SALUS SECURITY

FEB 2023

CODE SECURITY ASSESSMENT

BINARYX

Overview

Project Summary

- Name: BinaryX
- Version: commit 89a0bd9
- Platform: BSC
- Address: 0x5b1f874d0b0C5ee17a495CbB70AB8bf64107A3BD
- Language: Solidity
- Repository: https://github.com/vic-dev-ops/bnx-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	BinaryX
Version	v1
Type	Solidity
Dates	Feb 15 2023
Logs	Feb 15 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	4
Total	8

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	Centralization risk	Medium	Centralization	Resolved
2	Missing burner address field in AncestorBurnt() event	Low	Auditing and Logging	Acknowledged
3	User should be able to call getMinter()	Low	Access Control	Acknowledged
4	Unchecked ERC20 transfer	Low	Business Logic	Acknowledged
5	Use of floating compiler version	Informational	Configuration	Acknowledged
6	Mismatch between contract name and filename	Informational	Code Quality	Acknowledged
7	Can use immutable to save gas	Informational	Code Quality	Acknowledged
8	Code with no effect	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Centralization risk	
Severity: Medium	Category: Centralization
Target: - BinaryxToken.sol	

Description

In the BinaryxToken contract, there is a privileged **owner** role.

The owner of the BinaryxToken contract:

- can add or remove minters. The minters can mint *_newSupply* amount of tokens in total.
- can add addresses to or remove them from the blocklist. If an address is blocked, then transfers to and from that address are forbidden.

A malicious or compromised owner can lock a user's BNX token by adding the user's address to the blocklist. If the privileged owner account is a plain EOA account, this can be worrisome and pose a risk to the users.

Recommendation

We recommend transferring the privileged owner role to a community-governed DAO, or at least to a multisig account. In addition, a timelock-based mechanism can be implemented to demonstrate the project owners' commitment to the ongoing health of this project.

Status

This issue has been addressed by the team. The owner role of the BinaryxToken contract has been transferred to a multisig account.

2. Missing burner address field in AncestorBurnt() event

Severity: Low	Category: Auditing and Logging
Target: - BinaryxToken.sol	

Description

BinaryxToken.sol:L253

```
event AncestorBurnt(uint256 amount);
```

BinaryxToken.sol:L308-L313

```
function burnAncestor(uint256 amount) external {
    address account = _msgSender();
    IERC20(_ancestor).transferFrom(account, address(0xdEaD), amount);
    _mint(account, amount * 100);
    emit AncestorBurnt(amount);
}
```

The AncestorBurnt() event is emitted when a user burns the old BNX token to get the new BNX token. However, the AncestorBurnt() event does not record the burner's address, making it difficult to track the users' burning behavior.

Recommendation

Consider adding a burner address field to the AncestorBurnt() event, also consider adding the indexed modifier to this burner address field to make it more accessible to off-chain tools that parse events.

```
event AncestorBurnt(address indexed burner, uint256 amount);
```

Status

This issue has been acknowledged by the team.

3. User should be able to call getMinter()

Severity: Low	Category: Access Control
Target: - BinaryxToken.sol	

Description

BinaryxToken.sol:L297-L300

```
function getMinter(uint256 _index) external view onlyOwner returns (address){
    require(_index <= getMinterLength() - 1, "BSCToken: index out of bounds");
    return EnumerableSet.at(_minters, _index);
}
```

The getMinter() function is modified by the onlyOwner modifier, meaning that only the owner can call getMinter() to query the minter address at the _index position.

However, the users should be able to query the minter's address with getMinter(), just as they can query the owner's address with owner().

Recommendation

Consider removing the onlyOwner modifier from the getMinter() function.

Status

This issue has been acknowledged by the team.

4. Unchecked ERC20 transfer

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

Description

BinaryxToken.sol:L308-L313

```
function burnAncestor(uint256 amount) external {
    address account = _msgSender();
    IERC20(_ancestor).transferFrom(account, address(0xdEaD), amount);
    _mint(account, amount * 100);
    emit AncestorBurnt(amount);
}
```

Boolean return value for transferFrom() is not checked.

Several ERC20 tokens do not revert on transfer failure and return false instead. If *_ancestor* is an ERC20 token which returns false on transfer failure rather than revert, then a malicious user could free mint the BNX token via burnAncestor(): the malicious user first approves zero amount of _ancestor token for the BinaryxToken contract, then the user calls burnAncestor(), the transferFrom() function inside the burnAncestor() returns false, because this return value is not checked the following _mint() logic is executed, resulting in a free mint.

Fortunately, the _ancestor variable is expected to be set to the address of the old BNX token. Upon further investigation, we have found that the transferFrom() in the old BNX token contract reverts in case of failure. Therefore, when the old BNX token is the _ancestor, the unchecked transferFrom return value in the burnAncestor() function is unlikely to cause the aforementioned issue.

Nevertheless, we still recommend that the developers not rely on the assumption of the _ancestor, and instead add a check on the return value of transferFrom().

Recommendation

Consider ensuring that the transferFrom return value is checked.

```
bool success = IERC20(_ancestor).transferFrom(account, address(0xdEaD), amount);
require(success);
```

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Use of floating compiler version

Severity: Informational	Category: Configuration
Target: - BinaryxToken.sol	

Description

BinaryxToken.sol:L3

```
pragma solidity ^0.8.0;
```

The BinaryxToken contract uses a floating compiler version ^0.8.0.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

Consider locking the pragma version.

Status

This issue has been acknowledged by the team.

6. Mismatch between contract name and filename

Severity: Informational	Category: Code Quality
Target: - BinaryxToken.sol	

Description

BinaryxToken.sol:L242

```
contract WindToken is DelegateERC20, Ownable2Step {
```

The entry contract name is WindToken while the filename is BinaryxToken. It is best practice to use the same name for the entry contract and the filename.

Recommendation

Consider changing the entry contract name from WindToken to BinaryxToken.

Status

This issue has been acknowledged by the team.

7. Can use immutable to save gas

Severity: Informational	Category: Code Quality
Target: - BinaryxToken.sol	

Description

The BinaryxToken contract when deployed has a few fixed storage variables.
BinaryxToken.sol:L255-L263

```
constructor(
    address ancestor,
    uint256 ancestorMaxSupply,
    uint256 newSupply
) ERC20("BinaryX", "BNX") {
    _ancestor = ancestor;
    _maxSupply = (ancestorMaxSupply * 100 + newSupply) * 1e18;
    _newSupply = newSupply * 1e18;
}
```

The _maxSupply and _ancestor storage variables are defined in the contract.
BinaryxToken.sol:L245

```
uint256 public _maxSupply;
```

BinaryxToken.sol:L248

```
address public _ancestor;
```

But they are never changed.

Recommendation

Consider setting _maxSupply and _ancestor storage variables as **immutable** for a considerable gas improvement.

```
uint256 public immutable _maxSupply;
address public immutable _ancestor;
```

Status

This issue has been acknowledged by the team.

8. Code with no effect

Severity: Informational	Category: Redundancy
Target: - BinaryxToken.sol	

Description

BinaryxToken.sol:L9

```
pragma experimental ABIEncoderV2;
```

ABI coder v2 is activated by default since Solidity v0.8.0. The pragma `pragma experimental ABIEncoderV2;` is deprecated and has no effect.

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in commit 89a0bd9:

File	SHA-1 hash
BinaryxToken.sol	7874319365cba58999eb003fcd5f51659ed0893c