SALUS SECURITY

FEB 2025

CODE SECURITY ASSESSMENT

DEVHOOKED

Overview

Project Summary

- Name: DEVHooked - release
- Platform: The BSC Blockchain
- Language: Solidity
- Repository:
  - https://github.com/DEVHooked/release
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	DEVHooked - release
Version	v2
Type	Solidity
Dates	Feb 27 2025
Logs	Feb 26 2025; Feb 27 2025

Vulnerability Summary

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

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	Precision loss	Low	Numerics	Resolved
2	Gas optimization suggestions	Informational	Gas Optimization	Resolved
3	Missing zero address checks	Informational	Data Validation	Resolved
4	Missing events for functions that change critical state	Informational	Logging	Resolved

2.2 Notable Findings

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

1. Precision loss	
Severity: Low	Category: Numerics
Target: - contracts/CheckinSocial.sol	

Description

Solidity's integer division truncates the result. Thus, performing division before multiplication can lead to precision loss. Throughout the codebase, one instance that can lead to precision loss were identified:

contracts/CheckinSocial.sol:L142

```
hookToken.safeTransfer(user, share / PRECISION_FACTOR * amountPerShare);
```

Recommendation

Performing multiplication before division is generally better to avoid loss of precision. As such, consider performing multiplication before division.

Status

This issue has been resolved by the team with commit 2e41fc0.

2.3 Informational Findings

2. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - contracts/CheckinSocial.sol	

Description

Finding 1: It is better to remove the redundant codes to ensure good reading and gas efficiency. Because the code assigns the original value to the variable.

contracts/CheckinSocial.sol:L54

```
roundStatus = RoundStatus.NotStarted;
```

contracts/CheckinSocial.sol:L76-78

```
totalRewardShare = 0;
settleIndex = 0;
rewardPerShare = 0;
```

Finding 2: Memory reading saves more gas than storage reading multiple times when the state is not changed. So caching the storage variables in memory and using the memory instead of storage reading is effective.

contracts/CheckinSocial.sol:L173-L182

```
for (uint256 i = settleIndex; i < settleIndex + batchSize && i < totalParticipants; i++)
{
    address user = checkinUsers[i];
    uint256 userCheckinDays = userCheckinInfo[user].checkinDays;
    if (userCheckinDays >= targetCheckinDays) {
        uint256 reward = balanceOf(user) * rewardPerShare / PRECISION_FACTOR ;
        hookToken.safeTransfer(user, reward);
        uint256 balance = balanceOf(user);
        _burn(user, balance);
    }
}
```

Recommendation

Consider using the above suggestions to save gas.

Status

This issue has been resolved by the team with commit 2e41fc0.

3. Missing zero address checks	
Severity: Informational	Category: Data Validation
Target: - contracts/CheckinSocial.sol	

Description

It is considered a security best practice to verify addresses against the zero address. However, this precautionary step is absent in the `` `CheckinSocial` `` contract.

contracts/CheckinSocial.sol:L51-L55

```
constructor(IERC20 _hookToken, address _signer) ERC20("HOOK Learn Challenge", "HLC") {
    hookToken = _hookToken;
    signer = _signer;
    roundStatus = RoundStatus.NotStarted;
}
```

contracts/CheckinSocial.sol:L202-L204

```
function setSigner(address _signer) public onlyOwner {
    signer = _signer;
}
```

Recommendation

Consider adding zero address checks for above address variables.

Status

This issue has been resolved by the team with commit 2e41fc0.

4. Missing events for functions that change critical state	
Severity: Informational	Category: Logging
Target: - contracts/CheckinSocial.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 theCheckinSocial, event is lacking in the `setSigner` function.

contracts/CheckinSocial.sol:L202-L204

```
function setSigner(address _signer) public onlyOwner {
    signer = _signer;
}
```

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been resolved by the team with commit 2e41fc0.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 2d4b80d:

File	SHA-1 hash
contracts/CheckinSocial.sol	d2b2d527005006ae115469b99c71e0634b6d4bc4