SALUS SECURITY

JUN 2025

CODE SECURITY ASSESSMENT

TAGGER

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Tagger
Version	v2
Type	Solidity
Dates	Jun 16 2025
Logs	Jun 11 2025; Jun 16 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	5
Total informational issues	1
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	Staking tokens may change at the wrong time	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Unable to reduce user points	Low	Business Logic	Resolved
4	Missing fee on transfer token support	Low	Business Logic	Acknowledged
5	Boundary errors in numerical checks	Low	Data Validation	Resolved
6	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Business Logic	Resolved
7	Missing events for functions that change critical state	Low	Logging	Acknowledged
8	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved

2.2 Notable Findings

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

1. Staking tokens may change at the wrong time	
Severity: Medium	Category: Business Logic
Target: - StakingContract.sol	

Description

The `setStakingToken` function is restricted to being called only by the `owner`. It includes a modifier `onlyWhenNoActiveStake`, which checks whether the `msg.sender` currently has any active stake before allowing a staking token update.

However, this logic fails to account for stakes held by other users. As a result, the `stakingToken` can be changed even when non-owner users have active stakes. This leads to a dangerous scenario:

1. A user deposits TokenA before the change
2. The owner changes the stakingToken to TokenB.
3. The user then withdraws and receives TokenB instead of their original TokenA.

This inconsistency between deposit and withdrawal tokens can result in loss of funds or unintended token swaps.

Recommendation

It is recommended to revise the logic of the `onlyWhenNoActiveStake` modifier to ensure that no non-owner users have active stakes before changing staking tokens.

Status

This issue has been resolved by the team at address
0x1ce037db9f9f9cae8f68a5ccb04fa39adf15eeba

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - All	

Description

In these contracts, privileged roles such as `` `owner` ``, `` `admin` ``, and `` `operator` `` are granted significant authority. These roles have the ability to modify critical parameters, including but not limited to: Increasing user points, Changing the stakingToken, Minting NFTs, Arbitrarily signing tasks.

If the private keys associated with any of these privileged accounts are compromised, malicious actors could perform unauthorized operations, resulting in significant losses for both users and the project.

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. Unable to reduce user points	
Severity: Low	Category: Business Logic
Target: - ClaimTagV2.sol	

Description

The `updateUserPoints` function is used to increase a user's balance in the `ClaimTagV2` contract, but it can only increase the balance and cannot decrease it. Although the inability to decrease the balance may be an intended behavior, if points are mistakenly added to the wrong user, there is no function available to reverse this operation.

Recommendation

Consider adding a function that allows decreasing user points.

Status

This issue has been resolved by the team.

4. Missing fee on transfer token support	
Severity: Low	Category: Business Logic
Target: - StakingContract.sol	

Description

Since the `stakingToken` can be changed, it is possible that a fee-on-transfer token may be used as the staking token. However, in the `stake` function, the parameter `amount` is directly treated as the amount received by the contract. While this is generally acceptable for standard tokens, in the case of fee-on-transfer tokens, the actual amount received will be less than the transferred amount. Using the transfer amount directly can result in inaccurate accounting or may cause users to be unable to withdraw correctly.

Recommendation

Consider adjusting the functions to compare the contract's token balance before and after transfers to handle tokens with fees accurately. This ensures correct balance tracking and calculations.

Status

This issue has been acknowledged by the team.

5. Boundary errors in numerical checks	
Severity: Low	Category: Data Validation
Target: - ClaimTagV2.sol - TagTask.sol	

Description

The `claimTokens` function in the `ClaimTagV2` contract, which is used for user withdrawals, has an error in its bounds for token balance checks. The contract bounds the checksum only if the contract balance is greater than the user's `points`, but correctly this should be executed successfully whenever the contract balance is greater than or equal to `points`, not greater, which could lead to some users failing to claim when there are sufficient funds.

ClaimTagV2.sol:L73 - L84

```
function claimTokens() external nonReentrant whenNotPaused {
    ...
    uint256 tokenBalance = IERC20(tagToken).balanceOf(address(this));
    require(tokenBalance != 0 && tokenBalance > points, "Not enough Tag");
    ...
}
```

There is an error in the `uploadUserDailyTaskInfo` function in the `TagTask` contract that checks the signature and its timeliness. The contract checks whether the signature expiration time is greater than or equal to the current time. The correct approach is to only pass the check if the signature expiration time is greater than the current time, and to default to signature expiration if the time is equal.

TagTask.sol:L119 - L122

```
function uploadUserDailyTaskInfo(
...
) external nonReentrant whenNotPaused {
    ...
    require(
        taskSignatureAndExpiry.expiry >= block.timestamp,
        "Upload task signature expired"
    );
    ...
}
```

Recommendation

It is recommended that the check statement in the contract be modified based on the contract design.

Status

This issue has been resolved by the team.

6. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	
Severity: Low	Category: Business Logic
Target: - StakingContract.sol	

Description

There may not be a return value in the transfer function for non-ERC20 compliant tokens to indicate a successful or failed transfer. All token transfers in the `` `StakingContract` `` contract directly determine whether the functions `` `transfer` `` and `` `transferFrom` `` return values are true or not, which can lead to the inability to use tokens such as `` `USDT` ``.

Recommendation

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

Status

This issue has been resolved by the team at address
0x1ce037db9f9f9cae8f68a5ccb04fa39adf15eeba

7. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - StakingContract.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 `StakingContract` contract, events are lacking in the `setOperator()`, `setStakingTimes()`, `setCooldownPeriod()`functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

8. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - StakingContract.sol	

Description

The `StakingContract` 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 resolved by the team at address
0x1ce037db9f9f9cae8f68a5ccb04fa39adf15eeba

Appendix

Appendix 1 - Files in Scope

This audit covered the following files from address

StakingContract:

- proxy:0x2cE7f0afC55C0f2B5EF4e63FeA7495eA6e0910E8
- Implementation:0x3804bfb7809592d062260643349bf0f4c9043549

TagNFT:

- proxy:0x6Fd98C5dee0A542647fEeF910d756D826e402b0D
- Implementation:0xa052760ec06d830b207b5a194a0a09d30738b453

ClaimTag:

- proxy:0x6Bc83135946d784D44ef0aAe603fE913Dc4E9986
- Implementation:0xfc658706a2ffce20621678582b9137d2e613cd85

StakingTag:0x1ab2c20271e03715e55dfb69086534a12f0727d9

File	SHA-1 hash
StakingContract.sol	2bc92c590b9c9feb18ec93d21bb868c8c1aee07d
ClaimTagV2.sol	b148f73df80adbca2122bf1b4f4de74020266a3c
TagNFT.sol	8472985c1d8b35288b47092316bf6db3a35e5f58
TagTask.sol	e6f327833a23e8063fa1706e8b9e0612b8bb1537