SALUS SECURITY

DEC 2024

CODE SECURITY ASSESSMENT

MINERX

Overview

Project Summary

- Name: Minerx
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Minerx
Version	v3
Type	Solidity
Dates	Dec 18 2024
Logs	Dec 16 2024; Dec 17 2024; Dec 18 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	3
Total informational issues	1
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	Insecure signature usage in the buyMinerNFTs() function leading to potential front-running exploits	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Cross-chain replay attacks are possible with `buyMinerNFTs()`	Low	Business Logic	Acknowledged
4	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky External Calls	Acknowledged
5	Missing zero address checks	Low	Data Validation	Resolved
6	Missing events for functions that change critical state	Informational	Configuration	Resolved

2.2 Notable Findings

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

1. Insecure signature usage in the buyMinerNFTs() function leading to potential front-running exploits

Severity: Medium	Category: Business Logic
Target: - minerx.sol	

Description

The `buyMinerNFTs()` function is vulnerable to front-running attacks due to an insecure signature verification process. Specifically, the verifySignature() function does not bind the buyer's address to the signed message. As a result, when a legitimate user attempts to purchase MinerNFTs, malicious actors can observe the signed message in the transaction pool and front-run the transaction using the same signature. This undermines the security of the signature system.

This vulnerability can lead to two critical issues:

- Malicious users can fraudulently gain the purchasing power to acquire MinerNFTs.
- Legitimate users may lose their ability to complete their intended purchases.

Recommendation

To prevent front-running and ensure the integrity of the system, incorporate buyer-specific binding into the signed messages.

Status

The team has resolved this issue.

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - minerx.sol	

Description

The `minerx` contract has privileged accounts, The `admin` can withdraw all tokens.

If the admin's private key is compromised, an attacker could arbitrarily mint or burn tokens. Moreover, if these privileged accounts are standard externally-owned accounts (EOAs), it greatly increases the risk to 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. Cross-chain replay attacks are possible with `buyMinerNFTs()`

Severity: Low	Category: Business Logic
Target: - minerx.sol	

Description

minerx.sol:L192-L234

```
function buyMinerNFTs(
    uint256 batchId,
    bytes32[] calldata messages,
    bytes[] calldata signatures
) external nonReentrant {
    require(messages.length > 0, "e5");
    require(signatures.length > 0, "e6");
    require(messages.length == signatures.length, "e7");

    // verify message
    for (uint256 i = 0; i < messages.length; i++) {
        require(usedVerificationMessages[messages[i]] == false, "e8");
        require(usedVerificationSignatures[signatures[i]] == false, "e9");
        require(verifySignature(messages[i], signatures[i], messageSigner), "e10");
        usedVerificationMessages[messages[i]] = true;
        usedVerificationSignatures[signatures[i]] = true;
    }
    ...
}
```

The `buyMinerNFTs()` function is used to validate the signature of the buyer. However, the critical chainId information that prevents cross-chain replay is missing from the signature field data.

This could lead to a cross-chain replay vulnerability in the `minerx` contract when deployed on multiple chains with the same signer address. An attacker could exploit a single signature across multiple chains, resulting in the signature verification process being compromised.

Recommendation

It is recommended that signatures follow the EIP712 standard and include the chainId as a signature domain to prevent cross-chain replay attacks.

Status

This issue has been acknowledged by the team.

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

Severity: Low	Category: Risky External Calls
Target: - minerx.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.

5. Missing zero address checks

Severity: Low	Category: Data Validation
Target: - minerx.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or configuration. However, this check is missing for the signer address variables (e.g., messageSigner and redeemMessageSigner). If the signer address is set to the zero address, it can lead to signature misuse because an invalid signature recovered by the recover function would also return address(0).

Recommendation

Consider adding zero address checks for address variables.

Status

The team has resolved this issue.

2.3 Informational Findings

6. Missing events for functions that change critical state

Severity: Informational	Category: Logging
Target: - minerx.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 `minerx.sol`, events are lacking in the privileged setter functions (e.g. createBatch, createPowerBatch()).

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit :

File	SHA-1 hash
minerx.sol	91886f091a0bfc13348e590ae658c4840cc4bcbe

And we audit the following fixed version:

File	SHA-1 hash
minerx.sol	6ac005382467c60b640f4bac2df6ab9a2f1068a8