PUBLIC

Code Assessment

of the Fee Splitter Smart Contracts

September 25, 2024

Produced for

Curve

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Curve team,

Thank you for trusting us to help Curve with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Fee Splitter according to Scope to support you in forming an opinion on their security risks.

Curve implements fee splitter to distribute fees (in crvUSD token) from the crvUSD stablecoin markets to different recipient according to configured weights.

The most critical subjects covered in our audit are denial of service, correct access control and correct usage of the new Vyper modules. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	5
• Code Corrected	5

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Fee Splitter repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

fee-splitter

V	Date	Commit Hash	Note
1	28 August 2024	581b8978f91e426c648cf6243420fee5276166b7	Initial Version
2	23 September 2024	59077063bfc658189ec02923cbc7f72dd3380084	Version 2
3	25 September 2024	74a06ef438e8aeb096f54041d54afe489cf92f67	Version 3

snekmate

V	Date	Commit Hash	Note
1	26 June 2024	feb2dc084c7d817b0d93cbd533396881ba24bb30	snekmate v0.1.0

For the vyper smart contracts, the compiler version `0.4.0` was chosen.

The following contracts where included in scope:

`fee-splitter`:

- `contracts/fee_splitter/Controller.vyi`
- `contracts/fee_splitter/ControllerFactory.vyi`
- `contracts/fee_splitter/ControllerMulticlaim.vy`
- `contracts/fee_splitter/DynamicWeight.vyi`
- `contracts/fee_splitter/FeeSplitter.vy`

`snekmate`:

- `src/snekmate/auth/ownable.vy`

As of (Version 2), the `fee-splitter` contracts were moved to the following locations:

- `contracts/interfaces/IController.vyi`
- `contracts/interfaces/IControllerFactory.vyi`
- `contracts/interfaces/IDynamicWeight.vyi`
- `contracts/ControllerMulticlaim.vy`
- `contracts/FeeSplitter.vy`

2.1.1 *Excluded from scope*

Anything not listed in the scope is considered out of scope. |

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Curve offers a fee splitter to distribute fees (in crvUSD token) from the crvUSD stablecoin markets to different recipient according to configured weights.

The `FeeSplitter` module contain the main logic of the system and uses two submodules `ownable` and `ControllerMulticlaim` to delegate some logic. The contract defined one trusted role, the `owner` which can edit the list of recipient and their weight. The state modifying entry points of the contract are the following:

- `ControllerMulticlaim.update_controllers()` allow for anyone to update the set of allowed controllers by fetching the new controllers not yet registered by the `FeeSplitter` from the `ControllerFactory`.
- `FeeSlitter.dispatch_fees()` is the main entry point of the system. The unpermissioned function allows for anyone to distribute the fees from a provided list of controllers which are ensured to be approved by the system. If no list is provided, all approved controllers are used. The function will first claim the fee from each controllers before transferring a fraction of the total to each recipients according to their respective weight. In case a recipient defines a `weight()` function, the weight is dynamically fetched from them and is capped by the weight defined by the owner. The excess funds not claimed by dynamic recipient is sent to the last recipient of the list as `excess`.
- `FeeSplitter.set_receivers()` is only callable by the admin and allows to set the list of recipients and their weight.
- `ownable.transfer_ownership()` allows the owner to transfer the ownership of the contract to another address.
- `ownable.renounce_ownership()` allows the owner to renounce the ownership of the contract.

Calling the function `dispatch_fees` will be incentivized through the "hooker" infrastructure that is already being used by Curve for the fee burners.

2.2.1 Trust Model

- The `owner` is fully trusted.
- Receivers are trusted not to behave maliciously.
- `crvUSD`, the `ControllerFactory` and the different `Controller`, are assumed to be the contract defined by the curve stablecoin system (and not the curve lending system).
- It is assumed that the `FeeSplitter` is the `fee_receiver` of the `ControllerFactory`.

2.2.2 Changes in versions

In Version 2 and Version 3, the contract were updated to fix issues found during the audit, no new features were added.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0

High-Severity Findings	0

Medium-Severity Findings	0

Low-Severity Findings	5

- Contradiction in Condition Code Corrected
- Incorrect Interface Code Corrected
- Non-Indexed Events Code Corrected
- Receiver Can DoS the Distribution Code Corrected
- MAX_CONTROLLERS Mismatch Between ControllerFactory and FeeSplitter Code Corrected

Informational Findings	2

- Unused Interface Code Corrected
- Unused Event Code Corrected

6.1 Contradiction in Condition

Design Low Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-001

In `FeeSplitter._is_dynamic()`, the condition `len(response) > 32` is always false as `response` is a `Bytes[32]`.

Code corrected

The condition was removed from the contract.

6.2 Incorrect Interface

Correctness Low Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-002

The `FeeSplitter` defines `DYNAMIC_WEIGHT_EIP165_ID` as `0x12431234` which seems to be a placeholder as commented above. It should be replaced with the actual interface ID intended for `DynamicWeight`.

