SALUS SECURITY

DEC 2024

CODE SECURITY ASSESSMENT

VERIO LABS

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	VerioLabs - VerioIP
Version	v2
Type	Solidity
Dates	Dec 25 2024
Logs	Dec 24 2024; Dec 25 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	4
Total Low-Severity issues	3
Total informational issues	4
Total	12

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	Logic Error in Inner and Outer Loops	High	Business Logic	Resolved
2	Stake logic error	Medium	Business Logic	Resolved
3	The _handleunstake function may have dos problems	Medium	Business Logic	Resolved
4	The unstake operation could face DoS risks	Medium	Business Logic	Resolved
5	Centralization risk	Medium	Centralization	Acknowledged
6	Implementation contract could be initialized by everyone	Low	Business logic	Resolved
7	Use call instead of transfer for native tokens transfer	Low	Business logic	Resolved
8	Missing events for functions that change critical state	Low	Business logic	Resolved
9	Config limit	Informational	Redundancy	Resolved
10	Missing two-step transfer ownership pattern	Informational	Numerics	Resolved
11	Report a wrong error message	Informational	Inconsistency	Resolved
12	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Logic Error in Inner and Outer Loops	
Severity: High	Category: Business Logic
Target: - src/delegation/Delegator.sol	

Description

In the inner loop of the `` `_redelegate()` `` function, the variable `` `j` `` is not properly handled with the outer loop's `` `i` ``, and it serves no purpose here. `` `i` `` represents `` `_attempts` ``, meaning "Number of redelegation attempts to make" indicating the number of iterations for redelegating, which contradicts the intended role of `` `j` ``.

src/delegation/Delegator.sol:L612 - L644

```
for (uint256 i = 0; i < _attempts; i++) {
               ...
    if (fromValidatorDelegations.distribution.instant > 0) {
                        ...
    } else {
        for (uint256 j = 0; i < bondedMaturities.length; j++) {
            IIPTokenStaking.StakingPeriod maturity = bondedMaturities[i];
            ...
        }
    }
```

Recommendation

Consider modifying the loop logic of the inner loop variable.

```
for (uint256 j = 0; j < bondedMaturities.length; j++) {
    IIPTokenStaking.StakingPeriod maturity = bondedMaturities[j];
```

Status

The team has resolved this issue in commit 3a0b05f.

2. Stake logic error

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

Description

The user's stakes are divided into demand and three fixed staking modes. After reaching the stake limit for one mode, it moves to the next. However, there is a logic flaw in the function when checking if the staking threshold is met: it only considers the current staked amount, while the correct approach should also account for the `_amount` to be staked, otherwise, the stake may exceed the desired `bps`.

src/delegation/Delegator.sol:L162 - L184

```
uint256 targetInstantAmount = _calculateProportion(
    globalDistribution.total + delegation.amount,
    distributionConfig.configByMaturity[IIPTokenStaking.StakingPeriod.FLEXIBLE].bps
);

if (globalDistribution.totalByMaturity[IIPTokenStaking.StakingPeriod.FLEXIBLE] <
targetInstantAmount) {
    return delegation;
}

if (delegatorConfig.bondedMaturitiesEnabled) {
    // iterate over reverse maturity order (try longest first)
    for (uint256 i = bondedMaturities.length - 1; i >= 0; i--) {
        IIPTokenStaking.StakingPeriod maturity = bondedMaturities[i];
        MaturityConfig memory maturityConfig =
distributionConfig.configByMaturity[maturity];
        uint256 targetAmount =
            _calculateProportion(globalDistribution.total + delegation.amount,
maturityConfig.bps);
        if (globalDistribution.totalByMaturity[maturity] < targetAmount) {
            delegation.maturity = maturity;
            delegation.duration = maturityConfig.duration;
            delegation.endTimestamp = block.timestamp + maturityConfig.duration;
            break;
        }
    }
}
```

Recommendation

Consider including the `_amount` to be staked in the threshold checking logic.

Status

The team has resolved this issue in commit 3a0b05f.

3. The _handleunstake function may have DoS problems	
Severity: Medium	Category: Business Logic
Target: - src/delegation/Delegator.sol	

Description

The `Delegator` contract allows the administrator to manually turn the time deposit option on and off, and the `_handleunstake ` function verifies that the time deposit option is currently turned on when it performs the unstake operation. If the time deposit option is turned off after the time deposit has been open for a certain period of time, other users will not be able to perform the `unstake` operation normally after a user has finished withdrawing the current portion of the deposit.

src/delegation/Delegator.sol:L347 - L354

```
function _handleUnstake(uint256 _amount) internal returns (uint256 excessUnstake) {
    uint256 remainingUnstake;
    (remainingUnstake, excessUnstake) = _handleInstantUnstakes(_amount);
    if (remainingUnstake > 0 && delegatorConfig.bondedMaturitiesEnabled) {
        (remainingUnstake, excessUnstake) = _handleBondedUnstakes(remainingUnstake);
    }
    require(remainingUnstake == 0, "Delegator: Could not unstake full amount");
}
```

Recommendation

It is recommended that the `_handleunstake ` function does not check whether the `delegatorConfig.bondedMaturitiesEnabled` variable is `true` or not.

Status

The team has resolved this issue in commit 3a0b05f.

4. The unstake operation could face DoS risks	
Severity: Medium	Category: Business Logic
Target: - src/delegation/Delegator.sol	

Description

src/delegation/Delegator.sol:L400 - L410

```
function _calculateInstantUnstakeAmount(uint256 _existingStake, uint256 _unstakeAmount)
    internal
    view
    returns (uint256)
{
    uint256 unstakeAmount = _unstakeAmount < _existingStake ? _unstakeAmount :
_existingStake;
    if (!_stakeThresholdMet(unstakeAmount)) {
        unstakeAmount = delegatorConfig.stakeThreshold;
    }
    return unstakeAmount;
}
```

When the `_existingStake` in the contract is less than the amount the user expects to unstake, the actual unstake amount will be updated to `_existingStake`. However, if this value is less than `stakeThreshold`, it will be reassigned to `stakeThreshold`, which could result in the unstake operation being subjected to a DoS risk.

Recommendation

Consider addressing the case where `_existingStake` is less than `stakeThreshold` in the function to prevent this issue.

Status

The team has resolved this issue in commit 3a0b05f.

5. Centralize risk	
Severity: Medium	Category: Centralization
Target: - All	

Description

The contract contains privileged addresses, `admin` and `owner`. The `admin` has the authority to modify all critical parameters within the contract, such as setting the fee to 100%, changing the treasury, altering the stake pool, and more. Furthermore, the `owner` has the ability to modify the `admin`.

If the private key of either `admin` or `owner` is compromised, an attacker could manipulate these critical parameters, disrupt the normal operation of the contract, and harm the interests of regular 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.

6. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - All	

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 `Delegator`, `StakePool`, `WithdrawalPool`, `VerioIP`, `InsurancePool`, `RewardPool` 's implementation contract.

Recommendation

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

Status

The team has resolved this issue in commit 3a0b05f.

7. Use call instead of transfer for native tokens transfer

Severity: Informational	Category: Business logic
Target: - src/insurance/InsurancePool.sol - src/rewards/RewardPool.sol - src/withdrawal/WithdrawalPool.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.

Recommendation

Consider using a `call` instead of `transfer` for sending native tokens.

Status

The team has resolved this issue in commit 3a0b05f.

8. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - All	

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 `` `Delegator` ``, `` `StakePool` ``, `` `WithdrawalPool` ``, `` `VerioIP` ``, `` `InsurancePool` ``, `` `RewardPool` `` contracts, events are lacking in the privileged setter functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 3a0b05f.

2.3 Informational Findings

9. Config limit	
Severity: Informational	Category: Configuration
Target: - src/delegation/Delegator.sol	

Description

In the `_trySetDistributionConfig` function, if there is a duplicate `IIPTokenStaking.StakingPeriod.FLEXIBLE` in the `_maturityConfigs` parameter. It will cause the previous configuration to be overwritten.

src/delegation/Delegator.sol:L94 - L117

```
function _trySetDistributionConfig(MaturityConfig[] calldata _maturityConfigs) internal
{
    // inverting this order has the effect of making the longest maturity
    // the first to be processed when unstaking
    // and the last to be processed when staking
    uint256 lastMaturityDuration = 0;
    uint256 bpsSum = 0;
    for (uint256 i = 0; i < _maturityConfigs.length; i++) {
        if (_maturityConfigs[i].maturity == IIPTokenStaking.StakingPeriod.FLEXIBLE) {

distributionConfig.configByMaturity[IIPTokenStaking.StakingPeriod.FLEXIBLE] =
_maturityConfigs[i];
            bpsSum += _maturityConfigs[i].bps;
        } else {
            ...
        }
    }
    require(bpsSum == 10000, "Delegator: Maturity bps must sum to 10000");
}
```

Recommendation

It is recommended to check that `_maturityConfigs` does not have duplicate `maturity`.

Status

The team has resolved this issue in commit 3a0b05f.

10. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - All	

Description

The `Delegator`, `StakePool`, `WithdrawalPool`, `VerioIP`, `InsurancePool`, `RewardPool` contract inherits from the `OwnableUpgradeable` contract. These contracts do 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 3a0b05f.

11. Report a wrong error message	
Severity: Informational	Category: Inconsistency
Target: - src/delegation/Delegator.sol	

Description

Report a wrong error message, the message should not be a non-zero address, but not null.

src/delegation/Delegator.sol:L551 - L557

```
function _addValidator(bytes memory _validator) internal {
    require(!validatorRegistry.delegations[_validator].exists, "Delegator: Validator
already exists");
    require(_validator.length == 65, "Delegator: Validator must be 65 byte SECP256K1
uncompressed public key");
    require(keccak256(_validator) != keccak256(bytes("")), "Delegator: Validator cannot
be the zero address");
    validatorRegistry.validators.push(_validator);
    validatorRegistry.delegations[_validator].exists = true;
}
```

Recommendation

It is recommended that this be changed to an appropriate error message.

Status

The team has resolved this issue in commit 3a0b05f.

12. 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 c3ff599 :

File	SHA-1 hash
src/delegation/Delegator.sol	a78834554ea192cd22ab738e240a245bfb67b3e1
src/insurance/InsurancePool.sol	34c6eb4221069b730180c1d610ac1ce4522f8b47
src/rewards/RewardPool.sol	d5f746d067e437e5fed7864410e2d878ccfa534d
src/staking/StakePool.sol	c9957323f16e101da7f0a97ffc94b96ad857aa10
src/token/VerioIP.sol	ab904972a933bc7ac559f8fd259b43b6c6de86ea
src/withdrawal/WithdrawalPool.sol	c3e4fabb225a7975e4502d69e5598572711e4aa5