Gateway - HP
Security Assessment Findings Report

Date: November 10, 2024
Version 0.1

Contents

1 Confidentiality statement

This document is the exclusive property of Sub7 Security and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Sub7 Security and Sub7 Security.

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Humanity Protocol

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

1 (One) Security Auditors/Consultants were engaged in this activity.

5.1 Scope

https://github.com/gateway-fm/hi-rewards-contract.git

5.2 Timeline

From 01 November 2024 to 10 November 2024

5.3 Summary of Findings Identified

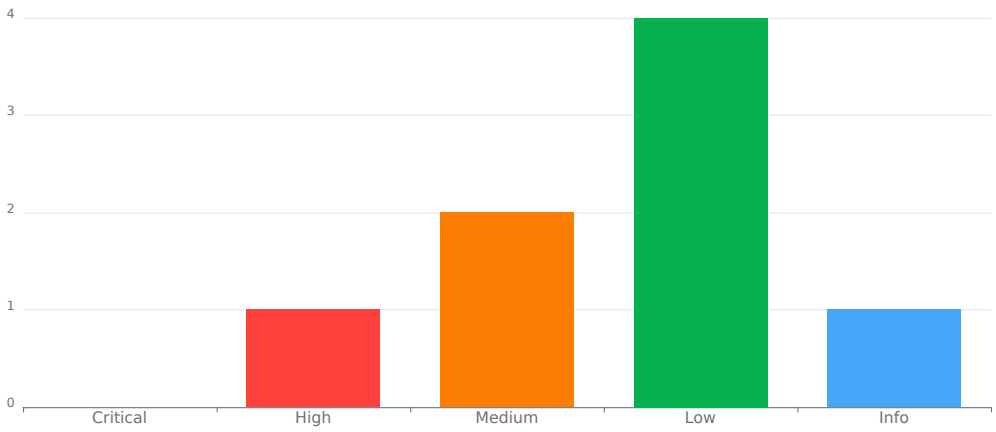

Figure 1: Executive Summary

 # 1 High Incorrect Epoch Counting After Contract Stop and Start – ***Fixed***

 # 2 Medium MIssing check in `processBatch` function if the refferrer address a user. – ***Fixed***

 # 3 Medium Missing Duplicate User Check in processBatch Function – ***Fixed***

 # 4 Low Incorrect Access Control Inheritance for Upgradeable Contract – ***Fixed***

 # 5 Low Start Timestamp Can Be Set Before Last Stop Time which can lead to wrong calculations – ***Fixed***

 # 6 Low Use a 2-step ownership transfer pattern – ***Fixed***

 # 7 Low Initializers could be front-run – ***Fixed***

 # 8 Info Gas optimizations – ***Fixed***

5.4 Methodology

SUB7's audit methodology involves a combination of different assessments that are performed to the provided code, including but not limited to the following:

Specification Check

Manual assessment of the assets, where they are held, who are the actors, privileges of actors, who is allowed to access what and when, trust relationships, threat model, potential attack vectors, scenarios, and mitigations. Well-specified code with standards such as NatSpec is expected to save time.

Documentation Review

Manual review of all and any documentation available, allowing our auditors to save time in inferring the architecture of the project, contract interactions, program constraints, asset flow, actors, threat model, and risk mitigation measures

Automated Assessments

The provided code is submitted via a series of carefully selected tools to automatically determine if the code produces the expected outputs, attempt to highlight possible vulnerabilities within non-running code (Static Analysis), and providing invalid, unexpected, and/or random data as inputs to a running code, looking for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Examples of such tools are Slither, MythX, 4naly3er, Sstan, Natspec-smells, and custom bots built by partners that are actively competing in Code4rena bot races.

Manual Assessments

Manual review of the code in a line-by-line fashion is the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found. This intensive assessment will check business logics, intended functionality, access control & authorization issues, oracle issues, manipulation attempts and multiple others.

Security Consultants make use of checklists such as SCSVS, Solcurity, and their custom notes to ensure every attack vector possible is covered as part of the assessment

6 Findings and Risk Analysis

6.1 Incorrect Epoch Counting After Contract Stop and Start

Severity: High
Status: Fixed

Description

The `Rewards` contract incorrectly calculates epoch numbers when restarted after being stopped. The contract skips an epoch in the transition, leading to a gap in epoch numbers and potential missed reward claims.

