Audit Report

# Skip Protocol Swap Contracts

v1.0

August 20, 2023

# Table of Contents

## License

# Disclaimer

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 Skip Protocol Inc. to perform a security audit of the Skip Protocol Swap Contracts.

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/skip-mev/swap-contracts |
| Commit | `15651c376414c5ddb1f18ea816a77e126db8f80f` |
| Scope | All contracts were in scope. |

## 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

The functionality associated with this audit is designed to facilitate cross-chain experiences, like swaps and transfers between any two IBC-enabled chains and tokens in as few transactions as possible, with reliable multi-chain relaying, packet tracking, and more.

# 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-Medium** | The codebase approached complex interchain interactions with a low-complexity approach. |
| Code readability and clarity | **Medium-High** | The codebase contained thorough documentation and was written clearly. |
| Level of documentation | **Medium-High** | The codebase and control flows were thoroughly documented |
| Test coverage | **High** | `93.10%` test coverage, `2309/2480` lines covered. |

# Summary of Findings

| No | Description | Severity | Status |
|---|---|---|---|
| 1 | Failure in the Sudo handler causes funds to be left in the `ibc-transfer` contract, allowing attackers to steal them | Critical | Resolved |
| 2 | Swap contracts do not perform sender validation | Major | Resolved |
| 3 | Leftovers in contracts allow other users to retrieve them | Major | Resolved |
| 4 | `verify_and_create_contract_call_msg` does not prevent self-calls | Minor | Resolved |
| 5 | Permissionless `TransferFundsBack` message | Minor | Resolved |
| 6 | `query_simulate_swap_exact_coin_out` might return incorrect results if incorrect pair address is provided | Minor | Acknowledged |
| 7 | The entry point contract's instantiate function does not check for duplicate swap venues | Minor | Resolved |
| 8 | Unused errors in codebase | Informational | Resolved |

# Detailed Findings

## 1. Failure in the Sudo handler causes funds to be left in the `ibc-transfer` contract, allowing attackers to steal them

**Severity: Critical**

Both the Osmosis and Neutron `ibc-transfer` contracts do not validate the caller of the `IbcTransfer` message to be the `entry-point` contract. This is problematic because the `entry-point` contract is responsible for ensuring the funds are sent according to the specified amount, as seen in `contracts/entry-point/src/execute.rs:312-316`.

Specifically, the current `ibc-transfer` implementation allows the caller to send zero funds while specifying a valid coin amount that exists in the contract. Normally, this should not be a problem since the contract is not intended to hold any funds, as the funds are either sent to other chains through IBC or refunded back to the recovery address in case of an acknowledgment error or timeout.

The issue arises when there is an error when refunding the funds to the recovery address. In this case, the funds will be left in the contract and not refunded to the recovery address. This allows an attacker to execute the `IbcTransfer` message while specifying the `coin` parameter as the leftover funds and steal them by specifying the receiver as their address on another chain. This is possible because the `ibc-transfer` contract does not check that the `coin` parameter is what is provided in the `info.funds` of the message, as this check is performed by the `entry-point` contract.

The above situation would likely happen when there is an error when refunding the recovery address using the bank message. For example, suppose Neutron governance updates the timeout fee to 0 funds. This would cause the bank message to fail as Cosmos SDK disallows sending a zero amount of funds (see `contracts/networks/neutron/ibc-transfer/src/contract.rs:182`).

Additionally, this might happen when a user calls the `ibc-transfer` contract directly without going through the `entry-point` contract. Since the contract assumes the `recover_address` is validated by the caller in line `84`, an error will occur in line `211` if the provided address is invalid. This will create a situation where the `ibc-transfer` contract is holding funds.

Consequently, an attacker can steal funds from the contract by specifying the amount of `coin` held and sending them over to an address they control on another chain.

### Recommendation

We recommend adding validation in both `IbcTransfer` messages to ensure that only the `entry-point` contract is authorized to call them.

**Status: Resolved**

## 2. Swap contracts do not perform sender validation

**Severity: Major**

Both the Osmosis and Neutron `swap` contracts do not validate that the caller of the `execute_swap` function is the `entry-point` contract. The `entry-point` contract performs essential validations and properly constructs the `Swap` message.

One example of a potential misconfiguration that could be introduced by calling the `swap` contract directly is that the `create_astroport_swap_msg` function in `contracts/networks/neutron/swap/src/contract.rs:121-123` does not enforce a `max_spread` or `minimum_receive` value. In contrast, the `entry-point` contract's `execute_post_swap_action` function will effectively enforce a slippage check. If a user calls the `swap` contract directly, there is no guarantee of the amounts to be received.

This issue also exists in `contracts/networks/osmosis/swap/src/contract.rs:115`, as the osmosis swap is not constructed with a `token_out_min_amount`.

Consequently, users will be exposed to sandwich attacks if they call the `swap` contracts without going through the `entry-point` contract.

### Recommendation

We recommend adding validation to ensure that only the `entry-point` contract may call the `execute_swap` function of both contracts.

**Status: Resolved**

## 3. Leftovers in contracts allow other users to retrieve them

**Severity: Major**

In the `entry-point` contract, there is validation to ensure all funds are sent over to other chains or contracts. It is important to ensure no leftover funds remain in the contract because anyone can retrieve them. Below we demonstrate several scenarios that will cause funds to remain in the `entry-point` contract.

