Audit Report

Out

v1.0

January 27, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Out to perform a security audit of the Out CosmWasm contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/stephanOUT/rust-contract
Commit	`78f20ed3f775fbdde424d67550dcc01f663efe28`
Scope	All contracts were in scope.
Fixes verified at commit	`0e2cc8f239669493189f5fbdabe7c78f4b40d7f3` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Out is an all-in-one social dApp for Web3 allowing users to buy and sell shares of other Out users.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	Medium-High	-
Level of documentation	Low	We recommend adding more explanatory comments to the code to ensure that the intents and decisions are well documented.
Test coverage	N/A	Determining the test coverage with `cargo tarpaulin` was not possible due to the test suite in `inj_test.rs` failing.

Summary of Findings

No	Description	Severity	Status
1	The contract can be drained completely, and attackers can pay in esoteric currencies	Critical	Resolved
2	Zero-amount token transfers will fail, preventing shares from being traded	Minor	Resolved
3	Missing address validation	Minor	Resolved
4	Economic risk due to unsustainable pricing model	Minor	Acknowledged
5	Inability to change the contract owner	Informational	Acknowledged
6	The `toggle_trading` function potentially emits misleading events	Informational	Resolved
7	Remove unused imports, errors, and variables	Informational	Resolved
8	Use of magic numbers and hardcoded values	Informational	Resolved
9	Overflow checks not enabled for release profile	Informational	Resolved

Detailed Findings

1. The contract can be drained completely, and attackers can pay in esoteric currencies

Severity: Critical

In `src/user/execute/buy_shares.rs:47,85` `info.funds[0].amount` is used to determine the amount of `inj` sent to the contract, but it is never checked if it is actually denominated in `inj`. Consequently, the current payment system in the contract allows users to pay in any denomination. This flexibility leads to two critical consequences:

1. Diverse Currency Acceptance: Users can pay with a currency different from the intended one.
2. Exploitation of Excessive Payment Refunds: More concerning is the vulnerability arising from the mechanism of refunding excessive payments. When a user overpays in a currency other than the desired `inj`, the system automatically calculates and refunds the excess amount in `inj`. This process can be exploited by a malicious attacker. By intentionally overpaying with a different currency, an attacker can trigger a refund that drains the entire balance of the contract, as the refund is made in `inj`.

Recommendation

We recommend checking that the transaction contains only the expected coin and no additional native tokens using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

2. Zero-amount token transfers will fail, preventing shares from being traded

Severity: Minor

In many instances, the `BankMsg::Send` message is used to transfer collected fees, denominated in `inj` tokens, to the respective recipients. However, if the transfer amount is zero, the `BankMsg::Send` message will fail, causing the transaction to revert and shares not to be bought or sold. While this is explicitly checked for the `referral_fee` fee in `src/user/execute/sell_shares.rs:76`, such a check is missing in all other instances:

- `src/user/execute/buy_shares.rs:69-72`, `74-77`, and `78-81`.
- `src/user/execute/sell_shares.rs:66-69` and `71-74`.

Recommendation

We recommend only sending a `BankMsg::Send` message if the coin transfer amount is greater than zero.

Status: Resolved

3. Missing address validation

Severity: Minor

In many instances, addresses supplied as type `Addr` to messages are not validated and are used as-is. While any invalid addresses supplied to the `buy_shares` and `sell_shares` function will error in the `BankMsg::Send` message, it is considered best practice and a better user experience to validate addresses early on and error with a meaningful error message.

For example, in `src/owner/execute/set_fee_destination.rs:15`, a configuration error of a wrong `state.protocol_fee_destination` address could lead to every transaction failing.

Recommendation

We recommend validating addresses with the `deps.api.addr_validate` utility function.

Status: Resolved

4. Economic risk due to unsustainable pricing model

Severity: Minor

The protocol's pricing curve follows an exponential rate of growth based on a formula with some magic numbers. There is no clear justification for the use of those numbers and the reason the price is an exponential function of supply.

For example, an early investor, specifically the first share buyer, would be in a 125% profit, even if they sell when the total supply is two. Their downside is also limited to the maximum amount of fees, which could be up to 25%. This makes early investors very well positioned to profit quickly and leaves later investors vulnerable to significant losses if people on a significant profit decide to materialize it.

In addition, the maximum amount of fees sets the spread at a hefty 25%, significantly increasing the margin of potential loss for users.

Also, it is worth noting that there is no cap on the supply, which would have been a relief factor in case the demand for the shares dropped.

Additionally, the potential for exploitation by individuals with access to automated trading tools or a substantial social media following is aggravated due to the exponential dynamics. These parties may leverage the protocol to artificially generate hype, manipulating market dynamics.

Nature of the Issue:

This concern, primarily economic, mirrors scenarios observable in low-liquidity markets without a predefined pricing curve. In such environments, practices like wash-trading can artificially inflate prices before a sell-off, or referrals are compensated through unofficial channels. It's important to note that while this issue is significant, it is classified as minor as it could arise in other low-liquidity markets without a pricing curve.

Nevertheless, by offering only an exponential curve, the pricing dynamics are preset, and the protocol also faces some reputational risk due to not offering a predictable environment for all participants, safeguarding against potential manipulative practices and fostering long-term viability.

Additionally, it's important to note that the pricing mechanism is particularly vulnerable to frontrunning and transaction ordering MEV: If an institution/block producer is able to frontrun, they are facing a determined price for each transaction. For buying transactions, the user will be only affected if they overpay because the transaction will fail otherwise, but for selling transactions, institutional traders might have a big advantage, prioritizing their own trades and thus creating larger losses for the user.

Recommendation

A redesign of the pricing logic is advised to mitigate incentives for exponential price dynamics. We recommend solutions with a balanced and sustainable pricing model - for example, locally exponential dynamics can be mitigated with logistic price curves.

Status: Acknowledged

Team Reply: *"The client states that an exponential bonding curve formula was adopted in order to achieve a highly attractive value proposition for its owner. This model allows prices to be determined by the market itself, where the 'supply and demand' automate the price, preventing manipulation of users determining prices they are willing to convert at, and instead only state their desired conversion amounts that determine the price.*

The team has considered the spread impact and therefore set it to 6% to prevent any significant loss on users end. Additionally, they state that the actual impact of selling would only bring down prices by 2% approximately in which they believe it to be a normal movement.

The team has acknowledged that they didn't limit the supply for two reasons. Mainly where limited supplies cause scarcity and possible price manipulation. Secondly, with growth reaching an 'almost 0' value causing no impact on the price.

Overall, the client believes that the vouching model provides an instant liquidity for the users and is considered safe while imposing no security risks or buying pressure."

5. Inability to change the contract owner

Severity: Informational

During the contract instantiation in the `instantiate` function in `src/contract.rs:37`, the contract owner is set to the message caller, `info.sender`. However, the contract can not be changed to a different address, presenting a risk to the contract ownership in case the owner account keys are compromised or a change of ownership is desired.

A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercase.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

Team Reply: *"The client believes that the disadvantages of implementing the possibility of changing the contract owner outweigh its advantages. This function, while allowing the rescue of the contract, can simultaneously be used by the hacker to transfer it to his own wallet, resulting in a complete loss of control.*

In its current state, the team has always a partial control over the contract in case of unfortunate events, which can be helpful in preventing certain damages. Additionally, the team has taken high-security measures to increase the safety of the contract wallet.

While it is not the client's focus, the team is aware of the possibility of transferring the contract. Under such conditions, if they exist, the team believes that legal measures will be taken to ensure the safety and transparency of information transfer."

6. The `toggle_trading` function potentially emits misleading events

Severity: Informational

The function `toggle_trading` in `src/owner/execute/toggle_trading.rs:5` is used to change the `trading_is_enabled` boolean state variable and subsequently prevent or allow buying and selling shares. This function emits events each time the state is set. However, the function could just reset the value of `trading_is_enabled` to the existing value. This would emit an event that off-chain participants may find confusing, as it might be assumed that this is a change in the state.

Recommendation

Consider only allowing an update of the `trading_is_enabled` variable when the new value is not equal to the current one.

Status: Resolved

7. Remove unused imports, errors, and variables

Severity: Informational

There are instances of unused imports, errors, and variables in the codebase:

- Unused error `InsufficientPayment` in `src/error.rs:28`.
- Unused error `BuySellQuantityLimitExceeded` in `src/error.rs:25`.
- Unused error `SerializationError` in `src/error.rs:19`.
- Unused error `NotFound` in `src/error.rs:16`.
- Unused error `CustomError` in `src/error.rs:13`.
- Unused variable `shares_holders` in `src/user/execute/buy_shares.rs:36`.
- Unused variable `shares_balance` in `src/user/execute/sell_shares.rs:19`.
- Unused import `StdError` in `src/user/execute/buy_shares.rs:6`.

Recommendation

We recommend removing unnecessary imports, errors, and variables to increase the code's readability and efficiency.

Status: Resolved

8. Use of magic numbers and hardcoded values

Severity: Informational

Throughout the codebase, there are many instances of magic numbers:

- Hardcoded denom `inj` in `src/user/execute/buy_shares.rs:71`, `76`, `80`, `89`, and `src/user/execute/sell_shares:63`, `68`, `73`, `79`.
- Magic fee limit values in `src/owner/execute/set_protocol_fee_percent.rs:14,36`,

`src/owner/execute/set_referral_fee_percent.rs:14,35`, `src/owner/execute/set_subject_fee_percent.rs:14,35`, and `src/contract.rs:36-43`.

- Magic values in pricing curve in `src/util.rs:13`.
- Magic values in the instantiation in `src/contract.rs:38-43`

Recommendation

We recommend removing hardcoded values and replacing them with constants.

Status: Resolved

9. Overflow checks not enabled for release profile

Severity: Informational

The contract in scope of the review does not enable `overflow-checks` in `Cargo.toml` for the release profile.

Recommendation

We recommend enabling overflow checks.

Status: Resolved