SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

NETWORK3

Overview

Project Summary

- Name: Network3 - miner pos
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Network-3/miner-pos/
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Network3 - miner pos
Version	v2
Type	Solidity
Dates	Mar 19 2025
Logs	Mar 19 2025; Mar 19 2025

Vulnerability Summary

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

Contact

E-mail: support@salusec.io

SALUS

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.

SALUS

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	Rewards can still be claimed after the maturity of a fixed deposit	Medium	Business Logic	Acknowledged
2	Centralization risk	Medium	Centralization	Acknowledged
3	Incorrectly recording time can result in users losing rewards	Low	Business Logic	Acknowledged
4	Users may be unable to withdraw their interest or even the principal amount normally	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. Rewards can still be claimed after the maturity of a fixed deposit	
Severity: Medium	Category: Business Logic
Target: - src/stake.sol	

Description

The `calculateReward()` function does not verify whether the user's fixed deposit has matured. Upon the maturity of a fixed deposit, it should automatically be converted into a current deposit. However, under the current function logic, the user's rewards continue to be calculated based on the fixed deposit interest rate even after the deposit has matured.

In this scenario, users are able to claim rewards that are higher than expected.

Recommendation

Consider using the flexible deposit interest rate to calculate interest or cease interest accrual after the fixed deposit has matured.

Status

This issue has been acknowledged by the team.

2. Users can engage in arbitrage through borrowing	
Severity: High	Category: Business Logic
Target: - src/stake.sol	

Description

Due to the fact that the interest rate for `$n3` tokens is significantly higher than the borrowing rates in the lending pool, if `$n3` tokens possess liquidity, users could borrow a substantial amount of other tokens (such as `$USDT`, `$ETH`, etc.), exchange them for `$n3` tokens, and then stake these into a contract to earn interest. Upon maturity, they could withdraw the funds, using a portion to repay the loan and its interest, while the remainder would constitute illicit profits gained from arbitrage.

Recommendation

Consider implementing a deposit cap for users or adopting other measures to prevent arbitrage opportunities.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - src/stake.sol	

Description

The `emergencyWithdraw()` function permits the `owner` to withdraw all `$n3` tokens from the contract, including the principal deposited by users.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

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. Incorrectly recording time can result in users losing rewards	
Severity: Low	Category: Business logic
Target: - src/stake.sol	

Description

The `calculateReward` function calculates the user's reward according to the number of whole days, but the time recorded in the `claimReward` function is not recorded according to the number of whole days in the past, which causes the user to lose a part of the reward generated by the time.

Recommendation

It is recommended that the `claimReward` function also record the user's withdrawal time by full days.

Status

This issue has been acknowledged by the team.

4. Users may be unable to withdraw their interest or even the principal amount normally	
Severity: Low	Category: Business logic
Target: - src/stake.sol	

Description

According to the test code, the amount of rewards in the contract is a fixed value. This could potentially lead to insufficient contract balance when users attempt to `` `unstake()` `` or `` `claimReward()` ``, resulting in users being unable to withdraw their funds normally.

```
function setUp() public {
    ...
    n3Token.transfer(address(stake), 100000 * 10 ** 18); // 为合约提供奖励代币
}
```

Recommendation

Consider dynamically adjusting the contract's reward balance to ensure that users' withdrawals can proceed normally.

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.

SALUS

6. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - src/stake.sol	

Description

```
pragma solidity ^0.8.22;
```

The `Stake` contract uses a floating compiler version `^0.8.22`

Using a floating pragma `^0.8.22`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 cdcd8c0:

File	SHA-1 hash
src/stake.sol	80be5128e71bffa44ae99673de766ed15de75c69