SALUS SECURITY

SEP 2023

CODE SECURITY ASSESSMENT

HOOKED PROTOCOL

Overview

Project Summary

- Name: Hooked Protocol - Release Contracts
- Version: commit da66d34
- Platform: BNB Smart Chain
- Language: Solidity
- Repository: https://github.com/DEVHooked/release
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Hooked Protocol - Release Contracts
Version	v2
Type	Solidity
Dates	Sep 27 2023
Logs	May 09 2023; Sep 27 2023

Vulnerability Summary

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

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	Redundant codes	Informational	Redundancy	Resolved
2	Gas optimization recommendations	Informational	Gas inefficiencies	Resolved
3	Floating compiler version	Informational	Configuration	Resolved

2.2 Notable Findings

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

No significant issues are found.

2.3 Informational Findings

1. Redundant codes

Severity: Informational	Category: Redundancy
Target: - contracts/Ecosystem.sol - contracts/PrivateSale.sol	

Description

1. The Context contract is imported and inherited by the Ecosystem and PrivateSale contracts, but its functionalities are not used in these contracts. Therefore, the Context contract is unnecessary and can be taken out of the system.

2. The ERC20Released events in Ecosystem and PrivateSale have two parameters: token and amount. However, the passed-in token value stays the same because the _token variable is immutable once the contract is deployed. Therefore, there is no need to include the token parameter in the ERC20Released event.

3. Inaccurate comments:
release/contracts/Ecosystem.sol:L81

```
/**
 * @dev Calculates the amount of tokens that has already released. Default
implementation is a linear curve.
 */
function _releaseSchedule(uint256 totalAllocation, uint64 timestamp) internal view
virtual returns (uint256) {
```

release/contracts/PrivateSale.sol:L79

```
/**
 * @dev Calculates the amount of tokens that has already released. Default
implementation is a linear curve.
 */
function _releaseSchedule(uint256 totalAllocation, uint64 timestamp) internal view
virtual returns (uint256) {
```

The comments highlighted above are inaccurate, since the Ecosystem and PrivateSale contracts use a non-linear vesting schedule with monthly batch releases. Therefore, they can be removed.

Recommendation

Consider removing the redundant codes.

Status

The team has resolved this issue with commit bd9a6d3.

2. Gas optimization recommendations

Severity: Informational	Category: Gas inefficiencies
Target: - contracts/Ecosystem.sol - contracts/PrivateSale.sol	

Description

1. The following variables could be declared constant instead of immutable, since they are set at compile time:
   - the _start variable in Ecosystem
   - the _duration variable in Ecosystem
   - the _start variable in PrivateSale
   - the _duration variable in PrivateSale

2. The release() functions in both Ecosystem and PrivateSale could be declared external instead of public, as they are not used internally.

3. The _releaseSchedule() functions in both Ecosystem and PrivateSale contain for loops that do not cache the length of the _unlockTimestamps array in memory. It is advisable that if a variable from the contract's storage is going to be read within a function several times, a copy in memory should first be created since reading to the storage directly is expensive. Specifically speaking, instead of using _unlockTimestamps.length in the for loop, it's recommended to cache the array length before the loop, and use the cached value to save gas.

4. The `timestamp > start() + duration()` checks in Ecosystem and PrivateSale can be replaced by `timestamp >= start() + duration()` to save gas. This will return early when timestamp equals exactly start() + duration().

Recommendation

Consider applying the gas-saving recommendations.

Status

The team has resolved this issue with commit bd9a6d3.

3. Floating compiler version

Severity: Informational	Category: Configuration
Target: - contracts/Ecosystem.sol - contracts/PrivateSale.sol	

Description

```
pragma solidity ^0.8.9;
```

The Ecosystem and PrivateSale contracts use a floating Solidity compiler version, ^0.8.9.

However, we discourage this practice. It's best to deploy contracts with the same compiler version and flags that they have been thoroughly tested with. Locking the compiler version helps to prevent contracts from being accidentally deployed using an outdated compiler version, which could introduce bugs that have a negative impact on the system..

Recommendation

It is recommended to use a locked Solidity compiler version.

Status

The team has resolved this issue with commit bd9a6d3.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit da66d34:

File	SHA-1 hash
contracts/Ecosystem.sol	d105efc0065e86b93f1859b89ba9a09f728f3c56
contracts/PrivateSale.sol	bb88ebcfe810858e9695d3cda331dcc519f34bdf