

LayerZero

Stargate Fee Library V4

by Ackee Blockchain

June 28, 2022

Contents

1. Document Revisions	3
2. Overview	4
2.1. Ackee Blockchain	4
2.2. Audit Methodology	4
2.3. Review team	5
2.4. Disclaimer	5
3. Executive Summary	6
4. System Overview	7
4.1. Contracts	7
4.2. Actors	10
4.3. Trust model	11
5. Vulnerabilities risk methodology	12
5.1. Finding classification	12
6. Findings	14
M1: Unchecked cast	15
W1: Wrong version	16
W2: Renounce ownership	17
W3: Commented-out code	18
W4: Solidity optimizer	19
I1: Unnecessary variable allocation	20
I2: Missing documentation	21
Appendix A: How to cite	22

1. Document Revisions

1.0	Final report	June 28, 2022
-----	--------------	---------------

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

[Ackee Blockchain](#) is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge & we run a free certification course [Summer School of Solidity](#) and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, [Rockaway Blockchain Fund](#).

2.2. Audit Methodology

1. Technical specification/documentation - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. Tool-based analysis - deep check with automated Solidity analysis tools and Slither is performed.
3. Manual code review - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. Local deployment + hacking - the contracts are deployed locally and we try to attack the system and break it.
5. Unit and fuzzy testing - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Members Name	Position
It's a pleasure to meet you	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

LayerZero engaged [Ackee Blockchain](#) to conduct a security review of the Stargate Fee Library V4 (private repository) with a total time donation of 4 engineering days. The review took place between June 22 and June 28, 2022.

We began our review by using static analysis tools then we took a deep dive into the logic of the contracts. The project uses private dependencies, and we could not execute unit tests. Therefore we isolated the audited contract `StargateFeeLibraryV04.sol` using Brownie and performed our custom arithmetic tests. During the review, we paid particular attention to:

- ¥ ensuring the arithmetic of the system is correct,
- ¥ ensuring access controls are not too relaxed or too strict,
- ¥ looking for common issues such as data validation.

Our review resulted in 7 findings, ranging from Info to Medium severity. The most severe one [M1](#) can lead to an integer overflow if the contract owner sets the `depegThreshold` value too high.

Ackee Blockchain recommends LayerZero:

- ¥ be aware of an integer overflow when casting `uint256` to `int256`,
- ¥ remove commented-out code,
- ¥ create the NatSpec documentation.

4. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

4.1. Contracts

Contracts we find important for better understanding are described in the following section.

StargateFeeLibraryV04.sol

The Stargate Fee Library is used in the Stargate protocol to calculate pool fees.

Addresses in the `whitelist` mapping are excluded from the protocol and LP fees.

The modifier `notDepegged`, which is used in the `getFees` function, reverts the function execution if the price from the pool's price feed drops below the `depegThreshold` value.

`getFees`

1. Gets the pool from the factory.
2. Gets the swap path from the pool.
3. Gets the token balance from the pool divided by the convert rate.
4. Gets the pool's LP token total liquidity.
5. Calculates the equilibrium reward using the `_getEqReward` function.
6. Calculates the equilibrium fee using the `_getEquilibriumFee` function.

7. Return calculated values if the `_from` address is in the whitelist.
8. Otherwise calculates an additional protocol fee and LP fee.

`_getEqReward`

1. Returns `0` if `_lpAsset < _currentAssetSD`.
2. Calculates `poolDeficit` by subtracting `_currentAssetSD` from `_lpAsset`.
3. Returns `min(_rewardPoolSize * _amountSD / poolDeficit, _rewardPoolSize)` if assets in the pool are <75% of provided liquidity and amount is > 2% of pool deficit.
4. Otherwise returns `0`.

The following image shows our simulation of the `_getEqReward` behavior.

`_getEquilibriumFee`

1. Check if the `balanceBefore` is greater or equal than `amountSD`, reverts if not.
2. Calculates the `balanceAfter` by subtracting `amountSD` from `balanceBefore`.

3. Calculates `safeZoneMin` and `safezoneMax` from `idealBalance` using `DELTA_1`, `DELTA_2` and `DENOMINATOR` constants.
4. Depending if the `afterBalance` is inside the safe zone (between `safeZoneMin` and `safezoneMax`), the function returns `eqFee` calculated using the `_getTrapezoidArea` function.

The following image shows our simulation of the `_getEquilibriumFee` behavior.

