QuillAudits

AUDIT REPORT

February 2025

For

hivello

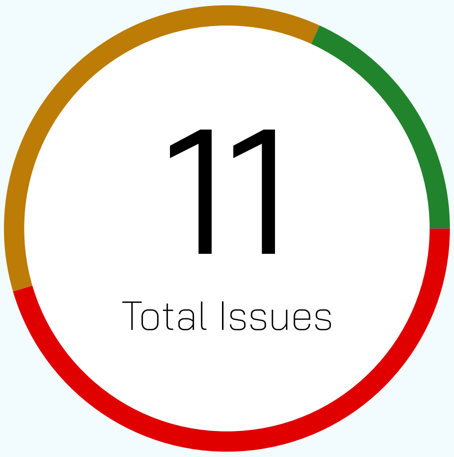
Table of Content

Executive Summary

Project name	Hivello
Overview	This protocol enables users to stake tokens and earn rewards through a tiered staking system. Key Features : Users can stake tokens into different tiers Each tier has specific requirements and benefits Rewards are distributed based on APY and lock duration Supports early unstaking (can be enabled/disabled)
Timeline	29th January 2025 - 7th February 2025
Update code Received	2025-02-07
Second Review	7th February 2025
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Blockchain	Solana
Audit Scope	The scope of this Audit was to analyze the Hivello Smart Contracts for quality, security, and correctness.
Source code	https://github.com/hivello/staking-contract/tree/main/programs
Contracts In Scope	1] programs/locking/src/lib.rs 2] programs/locking/src/utils 3] programs/locking/src/instructions 1. instructions/admin 2. instructions/user 4] programs/locking/src/events

Branch	Main
Commit Hash	701922a329b9bbdca699ee5080fb8fc09a0409f2

Number of Issues per Severity



🟥	High	5 (45.45%)
🟧	Medium	4 (36.36%)
🟩	Low	2 (18.18%)
🟪	Informational	0 (0.00%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	**5**	**4**	**2**	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

We have scanned the solana program for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- ✓ Signer authorization
- ✓ Account data matching
- ✓ Sysvar address checking
- ✓ Owner checks
- ✓ Type cosplay
- ✓ Initialization
- ✓ Arbitrary cpi
- ✓ Duplicate mutable accounts
- ✓ Bump seed canonicalization
- ✓ PDA Sharing
- ✓ Incorrect closing accounts
- ✓ Missing rent exemption checks
- ✓ Arithmetic overflows/underflows
- ✓ Numerical precision errors
- ✓ Solana account confusions
- ✓ Casting truncation
- ✓ Insufficient SPL token account verification
- ✓ Signed invocation of unverified programs

Techniques and Methods

Throughout the audit of Solana Programs, care was taken to ensure:

- The overall quality of code.
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.

Techniques and Methods

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

🟥 High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

🟧 Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

🟩 Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

🟪 Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** Security vulnerabilities identified that must be resolved and are currently unresolved.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

