BEOSIN
Web3 Security & Compliance

veSONIC

Smart Contract Security Audit

No. 202412091020

Dec 09th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing,1 Medium risk, 1 Low risk,2 info items were identified in the veSONIC project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Medium	Fixed : 1	Acknowledged: 0
Low	Fixed :0	Acknowledged: 1
Info	Fixed : 0	Acknowledged: 2

- **Project Description:**

The project audited this time is a token lock-up project where users can exchange tokens for corresponding rve-tokens to lock them up. Unlike typical lock-up mechanisms, rve-sonic's lock-up allows gradual claims based on predefined portion percentages and durations. Users can destroy the corresponding amount of rve-tokens to claim an equal amount of tokens when each portion's duration is reached. In addition to obtaining rve-tokens through the convert function, the project team can also distribute rve-tokens via the `generate_token` function. The team can pre-add tokens using `add_token_supply` to issue rve-tokens to users, and these rve-tokens can also be claimed when their respective durations are met.

In the December 9, 2024 update, the project team split the claim process into two parts: submitting a claim request and executing the claim request. Currently, after users perform a conversion, if they want to claim a specific number of tokens, they need to submit a corresponding claim request. This request will divide the tokens to be claimed into multiple portions based on predefined portions. After the duration for all the requested portions has been reached, users can withdraw their tokens using the `request_token_claim` function. It's important to note that if not all portions have reached their withdrawal time, the tokens in the remaining portions will not be withdrawable.

1 Overview

1.1 Project Overview

Project Name	veSONIC
Project Language	Rust
Platform	Solana
Github Link	https://github.com/mirrorworld-universe/rve-sonic-program-library

Commit	805e4ff21c52ecfbe0f943f490761942e6a5e1a4(origin)
	f854a1b7f1821de065b835e33b8fc73b7be18f40(fixed)
	cd30f54494d87915c3149bb40af9abc16711f638(update)

1.2 Audit Overview

Audit work duration: Nov 11, 2024 – Nov 18, 2024, Dec 9, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
veSONIC-01	Unverified Transfer Addresses	Medium	Fixed
veSONIC-02	Centralization Risk	Low	Acknowledged
veSONIC-03	Overflow Risk	Info	Acknowledged
veSONIC-04	Redundant Accounts	Info	Acknowledged

Finding Details:

[veSONIC-01] Unverified Transfer Addresses

Severity Level	**Medium**
Lines	convert_token.rs#133-153
Type	Business Security
Description	In the convert_token instruction, both the `rve_token_config_token_account` and `user_token_account` are passed as arguments in ctx but their authority and mint are not verified. This could allow an attacker to spoof the `rve_token_config_token_account` and convert tokens, effectively gaining access to rev tokens without cost.

```
    let rve_token_config_token_account = &ctx.remaining_accounts[0];
    let user_token_account = &ctx.remaining_accounts[1];
    let signer_seeds = &[
        RVE_TOKEN_CONFIG_ACCOUNT_PREFIX.as_ref(),
        rve_token_name.as_ref(),
        token_mint_account_key.as_ref(),
        &[rve_token_config_bump],
    ];
    let signer = &[&signer_seeds[..]];
    // 1: Transfer user Token for rve token
    let transfer_user_token_cpi_accounts = TransferChecked {
        from: user_token_account.to_account_info(),
        mint: ctx.accounts.token_mint_account.to_account_info(),
        to: rve_token_config_token_account.to_account_info(),
        authority: ctx.accounts.user.to_account_info(),
    };
    let transfer_user_token_cpi_context =
CpiContext::new(ctx.accounts.token_program.to_account_info(),
transfer_user_token_cpi_accounts);
    transfer_checked(transfer_user_token_cpi_context, amount,
ctx.accounts.token_mint_account.decimals)?;
```

Recommendation	It is recommended to verify the authority, mint, and other relevant data for both `rve_token_config_token_account` and `user_token_account`.
Status	**Fixed.**

[veSONIC-02] Centralization Risk

Severity Level	Low
Lines	generate_token.rs Withdraw_token_supply.rs
Type	Business Security
Description	In the withdraw_token_supply and generate_token instructions, the admin can directly withdraw tokens locked by users. This could result in users being unable to withdraw their tokens via the claim_token instruction once the lockup period expires, leading to potential user losses.
Recommendation	It is recommended to add a parameter in add_token_supply to track the available extra supply that can be added. When tokens are withdrawn in withdraw_token_supply or generate_token, the claim_amount should be deducted from the available extra supply to protect user locked tokens.
Status	**Acknowledged.**

[veSONIC-03] Overflow Risk

Severity Level	Info
Lines	initialize_rve_token_config.rs#100 update_rve_token_config.rs#53
Type	General Vulnerability
Description	In the `initialize_rve_token_config` and `update_rve_token_config` instructions, the total_percentage is stored as a u16, which could pose an overflow risk. `total_percentage += claim_portion_percentage;`
Recommendation	It is recommended to use methods like checked_add to prevent potential overflow risks.
Status	**Acknowledged.**

[veSONIC-04] Redundant Accounts

Severity Level	Info
Lines	add_token_supply.rs#13 claim_token.rs#98 Convert_token.rs#86 generate_token.rs#87 initialize_rve_token_config.rs#73 Initialize.rs#11,28 update_key.rs#12 update_rve_token_config.rs#12 update_user_index.rs#14 update_user.rs#48 withdraw_token_supply.rs#56
Type	Coding Conventions
Description	In several instructions, the fee_and_rent_payer and rent accounts are redundant. Typically, fee_and_rent_payer is used to pay for fees such as account creation or rent; however, in the add_token_supply instruction, fee_and_rent_payer is only passed as a Signer and is not directly used for any payments (e.g., rent or account initialization fees). This means it has no practical use. Meanwhile, Sysvar<Rent> is a system variable account used to fetch the current rent status (i.e., whether an account needs to pay rent, if the balance is sufficient, etc.). In several functions, the rent account is not used either, making it redundant.
Recommendation	It is recommended to remove the redundant accounts.
Status	**Acknowledged.**

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Critical**

Critical impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	SPL Token Standards
		Visibility Specifiers
		Lamport Check
		Account Check
		Signer Check
		Program Id Check
		Deprecated Items
		Redundant Code
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/