The `verify_and_create_user_swap_msg` function in `contracts/entry-point/src/execute.rs:366-368` ensures that the required `coin_in` amount is not greater than the received coin amount. This presents a case where if the `coin_in` amount is less than the received coin amount, the difference between them will remain in the contract.

The `verify_and_create_fee_swap_msg` function in lines `415-417` performs a swap for the required fees (presumably for the Neutron chain). Since there is no validation that all swapped fees will be fully used, leftovers might remain in the contract.

The `verify_and_create_ibc_transfer_adapter_msg` function in line `315` will distribute the IBC funds and the post-swapped amount. At the time of writing, Osmosis does not charge fees to perform IBC fund transfers. If the user provided IBC fees, the funds would remain in the `contracts/networks/osmosis/ibc-transfer` contract.

Consequently, excess funds will not be refunded to the user, allowing other users to receive them by performing another swap.

We classify this as a major issue because it would only happen due to user misconfigurations.

### Recommendation

We recommend adding a message that refunds the remaining funds to the caller in case of leftovers. The message can be added after dispatching the `PostSwapAction` message in `contracts/entry-point/src/execute.rs:91`.

**Status: Resolved**

## 4. `verify_and_create_contract_call_msg` does not prevent self-calls

**Severity: Minor**

The `verify_and_create_contract_call_msg` function in `contracts/entry-point/src/execute.rs:333` does not prevent a `contract_call_msg` from being constructed that calls the `entry-point` contract itself or any of the related `ibc-transfer` or `swap` contracts for various networks.

While very little can be done with this from an attacker's perspective, it is best practice to create a set of blocked addresses that fail in the `verify_and_create_contract_call_msg` function. This might become important as new functionality or new networks are added. As the protocol becomes more complex, this attack vector may be leveraged with other components to create an attack path.

### Recommendation

We recommend creating a set of blocked addresses that may not be called by the `verify_and_create_contract_call_msg` function.

**Status: Resolved**

## 5. Permissionless `TransferFundsBack` message

**Severity: Minor**

The `TransferFundsBack` messages in `contracts/networks/neutron/swap/src/contract.rs:57` and `contracts/networks/osmosis/swap/src/contract.rs:47` are permissionless.

We classify this issue as minor because the `swap` contracts only hold funds for the span of one transaction, so even though this message is callable by anyone, there won't be any funds in the contract to receive under normal circumstances. It would still be best practice to validate that the only caller of this message should be the contract itself.

### Recommendation

We recommend validating that the only caller of this message should be the contract itself for both contracts.

**Status: Resolved**

## 6. `query_simulate_swap_exact_coin_out` might return incorrect results if incorrect pair address is provided

**Severity: Minor**

In `contracts/networks/neutron/swap/src/contract.rs:226-247`, the `query_simulate_swap_exact_coin_out` function relies on the provided pool address to reverse-simulate the `denom_in` amount needed from the `denom_out` amount.

In the current validation, no validation ensures that the pool address holds both `denom_in` and `denom_out` assets. If a user provides a pair address that holds the `denom_out` but not the `denom_in`, the simulation succeeds with an incorrect denom returned. This is problematic because the coin will return as the operation's `denom_in` amount, not the other pool's asset.

For example, assume the `denom_in` is USDC while the `denom_out` is USDT. Naturally, the provided pool address should point to the USDC-USDT pair. If the user instead provided the OSMO-USDT pair, the simulation would return the amount denominated in OSMO. This is incorrect because the returned coin will be denominated in USDC, misleading users.

### Recommendation

We recommend querying the pool's contract address to ensure it holds both `denom_in` and `denom_out`. Using the example above, the pool address should hold USDC-USDT and not OSMO-USDT assets.

**Status: Acknowledged**

The client acknowledges that querying and verifying that each pool holds the tokens would be a more explicit way of simulating swaps, but they believe the extra gas consumption of each query and higher complexity of maintaining implementations of swap adapters outweighs the benefit of an early error return. Moreover, the fundamental invariant that the user will receive their min out is anyways guaranteed in the entry point contract. If the swap adapter fails to output the correct amount needed for the `ibc_fees`, the transaction will still fail.

## 7. The entry point contract's `instantiate` function does not check for duplicate swap venues

**Severity: Minor**

In the `instantiate` function in `contracts/entry-point/src/contract.rs:17`, `msg.swap_venues` should be checked for duplicates. Otherwise, when the values are stored in a map, the last duplicated element would overwrite the contract address of the first. It is best practice to check for duplicates and return errors if they exist such that the caller can revisit the instantiation message to ensure no incorrect state is introduced.

We classify this issue as minor because it can only be caused by the contract instantiator.

### Recommendation

We recommend ensuring that no duplicate venues exist in `msg.swap_venues`. This can be accomplished by checking the length of the map in the `instantiate` function and returning an error if it is not equal to the length of `msg.swap_venues`.

**Status: Resolved**

## 8. Unused error in codebase

**Severity: Informational**

In `contracts/networks/osmosis/swap/src/error.rs`, an `Overflow` error is defined, but it is not used. Unused code decreases the readability of the codebase.

**Recommendation**

We recommend removing the unused error mentioned above.

**Status: Resolved**