SALUS SECURITY

FEB 2025

CODE SECURITY ASSESSMENT

FOUR

Overview

Project Summary

- Name: Four - $FORM
- Platform: EVM-compatible chains
- Address: 0x5b73A93b4E5e4f1FD27D8b3F8C97D69908b5E284
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Four - $FORM
Version	v3
Type	Solidity
Dates	Feb 24 2025
Logs	Feb 20 2025, Feb 21 2025; Feb 24 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	2
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	BNX holders cannot burn their BNX	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Revisit the max supply amount	Low	Business Logic	Acknowledged
4	BNX holders may fail to burn their BNX	Low	Business Logic	Acknowledged
5	Suggest adding one sanity check in blacklist	Informational	Business Logic	Acknowledged
6	The delegate system lacks transitivity.	Informational	Business Logic	Acknowledged
7	Users must specify themselves as delegatee in order to receive votes.	Informational	Business Logic	Acknowledged
8	Magic numbers	Informational	Code Quality	Acknowledged

2.2 Notable Findings

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

1. BNX holders cannot burn their BNX

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

Description

In Four Token contract, the miner role can mint the $FORM directly via interface `mint`. And the normal BNX holders can burn their holding BNX to mint $FORM.

The problem here is that when BNX holders want to burn their BNX, the function `mint` will be triggered. And there is one modifier `onlyMinter` for the function `mint`. So normal users cannot burn their BNX.

FourToken.sol: L35-L45

```
function mint(address to, uint256 amount) public onlyMinter {
     _mint(to, amount);
    require(totalSupply() <= maxSupply(), "FOUR: total supply exceeds max supply");
}
function burnBNX(uint256 amount) public {
    address account = _msgSender();
    SafeERC20.safeTransferFrom(IERC20(BNX), account, address(0xdEaD), amount);
    emit BNXBurnt(account, amount);
    mint(account, amount);
}
```

Recommendation

User `_mint` in `burnBNX` function and do the related supply check.

Status

This issue has been resolved by the team.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - FourToken.sol	

Description

Four token contracts have one privileged account: owner. The owner role can add new minter and mint tokens.

If the private keys of these accounts are compromised, the malicious miner can mint any amount, which will block normal users' mint.

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. Revisit the max supply amount

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

Description

The $FORM token aims to encourage BNX holders to burn their BNX to mint the new $FORM. Considering that the ratio between BNX and FourToken is 1:1, we need to make sure that maximum supply in Four token should be larger than the BNX's current total supply.

Current total supply in BNX is 569541313 * 1e18. This supply is less than 580000000 * 1e18.

The problem is that the ancestor BNX token can be burned and mint some new BNX, then the BNX's total supply will increase. In current ancestor BNX(0x8C851d1a123Ff703BD1f9dabe631b69902Df5f97), the total supply is 6470113412384225165777359. The balance in address `0xdEaD` is 5779459304386803715213753. So there are some inactive ancestor BNX holders. If their inactive ancestor BNX tokens are burned to mint the BNX. Then the BNX supply will exceed the 580000000 * 1e18. This will cause some normal holders to not burn their BNX.

FourToken.sol: L28-L33

```
function maxSupply() public pure returns (uint256) {
    return 580000000 * 1e18;
}
```

WindToken.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);
}
```

Recommendation

Revisit the max supply amount.

Status

This issue has been acknowledged by the team.

4. BNX holders may fail to burn their BNX

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

Description

In $FORM, the miner can mint some $FORM via interface `mint`. The only limitation is that the current total supply can not exceed maxSupply.

The problem is that if the miner mint more Four tokens, e.g. (maxSupply - BNX's total supply + delta), normal BNX holders may fail to burn their BNX.

FourToken.sol: L28-L33

```
function mint(address to, uint256 amount) public onlyMinter {
    _mint(to, amount);
    require(totalSupply() <= maxSupply(), "FOUR: total supply exceeds max supply");
}
```

Recommendation

Add some limitations for the possible mint amount in mint function.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Suggest adding one sanity check in blacklist

Severity: Informational	Category: Business Logic
Target: - FourToken.sol	

Description

In FourToken.sol, we have one blacklist feature. The owner can set one address to the blacklist. If the address(0) is set to the blacklist, normal users cannot burn their BNX.

FourToken.sol: L28-L33

```
function blacklist(address account) external onlyOwner {
    _blacklist[account] = true;
    emit BlacklistUpdated(account, true);
}
```

Recommendation

Suggest adding one sanity check in the blacklist.

Status

This issue has been acknowledged by the team.

6. The delegate system lacks transitivity.

Severity: Informational	Category: Business Logic
Target: - DelegateERC20.sol	

Description

The protocol allows users to delegate their votes to other addresses, but the delegation lacks transitivity. For example, if user1 delegates their votes to user2, and user2 delegates their votes to user3, user2 cannot delegate the votes from user1 to user3. This may cause inconvenience for users.
DelegateERC20.sol: L164-L174

```
function _delegate(address delegator, address delegatee)
internal
{
    address currentDelegate = _delegates[delegator];
    uint256 delegatorBalance = balanceOf(delegator); // balance of underlying balances
(not scaled);
    _delegates[delegator] = delegatee;

    _moveDelegates(currentDelegate, delegatee, delegatorBalance);

    emit DelegateChanged(delegator, currentDelegate, delegatee);
}
```

Recommendation

It is suggested that when a delegate has already specified their delegation, the votes should be transferred to the delegate of the delegate.

Status

This issue has been acknowledged by the team.

7. Users must specify themselves as delegatee in order to receive votes.

Severity: Informational	Category: Business Logic
Target: - DelegateERC20.sol	

Description

When users want to receive votes, they must explicitly specify themselves as the `delegatee`; otherwise, the votes will not be recorded due to the `delegatee` being the zero address. This may cause inconvenience for users.
DelegateERC20.sol: L33-L44

```
function _mint(address account, uint256 amount) internal override virtual {
    super._mint(account, amount);

    // add delegates to the minter
    _moveDelegates(address(0), _delegates[account], amount);
}


function _transfer(address sender, address recipient, uint256 amount) internal override
virtual {
    super._transfer(sender, recipient, amount);
    _moveDelegates(_delegates[sender], _delegates[recipient], amount);
}
```

Recommendation

It is suggested that when the delegatee is not specified, the votes should be automatically added to the user's own account.

Status

This issue has been acknowledged by the team.

8. Magic numbers

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

Description

The contract contains magic numbers, which is detrimental to reading and understanding the code.
FourToken.sol: L24-L26

```
function maxSupply() public pure returns (uint256) {
    return 580000000 * 1e18;
}
```

Recommendation

It is recommended to replace magic numbers with constants that have meaningful variable names.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
DelegateERC20.sol	e84d713f7c85b286273fadcc77b2e0f8b780e26f
FourToken-0223.sol	54a039faaeb3c90663c73a732cc6a08e15b91378