SALUS SECURITY

APR 2024

CODE SECURITY ASSESSMENT

HOOKED PROTOCOL

Overview

Project Summary

- Name: Hooked Protocol - stHookV2
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/DEVHooked/release
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Hooked Protocol - stHookV2
Version	v2
Type	Solidity
Dates	Apr 26 2024
Logs	Apr 24 2024; Apr 26 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
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 **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Incorrect logic for calculating remaining days 6
2. Unbounded loop on array can lead to DoS 8
3. Incorrect balance check 10
4. Centralization risk 11

2.3 Informational Findings 12

5. Gas optimization 12
6. Redundant code 13
7. Missing two-step transfer ownership pattern 14
8. Missing zero address checks 15

Appendix **16**

Appendix 1 - Files in Scope 16

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	Incorrect logic for calculating remaining days	High	Business Logic	Acknowledged
2	Unbounded loop on array can lead to DoS	Medium	Business Logic	Resolved
3	Incorrect balance check	Low	Business Logic	Resolved
4	Centralization risk	Low	Centralization	Mitigated
5	Gas optimization	Informational	Gas Optimization	Resolved
6	Redundant code	Informational	Redundancy	Resolved
7	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
8	Missing zero address checks	Informational	Data Validation	Resolved

2.2 Notable Findings

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

1. Incorrect logic for calculating remaining days

Severity: High	Category: Business Logic
Target: - contracts/stHookV2.sol	

Description

contracts/stHookV2.sol:L95-L116

```
function increaseStakeAmount(uint256 _additionalAmount) public nonReentrant notPaused {
    uint256 currentDay = calculateDay(block.timestamp);
    require(currentDay == settledDay, "stHOOKV2: rewards not settled for today");

    require(_additionalAmount >= 1 ether, "stHOOKV2: cannot add less than 1 HOOK");
    UserInfo storage user = userInfo[msg.sender];
    require(user.lockTime != 0, "stHOOKV2: no active stake");
    require(block.timestamp < user.unlockDate, "stHOOKV2: tokens are already unlocked");

    claimReward();

    user.lockedAmount += _additionalAmount;
    uint256 additionalShares;

    additionalShares = calculateBoosterShare(_additionalAmount, (user.unlockDate -
block.timestamp) / 1 days + 1);
    user.boosterShare += additionalShares;
    totalShares += additionalShares + _additionalAmount;

    hookToken.safeTransferFrom(msg.sender, address(this), _additionalAmount);
    _mint(msg.sender, _additionalAmount);
    emit StakeAmountIncreased(msg.sender, _additionalAmount);
}
```

Even if there is only 1 second left until the unlock date, it will still be calculated as one day under this logic.
In this scenario, users can deposit tokens in the very short period before the lock period ends and earn a day's worth of rewards with less than a full day's time.

Recommendation

Consider correcting the calculation logic for the remaining days.
For example:

```
additionalShares = calculateBoosterShare(_additionalAmount, (user.unlockDate -
block.timestamp) / 1 days);
```

Status

This issue has been acknowledged by the team. It is expected that users can earn rewards on the first day of deposit.

2. Unbounded loop on array can lead to DoS

Severity: Medium	Category: Business Logic
Target: - contracts/stHookV2.sol	

Description

contracts/stHookV2.sol:L118-144

```
function extendLockPeriod(uint256 _additionalDays) public nonReentrant notPaused {
    ...
    uint256 originalLockDays = (user.unlockDate - user.lockTime) / 1 days;
    uint256 newLockDays = originalLockDays + _additionalDays;
    require(newLockDays <= MAX_LOCK_DAYS, "stHOOKV2: lock period exceeds maximum");

    removeElement(unlockAddresses[user.unlockDate], msg.sender);

    user.unlockDate = calculateUnlockDate(user.lockTime, newLockDays);
    require(user.unlockDate <= EXPIRED_DAY, "stHOOKV2: lock period exceeds expiration
date");

    ...
}
```

contracts/stHookV2.sol:L301-308

```
function removeElement(address[] storage array, address target) internal {
    for (uint256 i = 0; i < array.length; i++) {
        if (array[i] == target) {
            array[i] = array[array.length - 1];
            array.pop();
            break;
        }
    }
}
```

The unlockAddresses is a mapping variable that uses unlockDate as a key to find all users (stored as an array of addresses) whose lock period ends on a specific day.

However, when a user extends the lock period in the extendLockPeriod function, the original unlockDate may no longer be accurate.

Within the function, the removeElement function is called, which iterates through unlockAddresses[unlockDate] to find msg.sender and delete it. If this array is too long, users located towards the end of the array may not be able to successfully call this function to extend their lock period.

Recommendation

Consider using the EnumerableSet contract from OpenZeppelin to avoid the additional gas costs associated with loop-based queries.

Status

This issue has been resolved by the team with commit 9bab3cd.

3. Incorrect balance check

Severity: Low	Category: Business Logic
Target: - contracts/stHookV2.sol	

Description

contracts/stHookV2.sol:L317-322

```
function unpause() public onlyOwner {
    uint256 balance = hookToken.balanceOf(address(this));
    require(balance >= accruedRewards + totalSupply() , "stHOOKV2: insufficient contract
balance");
    paused = false;
    emit Pause(paused);
}
```

In the highlighted section of the above code, it is used to determine whether the contract has enough tokens to distribute rewards and the user's principal.

However, when calculating the expected reward value, the reward for the current day (rewardPerDay) has not been taken into account.

This may result in the contract not having enough tokens to pay out all the rewards, but not entering the pause state correctly.

Recommendation

Consider incorporating the rewards for the current day when calculating the expected reward.

For example, the if condition can be changed to the following:

```
require(balance >= accruedRewards + rewardPerDay + totalSupply() , "stHOOKV2:
insufficient contract balance");
```

Status

This issue has been resolved by the team with commit 9bab3cd.

4. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/stHookV2.sol	

Description

In the stHookV2 contract, there is a privileged owner role. The owner role has the ability to:

- pause/unpause the contract
- setting the rewardPerDay
- execute patch functions

If the owner's private key is compromised, the attacker can exploit the owner's role to pause the contract and transfer the owner to zero address, making the contract suspended forever. In this case, the user will not be able to withdraw and claim reward.

Recommendation

We recommend transferring the owner role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

Status

This issue has been mitigated by the team by using a multisig wallet as the owner.

2.3 Informational Findings

5. Gas optimization

Severity: Information	Category: Gas Optimization
Target: - contracts/stHookV2.sol	

Description

Cache array length in for loops can save gas.

contracts/stHookV2.sol:L219-226

```
function settleRewards() public notPaused {
    ...
    for (uint256 i = 0; i < unlockAddresses[settledDay].length; i++) {
        address userAddress = unlockAddresses[settledDay][i];
        UserInfo storage user = userInfo[userAddress];
        user.fixedReward = pendingReward(userAddress);
        user.rewardDebtPerShare = accRewardPerShare;
        totalShares -= user.boosterShare;
        user.boosterShare = 0;
    }

    ...
}
```

contracts/stHookV2.sol:L301-308

```
function removeElement(address[] storage array, address target) internal {
    for (uint256 i = 0; i < array.length; i++) {
        if (array[i] == target) {
            array[i] = array[array.length - 1];
            array.pop();
            break;
        }
    }
}
```

Recommendation

Consider Caching the array length in the stack.

Status

This issue has been resolved by the team with commit 9bab3cd.

6. Redundant code

Severity: Informational	Category: Redundancy
Target: - contracts/stHookV2.sol	

Description

contracts/stHookV2.sol:L13

```
bool public paused = false;
```

The default value is already false, so there is no need for additional assignment.

Recommendation

Consider removing the redundant codes.

Status

This issue has been resolved by the team with commit 9bab3cd.

7. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/stHookV2.sol	

Description

The stHOOKV2 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 acknowledged by the team.

8. Missing zero address checks

Severity: Informational	Category: Data Validation

Target:
- contracts/stHookV2.sol

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for the address variable hookToken.

contracts/stHookV2.sol:L54

```
constructor(IERC20 _hookToken, uint256 _rewardPerDay, uint256 _startDay) ERC20("Staked
Hook Token V2", "stHOOKV2") {
    require(_startDay == (_startDay - (_startDay % 1 days)), "stHOOKV2: invalid start
day");
    hookToken = _hookToken;
    rewardPerDay = _rewardPerDay;
    startDay = _startDay;
    settledDay = _startDay;
}
```

Recommendation

Consider adding zero address checks for the address variable hookToken.

Status

This issue has been resolved by the team with commit 9bab3cd.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 949f18d:

File	SHA-1 hash
contracts/stHookV2.sol	910a315eeb1e4ff639e9153f485ade6998b0531c