6.1.1 Technical Details

The issue occurs in the epoch calculation logic in `_currentEpoch()` and interaction between `stop()` and `start()` functions:

```
function _currentEpoch() internal view returns (uint256) {
    if (_contractActive) {
        return ((block.timestamp - _cycleStartTimestamp) / (1 days)) + 1 +
            _previousCycleLastEpochID;
    } else {
        return _previousCycleLastEpochID;
    }
}

function stop() external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_contractActive, "Rewards: contract not active");
    _previousCycleLastEpochID = _currentEpoch();
    _contractActive = false;
}

function start(uint256 startTimestamp) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(!_contractActive, "Rewards: contract already active");
    require(
        startTimestamp <= block.timestamp,
        "Rewards: start timestamp should be less or equal to current timestamp"
    );
    _cycleStartTimestamp = startTimestamp;
    _contractActive = true;
}
```

Reproduction test case:

```
describe("Epoch counting", function () {
    it("Should correctly count epochs across multiple stop/starts", async function () {
      const { rewards } = await loadFixture(deployFixture);

      // First cycle
      const startTime = await time.latest();
```

```
        console.log("Initial start time:", startTime);
        await rewards.start(startTime);
        expect(await rewards.currentEpoch()).to.equal(1);

        // Advance 5 days
        await time.increase(5 * Day);
        const timeAfter5Days = await time.latest();
        console.log("Time after 5 days:", timeAfter5Days);
        const epochBeforeFirstStop = await rewards.currentEpoch();
        expect(epochBeforeFirstStop).to.equal(6);
        await rewards.stop();

        // Second cycle that is started immediately
        const secondStartTime = await time.latest();
        console.log("Second start time:", secondStartTime);
        await rewards.start(secondStartTime);
        const epochAfterFirstStart = await rewards.currentEpoch();
        expect(epochAfterFirstStart).to.equal(7); // Should be 6 since no time passed
      });
});
```

6.1.2 Impact

1. Users can miss claiming rewards for skipped epochs
2. Reward distribution becomes inconsistent
3. Historical data and analytics based on epoch numbers become inaccurate

6.1.3 Root Cause

When the contract is stopped and immediately restarted: 1. `stop()` saves the current epoch (6 in test case) as `_previousCycleLastEpochID` 2. `start()` sets a new `_cycleStartTimestamp` 3. `_currentEpoch()` calculation adds 1 to the days difference plus the previous epoch 4. This causes an immediate jump to the next epoch even when no time has passed

Location

implementation/rewards/Rewards.sol#L60-L79

Recommendation

Modify the `_currentEpoch()` function to handle continuous epoch counting:

Comments

6.2 MIssing check in `processBatch` function if the refferrer address a user.

Severity: Medium
Status: Fixed

Description

7 Smart Contract Audit Report

7.1 Missing Referrer Validation in processBatch Function

7.1.1 Issue Description

The `processBatch` function in the VC contract lacks validation for referrer addresses, unlike the `_register` function. This missing check allows the registration of users with invalid referrers who aren't registered in the system.

7.1.2 Technical Details

In the `_register` function, there's a proper validation:

```
require(referrerAddress == address(0) || _isUserExist(referrerAddress), "VC: referrer does
    not exist");
```

However, the `processBatch` function lacks this check when processing user registrations:

```
function processBatch(User[] calldata users) external onlyRole(REGISTRATION_ADMIN_ROLE) {
    for (uint256 i = 0; i < users.length; i++) {
        User memory user = users[i];
        _users[user.userAddr] = user;
        // Missing referrer validation here
        ...
    }
}
```

7.1.3 Impact

1. Users can be registered with referrers that don't exist in the system
2. When referral rewards are distributed, they could be sent to addresses that aren't legitimate participants
3. This undermines the integrity of the referral system
4. Potential economic impact if rewards are distributed based on referral relationships

Function is admin control so probability is low but the impact is high (irregular reward distribution), hence the medium severity.

Location

implementation/vc/VC.sol#L128-L143

Recommendation

Add the same referrer validation check to the `processBatch` function:

```
function processBatch(User[] calldata users) external onlyRole(REGISTRATION_ADMIN_ROLE) {
    for (uint256 i = 0; i < users.length; i++) {
        User memory user = users[i];
        require(
            user.referrerAddr == address(0) || _isUserExist(user.referrerAddr),
            "VC: referrer does not exist"
        );
        _users[user.userAddr] = user;
        _usersCountOnRegistration[user.userAddr] = _totalUsers;
        _totalUsers++;

        emit UserRegistered(user.userAddr, user.referrerAddr);

        if (user.verified) {
            _totalVerifiedUsers++;
            emit UserVerified(user.userAddr);
        }
    }
}
```

Comments

7.2 Missing Duplicate User Check in processBatch Function

Severity: Medium
Status: Fixed

Description

8 Smart Contract Audit Report

8.1 Missing Duplicate User Check in processBatch Function

8.1.1 Issue Description

The `processBatch` function lacks validation to prevent duplicate user registrations. While the `_register` function includes this check, `processBatch` allows the same user to be registered multiple times, leading to incorrect state updates.

