afiUSD Audit Report

Audited By: zuhaibmohd and 0xWeb3boy

20th August 2025

afiUSD Security Review Report

Introduction

A time-boxed security review of the **afiUSD** contracts was conducted by **zuhaibmohd** and **0xWeb3boy**, 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 0xWeb3boy

0xWeb3boy is an independent smart contract security researcher.
Check out his previous work or connect on X @0xWeb3boy.

About afiUSD

The **afiUSD** system is a sophisticated DeFi vault platform that allows users to deposit stablecoins and earn yield through automated asset management across multiple DeFi protocols.

Severity Classification

Severity / Likelihood	High	Medium	Low
High Impact	Critical	High	Medium

Severity / Likelihood	High	Medium	Low
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.

Scope

https://github.com/Artificial-Financial-Intelligence/afiUSD/tree/audit-fixes

Commit Hash - ebe77f9b7e88d7cde9a4845e4afd4eff7c5de508

Findings Summary

ID	Title	Severity	Status
L-01	Missing Validation & Initialization of `minDistributionInterval` Allows Bypass of Rate Limiting	Low	Fixed
L-02	Retroactive Application of `cooldownPeriod` to Existing Redeem Requests	Low	Acknowledged
L-03	Centralized Control of `distributeYield` by `REBALANCER_ROLE`	Low	Acknowledged

Detailed Findings

[L-01] - Missing Validation and Initialization of `minDistributionInterval` Allows Bypass of Distribution Rate Limiting**

Description

The `Yield` contract contains a vulnerability where the `minDistributionInterval` state variable is never initialized and lacks proper validation. This variable is intended to enforce a minimum time interval between yield distributions to prevent spam and abuse.

Issues Identified: 1. **Uninitialized Variable**: `minDistributionInterval` is declared but never set in the `initialize()` function, defaulting to `0` 2. **Missing Validation**: The `setMinDistributionInterval()` function lacks validation to prevent setting the interval to `0` 3. **Bypassed Protection**: The time restriction check in `distributeYield()` becomes ineffective when `minDistributionInterval` is `0`

Economic Impact

- **Fee Extraction Attacks**: Malicious rebalancers can call `distributeYield()` multiple times rapidly, potentially collecting fees on each call
- **Economic Manipulation**: The intended protection against rapid yield distribution is bypassed, potentially affecting vault economics

Attack Scenarios

1. **Accidental Exploitation**: Admin forgets to set `minDistributionInterval`, leaving it at `0`
2. **Malicious Admin**: Compromised admin account sets `minDistributionInterval` to `0`

Recommendations

1. Initialize in Constructor

```
function initialize(address _admin, address _rebalancer) public
initializer {
    __AccessControl_init();
    __UUPSUpgradeable_init();

    // Set a reasonable default value
```

```
    minDistributionInterval = 1 hours; // or appropriate default

    _grantRole(DEFAULT_ADMIN_ROLE, _admin);
    _grantRole(ADMIN_ROLE, _admin);
    _grantRole(REBALANCER_ROLE, _rebalancer);
}
```

2. Add Validation in Setter

```
function setMinDistributionInterval(uint256
_minDistributionInterval) external onlyRole(ADMIN_ROLE) {
    if (_minDistributionInterval == 0) revert
Errors.InvalidMinDistributionInterval();
    minDistributionInterval = _minDistributionInterval;
    emit MinDistributionIntervalSet(msg.sender,
_minDistributionInterval);
}
```

[L-02] - If `cooldownPeriod` is changed after the requestRedeem , user have to wait for long before they can withdraw

`requestRedeem` stores the redemption request of the user and initially if the `cooldownPeriod` is 1 day initially a user has to wait for just 1 day after the `requestRedeem` call , but if the `cooldownPeriod` is changed to 5 days after the request has been made , A user with retrospective request might not be able to withdraw/redeem because the change of `cooldownPeriod` applies to retrospective request as well when it should be applied to the prospective requests.

```
        if (block.timestamp < request.timestamp + cooldownPeriod)
revert Errors.CooldownNotFinished();
```

The above line will revert the withdraw/redeem.

[L-03] - distributeYield`is centralised and at the descretion of`REBALANCER_ROLE`

`distributeYield` function’s params are `amount` and `feeAmount` which means the amount of yield to be distributed is solely at the descretion of the `REBALANCER_ROLE`. The rebalancer role can choose to distribute only a portion of yield while charging arbitrary amount of fee unspecified to the user who deposited funds in the vault.