SALUS SECURITY

JAN 2025

CODE SECURITY ASSESSMENT

NETWORK3

Overview

Project Summary

- Name: Network3 - Stake
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Network-3/N3C-N3-50-50-1_90
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Network3 - Stake
Version	v2
Type	Solidity
Dates	Jan 26 2025
Logs	Jan 26 2025; Jan 26 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	2
Total	6

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Incorrect logic for checking totalStakedPerOption	Medium	Business logic	Resolved
2	Centralization risk	Low	Centralization	Acknowledged
3	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky External Calls	Acknowledged
4	Missing events for functions that change critical state	Low	Business logic	Acknowledged
5	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
6	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Incorrect logic for checking totalStakedPerOption

Severity: Medium	Category: Business Logic
Target: - src/stake.sol	

Description

When users stake, a portion of `$n3c` is directly converted to `$n3` based on the `swapRate` and sent to the user, while the remaining `$n3c` is recorded as part of the user's stake amount.

At the same time, the contract keeps track of `totalStakedPerOption` for each `optionIndex`. When users stake, this value is updated by adding the remaining `$n3c` from the previous step.

However, during validation, instead of using the remaining amount, the function incorrectly uses the total amount passed as a parameter.

src/stake.sol:L38 - L64

```
function stake(uint256 amount, uint256 optionIndex) external nonReentrant {
    ...
    // Check total staked amount for this option
    uint256 totalStaked = getTotalStaked(optionIndex);
    require(totalStaked + amount <= option.maxAmount, "Exceeds max amount for this
option");

    ...

    totalStakedPerOption[optionIndex] += amount - (amount * option.rate / 100);
    emit Staked(msg.sender, amount - (amount * option.rate / 100), optionIndex);
}
```

Recommendation

Consider modifying the validation condition or adjusting the accumulated amount added to `totalStakedPerOption` to ensure it correctly accounts for only the remaining `$n3c` quantity after conversion.

Status

The team has resolved this issue in the commit 9defb8a.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - src/stake.sol	

Description

There is an `owner` privileged account in the `Stake` contract, and the `owner` can change the value of configuration by calling the functions in the contract at will, such as `setN3Token()` and `pushStakeOption()`. If `owner`'s private key is compromised, an attacker can change the configuration. Even modify the owner's address to extract all assets from the contract through the `withdrawTokens()` function.

This could be worrisome if the privileged account is a regular EOA account.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

3. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Risky External Calls
Target: - src/stake.sol	

Description

Tokens not compliant with the `ERC20` specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the `SafeERC20` library implementation from `OpenZeppelin` and call `safeTransfer` or `safeTransferFrom` when transferring `ERC20` tokens.

Status

This issue has been acknowledged by the team.

4. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - src/stake.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `Stake` contract, events are lacking in the privileged setter functions (e.g. `withdrawTokens()`, `setN3Token()`, `setN3CToken()` and `pushStakeOption()`).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - src/stake.sol	

Description

The `Stake` contract inherits from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

6. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - src/stake.sol	

Description

```
pragma solidity ^0.8.0;
```

The `Stake` contract uses a floating compiler version `^0.8.0`.

Using a floating pragma `^0.8.0` statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit c13af56:

File	SHA-1 hash
src/stake.sol	ed0e5da30c18022b3e0eed071821ac7a6ef3d1bb