0xCommit

# Security Assessment Report

Perpie

Version: Final

Date: 1 Jan 2024

# Table of Contents

# License

# Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

# Introduction

## Purpose of this report

0xCommit has been engaged by **Perpie** to perform a security audit of several contracts of Perpie.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

# Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

**Repository :** https://github.com/perpie-io/contracts

| Version | Commit hash |
| --- | --- |
| Initial | 396e9407e13565c5c158bd65b7c828a23edbe882 |
| Fixed | e1611654858a63f1bfbd2d4f9b009720eda8d8d7 |

# How to Read This Report

This report classifies the issues found into the following severity categories:

| Severity | Description |
|---|---|
| **Critical** | A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service. |
| **Major** | A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service. |
| **Minor** | A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies. |
| **Informational** | Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share. |

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

# Overview

## Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

## Functionality Overview

Perpie contracts are module built on top of GMXV1, facilitating users to create positions and increase their positions on GMXV1.

# Summary of Findings

| Sr. No. | Description | Severity | Status |
|---|---|---|---|
| 1 | Funds can be locked into the FeeManager contract | High | Resolved |
| 2 | Return data for ERC20 transfer not consider | High | Resolved |
| 3 | Missing control to limit values for fees | Medium | Resolved |
| 4 | Lack of zero address check in constructor | Low | Resolved |

# Detailed Findings

## 1. Funds can be locked into the FeeManager contract

**Severity:** High

### Description

When a user sends native tokens to the FeeManager contract, it becomes impossible to retrieve those funds, and they become permanently trapped within the FeeManager contract.

### Remediation

To address this issue, there are two possible solutions:

- Add an administrative method to withdraw native tokens from the FeeManager contract.
- Remove the fallback and receive methods from the FeeManager contract if they serve no meaningful purpose in the overall system.

### Status

Resolved

## 2. Return data for ERC20 transfer not considered

**Severity:** High

### Description

The "_chargeTokenFee" function in the contract facilitates the transfer of fees using ERC20 tokens to the FeesManager. Currently, this function invokes "execTransactionFromModule" within a smart contract wallet, disregarding the return data from the ERC20 token's "transfer" function. This approach poses a risk because, for ERC20 tokens, it is crucial to verify that the return value is true to confirm a successful transfer. As a result, the contract may incorrectly assume that fees have been deducted even if the ERC20 transfer fails.

### Remediation

To address this issue and ensure proper handling of fee transfers, it is recommended to update the "_chargeTokenFee" function by replacing the use of "execTransactionFromModule" with "execTransactionFromModuleReturnData".

### Status

Resolved

## 3. Missing control to limit values for fees.

**Severity:** Medium

### Description

In the Fee Manager contract, there is currently no validation in place to ensure that the feesBps value set in the setFeesBps function is within the valid range of 0 to 10000 (representing 0% to 100% in basis points). This lack of validation can lead to over-calculation of fees if a value greater than 10000 is set.

### Remediation

To address this issue and prevent over-calculation of fees, you should implement a check within the setFeesBps function to validate that the input feesBps value falls within the 0 to 10000 range. Here's the remediation solution:

```
function setFeesBps(uint256 _feesBps) public onlyOwner {
    require(_feesBps >= 0 && _feesBps <= 10000, "Fees must be between 0 and 10000 BPS (0% to 100%).");

    feesBps = _feesBps;
}
```

With this modification, the setFeesBps function now includes a “require” statement to check that the _feesBps value is within the valid range of 0 to 10000. If the condition is not met, the function will revert, ensuring that only valid fee values are set within the specified range. This prevents over-calculation of fees and enforces a proper fee setting within the desired range of 0% to 100%.

### Status

Resolved

## 4. Lack of zero address check in constructor

**Severity:** Low

### Description

In the contract GMXV1FeesModule's constructor and its base contract PerpFeesModule, there is a missing check to ensure that zero addresses are not accepted when setting variables. This oversight can lead to unintended behavior or vulnerabilities if zero addresses are allowed in these critical variables.

### Remediation

To mitigate this issue, you should implement zero address checks in the constructor of GMXV1FeesModule and any relevant functions within the PerpFeesModule base contract. This can be achieved by adding a "require" statement that checks if the provided address is not the zero "address (address(0))". This ensures that zero addresses cannot be used, preventing potential issues related to uninitialized or invalid addresses.

### Status

Resolved