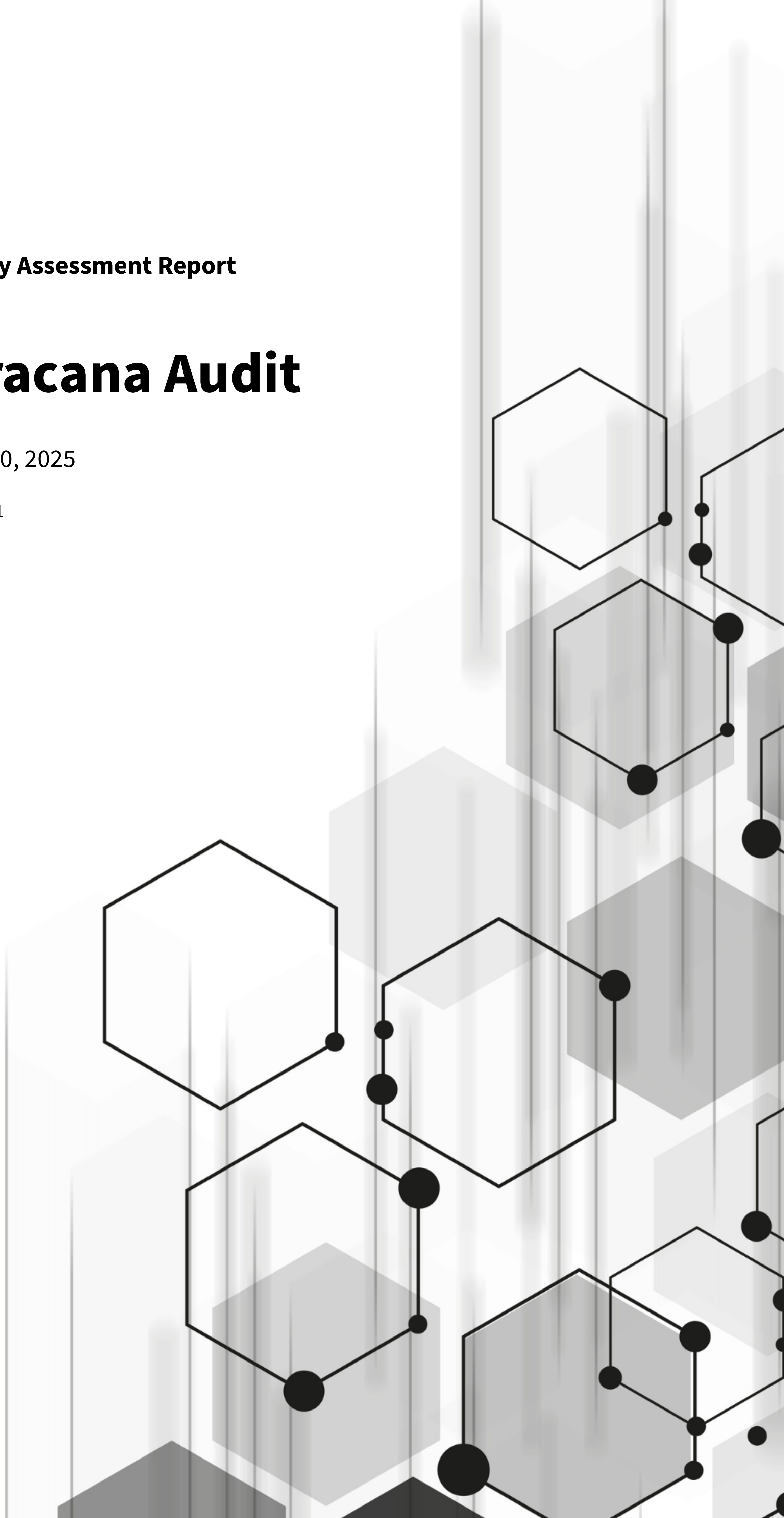
Security Assessment Report

Beracana Audit

March 10, 2025

Version 0.1



Contents

1 Confidentiality statement

2 Disclaimer

3 About Sub7

Founded in 2022, Sub7 Security is a pioneering cybersecurity company dedicated to creating innovative and efficient solutions for a secure digital future. In 2023, we established our presence in Luxembourg after a successful pitch of our cutting-edge platform, SecHub. Our solutions earned us the prestigious recognition of being named one of the top three cybersecurity solutions in Luxembourg, further cementing our position as a leader in the field.

