SALUS SECURITY

DEC 2024

CODE SECURITY ASSESSMENT

VERIO LABS

Overview

Project Summary

- Name: VerioLabs - VerioIPA
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/VerioLabs/VerioIPA
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	VerioLabs - VerioIPA
Version	v2
Type	Solidity
Dates	Dec 19 2024
Logs	Dec 16 2024; Dec 19 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	4
Total informational issues	5
Total	10

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	Part of the rewards may not be claimable	Medium	Business Logic	Resolved
2	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Business Logic	Resolved
3	Incomplete initialization	Low	Business Logic	Resolved
4	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
5	Centralization risk	Low	Centralization	Acknowledged
6	Stake logic mismatched	Informational	Business logic	Resolved
7	The return value is always 0	Informational	Numerics	Resolved
8	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved
9	Redundant Code	Informational	Redundancy	Resolved
10	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Part of the rewards may not be claimable

Severity: Medium	Category: Business Logic
Target: - src/IPAStaking.sol	

Description

According to the contract logic, rewards are claimed on a per-epoch basis.

Any staking period shorter than one epoch is considered invalid, meaning staking for 1.9 epochs and 1 epoch will result in the same reward.

In this situation, there exists a potential threat: if a user plans to claim rewards after staking for 2 epochs, an attacker could front-run by calling the `_distributeRewards()` function at 1.9 epochs. This would render the 0.9 epoch of staking invalid, causing some rewards to become unclaimable.

src/IPAStaking.sol:L624 - L670

```
function _distributeRewards(address ip) internal {
    ...
    for (uint256 i = 0; i < ipInfo.rewardPools.length; i++) {
        ...
        uint256 timePassed = block.timestamp - rewardPool.lastEpochTime;
            if (timePassed > 0 && rewardPool.totalDistributedRewards <
rewardPool.totalRewards) {
                uint256 rewards = (timePassed / config.epochDuration) *
rewardPool.rewardsPerEpoch;
                ...
            }
    ...
}
```

Recommendation

Consider restricting the access permissions of the `_distributeRewards()` function or modifying its logic to prevent malicious behavior by attackers.

Status

The team has resolved this issue in commit 08be90.

2. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Business Logic
Target: - src/IPAStaking.sol	

Description

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

The team has resolved this issue in commit 6b88775.

3. Incomplete initialization

Severity: Low	Category: Business Logic
Target: - src/IPAStaking.sol	

Description

The `IPAStaking` contract inherits from the `ReentrancyGuardUpgradeable` contract, but the `initialize()` function does not initialize the `ReentrancyGuardUpgradeable` contract.

Recommendation

Consider initializing the `ReentrancyGuardUpgradeable` contract within the `initialize()` function.

Status

The team has resolved this issue in commit 6b88775.

4. Implementation contract could be initialized by everyone

Severity: Low	Category: Business Logic
Target: - src/IPAStaking.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the `initialize()` function in `IPAStaking`'s implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the `_disableInitializers()` function in the constructor of the `IPAStaking` contract to automatically lock it when it is deployed.

Status

The team has resolved this issue in commit 6b88775.

5. Centralization risk

Severity: Low	Category: Centralization
Target: - src/IPAStaking.sol	

Description

The `IPAStaking` contract contains privileged addresses, `admin` and `whitelister`. The `admin` has the authority to arbitrarily add or remove `whitelisters` and `authorizedUsers`, execute the `freezeIP()` and `unfreezeIP()` functions, as well as modify other critical variables in the contract. Meanwhile, the `whitelister` has the ability to add or remove `IPs` in the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

6. Stake logic mismatched

Severity: Informational	Category: Business Logic
Target: - src/IPAStaking.sol	

Description

The stake logic has boundary cases that prevent reaching the maximum stake value. For example, if the minimum value is 300 and the maximum value is 1000, then when the stake reaches 701, the remaining space of 299 cannot be filled.

src/IPAStaking.sol:L284 - L312

```
function stake(address ip, bool tryCapped, uint256 amount)
    external
    whenNotPaused
    onlyWhitelistedIP(ip)
    whenIPNotFrozen(ip)
    nonReentrant
{
    require(!_shouldFreeze(ip), "IPAStaking: IP is under dispute and will be frozen");
    require(amount >= config.minStakeAmount, "IPAStaking: Amount must be greater than
minimum stake");
    ...

    require(pool.totalStakedInPool + amount <= pool.cap, "IPAStaking: Amount exceeds
cap");
    ...
}
```

Recommendation

Consider modifying the logic for the maximum and minimum stake values to eliminate the boundary cases that prevent reaching the maximum stake amount.

Status

The team has resolved this issue in commit 6b88775.

7. The return value is always 0

Severity: Informational	Category: Numerics
Target: - src/IPAStaking.sol	

Description

src/IPAStaking.sol:L618 - L620

```
function distributeRewards(address ip) external returns (uint256) {
    _distributeRewards(ip);
}
```

The return value exists in the definition of the `distributeRewards` function, but no value is assigned to it in the function, resulting in the return value always being zero.

Recommendation

It is recommended to give the correct return value in the function.

Status

The team has resolved this issue in commit 6b88775.

8. Missing two-step transfer ownership pattern

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

Description

The `IPAStaking` contract inherits from the `OwnableUpgradeable` 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 Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 6b88775.

9. Redundant Code

Severity: Informational	Category: Redundancy
Target: - src/IPAStaking.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following code are not being utilized:

src/IPAStaking.sol:L9

```
import "openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol";
```

src/IPAStaking.sol:L278 - L281

```
function _shouldSlash(address ip) internal returns (bool) {
    // slash when IP is disputed successfully
    return false;
}
```

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit 6b88775.

10. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.23;
```

All contracts use a floating compiler version `^0.8.23`.

Using a floating pragma `^0.8.23` statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 788bb35:

File	SHA-1 hash
src/IPAStaking.sol	386cc72ab8bd8c17fd4a81886033861cea57cc93