Missing Ownership Validation in Reward Claiming Function

Resolved

Path

programs/locking/src/instructions/user/handler_claim_rewards.rs

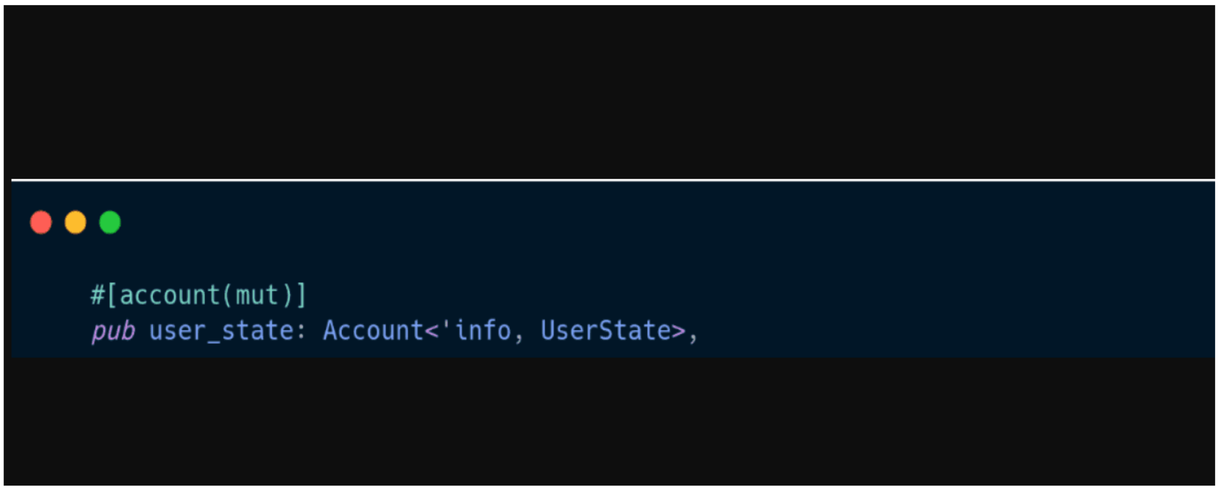
Function

claim_rewards

Description

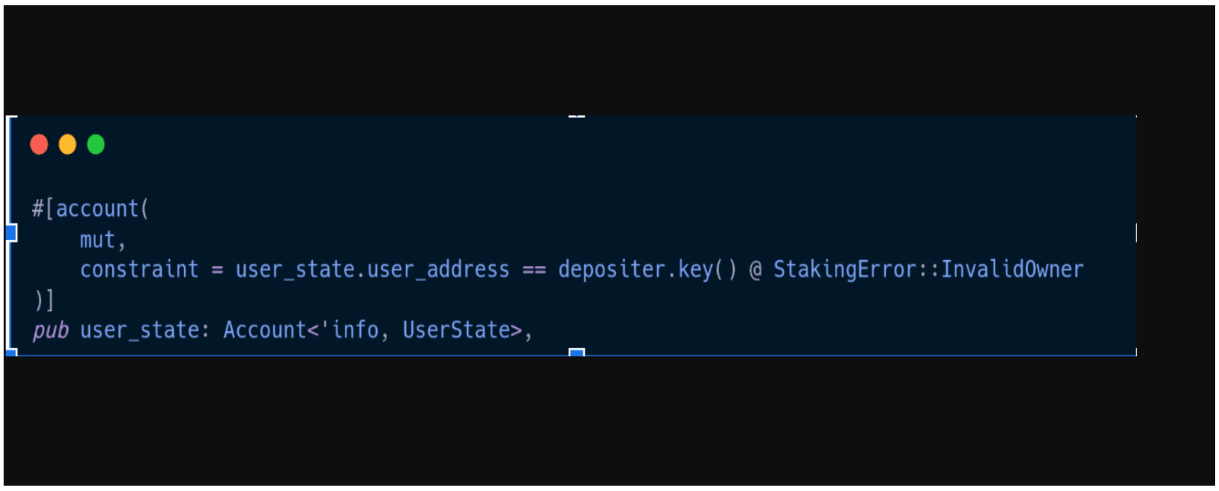
The staking contract's reward claiming function lacks ownership validation checks, which allows unauthorized users to claim rewards belonging to other users.
When calling the claim_rewards instruction, the contract does not verify if the caller is the actual owner of the UserState account.

An attacker can steal staking rewards from any user by:
Obtaining victim's UserState account address
Executing claim_rewards with their own payer_reward_token_ata but using victim's UserState
Successfully transferring victim's rewards to their account


```
#[account(mut)]
pub user_state: Account<'info, UserState>,
```

Recommendation

Add ownership validation in the claim_rewards instruction:



```
#[account(
    mut,
    constraint = user_state.user_address == depositer.key() @ StakingError::InvalidOwner
)]
pub user_state: Account<'info, UserState>,
```

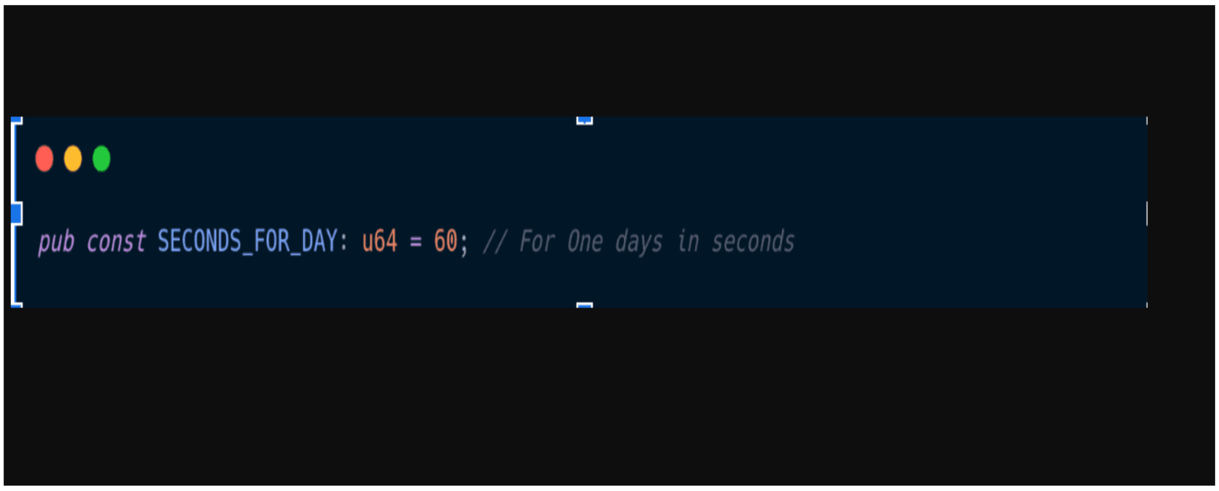
Incorrect Time Constant Leading to Invalid Days Calculation

