Custodia Security

Sting Vault Review

Conducted By: Ali Kalout, Ali Shehab

Contents

1. Disclaimer

A smart contract security review cannot ensure the absolute absence of vulnerabilities. This process is limited by time, resources, and expertise and aims to identify as many vulnerabilities as possible. We cannot guarantee complete security after the review, nor can we assure that the review will detect every issue in your smart contracts. We strongly recommend follow-up security reviews, bug bounty programs, and on-chain monitoring.

2. Introduction

Custodia conducted a security assessment of Sting's smart contract ensuring its proper implementation.

3. About Sting

Sting is a yield optimization system consisting of vault contracts and strategy contracts. The system allows users to deposit funds which are then deployed to generate yield according to the active strategy.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1. Impact

- High: Results in a substantial loss of assets within the protocol or significantly impacts a group of users.
- Medium: Causes a minor loss of funds (such as value leakage) or affects a core functionality of the protocol.
- Low: Leads to any unexpected behavior in some of the protocol's functionalities, but is not critical.

4.2. Likelihood

- High: The attack path is feasible with reasonable assumptions that replicate on-chain conditions, and the cost of the attack is relatively low compared to the potential funds that can be stolen or lost.
- Medium: The attack vector is conditionally incentivized but still relatively likely.
- Low: The attack requires too many or highly unlikely assumptions, or it demands a significant stake by the attacker with little or no incentive.

4.3. Action required for severity levels

- Critical: Must fix as soon as possible
- High: Must fix
- Medium: Should fix
- Low: Could fix

5. Security Assessment Summary

Duration: 03/03/2025 - 07/03/2025
Repository: JohnJurdak/StingSC
Commit: 79e34f01ffd94ead2787b31aef4dcdb9d461dfec

- src/*

6. Executive Summary

Throughout the security review, Ali Kalout and Ali Shehab engaged with Sting's team to review Sting. During this review, 11 issues were uncovered.

Findings Count

Severity	**Amount**
Critical	N/A
High	3
Medium	4
Low	4
Total Finding	**11**

Summary of Findings

ID	Title	Severity	Status
H-01	Strategies don’t handle Infrared Vaults’ Extra Rewards	High	Resolved
H-02	Retiring a strategy discards remaining token balances	High	Resolved
H-03	Unclaimed LP Pool Profits (Kodiak/BEX)	High	Resolved
M-01	Withdraw missing `whenNotPaused`	Medium	Resolved
M-02	`require(msg.sender == tx.origin)` blocks automation	Medium	Resolved
M-03	`Withdraw fee collected but unused`	Medium	Acknowledged
M-04	`chargeFees()` overcharges due to idle WBERA	Medium	Resolved
L-01	`getPricePerFullShare` can't handle LPs with decimals != 18	Low	Resolved
L-02	`chargeFees()` sends rewards to vault when vault is caller	Low	Resolved
L-03	Replace `approve(0)` with `forceApprove()`	Low	Resolved
L-04	`securityFee` is not updatable	Low	Resolved

7. Findings

7.1. High Findings

[H-01] Strategies don’t handle Infrared Vaults’ Extra Rewards

Severity:
High

Description:
Infrared vaults return rewards in multiple tokens, https://github.com/cantina-competitions/infrared-contracts/blob/65de7c256da60721a4ea6129bd8ed62815b260bc/src/core/InfraredVault.sol#L175-L177. However the strategies only account for rewards in iBGT.
Forcing the other rewards to be lost forever.

Recommendations:
Fetch and handle all reward tokens dynamically:

```
address[] memory rewards = IInfraredVault(lpVault).getAllRewardTokens();
for (uint i = 0; i < rewards.length; i++) {
    uint256 balance = IERC20(rewards[i]).balanceOf(address(this));
    if (balance > 0) {
        // swap or transfer based on desired logic
    }
}
```

[H-02] Retiring a strategy discards remaining token balances

Severity:
High

Description:
When `retireStrat` is called, tokens like WBERA, rewardToken, LP0, LP1 are left in the strategy contract. These unaccounted balances are not returned to the vault.

```
function retireStrat() external {
    require(msg.sender == vault, "!vault");
```

```
    IInfraredVault(lpVault).exit(); // claim all rewards and withdraw from pool

    chargeFees();
    addLiquidity();

    uint256 pairBal = IERC20(lpPair).balanceOf(address(this));
    IERC20(lpPair).transfer(vault, pairBal);
}
```

Recommendations:

Safely transfer all non-zero token balances back to the vault:

```
function _transferRemaining(address token) internal {
    uint256 bal = IERC20(token).balanceOf(address(this));
    if (bal > 0) IERC20(token).safeTransfer(vault, bal);
}

_transferRemaining(lpToken0);
_transferRemaining(lpToken1);
_transferRemaining(wbera);
_transferRemaining(rewardToken);
```

[H-03] Unclaimed LP Pool Profits (Kodiak/BEX)

Severity:

High

Description:

Profits accrued in BEX or Kodiak pools are not claimed unless exitPool() or removeLiquidity() is explicitly called. This causes loss of realized profit.

Recommendations:

Add logic to realize pool rewards during `retireStrat()`:

```
// Kodiak
IIslandRouter(islandRouter).removeLiquidity(...);

// BEX
IBexVault(bexVault).exitPool(poolId, ...);
```

7.2. Medium Findings

[M-01] Withdraw missing `whenNotPaused`

Severity:
Medium

Description:
Users can still withdraw while the vault is paused, potentially violating the intent of the pause functionality.

Recommendations:
Add the `whenNotPaused` modifier:

```
function withdraw(uint256 _shares) public nonReentrant whenNotPaused { ... }
```

[M-02] `require(msg.sender == tx.origin)` blocks automation

Severity:
Medium

Description:
Prevents contract-based callers like vaults or automated harvesters, limiting composability.

Recommendations:
Remove this restrictive check:

```
require(msg.sender == tx.origin, "!vault"); // REMOVE
```

[M-03] Withdraw fee collected but unused

Severity:
Medium

Description:
`securityFee` is deducted on withdraw, but left idle in the strategy contract, not sent or burned.

Recommendations:
Handle this security or send it to some fee recipient.

[M-04] `chargeFees()` overcharges due to idle WBERA

Severity:
Medium

Description:
Idle WBERA in the contract inflates the fee calculation when charging from swapped amounts.

Recommendations:
Use delta accounting:

```
uint256 wBeraBalBefore = IERC20(wbera).balanceOf(address(this));

uint256 toWbera = (IERC20(rewardToken).balanceOf(address(this)) * totalFee) / PERCENT_DIVISOR;
IUniswapV2Router02(uniRouter).swapExactTokensForTokensSupportingFeeOnTransferTokens(
    toWbera, 0, rewardTokenToWberaRoute, address(this), block.timestamp + 600
);

uint256 wBeraBal = IERC20(wbera).balanceOf(address(this)) - wBeraBalBefore;
```

7.3. Low Findings

[L-01] `getPricePerFullShare` can't handle LPs with decimals != 18

Severity:
Low

Description:
The return value will be inaccurate for LP tokens that use decimals other than 18, leading to incorrect vault share pricing.

Recommendations:
Normalize both `balance` and `totalSupply` to a common base to avoid decimal mismatch:

```
function getPricePerFullShare() public view returns (uint256) {
    return totalSupply() == 0
        ? 1e18
        : (balance() * 1e36) / (totalSupply() * 10 ** want().decimals());
}
```

[L-02] `chargeFees()` sends rewards to vault when vault is caller

Severity:
Low

Description:
When the vault calls `retireStrat()`, `msg.sender` is the vault, and fees meant for a user are sent to the vault.

Recommendations:
Handle the vault case explicitly:

```
IERC20(wbera).safeTransfer(
    msg.sender == vault ? treasury : msg.sender,
```

```
    callFeeToUser
);
```

[L-03] Replace `approve(0)` with `forceApprove()`

Severity:
Low

Description:
The manual approve(0) + approve(max) pattern is outdated and prone to race conditions and weird ERC20s.

Recommendations:
Use OpenZeppelin's `forceApprove()` pattern:

```
SafeERC20.forceApprove(IERC20(token), spender, amount);
```

[L-04] `securityFee` is not updatable

Severity:
Low

Description:
There's no function to adjust the securityFee, limiting configuration.

Recommendations:
Add an update method with max cap:

```
function updateSecurityFee(uint256 _fee) external onlyOwner {
    require(_fee <= MAX_FEE, "too high");
    securityFee = _fee;
}
```