`_getProtocolAndLpFee`

1. Calculates the `protocolFee` as `_amountSD * PROTOCOL_FEE / DENOMINATOR - _protocolSubsidy`.
2. Calculates the `lpFee` as `_amountSD * LP_FEE / DENOMINATOR`
3. If there are active emmissions (`allocPoint > 0`), `lpFee` is added to `protocolFee` and set to 0.
4. If `_lpAsset == 0` then `protocolFee` and `lpFee` are returned.
5. Otherwise the `currentAssetNumerated` is calculated as `_currentAssetSD * DENOMINATOR / _lpAsset`.

6. If the `currentAssetNumerated` is less than 50% then `protocolFee` and `lpFee` are set to 0
7. If the `currentAssetNumerated` is between 50% and 60% and the transfer does not drain the pathway below 60% of the ideal balance, then the protocol fee and LP fee get reduced linearly.

`_getTrapezoidArea`

Helper function for the `_getEquilibriumFee` function. Introduced in Stargate Fee Library V2 and did not change since then.

1. Checks if the balance is not out of bounds.
2. Calculates the `xBoundWidth` by subtracting `xLowerBound` from `xUpperBound`.
3. Calculates the `yStart` and `yEnd`.
4. Calculates the return value as `yStart + yEnd * deltaX / 2 / DENOMINATOR`.

4.2. Actors

This part describes actors of the system, their roles, and permissions.

Owner

The owner is an address that deploys the contract to the network and has special privileges in the contract:

- ¥ Transfer the ownership to another address,
- ¥ renounce the ownership (set the owner to `address(0)`),
- ¥ edit the whitelist,
- ¥ set pool ID to LP ID mapping,
- ¥ set pool ID to price feed address,

¥ set depeg threshold.

User

User role means any address which can call external functions (external addresses or contracts). The user can perform the following operations:

¥ get fees,

¥ get equilibrium reward,

¥ get equilibrium fee,

¥ get protocol and LP fee,

¥ get a trapezoid area (used in the `__getEquilibriumFee` function),

¥ get the library version.

4.3. Trust model

Users need to trust the owner regarding the special privileges listed above.

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood* which measures the probability of exploitability during runtime.

5.1. Finding classification

The full definitions are as follows:

Impact

High

Code that activates the issue will lead to undefined or catastrophic consequences for the system.

Medium

Code that activates the issue will result in consequences of serious substance.

Low

Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.

Warning

The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Info

The issue is on the border-line between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

High

The issue is exploitable by virtually anyone under virtually any circumstance.

Medium

Exploiting the issue currently requires non-trivial preconditions.

Low

Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overridden for purposes of readability, each finding contains:

- ¥ a *Description*,
- ¥ an *Exploit scenario*, and
- ¥ a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Type	Impact	Likelihood
M1: Unchecked cast	Overflow	High	Low
W1: Wrong version	Typo	Warning	N/A
W2: Renounce ownership	Access controls	Warning	N/A
W3: Commented-out code	Access controls	Warning	N/A
W4: Solidity optimizer	Compiler	Warning	N/A
I1: Unnecessary variable allocation	Gas optimization	Info	N/A
I2: Missing documentation	Documentation	Info	N/A

Table 1. Table of Findings

M1: Unchecked cast

Impact:	High	Likelihood:	Low
Target:	StargateFeeLibraryV04.sol	Type:	Overflow

Listing 1. Excerpt from

/contracts/libraries/StargateFeeLibraryV04.sol #L47-

L47[StargateFeeLibraryV04.notDepegged]

```
47         require(price >= int256(depegThreshold), "FeeLibrary:
    Ê _srcPoolId is depegged");
```

Description

OpenZeppelin `SafeMath` library handles the integer overflow/underflow during arithmetic operations but not during casting.

Exploit scenario

If the `depegThreshold` variable is greater than $2^{256} / 2 - 1$ (`int256` max. value), then the integer overflows during the casting, and the condition `price >= int256(depegThreshold)` passes.

Recommendation

Use safety check after the casting.

```
int256 intThreshold = int256(depegThreshold);
require(intThreshold > 0, "depegThreshold overflow");
```

[Go back to Findings Summary](#)

W1: Wrong version

Impact:	Warning	Likelihood:	N/A
Target:	StargateFeeLibraryV04.sol	Type:	Typo

Listing 2. Excerpt from

*/contracts/libraries/StargateFeeLibraryV04.sol #L289-
L289[StargateFeeLibraryV04.getVersion]*

```
289         return "2.0.0";
```

Description

`getVersion` function returns `"2.0.0"`, but previous versions of the library returns following values, which seems like a mistake in `StargateFeeLibraryV04`:

¥ `StargateFeeLibraryV01` - `"1.0.0"`

¥ `StargateFeeLibraryV02` - `"2.0.0"`

¥ `StargateFeeLibraryV03` - `"3.0.0"`

Recommendation

Double-check if `"2.0.0"` is the intended value.

