F Y E O

Security Code Review of Spree EVM

Spree

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

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

The assessment was conducted remotely by the FYEO Security Team. Testing took place on June 11 - June 16, 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 – Potential gas griefing attack
- FYEO-SPREE-02 – Code clarity
- FYEO-SPREE-03 – Consider adding increaseAllowance patterns or EIP-2612

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 EVM. 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/Spree-Finance/spree-points-evm with the commit hash 1d653931f797454c40107a7a5936eb35ad1c5ce6.

Remediations were submitted with the commit hash b6aec9749c02ffb63af44b7fba79224f601efc1b.

Files included in the code review

spree-points-evm/
└── contracts/
  └── spree/
  ├── CollateralVault.sol
  ├── SpreeFactory.sol
  └── SpreePoints.sol

Table 1: Scope

Technical Analyses and Findings

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

- 1 finding with LOW severity rating.
- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

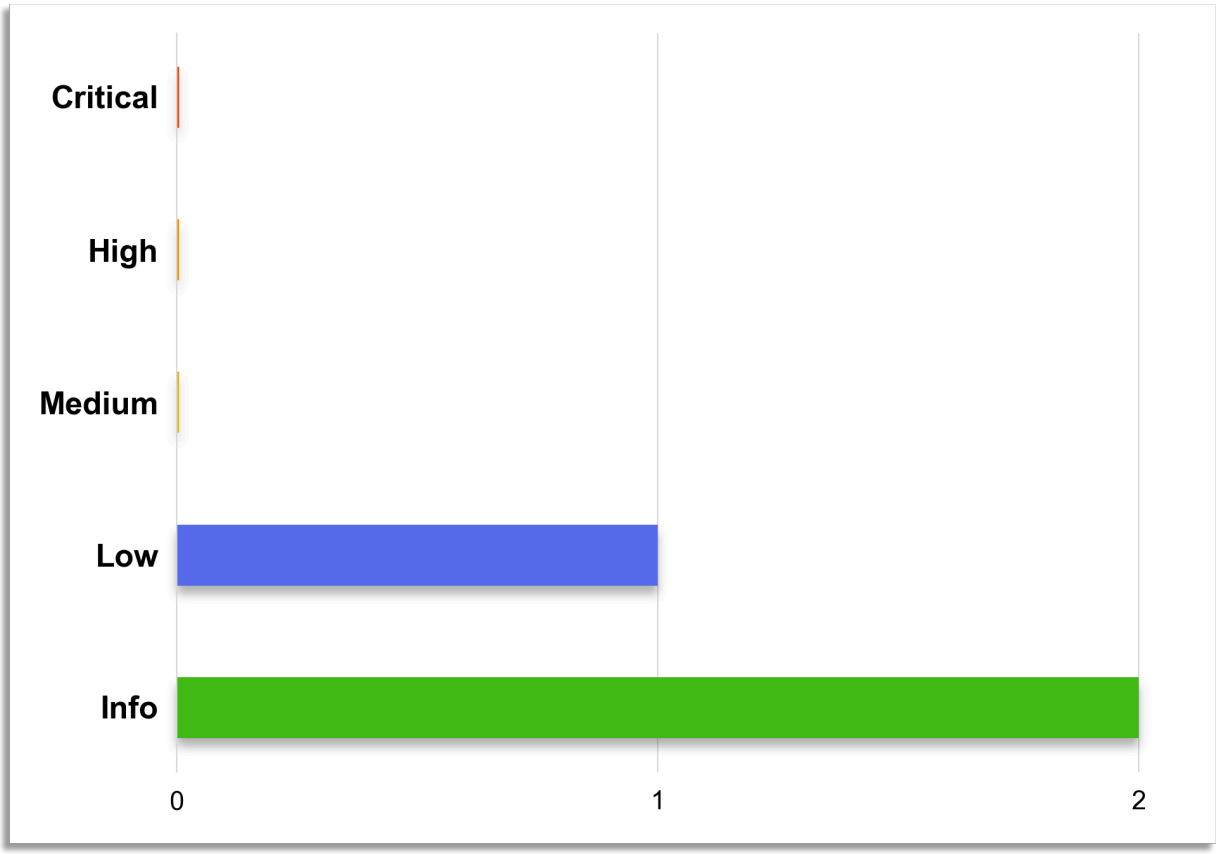

Figure 1: Findings by Severity

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	Status
FYEO-SPREE-01	Low	Potential gas griefing attack	Remediated
FYEO-SPREE-02	Informational	Code clarity	Remediated
FYEO-SPREE-03	Informational	Consider adding increaseAllowance patterns or EIP-2612	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

The update refactors the `Honey` stablecoin contracts into a new `SpreePoints` system. The logic shifts from a direct mint-and-redeem model to a permissioned, asynchronous process. Minting and redeeming `SpreePoints` is now restricted by whitelists managed by a `MANAGER_ROLE`. The redemption process has been redesigned to be a two-step flow: a user first submits a `requestToRedeem`, which escrows their `SpreePoints` in the factory contract, and an `EXECUTOR_ROLE` later finalizes the request, releasing the underlying collateral.

Furthermore, the `SpreePoints` token itself has been modified to be non freely transferable. All transfers, including `approve`, `transfer`, and `transferFrom`, are now gated by a `transferWhitelist` controlled by the contract's `DEFAULT_ADMIN_ROLE`. These changes transform the token from a permissionless stablecoin into a centrally managed points system where all significant on-chain actions are mediated by privileged roles.

