status: Remediated

Description

The current implementation of the `WhiteList` structure in the transfer-hook program uses a `Vec<Pubkey>` to store whitelisted addresses. While vectors are simple to implement, they scale poorly in Solana due to the platform's strict compute budget constraints (200,000 CU per transaction by default). Vector operations, such as iteration in `_remove_from_whitelist()` or containment checks in `_transfer_hook()`, have linear time complexity O(n), where `n` is the number of items. As the whitelist grows—potentially to hundreds thousands of users — the computational cost of these operations increases proportionally. This not only drives up the cost of transfers but also risks transaction failures when the whitelist exceeds a critical size (e.g., 1,000 entries), as the required compute units could surpass the budget, rendering transactions unprocessable.

Proof of Issue

File name: programs/transfer-hook/src/instructions/whitelist.rs
Line number: 95

```
pub struct WhiteList {
    pub authority: Pubkey,
    pub white_list: Vec<Pubkey>,
    pub freeze_transfer: bool,
}
```

Severity and Impact Summary

As the whitelist grows, transfer costs escalate, and the risk of transaction failure increases significantly. Beyond ~1,000 entries, transfers may become infeasible due to compute budget exhaustion, effectively creating a DoS condition for users. This scalability bottleneck undermines the system's usability and reliability in real-world scenarios with large user bases.

Recommendation

A more efficient approach is to leverage Merkle trees, where only the root is stored on-chain, while proofs are generated off-chain. This reduces on-chain storage costs significantly.

Using mappings or similar data structures with logarithmic or constant-time operations offers a more scalable and secure alternative.

Absence of event emission in key state update functions

Finding ID: FYEO-SPREE-03
Severity: Informational
Status: Remediated

Description

Some important functions that change system state, such as `_toggle_freeze()` and `_update_fees()`, lack event emission. This makes it difficult to audit and track changes because administrators and users do not receive transparent information about the changes made.

Proof of Issue

File name: programs/transfer-hook/src/instructions/whitelist.rs
File name: programs/spree_points/src/instructions/fees.rs

Severity and Impact Summary

This does not carry any threat.

Recommendation

Add event emission to each function where a state change occurs (e.g., switching state, updating commission parameters) to ensure that important changes are logged.

Duplicate accounts in whitelist

Finding ID: FYEO-SPREE-04
Severity: Informational
Status: Remediated

Description

The `_add_to_whitelist()` function does not check that the account being added is already in the whitelist.

Proof of Issue

File name: programs/transfer-hook/src/instructions/whitelist.rs
Line number: 7

```
pub fn _add_to_whitelist(ctx: Context<WhiteListInfo>) -> Result<()> {
    if ctx.accounts.white_list.authority != ctx.accounts.signer.key() {
        msg!("Only the authority can add to whitelist!");
        return Err(SpreeTokenError::Unauthorized.into());
    }
    ctx.accounts
        .white_list
        .white_list
        .push(ctx.accounts.user.key());
    ...
}
```

Severity and Impact Summary

Duplicate whitelist items are not a critical vulnerability. It may lead to excessive consumption of lamports due to increased account size.

Recommendation

Add a check to see if the account is in the whitelist.

Logging deletion of a non-existent account from whitelist

Finding ID: FYEO-SPREE-05
Severity: Informational
Status: Remediated

Description

In the `_remove_from_whitelist()` function, messages and an account deletion event are emitted regardless of whether the specified account is present in the whitelist.

Proof of Issue

File name: programs/transfer-hook/src/instructions/whitelist.rs
Line number: 34

```
pub fn _remove_from_whitelist(ctx: Context<WhiteListInfo>) -> Result<()> {
    if ctx.accounts.white_list.authority != ctx.accounts.signer.key() {
        msg!("Only the authority can remove from the white list!");
        return Err(SpreeTokenError::Unauthorized.into());
    }

    // Remove from whitelisted users
    ctx.accounts
        .white_list
        .white_list
        .retain(|&x| x != ctx.accounts.user.key());

    msg!(
        "Account removed from white list! {0}",
        ctx.accounts.user.key().to_string()
    );
    msg!(
        "White list length! {0}",
        ctx.accounts.white_list.white_list.len()
    );

    emit!(WhitelistRemoveEvent {
        authority: ctx.accounts.signer.key(),
        account_removed: ctx.accounts.user.key(),
    });

    Ok(())
}
```

Severity and Impact Summary

This is not a vulnerability, it can be misleading as the admin will receive confirmation of the deletion even if the account was not on the whitelist.

Recommendation

Before emitting the message and event, check if the whitelist has been changed (for example, compare the length of the vector before and after applying retain). If the length has not changed, you can print a message that the specified account is not found and not emit the delete event.

Our Process

Methodology

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

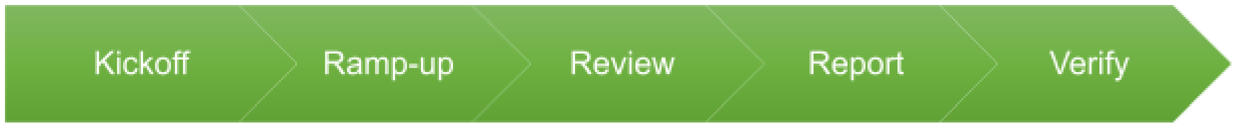

Figure 2: Methodology Flow

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