Vyral-fun – Escrow Contracts V2 Security Review Report

Audited By: zuhaibmohd

5th October 2025

Vyral-fun – Escrow Contracts V2 Security Review Report

Introduction

A time-boxed security review of the **Vyral-fun – Escrow Contracts V2** 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 (Twitter): @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 native or specified ERC20 token.
- Administrators can then distribute rewards from these requests to designated winners.

Severity Classification

Impact / Likelihood	High	Medium	Low
High Impact	Critical	High	Medium
Medium Impact	High	Medium	Low
Low Impact	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:** `f0300fe04b5a3e4b5ac7ebcb111d4a9f2886d177`
- **Fixes Review Commit Hash:** `053510640d6cb5ef13ba0ee62437855aa3409e71`

Scope

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

contracts/EscrowProxy.sol
contracts/EscrowLogic.sol

Findings Summary

ID	Title	Severity	Resolution
H-1	Fee Tracking Inconsistency in `topUpRequest` Function	**High**	Fixed
H-2	Fee Balance Desynchronization Leading to Fund Locking and Withdrawal Failures	**High**	Fixed
L-1	DoS Attack Vector in Reward Distribution Function	**Low**	Fixed
L-2	Unsafe Native ETH Transfer Using `.transfer()` Method	**Low**	Fixed
L-3	Missing ReentrancyGuard Initialization in EscrowLogic Contract	**Low**	Fixed
L-4	Incorrect Placement of Storage Gap Variable	**Low**	Fixed

---

H-1: Fee Tracking Inconsistency in `topUpRequest` Function

Description

An inconsistency exists in the fee tracking mechanism between the `createRequest` and `topUpRequest` functions in the `EscrowLogic` contract. While both functions correctly add fees to the global `s_feeBalances` mapping, the `topUpRequest` function fails to update the `YapRequest.fee` field in the struct, creating a significant accounting discrepancy.

In the `topUpRequest` function, additional fees are added to the global fee balance (`s_feeBalances[asset] += additionalFee`) but are not added to the individual request's fee field (`yapRequest.fee`).

Comparison with `createRequest`:

- `createRequest` correctly updates both `s_feeBalances[_asset] += _fee` and stores `fee: _fee` in the struct
- `topUpRequest` only updates `s_feeBalances[asset] += additionalFee` but omits `yapRequest.fee += additionalFee`

Code Evidence

```
// topUpRequest function (lines 215-216)
s_feeBalances[asset] += additionalFee;  //   Global fee balance updated
yapRequest.budget += additionalBudget;  //   Budget updated
//   MISSING: yapRequest.fee += additionalFee;
```

If additional fees were added via `topUpRequest`, affiliates can only claim rewards from the original fee amount, not the total fees collected. This means:

- Affiliates are denied legitimate rewards from additional fees
- The contract holds unclaimable fees that should be available for distribution
- Creates a permanent loss of funds for affiliate partners

Recommendation

Add the missing fee update in the `topUpRequest` function:

```
function topUpRequest(uint256 yapRequestId, uint256 additionalBudget, uint256 additionalFee)
    external
    payable
    nonReentrant
    returns (uint256, uint256, uint256, address, address)
{
    // ... existing validation code ...

    s_feeBalances[asset] += additionalFee;
    yapRequest.budget += additionalBudget;
    yapRequest.fee += additionalFee;  // ← ADD THIS LINE

    // ... rest of function ...
}
```

H-2: Fee Balance Desynchronization Leading to Fund Locking and Withdrawal Failures

Description

The EscrowLogic contract maintains two separate fee tracking mechanisms that become desynchronized during affiliate reward distributions:

1. **Global Fee Tracking**: `s_feeBalances[asset]` - tracks total accumulated fees per asset
2. **Per-Request Fee Tracking**: `yapRequest.fee` - tracks remaining fees for individual requests

The Critical Bug: In the `rewardAffiliateFromFees()` function (lines 334–344), when affiliate rewards are distributed:

- The function correctly deducts from `yapRequest.fee`
- **The function fails to deduct from `s_feeBalances[asset]`**

This creates a permanent desynchronization where `s_feeBalances[asset]` becomes inflated compared to the actual available fees.

Attack Scenario

1. **Setup Phase:**

   ```
   s_feeBalances[USDC] = 10,000 USDC
   YapRequest #1: fee = 5,000 USDC
   YapRequest #2: fee = 5,000 USDC
   ```

2. **Exploitation Phase:**

   ```
   rewardAffiliateFromFees():
   - YapRequest #1: reward = 3,000 USDC
   - YapRequest #2: reward = 2,000 USDC

   Result:
   - s_feeBalances[USDC] = 10,000 USDC (unchanged - BUG!)
   - YapRequest #1: fee = 2,000 USDC
   - YapRequest #2: fee = 3,000 USDC
   - Actual available fees = 5,000 USDC
   ```

3. **Impact Phase:**

   ```
   Owner withdraws 8,000 USDC
   Check: 8,000 <= 10,000   (incorrectly passes)
   Transfer fails   (insufficient actual balance)
   Fees become locked/unusable
   ```

Recommendation

Add missing fee balance deduction in `rewardAffiliateFromFees()`:

```
function rewardAffiliateFromFees(uint256 yapRequestId, address
affiliate, uint256 reward) external nonReentrant {
    // ... existing validation code ...

    s_yapRequests[yapRequestId].fee -= reward;
    s_feeBalances[yapRequest.asset] -= reward;  //   ADD THIS
LINE

    // ... rest of function ...
}
```

L-1: DoS Attack Vector in Reward Distribution Function

Description

The `rewardYapWinners` function in `EscrowLogic.sol` contains a Denial of Service (DoS) vulnerability that allows malicious actors to completely block reward distributions. If any winner address is a malicious contract that reverts on ETH transfers, the entire transaction fails, preventing all legitimate winners from receiving their rewards.

Recommendation

Evaluate implementing a pull-based reward claiming mechanism where winners actively claim their rewards rather than having them pushed to their addresses.

L-2: Unsafe Native ETH Transfer Using `.transfer()` Method

Description

The contract uses the deprecated `.transfer()` method for native ETH transfers in two critical functions:

1. `createRequest()`: `payable(msg.sender).transfer(msg.value - total);`
2. `topUpRequest()`: `payable(msg.sender).transfer(msg.value - total);`

This method has a fixed gas limit of 2300, insufficient for modern smart contracts, creating a DoS vulnerability for users who are contracts.

Recommendation

Replace `.transfer()` with the safer `.call{value: amount}("")` pattern.

```
(bool success,) = payable(msg.sender).call{value: msg.value -
total}("");
if (!success) {
    revert NativeTransferFailed();
}
```

L-3: Missing ReentrancyGuard Initialization in EscrowLogic Contract

Description

The `EscrowLogic` contract inherits from `ReentrancyGuardUpgradeable` but does not call `__ReentrancyGuard_init()` in its initializer. This may render the `nonReentrant` modifier ineffective and expose the contract to reentrancy.

Recommendation

Add `__ReentrancyGuard_init()` in the `initialize` function:

```
function initialize(address[] memory _admins, uint256
_currentYapRequestCount, address initialOwner)
    public
    initializer
{
    __Ownable2Step_init();
    __ReentrancyGuard_init(); // Add this line
    _transferOwnership(initialOwner);
}
```

L-4: Incorrect Placement of Storage Gap Variable

Description

The __gap variable is incorrectly placed in the middle of the contract’s storage layout instead of at the end. This can cause storage layout corruption during future upgrades.

Recommendation

Move the __gap variable to the end of the contract after all other variables and mappings.
