GLOBAL Interlink

SMART CONTRACT AUDIT

zokyo

March 27th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

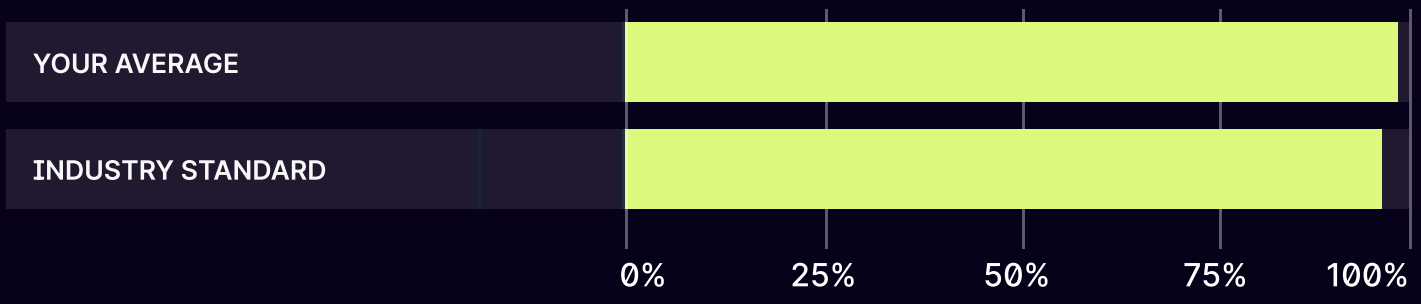
This document outlines the overall security of the Global Interlink (GIL) smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Global Interlink (GIL) smart contract's codebase for quality, security, and correctness.

Contract Status

There was 1 critical issue found during the audit. (See Complete Analysis)

Testable Code



98% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Sui network's fast-paced and rapidly changing environment, we recommend that the Global Interlink (GIL) team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Global Interlink (GIL) repository: https://github.com/Global-Interlink/smart-contracts/tree/54b25c16f4b8f8a723f94e6414e9ad245a330479/tokenomic/sources

Last commit: 54b25c16f4b8f8a723f94e6414e9ad245a330479

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- token_vesting.move
- utility.move

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Global Interlink (GIL) smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

There was one critical issue found during the audit, alongside two with high severity, two medium severity and two with low severity . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Global Interlink (GIL) team and the Global Interlink (GIL) team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Tx sender can steal the released tokens of any whitelisted user	Critical	Resolved
2	Whitelisted table balance and contract balance can be different resulting in locked funds for users	High	Resolved
3	Incorrect balance check	High	Resolved
4	Insufficient sanity checks for treasury creation	Medium	Resolved
5	Constant defined by a value that has not yet been published in the documentation	Medium	Resolved
6	The name of the function can cause a misunderstanding	Low	Resolved
7	Insufficient sanity check for add_to_whitelist	Low	Resolved

CRITICAL | RESOLVED

Tx sender can steal the released tokens of any whitelisted user

In contract token_vesting.move, function `release` is having the role of releasing the vested funds, first the function is checking that the parameter 'receiver' is a whitelisted address and after that is proceeding to calculate the amount that will be vested, adjust the storage and transfer the tokens, however there is one problem, the user that is checked for being whitelisted and is having his amount calculated is the 'receiver' variable, but the account that will actually receive the tokens is the function caller, there is where the problem is at L#225, the tokens in the end are not transferred to the receiver address but to the account that have called the function, which can be the receiver or not

Recommendation:

Implement proper sanity check by making sure the tx sender have the same address with the receiver or do not pass the receiver variable through the function parameters and just assign it the address of the caller

HIGH | RESOLVED

Whitelisted table balance and contract balance can be different resulting in locked funds for users

At this moment, the whitelisted address are added through the add_to_whitelist method and the funds of the pool are added through the deposit functionality, however these 2 steps pattern can become problematic for the contract integrity and the vested users, as the contract admin can add in the whitelist table tokens amount that do not reflect the true token balances in the contract, and users could be stuck with uncorrelated/hypothetical funds because there would be nothing for them to release from vesting

Recommendation:

Implement a check to never being able to add more hypothetical total funds in the table then the ones in contract balance or every time admin add a new user to the whitelist also transfer the necessary vested funds for that user directly in the same function logic.

HIGH | RESOLVED

Incorrect balance check

In contract utility.move, in function `merge_and_split` at line 31 there's an assertion between the base coin value and amount parameter. This is meant to ensure that there are enough remaining funds in the base coin balance, however there can be a case when the base coin remaining balance is equal to the amount parameter, so the assertion fails. This should not happen if the two values are equal

Recommendation:

Change the assertion from greater than to greater than or equal

```
public fun merge_and_split<T>(coins: vector<Coin<T>>, amount: u64
    let base = vector::pop_back(&mut coins);
    pay::join_vec(&mut base, coins);
    assert!(coin::value(&base) >= amount, E_INSUFFICIENT_FUNDS);
    (coin::split(&mut base, amount, ctx), base)
}
```

MEDIUM | RESOLVED

Insufficient sanity checks for treasury creation

In contract token_vesting.move, in function `create_treasury` there are no sanity checks for the `total_lock_in_days`, `vesting_period_in_days` and `initial_lock_in_days` variables. It would be ideal to add checks to prevent these values from being zero or being invalid combinations. Also, the `vesting_period_in_days` and `initial_lock_in_days` parameters are not checked to make sure they are not greater than the variable `total_lock_in_days` which could lead to unexpected behavior if the vesting period is longer than the total lock period. Even if these values are set in a function that's protected by admin rights, there is a chance that a mistake is made and not noticed before it leads to a possible issue.

Recommendation:

Add sanity checks to make sure that these parameters are set to the correct values.

MEDIUM | RESOLVED

Constant defined by a value that has not yet been published in the documentation

In contract utility.move the const EPOCH_SECONDS is supposed to be the time duration of an epoch in seconds. This const will affect the value returned by the functions `get_current_timestamp` and `get_timestamp_from_now_by_days`. The time duration of an epoch is present in the documentation just as an example, and there is no certainty that the value will remain the same.

Recommendation:

Be aware that the value of the const EPOCH_SECONDS comes from an example and not from the official documentation and there is no certainty that the value will remain the same and update the code accordingly.

LOW | RESOLVED

The name of the function can cause a misunderstanding

In contract utility.move, the function name "get_current_timestamp" could be misunderstood as a Unix timestamp, leading to potential confusion for developers who may rely on this function. It would be better to use a more specific name to avoid any confusion. By modifying the function name to "get_time_from_genesis", it will be clearer to developers that the function is not returning a Unix timestamp, but rather the time since the genesis block. This change will help to reduce confusion and potential errors in code that relies on this function.

Recommendation:

Modify the function name to "get_time_from_genesis" to more accurately reflect the function's purpose of calculating the time since the genesis block. This name will be more specific and easier for developers to understand, avoiding any potential confusion.

LOW | RESOLVED

Insufficient sanity check for add_to_whitelist

In contract token_vesting.move, in function `add_to_whitelist` there are no sanity checks for the whitelist_address and whitelist_amount vectors, there is one check that is making sure the vectors have the same length, however both of the vectors could have length `0` which will break the function invariant as this function will execute successful but the store will not be modified in any way.

Recommendation:
Add sanity checks to make sure that the arrays are not empty.

We are grateful for the opportunity to work with the Global Interlink (GIL) team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Global Interlink (GIL) team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo