Vyral-fun - Escrow Contracts Audit Report

Audited By: zuhaibmohd

5th June 2025

Vyral-fun - Escrow Contracts Security Review Report

Introduction

A time-boxed security review of the **Vyral-fun - Escrow Contracts** contracts was conducted by **zuhaibmohd**, with a focus on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where I attempt to find as many vulnerabilities as possible.

> **Note:**
> We cannot guarantee 100% security after the review or even that the review will uncover any vulnerabilities.
> It is recommended to conduct **subsequent reviews**, run **bug bounty programs**, and enable **on-chain monitoring** to ensure long-term security.

About zuhaibmohd

zuhaibmohd is an independent smart contract security researcher.
Check out his previous work or connect on X @zuhaib44.

About Vyral-fun - Escrow Contracts

The contracts implement an upgradeable escrow system for distributing rewards, where `EscrowProxy` is the proxy contract that delegates calls to the implementation contract `EscrowLogic`. `EscrowLogic` allows users to create "yap requests" with a budget and fee in a specified ERC20 token (Kaito token), and administrators can then distribute rewards from these requests to designated winners.

Severity Classification

Severity	Likelihood →	High	Medium	Low
Impact ↓				
High	**Critical**	High	Medium	
Medium	High	Medium	Low	
Low	Medium	Low	Low	

- **Impact:** The potential technical, economic, and reputational damage from a successful exploit.
- **Likelihood:** The probability that a given vulnerability will be discovered and exploited.
- **Severity:** The overall criticality based on the above two factors.

> **Informational**
> Findings in this category are recommended improvements to enhance code structure, usability, and overall system effectiveness.

Security Assessment Summary

- **Initial review commit hash:** `5e7b1d56f405184c7a9d60b109d646a87db1f077`
- **Fixes review commit hash:** `39568844e7cce77d416268e6528f800add3c3379`

Scope

The following smart contracts were included in the scope of the audit:

- `contracts/EscrowProxy.sol`
- `contracts/EscrowLogic.sol`

Findings Summary

ID	Title	Severity	Status
H-01	Insufficient fee validation allows creation of requests with minimal fees	High	Fixed
M-01	Unprotected token address update can lead to stuck funds and DOS in active YAP requests	Medium	Fixed
M-02	Missing storage gap in upgradeable contract creates risk of storage collision	Medium	Fixed
L-01	Ownable: Does not implement 2-Step-Process for transferring ownership	Low	Fixed

ID	Title	Severity	Status
L-02	Use of non-upgradeable `ReentrancyGuard` in `EscrowLogic`	Low	Fixed
L-03	Unnecessary initialization of `s_feeBalance` in `EscrowLogic.sol`	Low	Fixed
L-04	Incorrect Storage Slot for Admin Role Not Compliant with ERC1967	Low	Fixed
I-01	Redundant `owner` check instead of using `onlyOwner` modifier in proxy contract	Informational	Fixed
I-02	Unused `ApprovedWinner` struct increases contract size	Informational	Fixed
I-03	Inconsistent admin role management and missing event emissions	Informational	Fixed

Detailed Findings

[H-01] - Insufficient fee validation allows creation of requests with minimal fees (1 wei)

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L82-L88

Impact

The `createRequest` function only validates that the fee is non-zero but doesn't enforce a minimum fee threshold. This allows users to create yap requests with fees as low as 1 wei, which:

1. Could lead to economic losses for the protocol as the fees are meant to sustain operations
2. Makes the fee mechanism effectively optional since users can bypass its intended purpose with minimal cost

Description

The `createRequest` function in `EscrowLogic.sol` implements a fee mechanism where users must pay a fee along with their request budget. However, the current validation only checks if the fee is non-zero. As a result, users can create requests with a fee of just 1 wei.

This effectively nullifies the purpose of the fee mechanism as users can minimize their fee payment to an insignificant amount while still utilizing the escrow service.

Proof of Concept

```
function test_createRequestWithOneWeiFee() public {
    vm.startPrank(user1);
    kaitoToken.approve(address(escrowProxy), TOTAL_REQUEST_BUDGET);
    escrowProxyAsLogic.createRequest(REQUEST_BUDGET, 1); //1 wei fee
    vm.stopPrank();

    uint256 feeBalance = escrowProxyAsLogic.getFeeBalance();
    assertEq(feeBalance, 1);
}
```

Recommendation

Implement one of these solutions:

1. Define a minimum fixed fee amount.
2. Implement a percentage-based fee based on value of budget.

[M-1] - Unprotected token address update can lead to stuck funds and

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L177-L182

Impact

- Funds can become permanently stuck in the contract

- Denial of service for all active YAP requests

Description

The `resetKaitoAddress` function allows the owner to change the underlying token address (`kaitoTokenAddress`) without any checks on existing active YAP requests. This can lead to a situation where funds become inaccessible or the contract becomes unusable for users with active requests.

For example:

1. A YAP request is created with WETH as the token (original `kaitoTokenAddress`)
2. Users deposit WETH for their YAP requests
3. Owner calls `resetKaitoAddress` to change to WBTC
4. All existing YAP requests still hold WETH, but the contract now tries to transfer WBTC
5. When `rewardYapWinners` is called for existing requests, it will attempt to transfer WBTC instead of the originally deposited WETH, causing transactions to fail

Recommendation

1. Add checks to prevent token address changes while active YAP requests exist.
2. Remove the ability to change the token address entirely if it's not strictly necessary for the protocol.DOS in active YAP requests

[M-02] - Missing storage gap in upgradeable contract creates risk of storage collision

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowProxy.sol#L15
https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L19

Impact

Future upgrades to the contract could lead to storage layout corruption, potentially resulting in:

- Incorrect state reading/writing
- Loss of funds
- Complete contract malfunction
- Inability to safely upgrade the contract

Description

The `EscrowLogic` contract is designed to be upgradeable but lacks a storage gap variable. Storage gaps are crucial in upgradeable contracts as they reserve storage slots for future contract versions, preventing storage collision issues during upgrades.

If a future version of the contract needs to add new state variables, it would append them directly after the existing ones. This could cause storage conflicts with child contracts or break storage layout compatibility in future upgrades.

There is storage gap defined in the `EscrowProxy.sol` which is redundant and can be removed.

Additionally, the contract is missing a constructor with `_disableInitializers()`, It prevents the implementation contract from being initialized directly, which is a security best practice for upgradeable contracts.

Recommendation

1. Add a storage gap at the end of the contract:

```
contract EscrowLogic is Initializable, OwnableUpgradeable, ReentrancyGuar
    // existing state variables...

    // Reserve storage slots for future contract upgrades
    uint256[50] private __gap;
}
```

2. Add a constructor that disables initializers:

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

[L-1] - Ownable: Does not implement 2-Step-Process for transferring ownership

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowProxy.sol#L4
https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L7

Description

The contracts `EscrowProxy.sol` and `EscrowLogic.sol` do not implement a 2-Step-Process for transferring ownership.
So ownership of the contract can easily be lost when making a mistake when transferring ownership.

Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous.
So Consider using the Ownable2Step contract from OZ https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol and https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol for upgradable version.

The way it works is there is a `transferOwnership` to transfer the ownership and `acceptOwnership` to accept the ownership.

Recommendation

Implement 2-Step-Process for transferring ownership via `Ownable2Step` and `Ownable2StepUpgradable`.

[L-2] Use of non-upgradeable `ReentrancyGuard` in `EscrowLogic`

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L4

Description

The `EscrowLogic` contract is designed to be upgradeable (inherits from `Initializable` and uses `OwnableUpgradeable`), but it implements the non-upgradeable version of OpenZeppelin's `ReentrancyGuard`. This creates a potential risk as the `_status` variable used by `ReentrancyGuard` is initialized in its constructor, which doesn't execute in the context of proxy contracts. This could leave the reentrancy protection in an undefined state, potentially compromising the contract's security against reentrancy attacks.

Recommendation

Replace the standard `ReentrancyGuard` with its upgradeable counterpart `ReentrancyGuardUpgradeable` - https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/utils/ReentrancyGuardUpgradeable.sol

[L-3] - Unnecessary initialization of `s_feeBalance` in `EscrowLogic.sol`

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L64

Description

The `initialize` function explicitly sets `s_feeBalance` to 0, which is unnecessary since storage variables are automatically initialized to zero in Solidity. In upgradeable contracts, explicitly initializing storage variables to their default values is not only redundant but dangerous.

If the logic contract is updated and initialization of `s_feeBalance` is not removed and `initialize` function were to be called again it would reset the fee balance to zero, potentially leading to loss of fee tracking and funds.

Note: This does not pose an immediate security risk. If you want to keep it no issues, make sure to remove it in future upgrades.

Recommendation

Remove the explicit initialization of `s_feeBalance` in the `initialize` function.

[L-4] - Incorrect Storage Slot for Admin Role Not Compliant with ERC1967

Where

https://github.com/Vyral-fun/escrow-contracts/blob/ae212af6c07c15c5843674a165394947a5c3e706/src/EscrowProxy.sol#L14

Description

The contract uses a custom storage slot for the proxy admin (`PROXY_ADMIN_SLOT`) that doesn't follow the ERC1967 standard. According to ERC1967, the admin storage slot should be computed as `bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1)` to avoid potential storage collisions. Using a different pattern which could lead to compatibility issues with tools and standards that expect ERC1967 compliance.

Recommendation

Replace the current `PROXY_ADMIN_SLOT` with the ERC1967-compliant storage slot.
Code Fix:

```
// keccak256("eip1967.proxy.admin") - 1 = 0xb53127684a568b3173ae13b9f
bytes32 internal constant PROXY_ADMIN_SLOT = 0xb53127684a568b3173ae13
```

[I-1] - Redundant `owner` check instead of using `onlyOwner` modifier in proxy contract

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowProxy.sol#L48-L50

Description

In `EscrowProxy.sol`, the `upgradeTo` function implements a manual owner check using `if (msg.sender != owner())` with a custom `NotOwner` error. Since the contract inherits from OpenZeppelin's `Ownable`, this is redundant. The `Ownable` contract already provides the `onlyOwner` modifier which serves the same purpose.

Recommendation

Replace the manual owner check with the `onlyOwner` modifier.

```
function upgradeTo(address _newImplementation) external onlyOwner {
    if (_newImplementation == address(0)) {
        revert ImplementationRequired();
    }
    // ... rest of the function
}
```

[I-2] - Unused `ApprovedWinner` struct increases contract size

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L27

Description

The `ApprovedWinner` struct is defined in the `EscrowLogic` contract but is never used throughout the codebase. This struct contains three fields: `winner` (address), `amount` (uint256), and `approvalTime` (uint256). Including unused code increases the contract size unnecessarily, which leads to higher deployment costs and reduces code clarity.

Recommendation

Remove the unused `ApprovedWinner` struct to optimize contract size and improve code clarity. If this struct is intended for future use, it should be added when needed through a contract upgrade.

[I-3] - Inconsistent admin role management and missing event emissions

Where

https://github.com/Vyral-fun/escrow-contracts/blob/5e7b1d56f405184c7a9d60b109d646a87db1f077/src/EscrowLogic.sol#L237-L258

Description

1. The contract defines an `onlyAdmin` modifier but never uses it in any function, It can be called as part of the `rewardYapWinners()` by removing the if check.
2. Critical admin management functions (`addAdmin` and `removeAdmin`) lack event emissions, making it difficult to track admin role changes off-chain.

This creates inconsistency in the contract's role management system and reduces transparency for external observers.

Recommendation

1. Either utilize the `onlyAdmin` modifier where appropriate or remove it if unnecessary.
2. Add events for admin role changes to maintain transparency.
