ackee blockchain

GoblinGold

Nazare program

by Ackee Blockchain

21.9.2022

Contents

1. Document Revisions

0.1	Draft report	2.9.2022
1.0	Final report	21.9.2022

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 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, Rockaway Blockchain Fund.

2.2. Audit Methodology

The Ackee Blockchain auditing process follows a routine series of steps:

1. **Code review**
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

    missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows, ...

c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.

d. Review of best practices to improve efficiency, clarity, and maintainability.

2. **Testing and automated analysis**

   a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trdelnik.

3. **Local deployment + hacking**

   a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are characteristic of each program audited. However, when trying to attack, we rely on the information gained from previous steps and our rich experience.

2.3. Review team

Member's Name	Position
Tibor Tribus	Lead Auditor
Vladimír Marcin	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

GoblinGold is a yield aggregating vaults platform built on Solana, automating the selection of Orca's whirlpools parameters. Enabling liquidity providers to unlock their liquidity by minting LP-token instead of the Whirlpools NFT.

GoblinGold engaged Ackee Blockchain to conduct a security review of their ggoldca program with a total time donation of 10 engineering days. The review took place between 15. July and 30. July 2022.

The scope included the following repository with a given commit:

- **Repository**: goblingold
- **Commit**: `bb54cee6e7b0ca61cdec5c625a1994b00cac5c06`
- **Tag**: `audit1`
- **Program**: `ggoldca`

The beginning of the audit was dedicated to understanding the entire Goblin Gold platform. We then took a deep dive into the `ggoldca` program. We paid particular attention to:

- Is the correctness of the program ensured (does it correctly implement the project specifications)?
- Do the program correctly use dependencies or other programs they rely on (e.g. Orca's, SPL dependencies)?
- Is the code vulnerable to economic attacks?

Our review resulted in only two informational findings. Overall code quality is high and follows a high standard of Orca's code base.

Supporting documentation sufficiently describes the system and has significantly helped us understand the system overview. The tests

sufficiently verified the basic functionality and helped us understand the system. The tests were also a good starting point in our attempts to attack the audited program.

Ackee Blockchain recommends GoblinGold:

- Expand the documentation describing dynamic fee design, which can result in users getting penalty fees.

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. Programs

Nazare program

The `nazare` program is an on-chain program that builds on top of the Orca DEX. Nazare programs leverage Orca's Concentrated Liquidity product Whirlpools and aim to eliminate the tedious manual selection of whirlpools parameters with the creation of a vault that manages the price range for the liquidity, and auto compound fees and rewards. Users are motivated to keep their liquidity deposited for at least two entire periods in order to get some rewards. For more information see docs.

4.2. Actors

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

Authority

Authority has the right to create a global state, vault, open positions, close positions, and trigger a rebalance between two open positions. Authority can set protocol parameters such as the vault fees and market rewards.

User

Users can deposit liquidity which results in getting back LP tokens representing their share in the pool. Users can decide to withdraw their liquidity by burning LP tokens and collecting the rewards from the position.

Anyone can call the functions `collect_fees`, `collect_rewards`, `swap_rewards`, `transfer_rewards` and `reinvest`. They are checked in order to behave correctly even for malicious actors.

4.3. Trust model

Users of the protocol have to trust the authority as it has the power to change protocol parameters.

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

		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.

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
I1: Redundant Signer constrain	Info	Info	N/A
I2: Missing tick interval check	Info	Info	N/A

Table 2. Table of Findings

I1: Redundant Signer constrain

Impact:	Info	Likelihood:	N/A
Target:	open_position.rs	Type:	Account constrain

Description

In `open_position` instruction, there's a redundant check whether the passed account is `signer`. This constrain is ensured by init constrain in Orca's `OpenPosition` context.

```
#[account(signer, mut)]
/// CHECK: whirlpool cpi
pub position_mint: AccountInfo<'info>,
```

Recommendation

Remove the redundant constraint.

Go back to Findings Summary

I2: Missing tick interval check

Impact:	Info	Likelihood:	N/A
Target:	open_position.rs	Type:	Application logic

Description

When the user calls `open_position` instruction, the current implementation does not check for the ticks index. This allows the user to send nonvalid tick indexes. However, the `open_position` instruction does a CPI call to the Orca program where a sufficient check for tick indexes is in place.

```
whirlpool::cpi::open_position(
        ctx.accounts.open_position_ctx(),
        whirlpool::state::position::OpenPositionBumps {
            position_bump: bump,
        },
        tick_lower_index,
        tick_upper_index,
    )?;
```

Recommendation

Add explicit check for `tick_lower_index < tick_upper_index`.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, GoblinGold: Nazare program, 21.9.2022.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq