PUBLIC

Code Assessment

of the scrvUSD Smart Contracts

December 03, 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 scrvUSD according to Scope to support you in forming an opinion on their security risks.

Curve adopted Yearn's vault to distribute rewards to crvUSD holders that deposit their tokens in the vault. The rewards' origin from fees generated by Curve's stablecoin system. If the vault registers a profit, the profit is paid to the users over time by issuing shares to the vault backed by the profit and burning these shares over time.

Throughout the engagement, the communication and cooperation with the Curve and Yearn teams were excellent. The Curve team was responsive and provided the necessary information to conduct the audit efficiently. Besides the audit we also supported the Curve team with questions and feedback on the codebase.

The general subjects covered were proper use of Yearn vault, access control, and correct accounting. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security. Yet, 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	1
• Code Corrected	1
Low-Severity Findings	2
• Acknowledged	2

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 scrvUSD repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

scrvusd

V	Date	Commit Hash	Note
1	18 October 2024	11c78c41d540f216e5064182ebafc9fffd6c21bb	Initial Version
2	04 Novembre 2024	1bb64cfb92513c388df7f1c4d4de87d7bcd438e2	Version 2

snekmate

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

For the Vyper smart contracts, The compiler versions `0.3.7` and `0.4.0` were chosen.

scrvusd

All files in the `contracts` directory were considered in scope for the assessment.

For the contracts in the `contracts/yearn` directory, only changes with the yearn vault v3 repository as of commit `953f8b663ed2658c9cc937d380e3b6beefdecd18` were considered in scope as the latter is assumed correct in the scope of this assessment.

snekmate v0.1.0

- `src/snekmate/auth/access_control.vy`
- `src/snekmate/auth/interfaces/IAccessControl.vyi`

2.1.1 Excluded from scope

Any code or files not included in the `contracts` directory were excluded from the scope of the audit.

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 adopted Yearn's V3 vault contract to allow users to deposit their crvUSD tokens and earn rewards. The rewards originate from Curve's stablecoin markets fees. The fees are collected from the controllers by the `FeeSplitter` contract, out of scope for this review, and later forwarded to the `RewardsHandler`, one of the `FeeSplitter`'s recipient. Anyone can call

`RewardsHandler.process_rewards()` to forward the contract's crvUSD balance to the vault and trigger the vault's accounting (`vault.process_report()`).

The share of fees, the `FeeSplitter` forward to the `RewardsHandler` is capped by some admin defined bound, but is dynamically calculated in `RewardsHandler.weight()`. The weight is proportional to the ratio between the crvUSD balance of the vault and the token's circulating supply. This effectively awards more reward to the vault when more crvUSD is deposited in it. While anyone can take a snapshot of the ratio, to mitigate manipulation, the value is subject to a time weighted average (TWA) computed in the `TWA` module. Snapshots can only be added if the minimum time interval (`min_snapshot_dt_seconds`) has elapsed.

After the vault has accounted for the rewards, it will release them linearly over a specified time period to avoid sudden price per share spikes. The rewards are distributed to the vault's shareholders by inflating the value of the shares. The distribution process is unique in that the vault will issue to itself shares for the rewards and, over time, burn(unlock) these shares, thereby releasing the rewards to the shareholders. Similarly, the vault may issue shares for fees and protocol fees to the `accountant` and `protocol_fee_recipient`. Another unique feature of this vault version is that the vault itself can act as a strategy. Curve intends to use the Vault without registered strategies. The vault will be the only strategy and therefore is not expected to make any losses. However, in this section, the system will be described for the generic use case too.

One of the vault's most important state variables is `total_idle` (and `total_debt` in the case of enabled strategies). It accounts for the assets the Vault received from strategies or users. Simultaneously, `total_idle` is comparable with the `total_debt` of a strategy for the vault (if the vault is its own strategy). To account for any increase (or rare decrease) in `total_idle`, the vault will issue shares to itself (and fee receivers). Over time, these shares will be burned to give the assets to the remaining vault shareholders.

In the following sections, we detail the contracts and their functionalities.

2.2.1 Vault Factory

The `VaultFactory` contract implements a factory that can deploy vaults in a permissionless manner. Anyone can call `deploy_new_vault` to create a vault using `create_minimal_proxy_to` to create an EIP-1167 proxy to the vault implementation. Multiple vaults can be deployed for a single underlying token. Addresses holding the `governance` role can shut down the factory to stop new deployments, set the protocol fee to be used by deployed vaults, or update the fee recipient. The transfer of the governance role can be done with `transferGovernance()` and `acceptGovernance()` in a two-step process.

Unpermissioned functions:

- `deploy_new_vault()`: Anyone can call this to deploy a new vault.

Roles and their privileged functions:

- `governance`:
  - `shutdown()`: To stop new deployments.
  - `setProtocolFee()`: Set the protocol fee for vaults.
  - `updateFeeRecipient()`: Update the fee recipient for vaults.
  - `transferGovernance()`: Transfer the governance role.
  - `acceptGovernance()`: Accept the governance role transfer.

2.2.2 Vault

The system uses Yearn Vault v3.0.3 to implement scrvUSD. Although Yearn's vault is designed to be modular and can be used with various external strategies, strategies are not intended to be used in this system. Instead, rewards are directly sent to the vault by the `RewardsHandler` and can be accounted for by having the vault report on itself. This means that the strategy and debt management parts of the vault are not used in this system, and it is expected that no strategies will be added to the vault.

The vault can be split into several "modules":

2.2.2.1 Role Management

Several roles are defined in the vault:

- `role_manager`, which can assign or revoke roles to addresses.
- `ADD_STRATEGY_MANAGER`, which can add strategies to the vault.
- `REVOKE_STRATEGY_MANAGER`, which can remove strategies from the vault.
- `FORCE_REVOKE_MANAGER`, which can force-remove a strategy causing a loss.
- `ACCOUNTANT_MANAGER`, which can set the accountant that assesses fees.
- `QUEUE_MANAGER`, which can set the default withdrawal queue.
- `REPORTING_MANAGER`, which can call reports for strategies or for the vault itself.
- `DEBT_MANAGER`, which can add and remove debt from strategies.
- `MAX_DEBT_MANAGER`, which can set the max debt for a strategy.
- `DEPOSIT_LIMIT_MANAGER`, which can set deposit limits and modules for the vault.
- `WITHDRAW_LIMIT_MANAGER`, which can set the withdraw limit module.
- `MINIMUM_IDLE_MANAGER`, which can set the minimum total idle the vault should keep.
- `PROFIT_UNLOCK_MANAGER`, which can set the profit_max_unlock_time.
- `DEBT_PURCHASER`, which can purchase bad debt from the vault.
- `EMERGENCY_MANAGER`, which can shut down the vault in an emergency.

The `role_manager` can assign or revoke roles using the functions `set_role()`, `add_role()` and `remove_role()`. Additionally, the role manager role can be transferred using the functions `transfer_role_manager()` and `accept_role_manager()`.

2.2.2.2 Share Management

This module manages the vault's shares, which represent user ownership of the vault's assets. The vault uses an ERC-20-like mechanism to issue and transfer shares to depositors based on their contributions. The key functions are:

- `deposit` and `mint` allow receiving vault shares for depositing assets. The caller can specify an alternative receiver for the shares. In case `auto_allocate` is not set, assets will sit idle in the vault (tracked by `total_idle`) until invested (if strategies are enabled) or redeemed/withdrawn (if the default behavior with the vault as its own strategy is followed).
- `withdraw` and `redeem` enable users to take their share of assets out of the vault. If strategies are present, triggering these functions might cause losses that the user needs to cover.

Shares are ERC20 tokens and can be transferred, and approvals can be given to transfer from. Certain view functions inform users about the current share-to-asset ratio (e.g., `convertToShares` and `convertToAssets`). In the case of profits, the amount is converted to shares and credited to the vault. Over time, the vault will burn its shares and thereby release the assets to all remaining shareholders (if the `profit_max_unlock_time` is non-zero).

2.2.2.3 Setters

Various setters are available to privileged roles in the vault:

- `setName()`: Can be called by the `role_manager` to set the name of the vault.
- `setSymbol()`: Can be called by the `role_manager` to set the symbol of the vault.
- `set_accountant()`: Can be called by the `ACCOUNTANT_MANAGER` to set the accountant.
- `set_default_queue()` and `set_use_default_queue()`: Can be called by the `QUEUE_MANAGER` to set the default withdrawal queue and enable or disable forcing the use of the default queue.
- `set_auto_allocate()`: Can be called by the `DEBT_MANAGER` to enable or disable automatic allocation of funds to the first strategy in the queue when depositing.
- `set_deposit_limit()`: Can be called by the `DEPOSIT_LIMIT_MANAGER` to set the deposit limit of the vault if no deposit limit module is used.
- `set_deposit_limit_module()`: Can be called by the `DEPOSIT_LIMIT_MANAGER` to set the deposit limit module of the vault.
- `set_withdraw_limit_module()`: Can be called by the `WITHDRAW_LIMIT_MANAGER` to set the withdraw limit module of the vault.
- `set_minimum_total_idle()`: Can be called by the `MINIMUM_IDLE_MANAGER` to set the minimum total idle assets the vault should keep.
- `setProfitMaxUnlockTime()`: Can be called by the `PROFIT_UNLOCK_MANAGER` to set the profit maximum unlock time, used to smooth profit distribution and avoid bumps in the price per share.

2.2.2.4 Strategy Management

Although this is not intended to be used in scrvUSD, strategies can be added and removed from the vault:

- `add_strategy()`: Can be called by an `ADD_STRATEGY_MANAGER` to add a strategy to the vault. If `add_to_queue` is set to true, the strategy is added to the `default_queue`, meaning that during withdrawal processing, assets might be withdrawn from the strategy if the vault idle assets and strategies with lower indexes in the queue cannot cover the withdrawal.
- `revoke_strategy()`: Can be called by a `REVOKE_STRATEGY_MANAGER` to remove a strategy from the vault. If the strategy is in the default queue, it is removed from the queue. The current debt of the strategy must be 0.
- `force_revoke_strategy()`: Can be called by a `FORCE_REVOKE_MANAGER` to remove a strategy from the vault. The current debt of the strategy can be non-zero.

2.2.2.5 Debt Management

The debt management module focuses on handling the allocation and modification of debt levels for strategies within the vault. It is not relevant in the default use case where no external strategies are used, and the vault serves as its own "strategy". The following functions are available:

- `update_max_debt_for_strategy()`: Sets the maximum debt level for a given strategy. Can only be called by the `MAX_DEBT_MANAGER` role.
- `buy_debt()`: Allows a governance role to purchase bad debt from the vault in emergencies. Can only be called by the `DEBT_PURCHASER` role. *(It is listed under reporting management in the code base)*
- `update_debt`: Rebalances the debt of a strategy based on specified target levels. It will try to invest or close investments to maintain the correct target debt levels for a given strategy. Can only be called by the `DEBT_MANAGER` role.

2.2.2.6 Reporting Management

The most important functionality to account for gains and losses and adjust the vault's shares is done via the function `process_report`. The function will check if the vault received assets and account for these assets as gains, query an `Accountant` contract for additional information on fees and refunds, and finally check if the given strategy made gains or losses since the last report. If no strategy is provided, it will check internally that any changes in the value of assets held within the vault are properly accounted for. When a profit is realized, the vault does not immediately make all profits available. Instead, it implements a profit-locking mechanism. This locks the profits and gradually unlocks them over a set period, known as the `profit_max_unlock_time`. This approach prevents sudden spikes in the vault's price per share, maintaining stability for all depositors and mitigating MEV. In case, losses were reported, the vault burns an amount of its existing shares proportional to the realized losses, adjusting the remaining shares held by depositors accordingly. However, there are checks to ensure that the vault does not burn more shares than it holds, if losses were to be too large to be covered by the locked profits, the price per share would have a sudden drop and users shares would lose value. If a strategy was involved in generating the loss, its debt levels are also updated accordingly to reflect the reduction in assets.

2.2.3 Periphery

2.2.3.1 Deposit Limit Module

The `DepositLimit` module is intended to be used by the Vault to enforce a deposit limit.

The `admin` role can:

- Assign or revoke the admin role to addresses using `set_admin()`.
- Assign or revoke the security agent role to addresses using `set_security_agent()`.
- Set a maximum deposit limit to be used by the Vault using `set_deposit_limit()`.

The `security agent` role can:

- (Un)pause deposits using `set_deposits_paused()`.

The function `available_deposit_limit()` is a view function intended to be called by the vault that returns the available deposit limit.

- If the deposits are paused, the function will return 0.
- Otherwise,
  - If the deposit limit is `max_value(uint256)`, the function returns `max_value(uint256)`.
  - Otherwise, it returns the difference between the deposit limit and the `totalAssets()` of the vault.

2.2.3.2 Stablecoin Lens

The `StablecoinLens` contract only has a single view function `circulating_supply()` that computes the circulating supply of crvUSD. This contract is needed since crvUSD's `totalSupply()` is incorrect as it takes into account all minted crvUSD, including those in the controllers or flashloans. The circulating supply is computed as the sum of the controllers' debts plus the sum of the peg keepers' debts.

2.2.3.3 Rewards Handler

The `RewardsHandler` is used to distribute rewards for scrvUSD. Any crvUSD sent to this contract is considered donated as rewards for depositors and will not be recoverable. This contract can receive funds to be distributed from the `FeeSplitter` (crvUSD borrow rate revenues) as well as other sources. The amount of funds that this contract should receive from the `FeeSplitter` is determined by computing the time-weighted average of the crvUSD vault balance over the token's circulating supply ratio.

The contract handles the rewards in a permissionless manner; anyone can take snapshots of the TVL and distribute rewards. In case of manipulation of the time-weighted average, the contract allows the `RATE_MANAGER` to correct the distribution rate of the rewards.

The permissionless entry points of the contract are:

- `take_snapshot()`: Takes a snapshot of the current ratio between the deposited crvUSD in the vault and the circulating supply of crvUSD and updates the time-weighted average.
- `process_rewards()`: Allows anyone to distribute the rewards to the crvUSD vault by sending the contract's crvUSD balance to the vault before calling the vault's `process_rewards()` function.

The following functions are restricted to the `RATE_MANAGER` role:

- `set_twa_snapshot_dt()`: Sets the minimal frequency for taking the time-weighted average snapshots.
- `set_twa_window()`: Sets the time window for the time-weighted average.
- `set_distribution_time()`: Function used to correct the distribution rate of the rewards. The distribution time affects the time it takes for the vault to stream the rewards and can be used to mitigate manipulation of the time-weighted average.
- `set_minimum_weight()`: Updates the minimum weight that the contract will request from the `FeeSplitter`.
- `set_scaling_factor()`: Updates the scaling factor used in the weight calculation.

The following function is restricted to the `LENS_MANAGER` role:

- `set_stablecoin_lens()`: Updates the address of the stablecoin lens to use in computing the circulating supply of crvUSD.

The following function is restricted to the `RECOVERY_MANAGER` role:

- `recover_erc20()`: Can be used to rescue any ERC-20 token that was sent to the contract by mistake. crvUSD is not recoverable.

Roles and access control can be managed with the functions `grantRole()`, `revokeRole()`, `renounceRole()` and `set_role_admin()`.

2.2.4 Trust model and roles

The following roles are assumed to be trusted to behave honestly and correctly at all times:

- The governance of `VaultFactory`.
- The `role_manager` and all other roles of the `VaultV3`.
- The `admin` and `security_agent` roles of the `DepositLimit`.
- The `DEFAULT_ADMIN_ROLE`, `RATE_MANAGER`, `RECOVERY_MANAGER` and `LENS_MANAGER` of the `RewardsHandler`.

The `RewardsHandler` is expected to be given both the `Vault`'s `REPORTING_MANAGER` and `PROFIT_UNLOCK_MANAGER` roles.

The Vault is expected to have no external strategies.

As vaults are created in a permissionless way, it is expected that only the legitimate vaults are used by the users. Namely, the underlying token should be ERC-20 compliant without behaviors such as double entry points, rebase mechanisms, transfer fees, irrational return values, or high decimals. Each token should carefully be tested before used.

The amount to be received by the `RewardsHandler` from the `FeeSplitter` can be influenced by the proportion of the crvUSD supply that is in the Vault. Although manipulations are partially mitigated by a time-weighted average, it is still possible to manipulate the rewards by sandwiching TWA snapshots with large deposits or withdrawals of crvUSD in the Vault. This is known and accepted by the Curve team; in this audit, it is not considered a vulnerability.

2.2.5 Changes in Version 2

In Version 2, the `DepositLimitModule` was deprecated and removed from the codebase. Other than that, no major changes were made to the contracts, except for fixing the issues found during the audit.

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	2

5.1 Incorrect Event

Correctness Low Version 2 Acknowledged

CS-CURVE_SCRVUSD-015

In `_process_report()`, in case the vault is reporting on itself, and the following conditions are met:

- No gains nor losses were accrued.
- The accountant emitted some non-zero refunds.