Resolved

Description

The SECONDS_FOR_DAY constant is incorrectly set to 60 seconds instead of the correct value of 86,400 seconds (24 hours * 60 minutes * 60 seconds). This error affects the calculation of days_locked in the staking.

The incorrect constant leads to significantly wrong calculations of locked days:
Current: Using 60 seconds as one day
When user commits for 1 day (86,400 seconds), the calculation results in:
86,400 / 60 = 1,440 days (incorrect)
Expected: Using 86,400 seconds as one day
86,400 / 86,400 = 1 day (correct)



```
pub const SECONDS_FOR_DAY: u64 = 60; // For One days in seconds
```

Recommendation

Update the constant to the correct value with 86400

Reversed Pausing Logic in Reward Claims Validation

Resolved

Path

programs/locking/src/utils/accessors.rs

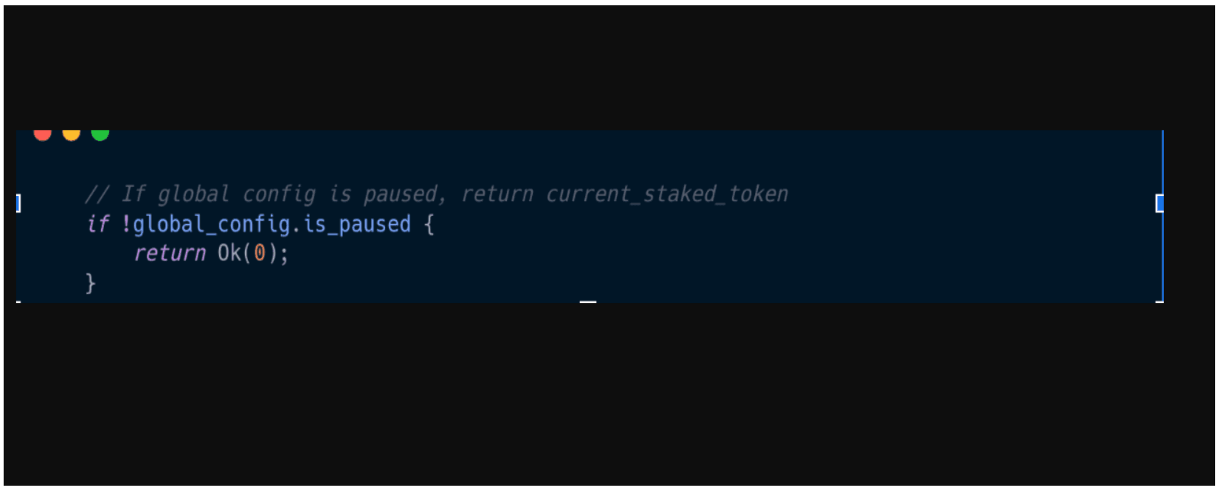
Function

validate_claim_rewards

Description

The validate_claim_rewards function has incorrectly implemented logic for the pausing mechanism. While the GlobalConfig struct includes an is_paused flag to prevent reward claims, the validation function implements the opposite behavior of what's intended.

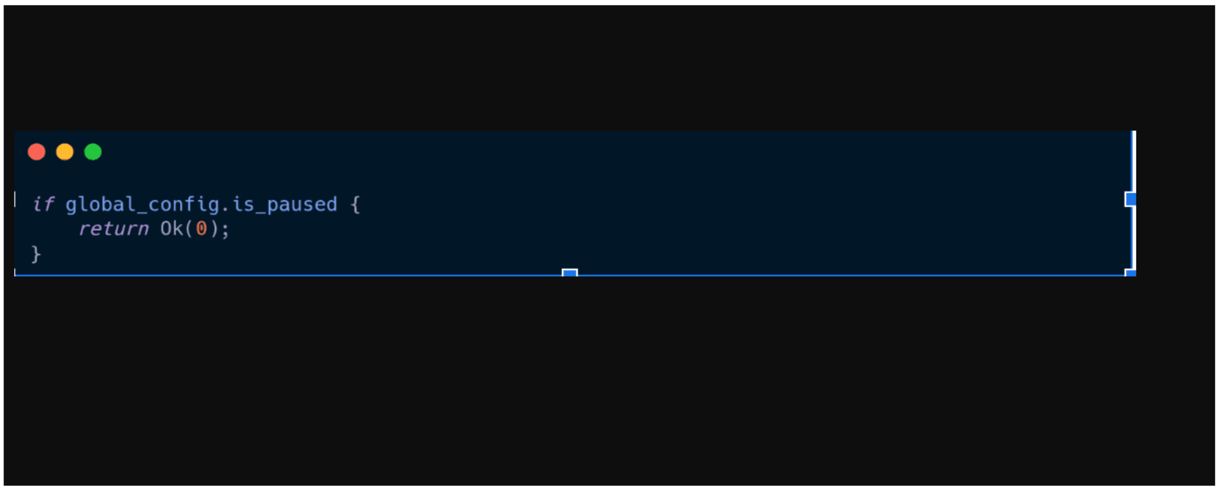
The reversed logic causes:
Users can claim rewards when system is paused (should be blocked)
Users are blocked from claiming when system is not paused (should be allowed)



```
// If global config is paused, return current_staked_token
if !global_config.is_paused {
    return Ok(0);
}
```

Recommendation

correct the if condition:



```
if global_config.is_paused {
    return Ok(0);
}
```

Precision Loss in Tier Requirements Calculation

Resolved

Path

programs/locking/src/utils/accessors.rs

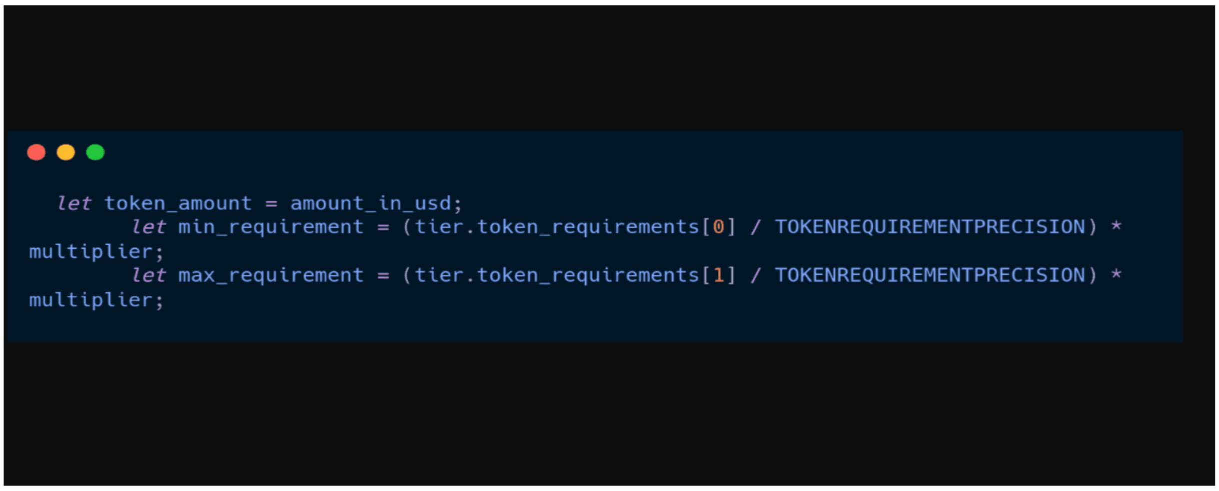
Function

assign_tier

Description

In the assign_tier function, the calculation order for tier requirements leads to precision loss. The current implementation divides token requirements by TOKENREQUIREMENTPRECISION (1000) before multiplication with the multiplier, causing potential rounding to zero.

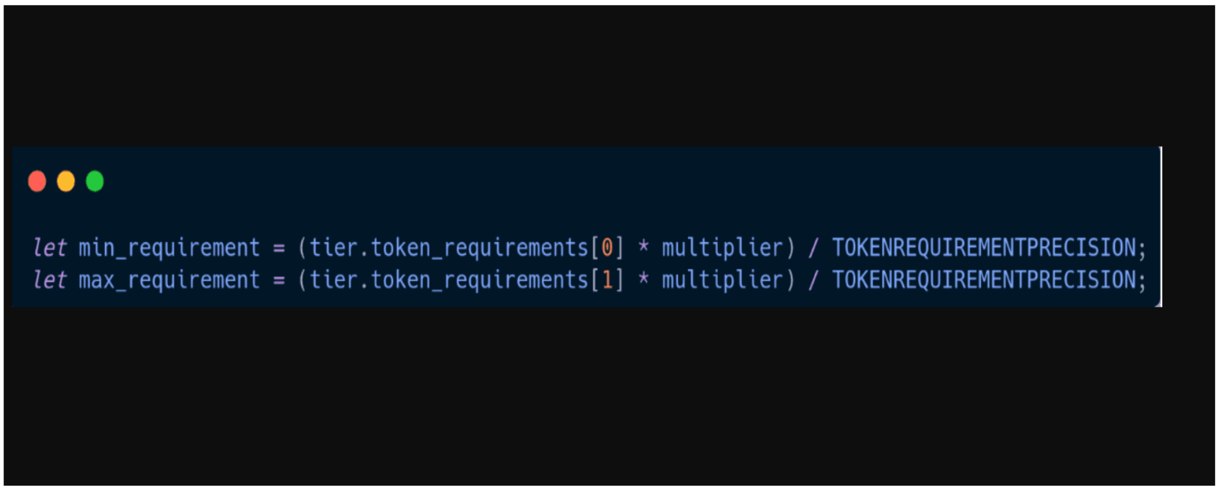
When tier.token_requirements values are less than 1000:
Division by TOKENREQUIREMENTPRECISION (1000) first results in 0
Subsequent multiplication with any multiplier yields 0
Function reverts with InvalidAmount error as no tier matches the requirements.



```
  let token_amount = amount_in_usd;
        let min_requirement = (tier.token_requirements[0] / TOKENREQUIREMENTPRECISION) *
multiplier;
        let max_requirement = (tier.token_requirements[1] / TOKENREQUIREMENTPRECISION) *
multiplier;
```

Recommendation

Change the calculation order to multiply before dividing:



```
let min_requirement = (tier.token_requirements[0] * multiplier) / TOKENREQUIREMENTPRECISION;
let max_requirement = (tier.token_requirements[1] * multiplier) / TOKENREQUIREMENTPRECISION;
```

Incorrect Default Pause State in Initialize Function

Resolved

Path

programs/locking/src/instructions/admin/handler_initialize_global_config.rs

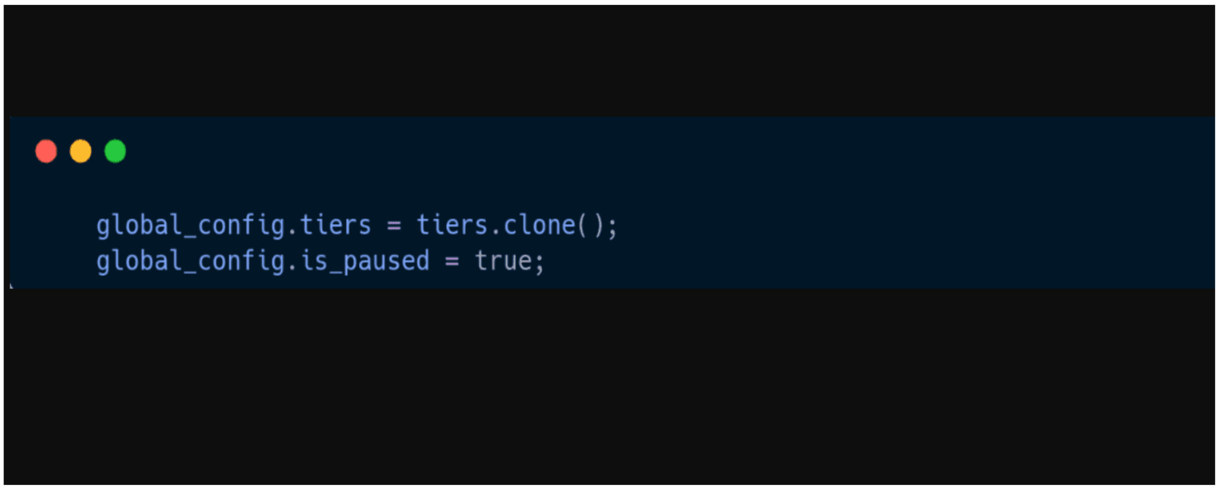
Function

initialize_global_config

Description

In the process function of InitializeGlobalConfig, the is_paused flag is set to true by default when initializing the global config. This immediately blocks users from claiming rewards after deployment.

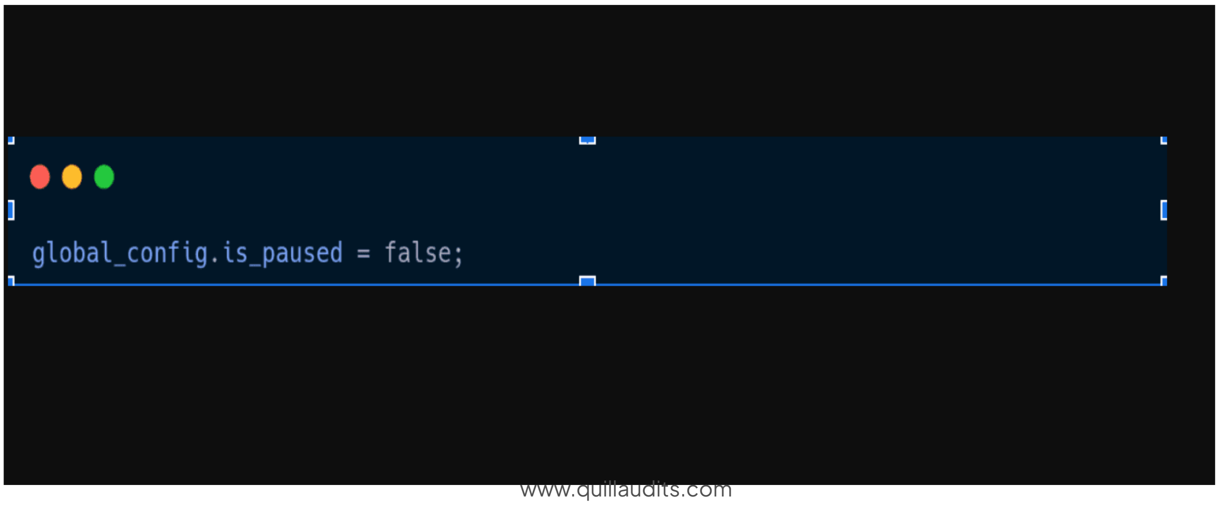
Setting is_paused = true during initialization causes:
Protocol starts in a non-functional state
Users cannot claim rewards immediately after deployment



```
global_config.tiers = tiers.clone();
global_config.is_paused = true;
```

Recommendation

Initialize the protocol in an active state :



```
global_config.is_paused = false;
```

Medium Severity Issues

Stake treasury and Reward treasury account creation DOS

Resolved

Path

programs/locking/src/instructions/admin/handler_initialize_global_config.rs,programs/locking/src/instructions/admin/handler_initialize_rewards.rs

Function

initialize_global_config,initialize_rewards

Description

The initialization of both stake_treasury and reward_treasury accounts uses the init constraint without checking if the accounts already exist. Since these accounts are PDA-derived token accounts, they can be created in advance by an attacker.

An attacker can cause a denial of service by:
Calculating the PDA addresses for treasury accounts using known seeds
Creating token accounts at these addresses before protocol deployment
Preventing the protocol initialization as init will fail if accounts exist

Reward Vault is Not Derived from Seeds

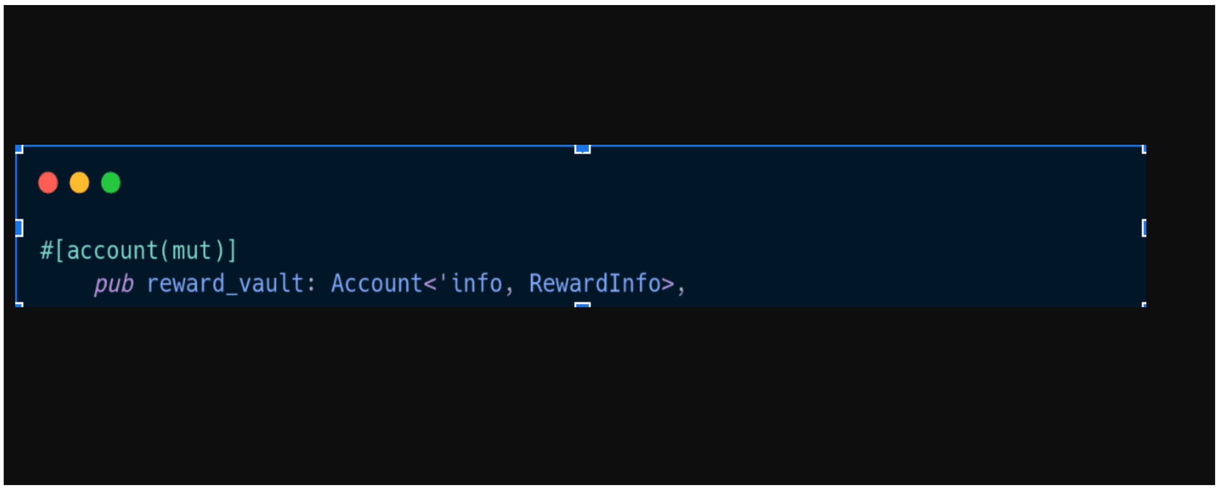
Resolved

Path

programs/locking/src/instructions/admin/handler_add_rewards.rs,
programs/locking/src/instructions/admin/handler_add_rewards.rs

Description

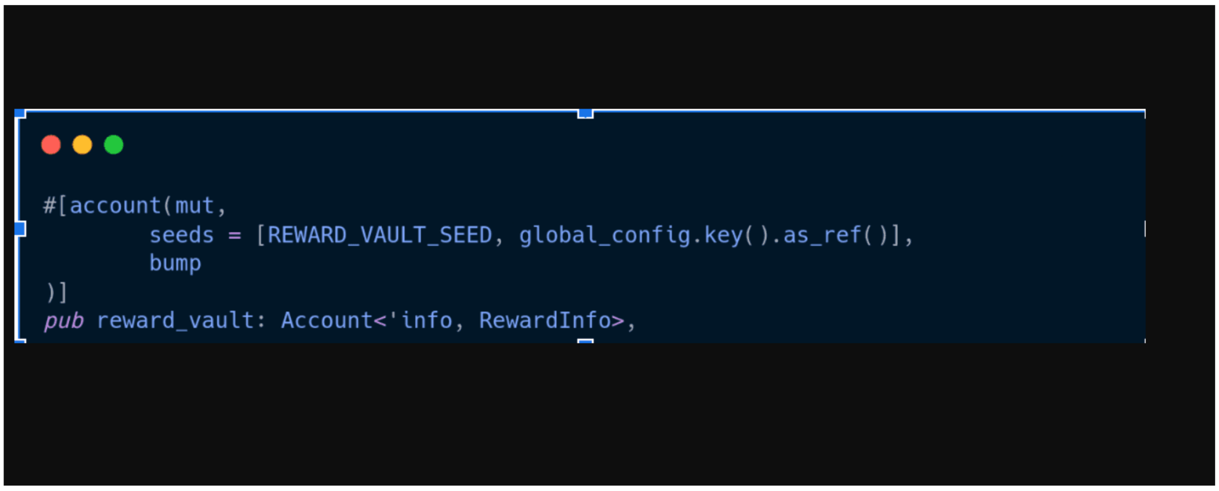
The reward_vault account in the handler_add_rewards and handler_claim_rewards instructions and is not verified to be derived from the expected seeds. This allows the function to accept any arbitrary account, which can lead to incorrect reward distribution.



```
#[account(mut)]
    pub reward_vault: Account<'info, RewardInfo>,
```

Recommendation

Derive the account from Seeds:



```
#[account(mut,
        seeds = [REWARD_VAULT_SEED, global_config.key().as_ref()],
        bump
)]
pub reward_vault: Account<'info, RewardInfo>,
```

global_config.total_staker Not Updating on User Closure.

Resolved

Path

programs/locking/src/instructions/user/handler_close_user.rs

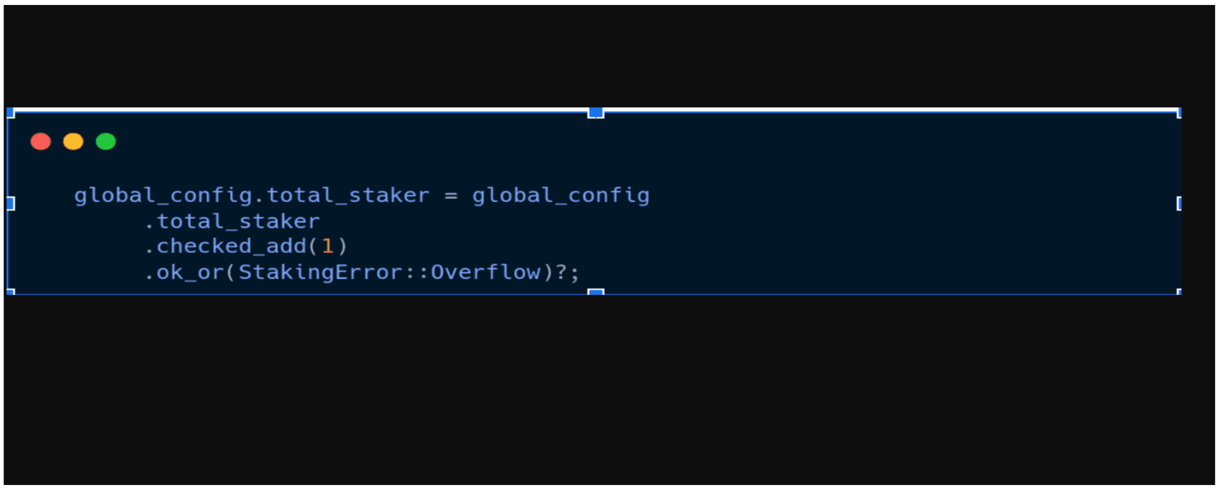
Function

close_staker

Description

The total_staker field in global_config is incremented when a user is initialized but not decremented when a user account is closed. This leads to an incorrect count of active stakers.

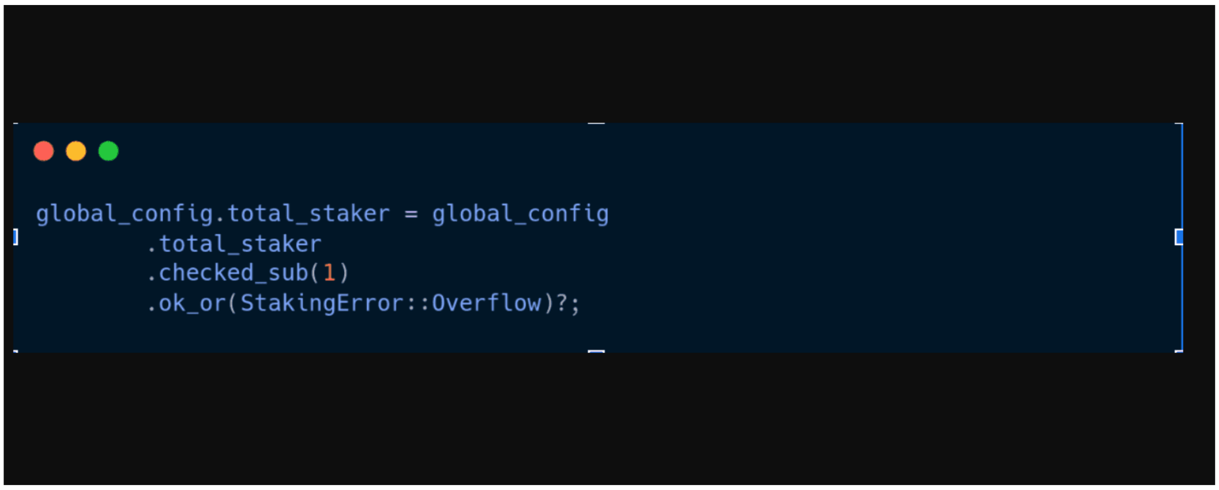
Code when initializing user_state :



```
global_config.total_staker = global_config
    .total_staker
    .checked_add(1)
    .ok_or(StakingError::Overflow)?;
```

Recommendation

Decrement total_staker when user account is closed:



```
global_config.total_staker = global_config
        .total_staker
        .checked_sub(1)
        .ok_or(StakingError::Overflow)?;
```

Missing multiple state updates claiming process.

Resolved

Path

programs/locking/src/utils/accessors.rs

Function

update_global_config,update_user_state

Description

During the staking process ,global_config.total_staked , user_state.stake_time, user_state.total_staked_token are updated correctly but when in reward claiming process these fields are not updated, leading to inconsistencies in global_config and user_state accounts.

Recommendation

Update these global_config.total_staked , user_state.stake_time fields in the update_global_config and update_user_state functions.

Low Severity Issues

Missing Check for total_reward > 0

Resolved

Path

programs/locking/src/instructions/user/handler_claim_rewards.rs

Function

process

Description

In the process function of the ClaimRewards implementation, there is no check to ensure that total_reward is greater than zero before executing the reward transfer logic. If total_reward is zero, the _transfer_from_vault function will still be called, leading to unnecessary computations, potential gas costs.

Recommendation

Add a check to ensure total_reward is greater than zero before proceeding with the reward transfer

No Verification of user_state in stake instruction.

Resolved

Path

programs/locking/src/instructions/user/handler_stake.rs

Function

process

Description

In the process function of Stake, there is no explicit check that the provided user_state account belongs to the actual staking user (payer)

Recommendation

Added a check to ensure that the user_state account is owned by the payer

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Hivello. We performed our audit according to the procedure described above.

Some issues of High, low and medium severity were found. In the End,Hivello Team,Resolved all Issues.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the submitted smart contract source code, including its compilation, deployment, and intended functionality.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

February 2025

For

hivello

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillhash.com