ackee blockchain

Overnight Finance

StrategyUs3UsdcWeth

by Ackee Blockchain

3.3.2023

Contents

1. Document Revisions

0.1	Draft report	February 13, 2023
1.0	Final report	March 3, 2022
1.1	Fix review	March 3, 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, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, 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, RockawayX.

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 Woke 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. Finding classification

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.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

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 a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline 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.

2.4. Review team

Member's Name	Position
Jan Kalivoda	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. 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

Overnight Finance is a protocol that presents yield-generating stablecoin (USD+) pegged to USDC. ETS is a sub-project targeting to yield generation. The scope of the audit is one of ETS's strategies and two libraries.

Revision 1.0

Overnight Finance engaged Ackee Blockchain to perform a security review of the specific strategy contract with a total time donation of 6 engineering days in a period between February 1 and February 10, 2023 and the lead auditor was Jan Kalivoda.

The audit has been performed on the commit `e7d61fa` on a private repository and the scope was the following:

- StrategyUs3UsdcWeth.sol
- UniswapV3StakeLibrary.sol
- AaveV3BorrowLibrary.sol

We began our review by using static analysis tools, namely Slither and Woke. We then took a deep dive into the logic of the contracts. For testing we involved Woke testing framework and Anvil development chain with a forked mainnet. During the review, we paid special attention to:

- if the strategy is susceptible to sandwich attack,
- 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 7 findings, ranging from Info to Medium severity. Since the scope was only the strategy contract and two of its dependencies, we have acted to other components as a black box. We also recommend performing an audit of other components. Namely, the BalanceMath contract is important for correct functionality and contracts containing public entrypoints for strategy contracts.

Ackee Blockchain recommends Overnight Finance:

- write a more exhaustive test suite,
- create proper documentation,
- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done between February 28 and March 3, 2023, on the given commit: `0fca062` and the scope was only the raised issues from the Revision 1.0.

See Revision 1.1 for the review of the updated codebase and additional information we consider essential for the current scope.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
M1: Missing data validation	Medium	1.0	Acknowledged
M2: Usage of deprecated function	Medium	1.0	Fixed
M3: Empty receive	Medium	1.0	Fixed
W1: Usage of `solc` optimizer	Warning	1.0	Acknowledged
I1: Borrow module is missing implementation for claiming rewards	Info	1.0	Fixed
I2: Documentation	Info	1.0	Fixed partially
I3: Unused function parameter	Info	1.0	Acknowledged

Table 2. Table of Findings

5. Report revision 1.0

5.1. 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.

Contracts

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

StrategyUs3UsdcWeth

The strategy implementation allows stake, unstake and claim rewards. It inherits from HedgeStrategy (public entrypoints), StakeModule (functions for UniswapV3StakeLibrary), BorrowModule (functions for AaveV3BorrowLibrary), and SwapModule (functions for QuickSwapV3SwapLibrary).

UniswapV3StakeLibrary

The library contains logic for interacting with UniswapV3.

AaveV3BorrowLibrary

The library contains logic for interacting with AaveV3.

Actors

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

Admin

The role is responsible for setting all parameters for the strategy contract.

Unit

The role can set a few parameters for the strategy contract, namely, slippages, lower and upper percent and needed health factor.

5.2. Trust model

Users should trust the Admin role to set the parameters correctly. The Admin role should be trusted to not set the parameters in a way that would cause the system to lose funds. Moreover, users should trust the overlying contract that implements the public entrypoints for the strategy contract that contains only internal methods for the logic.

M1: Missing data validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	StrategyUs3UsdcWeth	Type:	Data validation

Description

The project does not perform zero-address checks in the `setParams` function and lacks other forms of validation such as allowed ranges for integers.

Exploit scenario

By accident, an incorrect value is passed to the `setParams` function. Instead of reverting, the call succeeds.

Recommendation

Implement zero-address checks in the `setParams` function and consider adding allowed ranges for variables like `poolFee` where is not needed to use the whole integer range.

Client's response

Acknowledged by the client.

> We not add checks in method setParams because in most cases our contracts not have free spaces for these checks. If add these checks then we cannot deploy contract to chain. We setup params by deploy scripts and CLI tool whise has checks for it, also if we need to update these params on exist contract then use special test scripts for it.