The `StrategyReported` event will incorrectly report the `current_debt` as it will not include the refunds, although they were added to `total_idle`.

Acknowledged

Curve acknowledged the issue.

5.2 `previewDeposit()` Does Not Adhere to EIP-4626

Design Low Version 1 Acknowledged

CS-CURVE_SCRVUSD-002

When calling `previewDeposit()` with `max_value(uint256)`, `max_value(uint256)` is returned. However, doing the same for `deposit()` will mint shares for the asset balance of the depositor. Meanwhile, according to EIP-4626, `previewDeposit()`

```
MUST return as close to and no more than the exact amount of Vault shares that would
be minted in a deposit call in the same transaction. I.e. deposit should return the
same or more shares as previewDeposit if called in the same transaction.
```

Acknowledged

Curve is aware of the issue and decided to keep the code unchanged.

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	1

- Variable Override Code Corrected

Low-Severity Findings	0
Informational Findings	3

- Function Could Be Marked as Pure Code Corrected
- Misleading Documentation Code Corrected
- Missing Events Code Corrected

6.1 Variable Override

Correctness Medium Version 1 Code Corrected

CS-CURVE_SCRVUSD-001

In case `total_refunds` is non-zero in `VaultV3._process_report()` for the case that the vault is reporting itself, `self.total_idle` is increased by `total_refunds` but later overridden by setting it to `total_assets`. This means that when calling `process_report(self)` on a vault with both a gain and a non-zero `total_refund` the vault `total_idle` balance will be updated incorrectly.

When `total_refunds` is non-zero, shares are minted (or not burned if there were shares to be burned) for that refund and are supposed to be locked.

```
# Get the amount we will lock to avoid a PPS increase.
if gain + total_refunds > 0 and profit_max_unlock_time != 0:
    shares_to_lock = self._convert_to_shares(gain + total_refunds, Rounding.ROUND_DOWN)
```

Later, the refunds are pulled from the accountant and `total_idle` is updated given the new shares have to be backed by assets in `total_idle` to keep the `pps` constant (as the share are to be locked):

```
# Pull refunds

if total_refunds > 0:
    # Transfer the refunded amount of asset to the vault.
    self._erc20_safe_transfer_from(_asset, accountant, self, total_refunds)
    # Update storage to increase total assets.
    self.total_idle += total_refunds
```

However, if there were gain, `total_idle` is overwritten with `total_assets` which is the balance of the contract before pulling the refunds.

```
# Record any reported gains.

if gain > 0:
    # |no|: this will increase total_assets
    current_debt = unsafe_add(current_debt, gain)
    if strategy != self:
        self.strategies[strategy].current_debt = current_debt
        self.total_debt += gain
    else:
        self.total_idle = total_assets
```

This means that shares were minted but were not backed by assets, meaning that the `pps` is smaller than it should be, although there was no loss, and if the shares to be unlocked is small it could even be that the `pps` is decreasing.

Code corrected:

`total_idle` is now accounted correctly.

6.2 Function Could Be Marked as Pure

Informational **Version 1** **Code Corrected**

CS-CURVE_SCRVUSD-004

In `VaultV3`, `apiVersion()` is a view function, but it could be marked as `pure`.

Code corrected:

The function was declared `pure`.

6.3 Misleading Documentation

Informational **Version 1** **Code Corrected**

CS-CURVE_SCRVUSD-006

1. In `VaultV3`, the function `set_role()`, `add_role()` and `remove_role()` are misleading as they suggest to only be able to set, add or remove a role, respectively, although they can in theory add or remove multiple roles at once.
2. In `VaultV3`, `update_debt()`'s comment states that it returns the amount of debt added or removed, while it returns the new amount of debt for the given strategy.
3. In `StablecoinLens`, the documentation of `_circulating_supply()` states the following, although the internal function is exposed via the external function `circulating_supply()`:

   ```
   This function is not exposed as external as it can be easily manipulated and
   should not be used by third party contracts.
   ```

Code corrected:

All documentation has been updated to reflect the actual behavior of the functions.

6.4 Missing Events

Informational | Version 1 | Code Corrected

CS-CURVE_SCRVUSD-007

Several state changing functions do not emit events.

1. All functions in `DepositLimitModule`.
2. `VaultV3.transfer_role_manager()`.
3. `VaultV3.shutdown_vault()` does not emit the `RoleSet` event, although it changes the roles of the message sender.
4. `setName()` and `setSymbol()` in the `VaultV3`.

