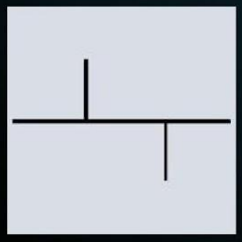
THREE SIGMA

Tokemak v2-core

Security Review



Disclaimer

Security Review

Tokemak v2-core

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Security Review

Tokemak v2-core

Table of Contents

Summary

Security Review

Tokemak v2-core

Summary

Three Sigma audited Tokemak in a 1.6 person week engagement. The audit was conducted from 21/07/2025 to 09/08/2025.

Protocol Description

Tokemak Protocol is a decentralized finance protocol that operates as a liquidity management system, where users deposit assets into Autopools (ERC-4626 compatible vaults that act as yield-generating indices) which automatically deploy capital across multiple DeFi destinations to optimize returns. The protocol uses Destination Vaults as standardized interfaces that wrap external protocols (like Curve, Balancer, Silo) to abstract their complexities while enabling reward collection and liquidation.

The audit scope is composed by three key components:

- SiloRewardLib - a comprehensive library managing gauge whitelisting and reward collection with configurable filtering strategies (all tokens, extra rewards only, or liquidation tokens only).
- SiloDestinationVault - the base vault that extends ERC4626 functionality to automatically collect and liquidate all Silo rewards from whitelisted gauges.
- SiloMainnetDestinationVault - a specialized mainnet implementation that splits reward handling by routing configured "extra reward tokens" to ExtraRewarders for distribution while sending remaining rewards for liquidation, enabling more sophisticated reward management strategies on mainnet deployments.

Scope

Security Review

Tokemak v2-core

Scope

Filepath	nSLOC
src\destinations\adapters\rewards\SiloRewardLib.sol	168
src\vault\SiloDestinationVault.sol	67
src\vault\SiloMainnetDestinationVault.sol	56
src\utils\SiloVaultWrapper.sol	184
src\utils\TokenRecoverable.sol	40
Total	**515**

Methodology

Security Review

Tokemak v2-core

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit, we classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes the general expected classification according to impact and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Impact / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Security Review

Tokemak v2-core

Project Dashboard

Application Summary

Name	Tokemak v2-core
Repository	https://github.com/Tokemak/v2-core
Commit	7accb69
Language	Solidity
Platform	Ethereum, Base, Sonic

Engagement Summary

Timeline	21/07/2025 to 09/08/2025
№ of Auditors	2
Review Time	1.6 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	3	3	0
None	1	1	0

Category Breakdown

Suggestion	0
Documentation	0
Bug	3
Optimization	0
Good Code Practices	1

Risk Section

Security Review

Tokemak v2-core

Risk Section

- **Partial Audit Scope**: The audit covers only specified contracts within the repository. Vulnerabilities in contracts outside this scope could introduce risks, potentially impacting overall system security.
- **Dependency on External Contracts**: The system relies on external contracts. Although integrations were reviewed, vulnerabilities or issues within these external systems could still pose risks.
- **Administrative Key Management Risks**: The system makes use of administrative keys to manage critical operations. Compromise or misuse of these keys could result in unauthorized actions and financial losses.

Findings

Security Review

Tokemak v2-core

Findings

3S-Tokemak-L01

LiquidationRow.claimsVaultRewards() can revert permanently if a FoT token is added to a program of a gauge

Id	3S-Tokemak-L01
Classification	Low
Impact	Low
Likelihood	Low
Category	Bug
Status	Addressed in #456688b.

Description

The **claimsVaultRewards** function in the **LiquidationRow** contract requires that the reward amounts returned from **vault.collectRewards()** will exactly match the actual token balances received by the contract. However, this assumption breaks in the case where a FoT token is used as a reward token in any of the incentive programs associated with a whitelisted gauge. If a new program is added to a gauge with such a token, the **collectRewards** function in **SiloRewardLib** will successfully claim the reward and transfer it to the **LiquidationRow**, but due to the transfer fee, the actual received balance will be lower than the expected amount.

This discrepancy causes **_increaseBalance** to revert with **Errors.InsufficientBalance**, since the updated total tracked balance exceeds the actual ERC20 balance.

```
    function _increaseBalance(address tokenAddress, address vaultAddress, uint256
balance) internal {
        Errors.verifyNotZero(balance, "balance");
        uint256 currentBalance = balances[tokenAddress][vaultAddress];
        uint256 totalBalance = totalTokenBalances[tokenAddress];
        uint256 newTotalBalance = totalBalance + balance;
        // ensure that this contract has enough balance to cover the new total balance
        uint256 balanceOfToken = IERC20(tokenAddress).balanceOf(address(this));
        if (newTotalBalance > balanceOfToken) {
            /**
```

```
            * @dev This should never happen, but just in case. The error is raised if the
updated total balance of a
            * specific token in the contract is greater than the actual balance of that token
held by the
            * contract.
            * The calling contract should transfer the funds first before updating the
balance.
            */
            revert Errors.InsufficientBalance(tokenAddress);
        }
```

As a result, **claimsVaultRewards** fails and no rewards are recorded, not even those from other valid programs. The issue is compounded by the fact that the destination vaults provides no mechanism to skip specific failing programs or tokens. Removing the entire gauge may not be feasible either, especially if it holds other not weird reward tokens. Ultimately, this results in a persistent denial of service for reward collection from the affected gauge.

Recommendation

A possible solution is to implement support for filtering out specific programs or to adjust the logic to properly handle FoT reward tokens.

3S-Tokemak-L02

Potential Dos due to dust in **siloVaultShares**

Id	3S-Tokemak-L02
Classification	Low
Impact	Low
Likelihood	Low
Category	Bug
Status	Addressed in #456688b.

Description

SiloVaultShares stores the total number of shares returned by Silo upon deposit, which are held in the contract. Due to rounding down, the shares added to **SiloVaultShares** during a **deposit** may be more than those subtracted during a **redeem**, which can result in dust remaining in **SiloVaultShares**.

This dust can be withdrawn via **recover**, for example, at a time when there are no active deposits. However, this should not be done because the value of **siloVaultShares** will not decrease, only the balance will. This will lead to a DoS of rewards in **claimRewardsToSender**.

```
 if (rewardToken != address(0)) {
        uint256 amount = IERC20(rewardToken).balanceOf(address(this));
        // Don't want to be able to siphon our actual holdings
        if (rewardToken == vaultToken) {
            amount -= siloVaultShares;
        }
        if (amount > 0) {
            SafeERC20.safeTransfer(IERC20(expectedRewardTokens[i]), msg.sender,
amount);
        }
    }
```

In that function, if the **rewardToken** is the same as the Silo vault token, **siloVaultShares** is subtracted from the amount. If the dust has been withdrawn from the contract and there are minimal rewards in the vault token, **amount - siloVaultShares** could be a negative value, causing the entire **claimRewardsToSender** function to revert.

Recommendation

No action is strictly required if the **recover** function is used carefully. However, a possible mitigation is to skip this reward token instead of reverting when **siloVaultShares >= amount**.

3S-Tokemak-L03

siloVaultDecimalOffset is not initialized

Id	3S-Tokemak-L03
Classification	Low
Impact	Low
Likelihood	High
Category	Bug
Status	Addressed in #456688b.

Description

The **siloVaultDecimalOffset** immutable variable in **SiloVaultWrapper** is declared but never initialized in the constructor, so its value will always be zero. Although this variable is not currently used within the contract, it is not an issue at the moment. However, if any external contract in the system reads this variable in the future, it could lead to problems.

Recommendation

Initialize siloVaultDecimalOffset in the constructor using the value from **ISiloVault(siloVault_).DECIMALS_OFFSET()**. This ensures the public variable accurately reflects the SiloVault's decimal offset.

3S-Tokemak-N01

Incorrect Storage Slot Calculation in SiloRewardLib

Id	3S-Tokemak-N01
Classification	None
Category	Good Code Practices
Status	Addressed in #7cf71d7.

Description

The storage slot calculation in **SiloRewardLib.sol** does not match the formula provided in the comment. The comment states the slot should be calculated as:

keccak256(abi.encode(uint256(keccak256("autopilot.storage.SiloRewardLib")) - 1)) & ~bytes32(uint256(0xff))

However, the actual SLOT constant is set to:

bytes32 private constant SLOT = 0x8b5e3b7c2a1d4f6e9c8b7a5d3e2f1c9b8a7d6f5e4c3b2a1d9f8e7c6b5a4d3e00;

When the formula from the comment is calculated correctly, it produces:

0xd777368586ae930ae191e530e87a4d24442db19a468df014057c22fc8e82aa00

The discrepancy suggests either the comment is outdated or the slot calculation was performed incorrectly during development.

Recommendation

Update the SLOT constant to match the calculation described in the comment:

// Before
bytes32 private constant SLOT =

```
0x8b5e3b7c2a1d4f6e9c8b7a5d3e2f1c9b8a7d6f5e4c3b2a1d9f8e7c6b5a4d3e00;
// After
bytes32 private constant SLOT =
0xd777368586ae930ae191e530e87a4d24442db19a468df014057c22fc8e82aa00;
```

Alternatively, if the current slot is intentionally different, update the comment to reflect the actual calculation method used.