We are the creators of SecHub, a transformative platform designed to deliver fast, transparent, and secure cybersecurity management. SecHub empowers clients by providing direct access to top-tier security researchers, real-time findings, and rapid issue resolution. Our services include smart contract audits, penetration testing, and a range of tailored security solutions. By significantly reducing audit timelines while

maintaining robust security, SecHub is revolutionizing the way organizations manage their digital risks.

Our team brings together a wealth of expertise spanning banking, finance, cybersecurity, law, and business management. We also work closely with a dedicated lawyer, ensuring compliance with regulatory standards and providing a comprehensive approach to security and legal considerations.

Our team members have professional experience at leading organizations such as ServiceNow, Polygon, Tokeny, Bitstamp, Kreditech, Mash, Deloitte, and Bitflyer, bringing invaluable insights from diverse industries.

At Sub7 Security, we are committed to innovation, trust, and resilience. By combining cutting-edge technology, industry expertise, and legal acumen, we deliver solutions that empower businesses and build a safer, more secure digital ecosystem.

Join us as we shape the future of cybersecurity.

4 Project Overview

Beracana focuses on providing users with a way to amplify their gains while participating in the Berachain ecosystem. At its core, Beracana facilitates undercollateralized lending, meaning users can deposit assets and borrow more than the value of their initial deposit, up to 5x the leverage. The protocol then directs these borrowed assets into liquidity farming strategies, optimizing BGT yields across the various dApps in the ecosystem.

Using Beracana's infrastructure, BERA and other assets can be deployed into various farming strategies. By leveraging into multiple dApps like Infrared, Kodiak, and Beradrome, users can simultaneously farm iBGT, oBERO, KDK, or any other farm tokens.

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

1 (One) Security Auditors/Consultants were engaged in this activity.

5.1 Scope

https://github.com/AuroBlocks-Studios/beracana-contracts.git

5.2 Timeline

From 20/02/2025 - 10/03/2025

5.3 Summary of Findings Identified

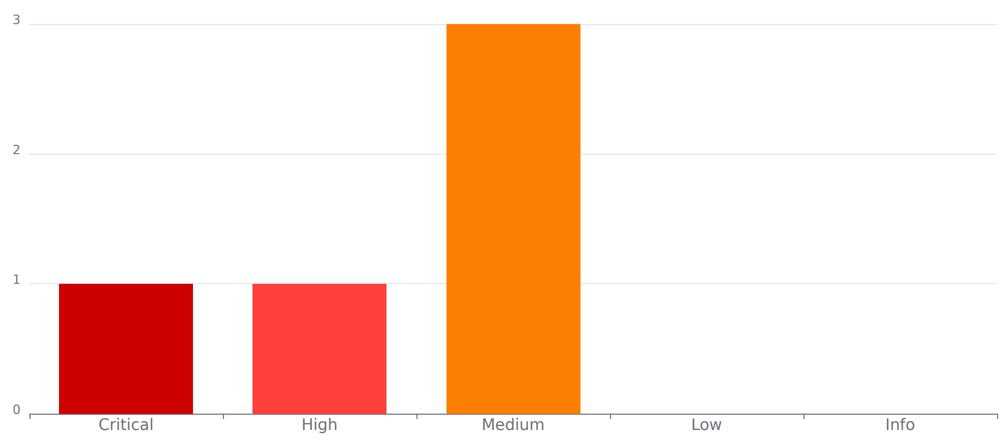


Figure 1: Executive Summary

# 1 Critical Deposit Function Fails to Mint Shares if Initial Tokens Are Transferred Externally – ***Fixed***

# 2 High `amountOutMinimum` Set to Zero in `addStrat` function Leads to Slippage Vulnerability – ***Fixed***

# 3 Medium Hardcoded or Fixed Slippage can lead to Loss of User funds – ***Fixed***

# 4 Medium Unchecked transfers return value – ***Fixed***

# 5 Medium On-Chain Testing Mode Allows Unauthorized Access – ***Fixed***

5.4 Methodology

SUB7's audit methodology involves a combination of different assessments that are performed to the provided code, including but not limited to the following:

Specification Check

Manual assessment of the assets, where they are held, who are the actors, privileges of actors, who is allowed to access what and when, trust relationships, threat model, potential attack vectors, scenarios, and mitigations. Well-specified code with standards such as NatSpec is expected to save time.

Documentation Review

Manual review of all and any documentation available, allowing our auditors to save time in inferring the architecture of the project, contract interactions, program constraints, asset flow, actors, threat model, and risk mitigation measures

Automated Assessments

The provided code is submitted via a series of carefully selected tools to automatically determine if the code produces the expected outputs, attempt to highlight possible vulnerabilities within non-running code (Static Analysis), and providing invalid, unexpected, and/or random data as inputs to a running code, looking for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Examples of such tools are Slither, MythX, 4naly3er, Sstan, Natspec-smells, and custom bots built by partners that are actively competing in Code4rena bot races.

Manual Assessments

Manual review of the code in a line-by-line fashion is the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found. This intensive assessment will check business logics, intended functionality, access control & authorization issues, oracle issues, manipulation attempts and multiple others.

Security Consultants make use of checklists such as SCSVS, Solcurity, and their custom notes to ensure every attack vector possible is covered as part of the assessment

6 Findings and Risk Analysis

6.1 Deposit Function Fails to Mint Shares if Initial Tokens Are Transferred Externally

Severity: Critical
Status: Fixed

Description

The `deposit` function in the contract relies on the `totalFunds()` value to determine the share allocation when minting tokens. However, if an attacker transfers 1 `honey` token directly to the contract before any deposits are made, the logic inside the function will fail in an unintended manner:

```
        uint256 share = total == 0 ? amount : amount.mul(totalSupply()).div(total);
        _mint(account, share);
```

1. Since `totalFunds()` is now 1 (due to the external transfer), the condition `total == 0` will be false.
2. The calculation `amount.mul(totalSupply()).div(total)` will evaluate with `totalSupply()` as `0`, since no shares have been minted yet.
3. As a result, the depositors receives `0` shares, as mint function tries to mint `0` shares everytime.
4. Without any shares, the depositor cannot withdraw, leading to a loss of funds.

This effectively locks user funds in the contract and prevents any future deposits from functioning correctly.

Location

master/contracts/LBGTBankV2.sol#L821-L822

master/contracts/LBGTBankV2.sol#L833-L834

Recommendation

Implement the ERC4626 standard, which provides a robust framework for tokenized vaults and includes safeguards against inflation attacks.

Comments

6.2 `amountOutMinimum` Set to Zero in `addStrat` function Leads to Slippage Vulnerability

Severity: High
Status: Fixed

Description

The `addStrat` function contains vulnerability where the `amountOutMinimum` parameter in the swap execution is set to `zero`. This allows for potential slippage attacks, enabling an attacker to front-run the swap and manipulate the price to drain funds from the contract. This exposes the contract to MEV (Maximal Extractable Value) and front-running attacks, where an attacker can manipulate liquidity pools to receive a near-zero output while still executing the swap.

```
        ISwapRouter.ExactInputSingleParams memory swapParamsTwo = ISwapRouter.
            ExactInputSingleParams({
            tokenIn: address(wbera),
            tokenOut: address(yeet),
            fee: 3000,
            recipient: address(this),
            deadline: deadline,
            amountIn: amountToSwap,
            amountOutMinimum: 0,    //@audit no slippage
            sqrtPriceLimitX96: 0
        });
```

Location

contracts/vault/LBGTVaultV3.sol#L622

Recommendation

Enforce a reasonable minimum output to protect against slippage

Comments

6.3 Hardcoded or Fixed Slippage can lead to Loss of User funds

Severity: Medium
Status: Fixed

Description

In vault contracts there is a hardcoded slippage values which is not adivsed as it can lead to loss of user funds when the volatility of token is high.

Location

contracts/vault/HoneyByusdVault.sol#L894

Recommendation

Consider allowing mechanism override/configure the existing slippage values.

Comments

6.4 Unchecked transfers return value

Severity: Medium
Status: Fixed

Description

ERC20 token transfers are performed using the transfer and transferFrom methods. These methods return a boolean value indicating the success of the operation, as per the ERC20 standard. However, the contract does not check these return values, which can lead to scenarios where token transfers fail.

Location

contracts/vault/HoneyByusdVault.sol#L808

Recommendation

Use SafeERC20 library from OpenZeppelin, which handles these inconsistencies and ensures compatibility.

Comments

6.5 On-Chain Testing Mode Allows Unauthorized Access

Severity: Medium
Status: Fixed

Description

Having testing mechanisms on-chain is bad because if testingMode is turned on, anyone can take advantage of it. This can lead to unauthorized access, bypassing of security restrictions, and potential exploitation.

Location

audit/contracts/BankV2.sol

Recommendation

Remove On-Chain Testing Mode, Use Off-Chain Testing Strategies.

Comments

sub7

FOLLOW US