8.1.2 Technical Details

In the `_register` function, duplicate users are prevented with:

```
require(!_isUserExist(userAddress), "VC: user already exist");
```

The `processBatch` function processes user registrations without this check:

```
function processBatch(User[] calldata users) external onlyRole(REGISTRATION_ADMIN_ROLE) {
    for (uint256 i = 0; i < users.length; i++) {
        User memory user = users[i];
        _users[user.userAddr] = user;            // No duplicate check
        _usersCountOnRegistration[user.userAddr] = _totalUsers;
        _totalUsers++;                           // Increments count even for duplicates

        emit UserRegistered(user.userAddr, user.referrerAddr);

        if (user.verified) {
            _totalVerifiedUsers++;               // Can increment verified count multiple
                times
            emit UserVerified(user.userAddr);
        }
    }
}
```

8.1.3 Impact

1. The same user can be registered multiple times
2. `_totalUsers` counter increases incorrectly for duplicate registrations
3. If the duplicate registration includes verified status, `_totalVerifiedUsers` also increases incorrectly
4. Multiple `UserRegistered` and `UserVerified` events are emitted for the same user
5. This could skew analytics and reward distributions that rely on accurate user counts

Location

implementation/vc/VC.sol#L128-L143

Recommendation

Add duplicate user check to the `processBatch` function:

```
function processBatch(User[] calldata users) external onlyRole(REGISTRATION_ADMIN_ROLE) {
    for (uint256 i = 0; i < users.length; i++) {
        User memory user = users[i];

        // Add check for existing user
        require(!_isUserExist(user.userAddr), "VC: user already exist");

        // Add check for valid referrer (combining with previous audit fix)
        require(
            user.referrerAddr == address(0) || _isUserExist(user.referrerAddr),
            "VC: referrer does not exist"
        );

        _users[user.userAddr] = user;
        _usersCountOnRegistration[user.userAddr] = _totalUsers;
        _totalUsers++;
```

```
17
18          emit UserRegistered(user.userAddr, user.referrerAddr);
19
20          if (user.verified) {
21              _totalVerifiedUsers++;
22              emit UserVerified(user.userAddr);
23          }
24      }
25  }
```

8.1.4 Test Cases

Test scenarios should include: 1. Attempt to batch register a list containing a duplicate user address 2. Attempt to batch register a user that was previously registered via single registration 3. Attempt to batch register a user that was previously registered in a different batch 4. Verify total user counts remain accurate after failed duplicate registrations 5. Verify verified user counts remain accurate when duplicate verified users are rejected

8.1.5 Additional Note

Consider adding a modifier or helper function to handle these common validation checks across both registration methods to maintain consistency and reduce code duplication.

Comments

8.2 Incorrect Access Control Inheritance for Upgradeable Contract

Severity: Low
Status: Fixed

Description

The `Rewards` contract uses `Initializable` for upgradeability but inherits from `AccessControl` instead of `AccessControlUpgradeable`.

8.2.1 Impact

Low severity - While the contract still functions, it doesn't follow best practices for upgradeable contracts. This could potentially cause issues in complex upgrade scenarios.

Location

implementation/rewards/Rewards.sol

Recommendation

Replace:

```
import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
```

with:

```
import { AccessControlUpgradeable } from "@openzeppelin/contracts-upgradeable/access/
    AccessControlUpgradeable.sol";
```

And update the contract inheritance:

```
contract Rewards is IRewards, AccessControlUpgradeable, Initializable {
```

Comments

8.3 Start Timestamp Can Be Set Before Last Stop Time which can lead to wrong calculations

Severity: Low
Status: Fixed

Description

`start()` allows setting `startTimestamp` before the last stop time, causing incorrect epoch progression.

Example: - Stop at day 10 → `_previousCycleLastEpochID = 10`
- Restart at day 5 - New epoch calculation = `((day 10 - day 5)/ 1 days)+ 1 + 10 = 16` - Instead of continuing from epoch 11, it jumps to epoch 16, completely skipping epochs 11-15

Location

implementation/rewards/Rewards.sol

Recommendation

```
uint256 private _lastStopTimestamp;

function stop() external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_contractActive, "Rewards: contract not active");
    _previousCycleLastEpochID = _currentEpoch();
    _lastStopTimestamp = block.timestamp; // track the last stop time
    _contractActive = false;
}

function start(uint256 startTimestamp) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
11      require(!_contractActive, "Rewards: contract already active");
12      require(startTimestamp <= block.timestamp, "Rewards: start timestamp should be less or
            equal to current timestamp");
13      require(startTimestamp >= _lastStopTimestamp, "Rewards: start timestamp should be
            after last stop"); // don't allow the start time to be greater than last stop time
            .
14      _cycleStartTimestamp = startTimestamp;
15      _contractActive = true;
16  }
```

Comments

8.4 Use a 2-step ownership transfer pattern

Severity: Low
Status: Fixed

Description

Ownership transfer is done in a single step, which could lead to accidentally transferring ownership to the wrong address with no way to recover.

8.4.1 Impact:

If the owner uses an incorrect address during ownership transfer, contract ownership could be permanently lost, blocking all admin functions.

8.4.2 Instances:

```
1  File: implementation/gasToken/GasToken.sol
2  7: contract GasToken is ERC20, Ownable {
3
4  File: implementation/rewardToken/RewardToken.sol
5  11: contract RewardToken is ERC20, Ownable {
```

Location

implementation/rewardToken/RewardToken.sol

implementation/gasToken/GasToken.sol

Recommendation

Implement a two-step ownership transfer where: 1. Owner calls function to nominate new owner 2. New owner must call separate function to accept ownership

Example using OpenZeppelin's Ownable2Step:

```
import {Ownable2Step} from "@openzeppelin/contracts/access/Ownable2Step.sol";

contract GasToken is ERC20, Ownable2Step {
  ...
}
```

Comments

8.5 Initializers could be front-run

Severity: Low
Status: Fixed

Description

The initialization functions can be called by anyone before the intended deployer, allowing attackers to take control of the contract.

8.5.1 Impact:

An attacker monitoring the mempool could front-run the initialization transaction and take control of critical contract parameters and admin roles.

8.5.2 Instances:

```
File: implementation/rewards/Rewards.sol
38:     function init(address vcContract, address tkn) external initializer {

File: implementation/vc/VC.sol
41:     function init() external initializer {
```

Location

implementation/vc/VC.sol

implementation/rewards/Rewards.sol

Recommendation

1. Deploy contracts using a factory pattern that initializes in the same transaction - this can be tackled in deployment script too.
2. Add access controls to initialization

Comments

8.6 Gas optimizations

Severity: Info
Status: Fixed

Description

8.6.1 [G-1] a = a + b is more gas effective than a += b for state variables

Saves 16 gas per instance.

```
File: implementation/rewards/Rewards.sol
158: _userBuffers[msg.sender] += _userClaims[msg.sender][_currentEpoch()].buffer;
177: _userBuffers[msg.sender] += _userClaims[msg.sender][_currentEpoch()].buffer;
205: _userBuffers[referrers[i]] += buffer;
208: _userClaims[referrers[i]][_currentEpoch()].buffer += buffer;
```

8.6.2 [G-2] Cache array length outside of loop

Saves extra sload/mload operation (100/3 gas) per iteration except first.

```
File: implementation/rewards/Rewards.sol
197: for (uint256 i = 0; i < referrers.length; i++)

File: implementation/vc/VC.sol
138: for (uint256 i = 0; i < users.length; i++)
```

8.6.3 [G-3] State variables should be cached in stack variables

Saves 100 gas per instance by reducing SLOAD operations.

```
File: implementation/rewards/Rewards.sol
153: _increaseBuffer(amount, IVC(_vcContract).getReferrersTree(msg.sender));
172: IERC20(_tkn).transfer(msg.sender, amount);
174: _increaseBuffer(amount, IVC(_vcContract).getReferrersTree(msg.sender));
```

8.6.4 [G-4] Use Custom Errors instead of Revert Strings

Saves ~50 gas per instance by avoiding string storage.

```
// Instead of
require(amount > 0, "Amount must be positive");

// Use
error AmountTooLow();
```

```
6  if (amount == 0) revert AmountTooLow();
```

Location

implementation/rewards/Rewards.sol

implementation/vc/VC.sol

Recommendation

Mentioned above

Comments