Gearbox Security Scan Results

by Pessimistic

This is not a security audit

This report is public

May 31, 2024

Abstract

This report considers the security of smart contracts of the Gearbox 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 Gearbox protocol.

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

The developers acknowledged six issues and commented on all of them.

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 Gearbox project on the following GitHub repositories:
    - core-v3, commit: b2628d77f17fecf71feb77ebb038d5350f26fca7, the scope of the scan excludes **test** folder;
    - bots-v3, commit: dc7fe47f5b0c05d24f8349ed41bdd72f4989bf40, the scope of the scan excludes **test** folder;
    - oracles-v3, commit: c6e4bd0a42331daeec599f3d8a688fab79f9879a, the scope of the scan excludes **test** folder;
    - integrations-v3, commit: 8f1ae29e14fa9c918b87e9ed9a2a6e93f3654dbe, the scope of the scan excludes **test** folder.
2. For the analysis of the protocol, we launched Slither v0.10.2 and Slitherin v0.6.1 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, acknowledged and commented on all of them.
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	1 issue found
AAVE Integration	1	Passed
Uniswap V2 Integration	7	Passed
OpenZeppelin	2	Passed
ERC-20	7	Passed
ERC-721	2	Passed
Known Bugs	15	Passed
Access Control	3	Passed
Arbitrary Call	5	Passed
Re-entrancy	6	2 issues found
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	Passed

Discovered Issues

Missing event

There are several contracts where setter functions do not emit an event:

- The `setActiveCreditAccount` and `setFlagFor` functions of the **CreditManagerV3.sol** contract;
- The `setQuotaRevenue` function of the **PoolV3.sol** contract.

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:*
>
> - *`setActiveCreditAccount` and `setFlagFor` — These are not significant enough state changes;*
> - *`setQuotaRevenue` — This derived value can be easily calculated based on the `updateQuota` and `updateTokenQuotaRate` events in the **PoolQuotaKeeperV3** contract.*

Block properties on the Arbitrum chain

In **LPPriceFeed**, **RedstonePriceFeed**, **AccountFactoryV3**, **CreditManagerV3** , **CreditLogic**, **PriceFeedValidationTrait**, **CreditConfiguratorV3**, **CreditFacadeV3**, **GearStakingV3**, **ControllerTimelockV3**, **PolicyManagerV3** , **QuotasLogic**, **PoolQuotaKeeperV3** and **PoolV3** contracts there are several functions rely on the `block.timestamp` or `block.number` value within the Arbitrum contract's code. This behaves differently than on Ethereum, since consecutive blocks can share the same `block.timestamp` and `block.number`. It is important to ensure that the contract logic remains correct despite these differences.

> *Comment from the developers: At the moment, we do not see a problem with this. In our codebase, we only assume `block.timestamp` and `block.number` are non-decreasing in time.*

Unsafe ERC20 interaction

There are several contracts where the return values of called ERC20 functions are not checked:

- The `submit` function ignores return values by `transferFrom` and `transfer` functions in the **LidoV1Gateway** contract;
- The `mint` function ignores return values by `transferFrom` and `transfer` functions in the **CEtherGateway** contract;
- The `redeem` function ignores return values by `transferFrom` and `transfer` functions in the **CEtherGateway** contract;
- The `redeemUnderlying` function ignores return values by `transferFrom` and `transfer` functions in the **CEtherGateway** contract;
- The `deposit` function ignores return value by `transferFrom` function in the **WrappedAToken** contract;
- The `withdraw` function ignores return value by `transfer` function in the **WrappedAToken** contract;
- The `constructor` function ignores return value by `approve` function in the **CurveV1StETHPoolGateway** contract;
- The `migrate` function ignores return value by `approve` function in the **GearStakingV3.sol** contract.

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.

> *Comment from the developers: In mentioned cases, we know in advance which token we are dealing with, and accordingly, we know its behaviour. The only exception here is **CEtherGateway** because **cETH** on mainnet in fact returns false when transfer fails instead of reverting, but this contract is not deployed on any of the chains.*

Unchecked return value

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

- The `rescue` function ignores return value by `functionCall` function in the **CreditAccountV3.sol** contract;
- The `openCreditAccount` function ignores return value by `creditAccountSet.add` function in the **CreditManagerV3.sol** contract;
- The `closeCreditAccount` function ignores return value by `creditAccountSet.remove` function in the **CreditManagerV3.sol** contract;
- The `withdrawUnderlying` function ignores return value by `withdraw` function in the **WrappedAToken** contract.

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

> *Comment from the developers:*
>
> - *We do not check the `functionCall` result because it is unnecessary, as the function is only used to withdraw stuck tokens from the account;*
> - *We do not check `creditAccountsSet.add` and `creditAccountsSet.remove` results because it is an invariant that they are always true: you can not take an account from the **factory** twice without closing it first and you can not close the same account twice without reopening it;*
> - *In the `withdrawUnderlying` function, we have confidence in Aave's ability to perform seamless one-to-one conversions. However, actually, this is not critical.*

Potential read-only reentrancy vulnerability

In the `updateBounds` function of the **LPPriceFeed** contract there is a possibility of read-only reentrancy vulnerability. In case of an external call to a malicious `reserveFeed` contract, it is possible to read outdated values of `latestRoundData`, `lowerBound` and `upperBound` before they are updated despite the `lastBoundsUpdate` variable's state being already updated.

> *Comment from the developers: We presume that `reserveFeed` is trusted, as it is our own contract.*

Read-only reentrancy vulnerability

There are several contracts where function's return values could be manipulated through external read-only reentrancy:

- The `ICurvePoolStETH(pool).get_virtual_price` function of the **CurveV1StETHPoolGateway** contract;
- The `ICurvePool(lpContract).get_virtual_price` function of the **CurveCryptoLPPriceFeed** contract;
- The `IBalancerStablePool(lpToken).getRate` function of the **BPTStablePriceFeed** contract.

It is crucial to verify that the pools are not being re-entered since price and rate formulas might be manipulated. Although only `view` functions are present in the current code, we recommend checking for reentrancy before using the returned values by these functions.

Comment from the developers:

- *In the **CurveV1StETHPoolGateway** contract, the `get_virtual_price` is just a `view` function for getting the pool's state. Actually, we do not use it anywhere;*
- *In **PriceFeed** contracts, there is no uniform way to prevent this across different pools, let alone different protocols, so we use strict boundaries for the return value. This allows manipulation of the value, but only within a strictly limited percentage.*

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Evgeny Bokarev, Junior Security Engineer
Nikita Kirillov, Product Owner
Evgeny Marchenko, CTO
Konstantin Zherebtsov, Business Development Lead

May 31, 2024