SALUS SECURITY

AUG 2025

CODE SECURITY ASSESSMENT

PLUTO

Overview

Project Summary

- Name: Pluto - Staking Vault
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Pluto - Staking Vault
Version	v2
Type	Solidity
Dates	Sep 09 2025
Logs	Sep 04 2025, Sep 09 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	7
Total	9

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	Centralization risk	Medium	Business Logic	Acknowledged
2	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
3	Use of floating pragma	Informational	Configuration	Resolved
4	Incomplete Docstrings	Informational	Code Quality	Resolved
5	Non-explicit Imports Reduct Code Readability	Informational	Code Quality	Resolved
6	Mapping Declaration Lacks Named Parameters for Improved Readability	Informational	Code Quality	Resolved
7	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Resolved
8	Missing State Change Validation	Informational	Data Validation	Resolved
9	Multiple Optimizable State Reads	Informational	Gas Optimization	Resolved

2.2 Notable Findings

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

1. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/core/StakingVault.sol	

Description

In `StakingVault` contract, there exists one privileged role, `OWNER`. This role has the authority to execute some key functions such as `emergencyWithdraw`.

If these roles' private keys are compromised, an attacker could trigger these functions to steal all funds.

contracts/core/StakingVault.sol:L346-L366

```
function emergencyWithdraw(
    address token,
    address to,
    uint256 amount
) external onlyOwner {
    if (to == address(0)) revert ZeroAddress();
    if (amount == 0) revert AmountMustBeGreaterThanZero();

    if (token == address(0)) {
        if (address(this).balance < amount)
            revert InsufficientNativeBalance();
        payable(to).sendValue(amount);
    } else {
        IERC20 tokenContract = IERC20(token);
        if (tokenContract.balanceOf(address(this)) < amount)
            revert InsufficientTokenBalance();
        tokenContract.safeTransfer(to, amount);
    }

    emit EmergencyWithdraw(token, to, amount);
}
```

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. Implementation contract could be initialized by everyone

Severity: Low	Category: Business Logic
Target: - contracts/core/StakingVault.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 functions in StakingVault's implementation contract.

Recommendation

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

Status

This issue has been resolved by the team.

2.3 Informational Findings

3. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - contracts/core/StakingVault.sol	

Description

```
pragma solidity ^0.8.20;
```

The Rainlink uses a floating compiler version ^0.8.20.

Using a floating pragma ^0.8.20 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 resolved by the team.

4. Incomplete Docstrings

Severity: Informational	Category: Code Quality
Target: - contracts/core/StakingVault.sol	

Description

The `StakingVault` of incomplete docstrings were identified across the codebase. Several functions/events do not document all of their parameters or return values. This reduces readability and maintainability of the code, and increases the effort required by developers, auditors, or integrators to fully understand the contract logic. When such functions are part of the contract's public API, incomplete documentation may further hinder external usage and integration.

Recommendation

It is recommended to provide complete documentation for all public functions and events, including details on their parameters and return values. Following the Ethereum Natural Specification Format (NatSpec) is highly encouraged to ensure consistency, readability, and usability. Comprehensive documentation improves long-term maintainability and facilitates both auditing and integration processes.

Status

This issue has been resolved by the team.

5. Non-explicit Imports Reduce Code Readability

Severity: Informational	Category: Code Quality
Target: - contracts/core/StakingVault.sol	

Description

The contract `StakingVault.sol` uses wildcard or global-style import statements such as `import "@openzeppelin/contracts/token/ERC20/IERC20.sol";`, which introduce all symbols from the imported file into the current compilation unit. While functional, this approach can reduce code readability and make it unclear which specific contracts, interfaces, or types are actually being used in the file. Explicit `import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";` declarations are generally preferred, as they make dependencies explicit and reduce the potential for namespace conflicts or unintentional symbol usage.

Recommendation

Consider refactoring a complete import statement to use named import syntax.

Status

This issue has been resolved by the team.

6. Mapping Declaration Lacks Named Parameters for Improved Readability

Severity: Informational	Category: Code Quality
Target: - contracts/core/StakingVault.sol	

Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In the `StakingVault` contract, the mapping `strategies`, `stakeRecords` and `userStakeRecords` is declared without named parameters, making the purpose of the key and value less obvious upon inspection.

contracts/core/StakingVault.sol:L29-L33

```
mapping(uint256 => Strategy) public strategies;

mapping(uint256 => StakeRecord) public stakeRecords;

mapping(address => uint256[]) public userStakeRecords;
```

Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

Status

This issue has been resolved by the team.

7. Lack of Security Contact Information for Responsible Disclosure

Severity: Informational	Category: Configuration
Target: - contracts/core/StakingVault.sol	

Description

The contract `StakingVault` does not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

This issue has been resolved by the team.

8. Missing State Change Validation

Severity: Informational	Category: Data Validation
Target: - contracts/core/StakingVault.sol	

Description

`StakingVault` is identified where functions update state variables without verifying whether the new value differs from the existing one. This can result in unnecessary state writes and event emissions, increasing gas costs and potentially impacting contract performance.

contracts/core/StakingVault.sol:L321-L341

```
function setOperator(address newOperator) external onlyOwner {
    if (newOperator == address(0)) revert ZeroAddress();

    address previousOperator = operator;
    operator = newOperator;

    emit OperatorUpdated(previousOperator, newOperator);
}

function setTreasury(address newTreasury) external onlyOwner {
    if (newTreasury == address(0)) revert ZeroAddress();

    address previousTreasury = treasury;
    treasury = newTreasury;

    emit TreasuryUpdated(previousTreasury, newTreasury);
}
```

Recommendation

It is recommended to verify whether the new value differs from the existing value before updating state variables. Perform the update only when the value actually changes to save gas and reduce redundant event emissions.

Status

This issue has been resolved by the team.

9. Multiple Optimizable State Reads

Severity: Informational	Category: Gas Optimization
Target: - contracts/core/StakingVault.sol	

Description

Multiple instances of storage reads in the codebase can be optimized. For example, some functions repeatedly read the same storage variable (SLOAD), incurring unnecessary gas costs. Storage access is one of the most expensive operations on Ethereum; caching the value in a memory variable before use can significantly reduce gas consumption.

contracts/core/StakingVault.sol:L218-L222

```
function batchActivateStake(uint256[] calldata recordIds) external {
    for (uint256 i = 0; i < recordIds.length; i++) {
        _activateStake(recordIds[i]);
    }
}
```

Recommendation

It is recommended to cache frequently accessed storage variables in memory before performing further operations.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
StakingVault.sol	9b7b1100f618df069689d2bb86e2d7efcac2583b

And we audited the fixed version of the project:

File	SHA-1 hash
StakingVault.sol	899c5fd7c638cdae03230ce4163a2b70e85cbaa2