Beefy Security Scan Results

by Pessimistic

This is not a security audit

This report is public

May 27, 2024

Abstract

This report considers the security of smart contracts of the Beefy protocol. Our task is to find and describe security issues using the static-analysis tools Slither and Slitherin and help resolve them.

The work is financially covered by the Arbitrum Foundation grant.

Disclaimer

Current work does not give any warranties on the security of the code. It is not an audit or its replacement. Performing this scan, we focused on finding as many crucial issues as possible rather than making sure that the protocol was entirely secure. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Summary

In this report, we described issues found in smart contracts of the Beefy protocol.

We scanned the codebase and manually rejected or verified all automated findings, revealing seven relevant issues.

The developers commented on two issues.

The entire process is described in the section below.

Scan process

Under the Arbitrum Foundation grant, we researched and developed Arbitrum-specific detectors. They became publicly available with Slitherin v0.6.0 release.

Workflow

This work consisted of five stages:

1. For the scan, we were provided with the Beefy project on a private GitHub repository, commit e6b02aaff3ba3d092620b8ef0d2c438739cac4ea.
2. For the analysis of the protocol, we launched Slither v0.10.1 and Slitherin v0.6.0 on the provided codebase.
3. One auditor manually checked (rejected or accepted) all findings reported by the tools. The second auditor verified this work. We shared all relevant issues with the protocol developers and answered their questions.
4. The developers reviewed the findings and gave comments on two issues.
5. We prepared this final report summarizing all the issues and comments from the developers.

Issue categories

Within the confines of this work, we were looking for:

- Arbitrum-specific problems;
- Standard vulnerabilities like re-entrancy, overflow, arbitrary calls, etc;
- Non-compliance with popular standards like ERC20 and ERC721;
- Some access control problems;
- Integration issues with some popular DeFi protocols;
- A wide range of code quality and gas efficiency improvement opportunities.

This scan does not guarantee that these issues are not present in the codebase.

Scan results

Issue category	Number of detectors	Status
Compilation	1	Passed
Arbitrum Integration	3	Passed
AAVE Integration	1	Passed
Uniswap V2 Integration	7	Passed
OpenZeppelin	2	Passed
ERC-20	7	Passed
ERC-721	2	Passed
Known Bugs	15	1 issue found
Access Control	3	Passed
Arbitrary Call	5	Passed
Re-entrancy	6	Passed
Weak PRNG	2	Passed
Upgradability	2	Passed
Ether Handling	3	Passed
Low-level Calls	2	Passed
Assembly	2	Passed
Inheritance	3	Passed
Arithmetic	2	Passed
Old Solidity Versions Bugs	10	Passed
Code Quality	15	2 issues found
Best Practices	4	1 issue found
Gas	7	3 issues found

Discovered Issues

Unsafe ERC20 interaction

The `initialize`, `addRewardToken`, `setRewardPool`, `resetRewardTokens` functions in the **BeefyQIVault** contract ignore the return values by ERC20 `approve` function.

According to the ERC20 token standard:

> *Callers MUST handle `false` from returns `(bool success)`. Callers MUST NOT assume that `false` is never returned!*

We recommend using the safe functions from the OpenZeppelin **SafeERC20** library to interact with ERC20 tokens.

Unchecked return value

There are several contracts where the return values of called functions are not checked:

- The `_swapRewardsToNative` function ignores return values by `UniV3Utils.swap` and `BalancerActionsLib.balancerSwap` functions in the **BeefyQIVault** contract;
- The `_removeLiquidity`, `_claimEarnings`, `_addLiquidity` functions ignores return values by `IUniswapV3Pool.burn`, `IUniswapV3Pool.collect` and `IUniswapV3Pool(pool).mint` functions in the **StrategyPassiveManagerUniswap** contract.

We recommend checking the return values to avoid incorrect state changes in case of unexpected behaviour of the called function.

Missing event

`setLpToken0ToNativePath` and `setLpToken1ToNativePath` setter functions in **StrategyPassiveManagerUniswap** contract and `initialize` function in the **BeefyQIVault** contract do not `emit` an `event`. Emmiting of event in setter functions allows contract owner and relevant parties to be notified about important state changes within the contract.

> *Comment from the developers: That is correct, setter functions should emit events.*

Constant variable

The `VAULT_ROLE` variable in the **BeefyQI** contract can be declared as `constant`. We recommend declaring it as `constant` to reduce gas consumption and improve code quality.

Immutable variable

The following variables are set during contract deployment and never change later:

- The `uniswapV3Quoter` and `oneUsd` variables in the **BeefyConcLiqLens** contract;
- The `instance` variable in the **BeefyVaultConcLiqFactory** contract;
- The `keeper` variable in the **StrategyFactory** contract.

We recommend declaring them as `immutable` to reduce gas consumption and improve code quality.

Functions visibility

There are several contracts where functions can be declared as `external` instead of `public`:

- `initialize`, `want`, `available`, `previewWithdraw`, and `previewDeposit` functions in the **BeefyVaultConcLiq** contract;
- `price`, `lpToken0ToNative`, and `lpToken1ToNative` functions in the **StrategyPassiveManagerUniswap** contract.

It helps to improve code readability and optimize gas consumption in the project.

> *Comment from the developers: Probably, they were not declared as `external` after we made the changes that allowed this.*

Potential outdated data access

The `resetRewardTokens` function in the **BeefyQIVault** contract deletes the `Reward` struct from the **BeefyBalancerStructs**, but it does not delete the `swapInfo` `mapping` values. This can lead to a problem in a rare scenario where outdated `swapInfo` data becomes accessible in the `_swapRewardsToNative` function after re-adding the token by `addRewardToken` function.

This analysis was performed by Pessimistic:

Egor Dergunov, Junior Security Engineer
Oleg Bobrov, Security Engineer
Evgeny Bokarev, Junior Security Engineer
Nikita Kirillov, Product Owner
Evgeny Marchenko, CTO
Konstantin Zherebtsov, Business Development Lead

May 27, 2024