> — Overnight Finance

Go back to Findings Summary

M2: Usage of deprecated function

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	StrategyUs3UsdcWeth, UniswapV3StakeLibrary, AaveV3BorrowLibrary, HedgeStrategy	Type:	Undefined behavior

Description

The contracts are using one or more functions from the following list:

- `baseToUsd`
- `usdToBase`
- `sideToUsd`
- `usdToSide`

These functions are using the deprecated `latestAnswer` function from the Chainlink aggregator. This function is deprecated and should not be used anymore. The `latestRoundData` function should be used instead.

The `latestAnswer` function does not error if no answer has been reached and returns 0 or can return stale prices when the API becomes deprecated.

Exploit scenario

The `latestAnswer` function returns 0, as a result, it causes the contracts to behave unexpectedly.

Recommendation

Replace the deprecated function with the recommended one and add proper validation for stale prices (checking `answeredInRound` against `roundId`, or `updatedAt` return values) and non-zero values (checking `answer` return value).

Fix 1.1

The `latestRoundData` function is used instead and there is added validation for round completeness, and if the obtained data are fresh according to round id.

The validation could be even more stringent, especially:

- checking for positive values:

```
require(price > 0, "Negative returned price");
```

- checking if the round is not too old:

```
require(timeStamp >= block.timestamp - HEARTBEAT_TIME , "Stale price
feed"); ①
```

① where `HEARTBEAT_TIME` is a constant that is set to a maximum desired freshness

Go back to Findings Summary

M3: Empty receive

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	StrategyUs3UsdcWeth	Type:	Locked Ether

Description

The contract contains an empty receive function that allows sending Ether to the contract address. However, there is no transfer function to obtain it back.

Exploit scenario

Bob sends 1 Ether to the contract address. The contract does not have a transfer function, so the Ether is locked forever.

Recommendation

If it is not desirable to receive Ether, revert on `receive`.

Fix 1.1

The empty receive function is removed.

Go back to Findings Summary

W1: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

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.

Vulnerability scenario

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

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Client's response

Acknowledged by the client.

> We tried to remove optimizations, but got a 2-fold increase in the size of contracts.
>
> — Overnight Finance

Go back to Findings Summary

I1: Borrow module is missing implementation for claiming rewards

Impact:	Info	Likelihood:	N/A
Target:	AaveV3BorrowLibrary	Type:	Missing implementation

Description

The `_claimRewards` function in the `AaveV3BorrowLibrary` library is missing implementation. As a result, no rewards can be claimed after calling this function.

Recommendation

Functions that are not implemented should be removed from the production code or should be properly documented why they are not implemented.

Fix 1.1

The `_claimRewards` function is now properly documented.

Go back to Findings Summary

I2: Documentation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Best practices

Description

The NatSpec documentation is missing. Functions should contain for example an explanation for function parameters and return values Also there is no other low-level documentation that could help to understand the code.

Recommendation

Cover all contracts and functions with NatSpec documentation and add general documentation for developers and auditors.

Fix 1.1

There are significantly more code comments, however, the full NatSpec documentation is still missing.

Go back to Findings Summary

I3: Unused function parameter

Impact:	Info	Likelihood:	N/A
Target:	StrategyUs3UsdcWeth	Type:	Dead code

Description

The `_claimRewards` function in the `StrategyUs3UsdcWeth` contract has an unused parameter `_to`. As a result, any address that is passed does not affect the function.

Recommendation

Remove the unused parameter or implement it.

Client's response

Acknowledged by the client.

> It's deprecated param, at the moment it deletion will lead to more refactoring, so we take note and get rid of it in the future
>
> — Overnight Finance

Go back to Findings Summary

6. Report revision 1.1

The following issues were fixed:

- M2: Usage of deprecated function,
- M3: Empty receive,
- I1: Borrow module is missing implementation for claiming rewards,

and the rest of the issues were acknowledged or fixed partially. For more information see each finding in Summary of Findings.

Appendix A: How to cite

Please cite this document as:

> Ackee Blockchain, Overnight Finance: StrategyUs3UsdcWeth, 3.3.2023.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq