Zellic

Prepared for
Garry Walsh
Ruji Holdings

Prepared by
Frank Bachman
Ayaz Mammadov
Zellic

November 28, 2024

Rujira

Smart Contract Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Ruji Holdings from October 30th to November 5th, 2024. During this engagement, Zellic reviewed Rujira's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any issues that could result in the loss of user funds?
- Does the reward distribution work correctly?
- Are proper access controls implemented in the contracts?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

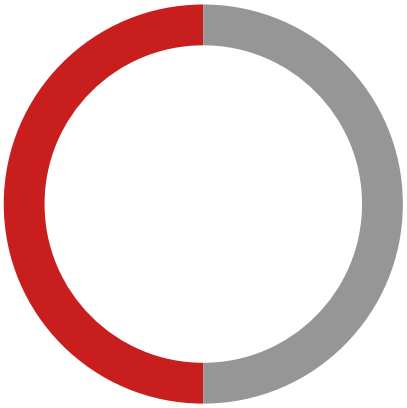
1.4. Results

During our assessment on the scoped Rujira contracts, we discovered two findings. One critical issue was found. The other finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Ruji Holdings in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	0
Medium	0
Low	0
Informational	1



2. Introduction

2.1. About Rujira

Ruji Holdings contributed the following description of Rujira:

> Rujira is the app-layer for THORChain, creating a smart contract platform for all L1 blockchains that THORChain connects to.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational"

finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Rujira Contracts

Type	Rust
Platform	Cosmos
Target	Rujira
Repository	https://gitlab.com/thorchain/rujira ↗
Version	`78506b97f60dd33da005ef40d507998fb09a8c46`

Programs	`contracts/rujira-merge/src/*`
	`contracts/rujira-revenue/src/*`
	`contracts/rujira-staking/src/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.5 person-weeks. The assessment was conducted by two consultants over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Frank Bachman
Engineer
Frank@zellic.io

Ayaz Mammadov
Engineer
ayaz@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

October 30, 2024	Kick-off call
October 30, 2024	Start of primary review period
November 5, 2024	End of primary review period

3. Detailed Findings

3.1. Withdrawal does not send tokens

Target	rujira-merge/state.rs		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The function responsible for withdrawing $RUJI from the protocol does not add a message sending the tokens to the user withdrawing.

```
#[cfg_attr(not(feature = "library"), entry_point)]
pub fn execute(...)
    ...
    ExecuteMsg::Withdraw { share_amount } => {
        nonpayable(&info)?;
        let amount = execute_withdraw(deps.storage, &config, time,
    &info.sender, share_amount)?;
        Ok(Response::default().add_event(
            Event::new("merge/withdraw")
                .add_attribute("account", info.sender)
                .add_attribute("shares", share_amount)
                .add_attribute("amount", amount),
        ))
    }
    ...
}
```

Impact

Users will not get their funds.

Recommendations

Add the relevant logic to send $RUJI to users withdrawing.

Remediation

Ruji Holdings remediated this issue in commit 86990dc0 ↗ by editing the relevant logic which adds `BankMsg::Send` to the response.

3.2. Revenue contract allows adding duplicate target denom

Target	rujira-revenue/contract.rs		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `SudoMsg` handler in the `rujira-revenue` contract does not check if a denom already exists in the `target_denoms` list before adding it.

```
pub fn sudo(deps: DepsMut, _env: Env, msg: SudoMsg) -> Result<Response,
    ContractError> {
    let mut config = Config::load(deps.storage)?;
    match msg {
[...]
        SudoMsg::AddTargetDenom(denom) => {
            config.target_denoms.push(denom);
            config.save(deps.storage)?;
            Ok(Response::default())
        }
```

This could result in a denom being accidentally added more than once.

Impact

The `target_denoms` list is used to distribute funds stored in the contract for each denom. A denom being duplicated in the list would result in duplicated `BankMsg::Send` messages being pushed. The transaction would always fail due to insufficient funds being present.

```
for target in config.target_denoms.clone() {
    distribute_denom(deps, &env, &config, &mut sends, target)?;
}
```

Recommendations

Use a HashSet for `target_denoms` or check if denom already exists before it's added.

Remediation

Ruji Holdings remediated this issue in commit 03820731 ↗ by changing the logic to use CosmWasm's `HashSet`s.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. First-deposit issue

Due to the truncations that exist within `execute_withdraw` and `execute_deposit`, there seems to be a possible path to exploit the first-deposit issue ↗, . However, this would require a first user that is able to inflate a single share to a large price. This is not a security issue in Rujira due to the initial four-week starting period in which decay will not exist; as such, there will be a large amount of seed liquidity, and truncation / share inflation is not possible in that period due to a 1:1 coupling of share/deposit price.

Ruji Holdings remediated this issue in commit a6410cc2 ↗ by changing the logic to work with `CosmWasm Dec` values instead of fixed integer calculations.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. rujira-merge

The rujira-merge contract allows exchanging merge tokens for $RUJI with a linear decay and a $RUJI reallocation mechanism.

The contract exposes two messages:

1. `ExecuteMsg::Deposit`, which allows depositing merge tokens in exchange for shares in the contract.
2. `ExecuteMsg::Withdraw`, which is used to withdraw $RUJI tokens from the contract by redeeming the user's shares in the pool.

The contract executes the `allocate` function before deposits/withdrawals. This calculates the unmerged supply of merge tokens and distributes the surplus $RUJI tokens proportionally across all existing mergers based on the result.

Invariants

Users are only able to withdraw $RUJI tokens proportional to their share in the pool.

Test coverage

- One large multitest in contract.rs.
- Several unit tests on the pool and relevant state-calculating functions.

Cases covered. Unit tests and execution tests verifying that the deposit/withdraw messages function correctly.

Cases not covered. A test case verifying that the user has received the expected amount of $RUJI tokens on withdraw.

Attack surface

There is a possibility of a first-deposit issue being present; however, due to the starting period of four weeks that allows users to enter the pools, in which no share-price decoupling from 1:1 can happen (no truncation can happen), such issue (4.1. ↗) seems unlikely.

5.2. rujira-revenue

The rujira-revenue contract is a smart contract that collects reward tokens and converts them into different assets to be distributed to $RUNE and $RUJI token holders.

An instance of rujira-revenue is deployed for each revenue token.

This is achieved through a structure where a privileged user sets actions, and then these actions are executed. Specifically, a function `ExecuteMsg::Run` is triggered, where these actions perform routing/swapping, and once the execution is finished, the remaining `targeted_tokens` are distributed among the stakeholders with reference to their stake.

Invariants

Addresses without the correct sudo privileges should not be able to set an important state that holds variables such as the

- executor (the address responsible for calling `ExecuteMsg::Run`),
- action (the steps/calls to make to convert/route certain tokens into other assets), and
- target denom (the denoms whose balance to check and distribute upon completion of execution).

Test coverage

- Large amount of testing coverage for authorization.
- Large amount of testing coverage for execution ordering.
- One test for distribution.
- Lack of testing/planning for failure of the messages.

Cases covered. All sudo function authorizations, execution ordering in one case, and dummy distribution.

Cases not covered. Integrated testing where one or more steps fail (slippage too high, how to reset, etc.).

Attack surface

There is very little attack surface due to the program being entirely privileged. The only possible issues that are out of scope are the correctness of the actions and the correct handling of keys, which should be controlled with the relevant protection (multi-sig wallet, timelock).

5.3. rujira-staking

The rujira-staking contract is a smart contract that is responsible for revenue distribution with built-in liquid staking.

Depositors have the option between:

- staking in an account for yield paid in `revenue_denom`, typically $USDC, periodically manually claiming, and
- minting a liquid token, representing a share of a pool, which claims `revenue_denom` and swaps to `bond_denom`, increasing the size of the pool over time.

Invariants

The revenue distribution between account/liquid should be proportional by weight.

Test coverage

- One large multitest in contract.rs.
- Several unit tests on the pool and relevant state-calculating functions.

Cases covered. A large test ensuring that entering different distributions remain correct in the life cycle of the protocol, unit tests, and dummy distribution.

Cases not covered. Calculated life-cycle tests ensuring that staking in one pool is not more profitable than another pool (to a certain degree, factoring the autoclaiming).

Attack surface

There is a possibility of a first-deposit issue being present; however, due to the starting period of four weeks that allows users to enter the pools, in which no share-price decoupling from 1:1 can happen (no truncation can happen), such issue (4.1. ↗) seem unlikely.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Thorchain network.

During our assessment on the scoped Rujira contracts, we discovered two findings. One critical issue was found. The other finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.