CD SECURITY

AUDIT REPORT

Beezie Stableswap
February 2025

Prepared by
Pelz
yotov721

Introduction

A time-boxed security review of the **Beezie** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Beezie Stableswap**

The Beezie stableswap update facilitates token exchanges on a Curve stableswap pool between `USDC` and `USDF`. It ensures that the pool is correctly configured with the expected two coins and determines the proper coin indices upon deployment. Users can perform a checkout by transferring USDC, having it exchanged via the pool to USDF, and then invoking an external target contract with custom calldata.

The protocol also includes owner-only functions to manage allowed target addresses and withdraw ERC20 tokens.

Severity classification

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

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 83d7904af4a0e397d8509be40854587ae7ce1fc5**

Scope

The following smart contracts were in scope of the audit:

- src/CurveStableswapCheckout.sol

The following number of issues were found, categorized by their severity:

- High: 0
- Medium: 0
- Low: 1
- Info: 3

Findings Summary

ID	Title	Severity
[L-01]	Incorrect assembly revert handling	Low
[I-01]	Missing event emission in state changing function	Info
[I-02]	Recommended adding reentrancy guard to function that does low level call	Info
[I-03]	Missing address zero check in `setTarget()` Function	Info

Detailed Findings

[L-01] Incorrect assembly revert handling

Description

The contract makes a low level call and reverts using assembly in the call fails. When the call to target fails, it returns an error message stored in memory (as `returnData`). This variable is a pointer to the memory location where the return data from the failed call is stored. The first 32 bytes of this data typically encode the length of the actual error message, followed by the error message itself.

`add(returnData, 32)` moves the pointer 32 bytes forward in memory. This effectively skips the length field and points directly to the start of the error message.

The second parameter to revert in the assembly block is supposed to be the length of the error message (in bytes). However, in the code above, it is incorrectly given as `returnData`, which is the pointer value rather than the length.

This will cause the revert to return an incorrect error message.

Recommendations

To retrieve the length of the error message, the first 32 bytes of `returnData` need to be read using `mload(returnData)`.

```
        (bool success, bytes memory returnData) = target.call(data);
        if (!success) {
            assembly {
-               revert(add(returnData, 32), returnData)
+               revert(add(returnData, 32), mload(returnData))
            }
        }
```

[I-01] Missing event emission in state changing function

Description

By standard events should be emitted by all state changing function should emit events. The `setTarget` function enables or disables an address from being callable by the low level call in `doCheckout()`.

Recommendations

Add event emission on the `setTarget` function

```
+    event LogTarget(address target, bool allowed);

...

function setTarget(address target, bool allowed) external onlyOwner {
    allowedTargets[target] = allowed;
+    emit LogTarget(target, allowed);
}
```

[I-02] Recommended adding reentrancy guard to function that does low level call

Description

The `doCheckout` function executes a low level call via `target.call(data)`. It can call only whitelisted targets, however it is recommended to add a reentrancy guard just in case the target gives control to another contract. That could happen unintentionally via NFT `safeTransfer`

Recommendations

```
+ import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

- contract CurveStableswapCheckout is Ownable {
```

```
+ contract CurveStableswapCheckout is Ownable, ReentrancyGuard {

...

-     function doCheckout(uint256 amount, uint256 minAmountOut, address
target, bytes memory data) external {
+     function doCheckout(uint256 amount, uint256 minAmountOut, address
target, bytes memory data) external nonReentrant {
```

[I-03] Missing address zero check in `setTarget()` Function

Summary

The `setTarget` function allows setting a zero address (`address(0)`) as a valid target. This could cause unintended behavior if calls are made to a zero address, potentially leading to lost funds or contract misbehavior.

Recommended Fix

Add a check to prevent setting `address(0)` Emit an event when the target is updated for better traceability