Code corrected:

1. The `DepositLimitModule` contract was removed from the codebase.
2. An event `UpdateFutureRoleManager` is now emitted.
3. Does emit the `RoleSet` now.
4. Curve informed us this is intentional.

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 Choice of `profit_max_unlock_time`

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-014

In case the vault's `profit_max_unlock_time` is not carefully chosen and too small, it could be that users are able to game the profit distribution by exiting the vault when no profit is being distributed and rejoin the vault after the `RewardsHandler` deposited new profits. Hence, the variable should be carefully chosen to mitigate this behavior.

Acknowledged

Curve acknowledged the issue.

7.2 Code Style

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-003

The codebase uses both snake case and sometimes camelCase for function names. While camelCase is required for function defined in EIPs like EIP20 or EIP4626. Some functions in the codebase are defined in snake_case like `set_auto_allocate` while others are defined in camelCase like `setName` although they are not defined in EIPs.

Acknowledged

The Yearn team anwsered:

```
Camel case is used for both standardized functions from EIP's as well
as functions who are mirrored in the V3 TokenizedStrategy interface to assure
consistency across both vault implementations. Then snake_case is used for functions
only used in the VaultV3 contracts.
```

7.3 Gas Optimizations

Informational **Version 1** **Code Partially Corrected**

CS-CURVE_SCRVUSD-005

The following optimizations were identified in the codebase:

DepositLimitModule

1. In `available_deposit_limit()`, `max_deposit_limit` is read from the storage three times in certain cases, it could be cached.

2. In `available_deposit_limit()`, `self.max_deposit_limit - vault_balance` could be unchecked.

TWA

1. In `_take_snapshot()`, the two conditions in `len(self.snapshots) == 0` and `self.last_snapshot_timestamp + self.min_snapshot_dt_seconds <= block.timestamp` could be swapped to avoid evaluating the first condition in most cases.
2. In `_compute()`, except for the first one, each iteration reads `self.snapshots[i_backwards + 1]` from storage, although it was already read in the previous iteration. It could be cached.
3. In `_compute()`, `len(self.snapshots)` is read twice from storage, it could be cached.

StablecoinLens

1. In `_circulating_supply()`, `factory.controllers(3)` is called twice, once to query its monetary policy, and once for its total debt, it could be cached.

VaultV3

1. In `_revoke_strategy()`, `loss` could be declared inside the `if` statement to save in memory expansion cost.
2. In both `add_role()` and `remove_role()`, `self.roles[account]` is loaded from storage twice, it could be cached.
3. Some state variables that are anyway always used together like `full_profit_unlock_date` and `last_profit_update` could be packed.
4. In `_redeem()`, the subtraction in `current_total_idle += (assets_to_withdraw - loss)` could be made unsafe.
5. In `buy_debt()`, `self.strategies[strategy].current_debt -= _amount` load the debt from storage while it was already cached in memory earlier in the function.
6. In `buy_debt()`, when logging the `DebtUpdated` event, the subtraction `current_debt - _amount` could be made unsafe.
7. In `_update_debt()`, the subtraction in the following code could be made unsafe:

```
if withdrawn < assets_to_withdraw and max_loss < MAX_BPS:
    # Make sure the loss is within the allowed range.
    assert assets_to_withdraw - withdrawn <= assets_to_withdraw * max_loss / MAX_BPS, "too much loss"
```

Code partially corrected:

- The `DepositLimitModule` was removed from the codebase.
- `TWA` optimizations were implemented except for 2.
- `StablecoinLens` optimizations were not implemented to improve readability.
- All gas optimizations were adopted for the `VaultV3` except no. 3.

7.4 Missing Sanity Checks

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-008

- In `DepositLimitModule`, the constructor and all setters do not perform any sanity checks on the provided arguments.

- In `StablecoinLens`, the constructor does not perform any sanity checks on the `factory`.
- In `RewardsHandler`, the constructor does not perform any sanity checks on the `_stablecoin`, `_vault` and `admin` addresses.
- In `VaultV3`, `update_debt` does not check if the provided strategy is a valid strategy, although providing an invalid strategy should have no effect in the current system.

Acknowledged

Curve acknowledged the issue.

7.5 Missing `implements` Statement

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-009

The `StablecoinLens` contract could implement the `IStablecoinLens` interface to ensure that the implementation matches the interface.

Acknowledged

Curve acknowledged the issue.

7.6 Spamming Snapshots

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-010

The `RewardsHandler()` is initialized with a TWA window of 1 week and a minimum time delta between snapshot of 600 seconds. Spamming snapshots every 600 seconds is possible, although costly in gas. That would mean creating `1008` measurements in a week. Simulations show that this would lead calls to `RewardsHandler.weight()` to consumes between 5 and 6 million gas. Care should be taken when modifying either the `twa_window` or the `min_snapshot_dt_seconds` as larger window or smaller time delta would increase the maximum gas cost of the function, which it could lead to a situation where the gas cost of the function is too high regarding the gas limit of a block, leading to a denial-of-service attack on the system by spamming snapshots.

Acknowledged

Curve acknowledged the issue.

7.7 `StablecoinLens` High Gas Cost

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-011

The function `StablecoinLens._circulating_supply()` will require a high amount of gas, which is proportional to the amount of controller and pegkeeper in the system.

- Given that the function is called when taking a snapshot via the `RewardsHandler`, it could be that users have not enough incentive to take snapshots, as they don't own enough shares of the vault, and/or the rewards from the stablecoin markets are too low to cover the gas cost.
- As the number of controllers and pegkeepers increases, it could be that the gas cost of the function is too high regarding the gas limit of a block.

As such, care should be taken that there are enough incentives for users to take snapshots, and that the amount of controllers and pegkeepers is such that the gas cost of the function is acceptable.

Acknowledged

Curve acknowledged the issue.

7.8 `_update_debt()` Does Not Check That the Strategy Is Valid

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-012

In `_update_debt()`, the function does not check that the strategy to update the debt for is valid. While it is not an issue in the current implementation as the invalid strategy's will have a null `max_debt` and `current_debt` values, it is a good practice to check that the strategy is valid before updating the debt.

Acknowledged

Curve acknowledged the issue.

7.9 `permit` Will Always Succeed for Address 0X29

Informational **Version 1** **Acknowledged**

CS-CURVE_SCRVUSD-013

Due to a known vulnerability in Vyper v0.3.7, the ecrecover precompile does not fill the output buffer if the signature does not verify, and instead some dirty bytes could be read.

In the case of the `VaultV3`, the dirty bytes happen to be `0x29`, the storage slot of the `nonces` HashMap. Meaning that calling the `permit` function with this as owner will always succeed, independently of the provided signature.

As such, `0x0000000000000000000000000000000000000029` should never be used as some special address, such as a burn address, given that anyone could drain the vault's shares sent to it.

Acknowledged

Curve acknowledged the issue.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Accountant Reports

Note Version 1

In `VaultV3._process_report()` the accountant might report `total_fees`. The fees are not a percentage but an absolute value. If this value exceeds the profits, the fees could reduce the price per share for users. Which user might circumvent by front-running the report processing.

8.2 Fee Share Distributed Might Be Worth Less or More Than the Intended Fee

Note Version 1

Fees are paid in shares not assets at the beginning of the report processing. Hence, in case the function later changes the price per share, the fee shares might be worth more or less than the intended fees to be taken.

1. In case of `profit_max_unlock_time == 0`, the profits are not locked and the price per share increases immediately, this means that if `X` assets were asked as fees, and the fee receiver received `Y` shares, immediately after the call to `process_rewards`, `convert_to_share(X) > Y` and the receiver would have received more than the intended fees.
2. Similarly, if there was to be losses that could not be covered by the locked profits, the price per share would decrease, and the fees would be worth less than the intended fees.

8.3 Fees Accumulate Compounded Interest

Note Version 1

Currently, fees are calculated based on an absolute amount of assets asked by the accountant. But shares are minted for these fees. Consequently, these shares will participate and get another share of the next reward payments. The amount paid in fees is therefore not a constant percentage but compounding over time.

8.4 Manipulation of Weight Supply

Note Version 1

The snapshots made by calling `RewardsHandler.take_snapshot()` can easily be manipulated by taking large loans, making large deposits in or withdraw from the vault. Although the time weighted average partially mitigates this issue, it should be noted that malicious actors can perform all aforementioned actions with flashloans. This means that in theory, that by manipulating snapshots over time, the malicious actor will control more and more of the TWA. This is known and acknowledged by the Curve team, and the function `set_distribution_time()` can be used to mitigate an ongoing attack.