[Go back to Findings Summary](#)

W2: Renounce ownership

Impact:	Warning	Likelihood:	N/A
Target:	StargateFeeLibraryV04.sol	Type:	Access controls

Description

The OpenZeppelin `Ownable` pattern contains the `renounceOwnership()` function which sets the owner address to `address(0)`. This could lead to irreversible damage to the contract. Thus, nobody will be able to become the owner again and call functions with the `onlyOwner()` modifier.

Exploit scenario

This is not a directly exploitable issue but can be considered as an unintended feature of the system. This function can be called accidentally or intentionally by a malicious owner.

Recommendation

We recommend using a multisig wallet for the owner to avoid an accidental `renounceOwnership()` call. This feature can also be disabled by overriding the `renounceOwnership()` function in contracts inherited from the `Ownable` contract.

[Go back to Findings Summary](#)

W3: Commented-out code

Impact:	Warning	Likelihood:	N/A
Target:	StargateFeeLibraryV04.sol	Type:	Access controls

Listing 3. Excerpt from

*/contracts/Libraries/StargateFeeLibraryV04.sol #L277-
L277[StargateFeeLibraryV04._getTrapezoidArea]*

```
277          // xStartDrift = xUpperBound.sub(xStart);
```

Listing 4. Excerpt from

*/contracts/Libraries/StargateFeeLibraryV04.sol #L280-
L280[StargateFeeLibraryV04._getTrapezoidArea]*

```
280          // xEndDrift = xUpperBound.sub(xEnd)
```

Description

The function `_getTrapezoidArea` contains commented-out code, which is not a good practice.

Recommendation

Remove unused and commented-out parts of code.

[Go back to Findings Summary](#)

W4: Solidity optimizer

Impact:	Warning	Likelihood:	N/A
Target:		Type:	Compiler

Description

The project uses `solc` optimizer. Enabling `solc` optimizer [may lead to unexpected bugs](#).

The Solidity compiler was audited in November 2018, and the audit [concluded](#) that the optimizer may not be safe.

Recently [the bug in the YUL optimizer](#) has been discovered in Solidity 0.8.13 and fixed in 0.8.15.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it can be possible to attack the protocol.

Recommendation

If it is not necessary, avoid using the optimizer.

[Go back to Findings Summary](#)

I1: Unnecessary variable allocation

Impact:	Info	Likelihood:	N/A
Target:	StargateFeeLibraryV04.sol	Type:	Gas optimization

Listing 5. Excerpt from

/contracts/Libraries/StargateFeeLibraryV04.sol #L101-

L101[StargateFeeLibraryV04.getFees]

```
101      (uint256 eqFee, uint256 protocolSubsidy) = _getEquilibriumFee
    Ê      (chainPath.idealBalance, chainPath.balance, _amountSD);
```

Listing 6. Excerpt from

/contracts/Libraries/StargateFeeLibraryV04.sol #L110-

L110[StargateFeeLibraryV04.getFees]

```
110      (uint256 protocolFee, uint256 lpFee) = _getProtocolAndLpFee
    Ê      (_amountSD, currentAssetSD, lpAsset, protocolSubsidy, srcPoolId,
    Ê      chainPath);
```

Description

Variable `eqFee` in the [first snippet](#) and `protocolFee`, `lpFee` in the [second snippet](#) are allocated unnecessarily.

Recommendation

Assign `eqFee`, `protocolFee`, and `lpFee` values directly to the function's return value (`Pool.SwapObjects`).

[Go back to Findings Summary](#)

I2: Missing documentation

Impact:	Info	Likelihood:	N/A
Target:	StargateFeeLibraryV04.sol	Type:	Documentation

Description

The code contains brief, understandable comments of important logic, but detailed documentation is missing.

Recommendation

We strongly recommend covering the code by NatSpec. High-quality documentation has to be an essential part of any professional project.

[Go back to Findings Summary](#)

Appendix A: How to cite

Please cite this document as:

[Ackee Blockchain](#), LayerZero: Stargate Fee Library V4, June 28, 2022.