Potential gas griefing attack

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

Description

Users can create claims for potentially minuscule amounts and force the `EXECUTOR_ROLE` to pay for iterating large amounts of data and could potentially cause a DoS situation if the gas limit is reached.

Proof of Issue

File name: contracts/spree/SpreeFactory.sol
Line number: 464

```
function requestToRedeem(
    address asset,
    uint256 spAmount,
    address receiver,
    bool expectBasketMode
)
    external
    whenNotPaused
    onlyRedeemWhitelisted
{
    _checkRegisteredAsset(asset);

    if (spAmount == 0)
        ZeroAmount.selector.revertWith();
    }
```

File name: contracts/spree/SpreeFactory.sol
Line number: 579

```
function finalizeRedeems(address account) external whenNotPaused returns (uint256[]
memory redeemed) {
    _checkRole(EXECUTOR_ROLE);

    // get all redeem requests of the account
    RedeemRequest[] storage _redeemRequests = redeemRequests[account];
    if (_redeemRequests.length == 0) {
        return redeemed;
    }

    redeemed = new uint256[](registeredAssets.length);
    for (uint256 reqIdx = 0; reqIdx < _redeemRequests.length; reqIdx++) {
        ...
```

Severity and Impact Summary

This vulnerability allows for a gas griefing attack, creating a potential Denial of Service vector. A malicious actor can create a large number of small redemption requests at a low cost, making the gas required for

the `finalizeRedeems` function exceed the block gas limit. This would prevent the `EXECUTOR_ROLE` from processing redemptions in batches, forcing them into a costly and inefficient one-by-one process and potentially delaying redemptions for all users.

Recommendation

It is strongly recommended to ensure that each claim is of some substantial value or to modify the `finalizeRedeems` function to process requests in bounded batches rather than iterating over the entire array. The function should accept parameters such as `startIndex` and `count` to allow the `EXECUTOR_ROLE` to process the request queue in manageable, gas predictable chunks. This mitigates the DoS vector and ensures the operational stability of the redemption process.

Remediation

The fix eliminates the unbounded loop over a user's entire redeem request array by collapsing `redeemRequests` from an array into a single, per account struct (with an `exists` flag) and removing both the old `finalizeRedeems` batch function and the multi-indexed `redeemRequestsCount` machinery. A new `minRedemRequest` threshold is enforced so that tiny, gas griefing requests can't be queued, and any account may only ever have one pending request at a time, so the executor no longer needs to iterate over a potentially huge array, and there's no way for an attacker to create enough sub-unit requests to gas-out the finalizer. This both bounds gas usage predictably and prevents the denial-of-service attack vector.

Code clarity

Finding ID: FYEO-SPREE-02
Severity: **Informational**
Status: **Remediated**

Description

Some minor concerns around comments and event emission.

Proof of Issue

File name: contracts/spree/SpreeFactory.sol
Line number: 296

```
function removeFromRedeemWhitelist(address account) external {
    _checkRole(MANAGER_ROLE);
    require(redeemWhitelist[account], "Address not redeem whitelisted");
    redeemWhitelist[account] = false;
    emit AddedToRedeemWhitelist(account);
}
```

File name: contracts/spree/SpreeFactory.sol
Line number: 530

```
// emit RedeemRequestApproved event
emit RedeemRequestRejected(account, _redeemRequest.receiver, _redeemRequest.asset,
_redeemRequest.amount, requestId);
```

Severity and Impact Summary

The impact is primarily on off-chain systems and code clarity. Emitting an incorrect event can mislead monitoring tools and block explorers, while stale comments can confuse developers, potentially leading to misunderstandings about the contract's behavior. There is no direct risk to user funds.

Recommendation

It is recommended to correct the event emission in the `removeFromRedeemWhitelist` function to emit a `RemovedFromRedeemWhitelist` event for accurate off-chain tracking. Additionally, the stale comment in the rejectRedeemRequest function should be updated to match the event being emitted.

Remediation

The team has addressed this concern by updating the comment and emitting the correct event.

Consider adding increaseAllowance patterns or EIP-2612

Finding ID: FYEO-SPREE-03
Severity: **Informational**
Status: **Acknowledged**

Description

Encourage the use of `increaseAllowance` / `decreaseAllowance` patterns, or implement EIP-2612 `permit` to avoid on-chain approvals. The `permit` pattern (EIP-2612) lets users approve an ERC20 allowance via an off-chain signature instead of an on-chain `approve` call, so they can do one fewer transaction.

Proof of Issue

File name: contracts/spree/SpreePoints.sol
Line number: 96

```
function approve(address spender, uint256 amount) public override returns (bool) {
    if (!isTransferWhitelisted(msg.sender)) {
        NotTransferWhitelisted.selector.revertWith(msg.sender);
    }
    if (!isTransferWhitelisted(spender)) {
        NotTransferWhitelisted.selector.revertWith(spender);
    }

    // Call the approve function.
    return super.approve(spender, amount);
}
```

Severity and Impact Summary

This is a recommendation for improving user experience and gas efficiency, not a direct vulnerability. The standard approve function requires users to submit two separate transactions for interactions with other contracts, increasing gas costs and complexity. It also has a known, albeit minor, front-running race condition risk that `increaseAllowance` / `decreaseAllowance` mitigates.

Recommendation

Consider implementing both EIP-2612 `permit` and the `increaseAllowance` / `decreaseAllowance` functions. EIP-2612 would improve the user experience by allowing for one transaction interaction. The increase/decreaseAllowance functions should be added to mitigate the well-known race condition vector associated with the standard approve function, providing a safer alternative for on-chain allowance management.

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.