QuillAudits

AUDIT REPORT

January 2025

For

SWARM

Table of Content

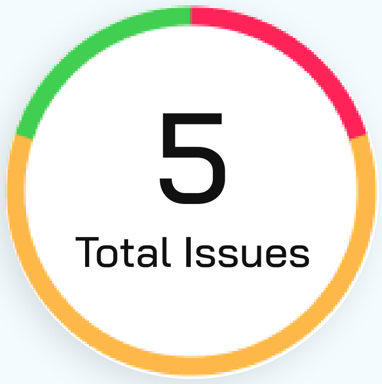
Executive Summary

Project Name	Swarm
Project URL	https://swarmnetwork.ai/
Overview	The **Swarm License Sale** system is built on the SUI blockchain and facilitates the sale of digital licenses while incentivizing referrals. It incorporates advanced features such as tiered pricing, referral incentives, and administrative controls, offering flexibility and scalability. A notable feature of the Swarm License Sale contracts is the **Referral Program**, which incentivizes users to promote the platform. Referrers earn cashback rewards when their referral codes are used for purchases. The cashback rate is dynamic, increasing as the referrer successfully brings more participants into the system. This multi-tiered referral system encourages user engagement and helps in organic community expansion.
Audit Scope	The scope of this Audit was to analyze the **Swarm Smart Contracts** for quality, security, and correctness. https://github.com/Swarm-Network/swarm-license-sale-SUI
Contracts in Scope	Swarm_license_sale Token
Commit Hash	8bf1089
Language	Move

Executive Summary

Blockchain	Sui
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	16th January 2025 - 23rd January 2025
Updated Code Received	23rd January 2025
Review 2	24th January 2025
Fixed In	f0ba3e2

Number of Issues per Severity



🟥	High	1 (20%)
🟧	Medium	3 (60%)
🟩	Low	1 (20%)
🟪	Informational	0 (0%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	1	2	1	0
Acknowledged	0	1	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Transaction-ordering Dependence
- ✓ Access Control
- ✓ Gas Usage
- ✓ Unchecked CALL Return Values
- ✓ Witness Type
- ✓ Denial of Service/Logical Oversights
- ✓ Business Logic Contradicting the Specification
- ✓ Timestamp Dependence
- ✓ Number of Rounding Errors
- ✓ Centralization of Power
- ✓ Arbitrary Token Minting
- ✓ The Flow of Capability
- ✓ Implicit Visibility Level
- ✓ Integer Overflow/Underflow by Bit Operations
- ✓ Code Clones, Functionality Duplication

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
- 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 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.

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 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.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Statistic Analysis.

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. Incorrect Total Distribution Update in Airdrop and Payment Functions

Resolved

Path

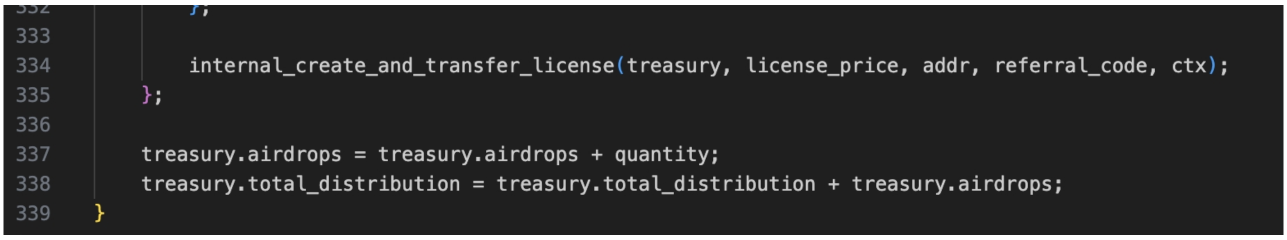
swarm_license_sale.move

Function Name

airdrop_license, internal_process_payment

Description

Both functions incorrectly update the **total_distribution** field. Instead of adding the specific **quantity** to the total distribution, the code adds the entire accumulated airdrops or sales values. This can lead to Inaccurate tracking of total token distribution which leads to price increase.



```
        };

        internal_create_and_transfer_license(treasury, license_price, addr, referral_code, ctx);
    };

    treasury.airdrops = treasury.airdrops + quantity;
    treasury.total_distribution = treasury.total_distribution + treasury.airdrops;
}
```

Recommendation

Modify both functions to directly add the **quantity** to **total_distribution**.

```
   treasury.airdrops = treasury.airdrops + quantity;
   treasury.total_distribution = treasury.total_distribution + quantity;
```

Medium Severity Issues

2. Admin cannot add a new Referral code to existing user in assign_custom_referral_code

Resolved

Path
swarm_license_sale.move

Function Name
assign_custom_referral_code

Description
The current implementation prevents re-assigning or updating a referral code for an existing user. The **internal_create_referrer** function uses **table::add()**, which fails if the key (wallet address) already exists in the table.

```
table::add(&mut treasury.wallet_addres_to_referrer_id, addr, referrer_id);
```

Recommendation
Modify the code to remove the existing entry before adding a new one:

```
if (table::contains(&treasury.wallet_addres_to_referrer_id, addr)) {
    table::remove(&mut treasury.wallet_addres_to_referrer_id, addr);
}
table::add(&mut treasury.wallet_addres_to_referrer_id, addr, referrer_id);
```

3. Price Reset issue in Zero-Quantity License Purchase

Resolved

Path

swarm_license_sale.move

Function Name

internal_process_payment

Description

```
while (i < quantity) {
    i = i + 1;

    let current_price = internal_calculate_price(prev_treasury_total_distribution + i);
    let next_price = internal_calculate_price(prev_treasury_total_distribution + i + 1);

    if (i == quantity) {
        final_current_price = current_price;
        final_next_price = next_price;
    };
}

treasury.price.current = final_current_price;
treasury.price.next = final_next_price;
//....
```

When quantity is 0, the function's loop does not execute, causing uninitialized price variables to default to 0. The final_current_price and final_next_price are set based on the last iteration, which never occurs with zero quantity. This results in resetting treasury prices to zero.

Recommendation

Add an explicit quantity validation.

4. Admin Can Increase License Prices Through Massive Airdrops

Acknowledged

Path

swarm_license_sale.move

Function Name

airdrop_license

Description

The current implementation allows admins to artificially inflate license prices by conducting massive airdrop. With no quantity restrictions, mass airdrops can push the total distribution beyond price threshold phases, dramatically increasing the license price for future purchases.

```
public entry fun airdrop_license(..., quantity: u64, ...) {
    // No quantity validation
    loop {
        // Create licenses without restrictions
        internal_create_and_transfer_license(...);
    }
}

fun internal_calculate_price(total_distribution: u64): u64 {
    // Prices increase based on total distribution
    if (total_distribution <= PHASE5_END) {
        return PHASE5_BASE + (increments * PHASE5_INCREMENT)
    };
}
```

Recommendation

Implement a maximum airdrop quantity limit.

Status

Acknowledged — Added 50 cap per tx.

Low Severity Issues

5. Insufficient Validation in revoke_admin_power() and approve_admin_power() Functions

Resolved

Path
swarm_license_sale.move

Function Name
revoke_admin_power(), approve_admin_power()

Description

```
public entry fun revoke_admin_power(...) {
    vec_set::remove(&mut admin_whitelist.wallets, &wallet);
}

public entry fun approve_admin_power(...) {
    vec_set::insert(&mut admin_whitelist.wallets, wallet);
}
```

Both functions lack proper validation before performing vec_set operations, which can lead to potential runtime failures.

Recommendation
Add existence checks before operations:

```
public entry fun revoke_admin_power(...) {
    if (vec_set::contains(&admin_whitelist.wallets, &wallet)) {
        vec_set::remove(&mut admin_whitelist.wallets, &wallet);
    }
}

public entry fun approve_admin_power(...) {
    if (!vec_set::contains(&admin_whitelist.wallets, &wallet)) {
        vec_set::insert(&mut admin_whitelist.wallets, wallet);
    }
}
```

Functional Tests

Some of the tests performed are mentioned below:

- ✓ swarm_license_sale::swarm_license_sale::test_airdrop_license
- ✓ swarm_license_sale::swarm_license_sale::test_approve_admin_power
- ✓ swarm_license_sale::swarm_license_sale::test_internal_calculate_price
- ✓ swarm_license_sale::swarm_license_sale::test_internal_calculate_referral_cashback_rate
- ✓ swarm_license_sale::swarm_license_sale::test_internal_generate_referral_code
- ✓ swarm_license_sale::swarm_license_sale::test_license_sale_state_init
- ✓ swarm_license_sale::swarm_license_sale::test_purchase_license
- ✓ swarm_license_sale::swarm_license_sale::test_purchase_multiple_licenses
- ✓ swarm_license_sale::swarm_license_sale::test_revoke_admin_power

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 Swarm. We performed our audit according to the procedure described above.

Some issues of High, low and medium severity were found.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Swarm. 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 Swarm. 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 Swarm 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.

QuillAudits

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

Follow Our Journey

AUDIT REPORT

January 2025

For

SWARM

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillhash.com