SALUS SECURITY

NOV 2024

CODE SECURITY ASSESSMENT

ING

Overview

Project Summary

- Name: InG - infinity ground checkin
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/xiaomailisten/infinity_ground_checkin
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	InG - infinity ground checkin
Version	v2
Type	Solidity
Dates	Nov 04 2024
Logs	Nov 02 2024; Nov 04 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	3
Total	5

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	Misuse of lastClaim leads to loss of reward for users	High	Business Logic	Resolved
2	Missing events for functions that change critical state	Low	Logging	Resolved
3	Missing zero address checks	Informational	Data Validation	Resolved
4	Use call instead of transfer for native tokens transfer	Informational	Business logic	Resolved
5	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. Misuse of lastClaim leads to loss of reward for users

Severity: High	Category: Business Logic

Target:
- infinity_ground_checkin/src/LevelRewards.sol

Description

infinity_ground_checkin/src/LevelRewards.sol:L56-L78

```
function _resetDaily(address user) internal {
    if (block.timestamp >= userInfo[user].lastClaim + 1 days) {
        userInfo[user].dailyRewards = 0;
    }
}

// 用户领取奖励
function claimReward(uint256 amount) external nonReentrant whenNotPaused {
    UserInfo storage user = userInfo[msg.sender];
    require(user.level > 0, "User level not set");
    require(amount > 0, "Invalid amount");

    _resetDaily(msg.sender);

    Level storage level = levels[user.level];
    require(user.dailyRewards + amount <= level.dailyLimit, "Daily limit exceeded");

    user.dailyRewards += amount;
    user.lastClaim = block.timestamp;

    usdt.safeTransfer(msg.sender, amount);
    emit RewardClaimed(msg.sender, amount);
}
```

In _resetDaily(), the function uses the lastClaim to determine whether to reset the accumulated rewards for the day. The team mistakenly assumed that lastClaim marked the start time for the day's rewards, while in reality, lastClaim updates with each new claim.

This misunderstanding prevents users from successfully resetting their daily rewards after one day, potentially causing them to lose a portion of their earned rewards.

Proof of Concept

The following tests demonstrate that users are unable to claim their earned rewards on the second day.

```
function testDailyLimitDos() public {
    uint256 entryProtocolTime = 1000;

    // Set user level to 1 (daily limit 2 USDT)
    rewards.setUserLevel(user1, 1);
```

```
    vm.startPrank(user1);

    vm.warp(entryProtocolTime);
    rewards.claimReward(1 * 10 ** 18);

    vm.warp(entryProtocolTime + 1 days - 1);
    rewards.claimReward(0.9 * 10 ** 18);

    vm.warp(entryProtocolTime + 2 days - 100);
    // Should have been available for claim the next day
    vm.expectRevert("Daily limit exceeded");
    rewards.claimReward(0.9 * 10 ** 18);
}
```

Recommendation

It is recommended that lastClaim be updated only in _resetDaily at the same time as dailyRewards.

Status

The team has resolved this issue in commit [ee60f64](ee60f64).

2. Missing events for functions that change critical state

Severity: Low	Category: Logging

Target:
- infinity_ground_checkin/src/Checkin.sol
- infinity_ground_checkin/src/LevelRewards.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 `Checkin.sol`, events are lacking in the following functions:

- updateAdmin()
- updateCheckinFee()
- withdraw()

In the `LevelRewards.sol`, events are lacking in the following functions:

- setSigner()

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit ee60f64.

2.3 Informational Findings

3. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - infinity_ground_checkin/src/LevelRewards.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 address variables `signer`.

Even worse, if the signer is set to the zero address, the signature can be exploited by anyone.

Recommendation

Consider adding zero address checks for address variables.

Status

The team has resolved this issue in commit ee60f64.

4. Use call instead of transfer for native tokens transfer

Severity: Informational	Category: Business logic
Target: - infinity_ground_checkin/src/Checkin.sol	

Description

The transfer function is not recommended for sending native tokens due to its 2300 gas unit limit which may not work with smart contract wallets or multi-sig. Instead, call can be used to circumvent the gas limit.

infinity_ground_checkin/src/Checkin.sol:L88-L93

```
function sendGas(address to) external {
    if (msg.sender != admin && msg.sender != owner()) {
        revert InvalidAdmin();
    }
    payable(to).transfer(gasFee);
}
```

Recommendation

Consider using call instead of transfer for sending native token.

Status

The team has resolved this issue in commit ee60f64.

5. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - infinity_ground_checkin/src/Checkin.sol - infinity_ground_checkin/src/LevelRewards.sol	

Description

The Checkin and LevelRewards 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

The team has resolved this issue in commit ee60f64.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 099716c:

File	SHA-1 hash
Checkin.sol	c45bbd444853d9ff40261e324574c1a85a5fae0a
LevelRewards.sol	de78bf7b702205d9d45815cffc30f151f1e76825