Code corrected

The placeholder has been replaced by the method id of weight().

6.3 Non-Indexed Events

Design Low Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-005

No parameters are indexed in the events of the FeeSplitter. It is recommended to index the relevant event parameters to allow integrators and dApps to quickly search for these and simplify UIs.

Code corrected

The receiver of FeeDispatched is now indexed.

6.4 Receiver Can DoS the Distribution

Design Low Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-003

If a receiver implements EIP-165 and state that it supports the DYNAMIC_WEIGHT_EIP165_ID but in reality either does not implement the function weight() or always revert when weight() is being called, the FeeSplitter will not be able to distribute the fees until the owner removes them from the receivers list.

Code corrected

As receiver are generally trusted, the receiver should never behave maliciously, however, to accommodate with potential issue in the implementation of a receiver, the code was updated such that when a call to weight() fails for some receiver, a LivenessProtectionTriggered event is emitted and the receiver is skipped in the distribution.

6.5 MAX_CONTROLLERS Mismatch Between ControllerFactory and FeeSplitter

Design Low Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-004

update_controller iterates over the new controllers added to the factory since the last time it was called and updated the ControllerMulticlaim state with the new controllers. This is done with:

```
for i: uint256 in range(new_len - old_len, bound=MAX_CONTROLLERS):
    i_shifted: uint256 = i + old_len
    c: Controller = Controller(staticcall factory.controllers(i_shifted))
    self.allowed_controllers[c] = True
    self.controllers.append(c)
```

However, if more than MAX_CONTROLLERS controllers were added to the factory, the call will revert as the loop's bound will be reached. This mean that it will be impossible to update the ControllerMulticlaim with the new controllers afterwards. This could happen as the ControllerFactory allows in theory up to 50000 controllers.

Code corrected

MAX_CONTROLLERS was updated to 50000 to match the ControllerFactory.

6.6 Unused Interface

Informational Version 2 Code Corrected

CS-CURVE_FEE_SPLITTER-011

IDynamicWeight.vyi is imported in FeeSplitter but never used. It could be removed.

Code corrected

IDynamicWeight.vyi is no longer imported in FeeSplitter.

6.7 Unused Event

Informational Version 1 Code Corrected

CS-CURVE_FEE_SPLITTER-007

In FeeSplitter, the event SetWeights is defined but never used.

Code corrected

The event was removed from the contract.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Loop Can Run Out of Gas

Informational Version 2 Acknowledged

CS-CURVE_FEE_SPLITTER-010

In the case `update_controllers()` is not called for a long time, and that a large number of controllers have been deployed by the controller factory, it could be that calling `update_controllers()` is too costly and would always revert. While it could be partially mitigated with access lists, there exists an amount of controllers that would make the function revert always.

Acknowledged

Curve acknowledged the behavior and precised that the factory's controllers creation is under the control of the DAO for minting markets.

7.2 Gas Savings

Informational Version 1 Code Partially Corrected

CS-CURVE_FEE_SPLITTER-006

The following gas optimization could be made:

1. In `FeeSplitter.dispatch_fees()`, several additions and subtractions could be made unsafe.
2. In `FeeSplitter._set_receivers()`, the weights addition could be made unsafe.
3. In `ControllerMulticlaim.update_controller()`, the `for` loop could be replaced with a loop with the following pattern to avoid for computing `i_shifted` at each iteration:

```
for i in range(start, end, bound=N):
    ...
```

Code partially corrected

The `for` loop in `update_controller()` was replaced with a loop with the pattern described above. The other optimizations were not implemented.

7.3 `_is_dynamic` Does Not Follow EIP-165

Informational Version 1 Acknowledged

CS-CURVE_FEE_SPLITTER-008

The function _is_dynamic does not follow the step described in EIP-165 to verify that the callee implements EIP-165.

This includes ensuring that:

1. contract.supportsInterface(0x01ffc9a7). does not revert and returns true.
2. contract.supportsInterface(0xffffffff). does not revert and returns false.

The current checks are not sufficient to ensure that the callee implements EIP-165 and the Dynamic weight interface as it could be a fallback function returning always true for example. The additional checks does not rules any possibility of a false positive but decrease greatly the probabilities. Another option that could be considered would be something similar to what is being used in EIP-3156 where the called function must return some magic value.

Acknowledged

Curve acknowledged the behavior.

7.4 dispatch_fees Can Leave Dust

Informational Version 1 Acknowledged

CS-CURVE_FEE_SPLITTER-009

The function FeeSplitter.dispatch_fees might not transfer the entire balance of crvUSD to receiver due to always rounding down when applying the weights. The dust will remain the contract and can be used the next time the function is called.

Acknowledged

Curve acknowledged the behavior and documented it in the contract.