ackee blockchain

LayerZero

LPStakingTime.sol, WidgetSwap.sol

by Ackee Blockchain

July 12, 2022

Contents

1. Document Revisions

1.0	Final report	12 July 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

Member’s Name	Position
Lukáš Böhm	Lead Auditor
Štěpán Šonský	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

Layer-zero engaged Ackee Blockchain to perform a security review of the two separate contracts with a total time donation of 4 engineering days in a period between June 30 and July 8, 2022. The lead auditor was Lukáš Böhm. The audit has been performed on the following commits:

- WidgetSwap.sol: `8938c7b`
- LPStakingTime.sol: `220b949`

We began our review using static analysis tools, namely `slither` and the `solc` compiler. This yielded several code quality improvements, such as I2: Public functions without internal calls. We then took a deep dive into the logic of the contracts. During the review, we paid particular attention to:

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

Our review resulted in 5 findings, ranging from Info to Medium severity.

The code quality is excellent, as with the previous Layer Zero audits. The code follows Solidity best practices and is well readable. The scope of the security review was two separate contracts for which LayerZero team provided a short description. Contracts contain partial NatSpec in-code documentation.

Ackee Blockchain recommends Layer-zero:

- adding complete NatSpec in-code documentation,

- log all contract’s state changes,
- address all other reported issues.

4. System Overview

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

4.1. Contracts

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

WidgetSwap.sol

The contract allows swapping tokens or ETH. In ETH swap function `swapETH`, internal function `_getAndPayWidgetFeeETH` is called, where fee calculation is happening and then sending fees to the fee collector. `swapETH` execution continues with actual swap on `stargateRouterETH`.

Similar logic is used in the `swapTokens` function for swapping the tokens. The fee is calculated in `_getAndPayWidgetFee`, where the tokens are sent to the contract, the fee is paid, and `stargateRouter` is approved to make actual swaps.

LPStakingTime.sol

The contract allows to stack LPToken. While depositing or withdrawing, LPToken user gets eToken. The amount of eToken is calculated concerning the last withdrawal time. The `emergencyWithdraw` is used for fast withdrawal without caring about rewards. The contract is a fork of LPStaking.sol. Instead of `block.number`, `block.timestamp` is used for compatibility with Optimism chain.

4.2. Actors

This part describes the system’s actors, roles, and permissions.

Owner

In LPStaking contract, the owner can add new pools with LP tokens, set allocation points for a given pool, and set eToken earned per second. These operations may change the contract’s state.

User

In WidgetSwap contract, a user can wrap stargate transfers that charge their own fee on top of LayerZero’s Stargate.

In LPStakingTime contract, a user can deposit and withdraw to a specific pool and check the pending reward.

4.3. Trust model

Users have to trust the Owner of LPStakingTime contract.

5. Vulnerabilities risk methodology

A `Severity` rating of each finding is determined as a synthesis of two sub-ratings: `Impact` and `Likelihood`. It ranges from `Informational` to `Critical`.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating 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:

Severity

Severity	Impact	Likelihood
Informational	Informational	N/A
Warning	Warning	N/A
Low	Low	Low
Medium	Low	Medium
Medium	Low	High
Medium	Medium	Medium
High	Medium	High
Medium	High	Low

Severity	Impact	Likelihood
High	High	Medium
Critical	High	High

Table 1. Severity of findings

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 overriden 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

	Severity	Impact	Likelihood
M1: Unchecked transfer	Medium	High	Low
W1: Floating pragma	Warning	Warning	N/A
W2: Lack of events	Warning	Warning	N/A
I1: Use immutable instead of constant	Info	Info	N/A
I2: Public functions without internal calls	Info	Info	N/A

Table 2. Table of Findings

M1: Unchecked transfer

Impact:	High	Likelihood:	Low
Target:	{WidgetSwap.sol}	Type:	Data validation

Description

_getAndPayWidgetFee function for getting and paying Widget fee uses transferFrom function of ERC20 token from user to WidgetSwap contract and then transfer function to the fee collector contract. ERC20 token is specific for a defined pool.

```
IERC20(token).transferFrom(msg.sender, address(this), _amountLD);

IERC20(token).transfer(_feeObj.feeCollector, widgetFee);
```

SafeERC20 helper checks the boolean return values of ERC20 operations and reverts the transaction if they fail. At the same time, it allows supporting some non-standard ERC20 tokens that do not have boolean return values.

If ERC20 token of the defined pool is a non-standard token and the contract does not use SafeERC20, nor does it appropriately handle the case of tokens returning false (rather than reverting) on failure conditions (such as insufficient allowance), it may lead to unexpected behavior.

Vulnerability scenario

For example, the transfer function used to move tokens to the fee collector may fail for whatever reason internal to the token at hand, and the failure is missed. As a result, the tokens are not sent and stay in the contract.

If ERC20 tokens of the pools are not non-compliant, the standard transfer works correctly.

Recommendation

Short term, make sure whether non-compliant tokens are used or not. If they are, use SafeERC20 wrapper.

Long term, always use SafeERC20 when interacting with external tokens. This will ensure the maximum support range for variously-behaving ERC20 tokens.

Go back to Findings Summary

W1: Floating pragma

Impact:	Warning	Likelihood:	N/A
Target:	{WidgetSwap.sol}	Type:	Compiler configuration

Description

The project uses solidity floating pragma: `^0.8.4`.

Vulnerability scenario

A mistake in deployment can cause a version mismatch and thus an unexpected bug.

Recommendation

Stick to one version and lock the pragma in all contracts. More information can be found in: swcregistry

Go back to Findings Summary

W2: Lack of events

Impact:	Warning	Likelihood:	N/A
Target:	{LPStakingTime.sol}	Type:	Logging

Description

LPStakingTime contract logs only `Deposit`, `Withdraw` and `EmergencyWithdraw` events. Functions `set`, `add` and `setETokenPerSecond` lacks of emits, although they make changes to the contract's state and arithmetics.

Recommendation

Log any values that on-chain and off-chain observers might be interested in. This ensures the maximum transparency of the protocol to its users, developers, and other stakeholders.

Go back to Findings Summary

I1: Use immutable instead of constant

Impact:	Info	Likelihood:	N/A
Target:	{WidgetSwap.sol}	Type:	Gas optimization

Description

Variable `TENTH_BPS_DENOMINATOR` is immutable, but its value is not assigned in the constructor.

For constant variables, the value has to be fixed at compile time, while for immutable, it can still be assigned at construction time. For these values, 32 bytes are reserved. Due to this, constant values can sometimes be cheaper than immutable values.

Recommendation

Use `constant` keyword for state variables that are not assigned in the constructor.

[Go back to Findings Summary](#)

I2: Public functions without internal calls

Impact:	Info	Likelihood:	N/A
Target:	{LPStakingTime.sol}	Type:	Gas optimization

Description

Some functions are declared public even though they are not called internally anywhere.

```
function add(uint256 _allocPoint, IERC20 _lpToken) public onlyOwner {}

function set(uint256 _pid, uint256 _allocPoint) public onlyOwner {}

function deposit(uint256 _pid, uint256 _amount) public {}

function withdraw(uint256 _pid, uint256 _amount) public {}

function emergencyWithdraw(uint256 _pid) public {}
```

Recommendation

If functions are not called internally, they should be declared as external. It helps gas optimization because function arguments do not have to be copied into memory.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, LayerZero: LPStakingTime.sol, WidgetSwap.sol, July 12, 2022.

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq