QuillAudits

Audit Report
October, 2024

For

hivello

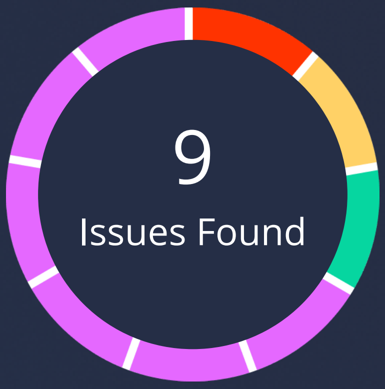
Table of Content

Table of Content

Executive Summary

Project Name	Hivello
Overview	The token-contract program facilitates SPL token operations such as creation, transfer, burning, and metadata management by interacting with the SPL token program and token metadata standards on Solana. The staking-contract provides staking functionality, allowing users to stake tokens into different tiers with various token requirements and commitment times, using price feed data for token valuation. Together, these contracts offer a complete solution for token management and staking with built-in validation and tiered staking systems.
Timeline	2nd October 2024 - 8th October 2024
Updated Code Received	11th October 2024
Second Review	11th October 2024 - 14th October 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Hivello Programs for quality, security, and correctness.
Contracts In-Scope	Token and Token Locking
Source Code	https://github.com/danii-authic/staking-contract/blob/main/programs/locking https://github.com/danii-authic/token-contract/blob/main/programs/spl_token
Branch	Main
Fixed In	**token contract** - 8d667710f8f44491ebb1d56430a235f7d927708d **staking contract** - c21b9a442c1018a1e1334d555ce1dbea67ffc04b

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	1	5

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 smart contracts, 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.
- 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 analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

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 analyzed, 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 behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of 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

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

These are the issues identified in the initial audit and have been successfully fixed.

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

1. Potential for Admin to Halt Operations by Closing Vault

Path

programs/locking/src/state.rs

Function

close_vault

Description

In the close_vault function, the admin has the ability to close the vault, which could prematurely halt all staking operations. This creates a significant risk, as the vault is central to managing the staking tiers, total staked tokens, and staker profiles. If the vault is closed while there are still active stakers or unclaimed rewards, it could result in unintended consequences, such as the loss of user funds, halted staking processes, and the inability to unstake or claim rewards.

```
pub fn close_vault(_ctx: Context<CloseVault>) -> Result<()> {
    let payer = _ctx.accounts.identity.key();
    require!(payer != ADMIN_PUBKEY, state::ErrorCode::AdminAccountOnly);
    Ok(())
}
```

Recommendation

Implement additional validation checks before allowing the vault to be closed. For example, ensure that there are no active staker profiles and that all staked tokens have been unstaked or claimed before the vault can be closed.

Status

Resolved

Medium Severity Issues

2. Incorrect Decimal Handling in Staking

Path

programs/locking/src/lib.rs

Function

stake

Description

In the stake function of the locking contract, the handling of token decimals is currently hardcoded to assume that the tokens being staked have 8 decimal places (token_decimals = 8). This approach introduces a significant flaw, as not all tokens on the Solana blockchain adhere to the same decimal structure. Different tokens can have varying numbers of decimal places, which represent the precision of the smallest divisible unit of the token. For instance, some tokens might have 6 decimals, while others could have 9 or 0. By hardcoding the decimal value, the contract assumes a fixed precision for all tokens, which can lead to miscalculations in staking amounts, rewards, and other related operations

Recommendation

Instead of hardcoding the decimal value to 8, dynamically retrieve the token's actual decimal precision directly from the token mint account when a token is staked. The token mint account holds essential metadata about the token, including the number of decimal places.

Status

Resolved

Low Severity Issues

3. Lack of Token Validation in Transfer Function

Path

programs/spl_token/src/lib.rs

Function

transfer_token

Description

In the transfer_token function, the current implementation does not perform adequate validation to ensure that the correct token type is being transferred. The function is designed to facilitate token transfers between accounts, but without validating that the token involved in the transfer matches the expected token type, there is a risk of unintended consequences. Specifically, any token account could be passed into the transfer function, and tokens other than the intended one could be transferred. The function logs a message indicating the token transfer, but since the token type is not validated, the log message could be misleading, showing that a transfer has occurred without specifying the correct token type. This lack of validation can introduce confusion and undermine the integrity of the contract.

Recommendation

Implement a check within the transfer_token function to ensure that the token account being used for the transfer corresponds to the intended token mint. This can be done by comparing the token mint of the from_account and to_account with the expected token mint address. This validation step ensures that only the correct token is involved in the transfer operation, preventing accidental or malicious transfers of unintended tokens.

Status

Resolved

Informational Issues

4. Magic Numbers in Token Creation

Description

In the create_token function, there are several instances where numeric values, such as space = 82, rent_fee = 1461600, and decimals = 8, are hardcoded directly into the function. These values, often referred to as “magic numbers,” are arbitrary constants that provide no context or explanation for their purpose. Hardcoding such values can significantly reduce the readability, flexibility, and maintainability of the code. Future developers or auditors may not easily understand the significance or calculation behind these numbers, making it harder to update or debug the system. For instance, if the rent or space requirements change in the underlying Solana protocol, these hardcoded values would need to be manually adjusted in every occurrence. Additionally, the fixed decimal value might not work for tokens with different decimal precision, creating inconsistency when minting or transferring tokens.

Recommendation

Replace Magic Numbers with Constants: Instead of hardcoding the values, define them as constants at the top of the file with descriptive names. For example, const TOKEN_SPACE: u64 = 82; or const RENT_FEE: u64 = 1461600;. This makes it clear what each value represents and makes the code easier to modify in the future.

Status

Resolved

5. Redundant Middleware Program

Description

The current token-contract is a middleware program, which wraps around the SPL Token program on Solana, provides functionality such as token creation, transfer, burning, and setting metadata. However, many of the functions implemented in this middleware are redundant, as they duplicate the capabilities that are already available directly from the SPL Token program. For example, the SPL Token program natively supports minting, transferring, and burning tokens, as well as managing token authorities. By creating a middleware that simply calls these SPL Token functions, this program adds an unnecessary layer of abstraction and complexity, without delivering any additional features or value. This not only increases the maintenance burden but also introduces additional points of potential failure, making the overall system more difficult to debug and manage. Furthermore, the extra layer of code could increase gas fees or execution time, since every call to the middleware is essentially a proxy to an already-existing SPL function. In essence, this program acts as a middleman that replicates standard Solana functionality, making it harder to justify its existence from a technical and operational standpoint.

Recommendation

The first step is to critically assess whether this middleware program adds any unique value or functionality that cannot be achieved by interacting directly with the SPL Token program. If the only purpose of the middleware is to wrap existing SPL Token functions, it may be more efficient to remove the middleware altogether and directly interface with the SPL Token program. This would simplify the architecture, reduce the risk of bugs, and lower operational overhead.

Status

Acknowledged

6. Missing Event Emissions for Critical Management Actions (e.g., Initialize, Update Tiers)

Path

programs/locking/src/lib.rs

Function

Initialize, update_tiers

Description

In the current implementation of the contract, there are no event emissions for critical management actions such as initialize (vault initialization) and update_tiers (updating the staking tiers). These actions represent important state changes in the contract, particularly for system administrators and auditors who need to track how the protocol evolves over time. The absence of events for these key management functions makes it difficult to monitor changes or create a historical log of important contract activities.

Recommendation

Implement an event emission in the initialize and update_tiers functions.

Status

Resolved

7. Tier Not Cleared After Unstaking All Tokens

Path

programs/locking/src/lib.rs

Function

unstake

Description

In the current implementation of the staking system, when a user unstakes all their tokens, the tier information in the staker profile is not cleared. This results in a situation where the user's staker profile still reflects their highest tier, even though they no longer have any tokens staked. This inconsistency can lead to several problems: it may falsely indicate that the user still qualifies for the highest tier benefits, potentially granting them access to features or rewards they are no longer eligible for. Moreover, since the tier information is not updated or cleared after the tokens are fully unstaked, the system does not accurately reflect the user's staking status. This creates confusion in the user interface, reporting tools, and any systems that rely on the staker profile data.

Recommendation

When a user unstakes all their tokens, the tier information in their staker profile should be reset or cleared.

Status

Resolved

8. Hardcoded Admin Key Prevents Key Rotation

Path

programs/locking/src/lib.rs

Function

ADMIN_PUBKEY

Description

In the current implementation of the contract, the administrator's public key (ADMIN_PUBKEY) is hardcoded directly into the contract. This creates a significant limitation in the event that the admin key is lost, compromised, or needs to be rotated for any reason. Because the admin key is statically embedded in the contract, there is no mechanism to update or change it without redeploying the entire contract. This can result in severe operational risks, as losing access to the admin key would leave the contract without a way to perform critical management actions such as updating tiers, modifying parameters, or closing the vault.

Recommendation

Instead of hardcoding the admin key, store the admin's public key as a mutable value within the contract's state. This allows the admin key to be updated or rotated dynamically without requiring a full contract redeployment.

Status

Resolved

9. Missing Tests for Critical Functions and Scenarios

Description

The current implementation of the contract lacks comprehensive unit tests and integration tests for several critical functions, particularly those related to core staking mechanisms, management actions, and edge cases.

Recommendation

Develop comprehensive unit tests that cover all key functions in the contract.

Status

Resolved

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 the Hivello codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the End, Hivello Team Resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Hivello smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Hivello smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Hivello to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2024

For

hivello

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com