SALUS SECURITY

DEC 2025

# CODE SECURITY ASSESSMENT

PRIME VAULT

# Overview

## Project Summary

- Name: Prime Vault
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/prime-vaults/prime-vaults-contract
- Audit Range: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Prime Vault |
| **Version** | v3 |
| **Type** | Solidity |
| **Dates** | Dec 22 2025 |
| **Logs** | Dec 18 2025, Dec 22 2025; Dec 22 2025 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 3 |
| **Total Medium-Severity issues** | 5 |
| **Total Low-Severity issues** | 1 |
| **Total informational issues** | 1 |
| **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 | Delay withdrawal will be blocked | High | Business Logic | Resolved |
| 2 | Users' shares may be locked forever | High | Business Logic | Resolved |
| 3 | Users may lose their expected rewards | High | Business Logic | Resolved |
| 4 | Missing claim rewards in delayed withdraw | Medium | Business Logic | Resolved |
| 5 | Compound rewards will extend the lock time | Medium | Business Logic | Resolved |
| 6 | Platform fees may be rounded down to 0 | Medium | Business Logic | Resolved |
| 7 | Centralization risk | Medium | Centralization | Acknowledged |
| 8 | Missing validation for buffer helper | Low | Business Logic | Resolved |
| 9 | Redundant code | Informational | Configuration | Resolved |

## 2.2 Notable Findings

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

### 1. Delay withdrawal will be blocked

| Severity: High | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/DelayedWithdraw.sol | |

### Description

In the DelayedWithdraw contract, users need to request withdrawal at first, then complete the withdrawal when the withdrawal request reaches maturity.

The `minimumAssets` will be calculated via the recorded `exchangeRateAtTimeOfRequest`. Based on current implementation, the share price will keep decreasing, this will cause the withdrawal cannot be completed.

contracts/core/DelayedWithdraw.sol:L379-L384

```
function _completeWithdraw(address account, WithdrawRequest storage req) internal
returns (uint256 assetsOut) {
    ...
    assetsOut = shares.mulDivDown(req.exchangeRateAtTimeOfRequest, ONE_SHARE);

    req.shares = 0;
    req.sharesFee = 0;
    teller.bulkWithdraw(shares, assetsOut, account);
    ...
}
```

### Recommendation

It is recommended to refactor the minimum asset check logic.

### Status

The team has resolved this issue in commit 1559c1b.

## 2. Users' shares may be locked forever

| Severity: High | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/DelayedWithdraw.sol | |

## Description

When users deposit assets in prime vaults, the minted share will be locked for one period. The problem here is that malicious users can deposit tiny amounts of underlying assets to the victims to extend their share lock period.

contracts/core/DelayedWithdraw.sol:L379-L384

```
function _afterPublicDeposit(address user, uint256 depositAmount, uint256 shares,
uint256 currentShareLockPeriod) internal {
    // Only set share unlock time if share lock period is greater than 0.
    if (currentShareLockPeriod > 0) {
        beforeTransferData[user].shareUnlockTime = block.timestamp +
currentShareLockPeriod;
    }
}
```

## Recommendation

Only allowed operators can deposit on behalf of users.

## Status

The team has resolved this issue in commit 1559c1b.

## 3. Users may lose their expected rewards

| Severity: High | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/DelayedWithdraw.sol | |

## Description

In Distributor contract, the operators can notify rewards via function `nofityRewardAmount`. The rewardRate is the actual reward amount per second.

When users deposit/withdraw/transfer shares, the accrued rewards amount will be calculated. The `rewardPerToken` will be calculated with one 18 precision decimal.

The problem here is that 18 precision decimal is not enough in some cases.
For example:
1. Underlying asset is 18 decimal, e.g. DAI.

2. The reward token is 6 decimal, e.g. USDC/USDT.

3. Assume the reward is 100 USDC per week. The rewardRate is 165.

4. Assume that total supply is 50000 * 1e18(50000 DAI).

5. Below calculation will be round down to 0 if users deposit/withdraw/transfer shares in 5 minutes.

contracts/core/Distributor.sol:L372-L386

```
function rewardPerToken(address _rewardsToken) public view returns (uint256) {
    if (vault.totalSupply() == 0) {
        return rewardData[_rewardsToken].rewardPerTokenStored;
    }
    return
        rewardData[_rewardsToken].rewardPerTokenStored +
        ((lastTimeRewardApplicable(_rewardsToken) -
rewardData[_rewardsToken].lastUpdateTime) * rewardData[_rewardsToken].rewardRate * 1e18)
/
            vault.totalSupply();
   }
```

## Recommendation

Use one higher precision decimal to avoid the potential round down issue.

## Status

The team has resolved this issue in commit 1559c1b.

## 4. Missing claim rewards in delayed withdraw

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/DelayedWithdraw.sol | |

## Description

When users request withdrawal, the related share will be transferred from users to the DelayedWithdraw contract. The DelayedWithdraw contract will earn some rewards in the withdrawal delay period.

The problem here is that the DelayedWithdraw contract misses one interface to claim these rewards. If the treasury is disabled, the operator will transfer tokens to distributor contracts as rewards. In this case, some rewards will be locked in the Distributor contract.

contracts/core/DelayedWithdraw.sol:L242-L268

```
function requestWithdraw(uint96 shares, bool allowThirdPartyToComplete) external
requiresAuth nonReentrant {
    WithdrawRequest storage req = withdrawRequests[msg.sender];
        // Users should not request a new withdraw if they have an existing pending
withdraw.
    if (req.shares > 0) revert DelayedWithdraw__WithdrawPending();
    // transfer from msg.sender to this contract.
    boringVault.safeTransferFrom(msg.sender, address(this), shares);
}
```

## Recommendation

Add one interface to claim rewards and return rewards back to the team.

## Status

The team has resolved this issue in commit [b1c85dc](b1c85dc).

## 5. Compound rewards will extend the lock time

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/DelayedWithdraw.sol | |

## Description

In Distributor contract, users can compound claimed rewards to re-deposit the reward asset into the prime vault.

The problem here is that this compound operation will extend this account's lock period. If users allow third parties to compound, then malicious users can compound to extend the lock time forever.

contracts/core/DelayedWithdraw.sol:L242-L268

```
function compoundReward(address _account) external updateReward(_account) requiresAuth {
    ...
    if (amountToCompound > 0) {
        asset.safeApprove(address(vault), amountToCompound);
        uint256 shares = teller.deposit(amountToCompound, 0, _account);
        emit CompoundReward(_account, address(asset), amountToCompound, shares, fee);
    }
}
```

## Recommendation

Consider using bulk deposit or only allow third parties to compound this account.

## Status

The team has resolved this issue in commit 1559c1b.

## 6. Platform fees may be rounded down to 0

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/AccountantProvider.sol | |

## Description

When users deposit/withdraw assets, the exchange rate will be updated, and the related platform fee will be calculated and accrued into `feesOwedInBase`.

The problem here is that if users deposit/withdraw frequently, the platform fee in one short time period may be rounded down to 0.

For example:
1. Total assets is 2 WBTC(2e8). Assume the platform fee is 1% APR.

2. The platform fee per second will be 0.06 satoshi.

contracts/core/DelayedWithdraw.sol:L242-L268

```
function _calculatePlatformFee(
    uint128 totalSharesLastUpdate,
    uint64 lastUpdateTimestamp,
    uint16 platformFee,
    uint256 currentExchangeRate,
    uint256 currentTotalShares,
    uint64 currentTime
) internal view returns (uint256 platformFeesOwedInBase) {
    if (platformFee > 0) {
        uint256 timeDelta = currentTime - lastUpdateTimestamp;
        uint256 assets = shareSupplyToUse.mulDivDown(currentExchangeRate, ONE_SHARE);
        uint256 platformFeesAnnual = assets.mulDivDown(platformFee, 1e4);

        platformFeesOwedInBase = platformFeesAnnual.mulDivDown(timeDelta, 365 days);
    }
}
```

## Recommendation

If the last time slot does not accrue any platform fee, do not update the `lastUpdateTimestamp`.

## Status

The team has resolved this issue in commit 1559c1b.

## 7. Centralization risk

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/BoringVault.sol<br>- contracts/core/Distributor.sol | |

## Description

In the BoringVault/Ddistirbutor contract, there exists some privileged roles. The privileged roles have the authority to execute some key functions such as `notifyRewardAmount`, `manage`.

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

contracts/core/BoringVault.sol:L65-L68

```
function manage(address target, bytes calldata data, uint256 value) external
requiresAuth returns (bytes memory result) {
    result = target.functionCallWithValue(data, value);
}
```

contracts/core/Distributor.sol:L194-209

```
function notifyRewardAmount(address _rewardsToken, uint256 reward) external onlyOperator
updateReward(address(0)) {
    if (block.timestamp >= rewardData[_rewardsToken].periodFinish) {
        // calculate the reward rate.
        rewardData[_rewardsToken].rewardRate = reward /
rewardData[_rewardsToken].rewardsDuration;
    } else {
        // If the previous reward period is still active, add the new reward to the
leftover amount
        uint256 remaining = rewardData[_rewardsToken].periodFinish - block.timestamp;
        uint256 leftover = remaining * rewardData[_rewardsToken].rewardRate;
        rewardData[_rewardsToken].rewardRate = (reward + leftover) /
rewardData[_rewardsToken].rewardsDuration;
    }
}
```

## 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.

## 8. Missing validation for buffer helper

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/core/TellerWithBuffer.sol | |

## Description

In TellerWithBuffer, the protocol admin can set one buffer for deposit/withdrawal. This will help to deposit assets to the strategies or withdraw assets from the strategies.

The buffer helper can be disallowed by the function `disallowedBufferHelper`. The problem here is that the function `disallowedBufferHelper` misses the validator whether the disallowed buffer helper is used now. This will cause the system to use one disallowed buffer.

contracts/core/TellerWithBuffer.sol:L101-L108

```
function setWithdrawBufferHelper(IBufferHelper _withdrawBufferHelper) external
onlyProtocolAdmin {
    if (allowedBufferHelpers[_withdrawBufferHelper] || _withdrawBufferHelper ==
IBufferHelper(address(0))) {
        bufferHelpers.withdrawBufferHelper = _withdrawBufferHelper;
        emit WithdrawBufferHelperSet(_withdrawBufferHelper);
    } else {
        revert TellerWithBuffer__BufferHelperNotAllowed(_withdrawBufferHelper);
    }
}
function disallowBufferHelper(IBufferHelper _bufferHelper) external onlyProtocolAdmin {
    allowedBufferHelpers[_bufferHelper] = false;
    emit BufferHelperDisallowed(_bufferHelper);
}
```

## Recommendation

Add one validator in the function `disallowBufferHelper`.

## Status

The team has resolved this issue in commit 1559c1b.

# 2.3 Informational Findings

### 9. Redundant code

| Severity: Informational | Category: Redundancy |
| --- | --- |
| Target:<br>- contracts/core/Teller.sol | |

## Description

In the Tellor contract, function `_handlePermit` aims to support deposit with permit. The problem here is that the contract misses one public permit deposit function.

contracts/core/Teller.sol:L295-L301

```
function _handlePermit(uint256 depositAmount, uint256 deadline, uint8 v, bytes32 r,
bytes32 s) internal {
    try asset.permit(msg.sender, address(vault), depositAmount, deadline, v, r, s) {}
catch {
        if (asset.allowance(msg.sender, address(vault)) < depositAmount) {
            revert Teller__PermitFailedAndAllowanceTooLow();
        }
    }
}
```

## Recommendation

Remove unnecessary code.

## Status

The team has resolved this issue in commit 1559c1b.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit 9e41279:

| File | SHA-1 hash |
|---|---|
| ManagerWithMerkleVerification.sol | 36ef30344c512112df5e864ed7dd239027b42c96 |
| Teller.sol | 172ebf5cde3e221b735f6eb435521c703002e7a6 |
| Distributor.sol | 5f21d95e65c35eb6c7804fd1e6ae94a8ed765432 |
| BoringVault.sol | b9b62c69d3709f9e49be751e23783b7fbbfda705 |
| TellerWithBuffer.sol | a71af6fd98e831f48088a99dda6d24d1b4442bff |
| AccountantProviders.sol | 0733fe0ddc173732c090ecb54c12e81b152e3152 |
| DelayedWithdraw.sol | f7f2ddc83432d43d497e571b86c5368ef89d145c |
| PrimeAuth.sol | 4ffd3d5bf89957b2f7f6740e6b988aea66126b6e |
| RolesAuthority.sol | 3ab5360f0cc84a5af067af6d0faa7c4033be4474 |
| PrimeRBAC.sol | 62925dac8924b9eef11b49cc1ea8b9771595502a |