4th05

Aegis

Security Review Report

3 May 2025

Security Review Report

4th05

May 3, 2025

Table of Contents

Protocol Summary

Aegis.im is a Bitcoin-backed, delta-neutral stablecoin with real-time transparency, built-in funding-rate yield, and zero reliance on the fiat banking system. This contest focuses on the core YUSD contracts—mint/redeem, rewards distribution, and supporting libraries

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 try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Overview

Contest platform	Sherlock
LOC	923
Language	Solidity
Commit	b81dcb1b227073a485b642dcbbf719df6c8b81d4
Previous audits	Hacken audit

Scope

- AegisConfig.sol
- AegisMinting.sol
- AegisOracle.sol
- AegisRewards.sol
- YUSD.sol
- ClaimRewardsLib.sol
- OrderLib.sol

Issues found

Severity	Number of issues found
High	0
Medium	0
Low	1
Info	0

Findings

[L1] The `_deduplicateOrder` function may revert using an `order` nonce value greater than `type(uint64)max`

Summary

In `AegisMinting::_deduplicateOrder` a first time use of a nonce greater than `type(uint64).max` may cause the function to revert because of the truncation made in the explicit conversion of the `nonce` parameter from `uint256` to `uint64`.

Relevant Github link

https://github.com/sherlock-audit/2025-04-aegis-op-grant/blob/main/aegis-contracts/contracts/AegisMinting.sol#L63
L650

Root Cause

The `nonce` input parameter of the `_deduplicateOrder` is explicitly converted to an `uint64` variable type. However, the number that comes out from this explicit conversion could be already used by the user, causing the function to revert. This because of the truncation made during the conversion when the `nonce` is greater than `type(uint64).max`.

```

   function verifyNonce(address sender, uint256 nonce) public view returns (uint256, uint256, uint256) {
     if (nonce == 0) revert InvalidNonce();
@>   uint256 invalidatorSlot = uint64(nonce) >> 8;
     uint256 invalidatorBit = 1 << uint8(nonce);
     uint256 invalidator = _orderBitmaps[sender][invalidatorSlot];
     if (invalidator & invalidatorBit != 0) revert InvalidNonce();

     return (invalidatorSlot, invalidator, invalidatorBit);
   }

   /// @dev deduplication of user order
   function _deduplicateOrder(address sender, uint256 nonce) private {
@>   (uint256 invalidatorSlot, uint256 invalidator, uint256 invalidatorBit) = verifyNonce(sender, nonce);
     _orderBitmaps[sender][invalidatorSlot] = invalidator | invalidatorBit;
   }
```

Internal Pre-conditions

A specific nonce has already been used by the user.

For instance `nonce=1` has already been used.

Attack Path

The same user tries to either call `mint` or `depositIncome` functions using as nonce `18446744073709551617`. The nonce will be truncated during the conversions to both `uint64` and `uint8` type thus causing the function to revert.

`invalidatorSlot=0 invalidator=0 invalidatorBit=2`

Impact

Both `AegisMinting::mint` and `AegisMinting::depositIncome` called by the user will revert when they should not (because it would have been the first time in which the user provide the number `18446744073709551617` as nonce).

PoC Contract used:

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.26;
contract Aegis_Conversion {

mapping(address => mapping(uint256 => uint256)) private _orderBitmaps;
address sender = address(1);

function verifyNonce(uint256 _nonce) public view returns (uint256,
    uint256, uint256) {
    if (_nonce == 0) revert();
    uint256 invalidatorSlot = uint64(_nonce) >> 8;
    uint256 invalidatorBit = 1 << uint8(_nonce);
    uint256 invalidator = _orderBitmaps[sender][invalidatorSlot];
    if (invalidator & invalidatorBit != 0) revert();

    return (invalidatorSlot, invalidator, invalidatorBit);
  }

  /// @dev deduplication of user order
  function deduplicateOrder(uint256 _nonce) public {
    (uint256 invalidatorSlot, uint256 invalidator, uint256
        invalidatorBit) = verifyNonce(_nonce);
    _orderBitmaps[sender][invalidatorSlot] = invalidator |
        invalidatorBit;
  }
}
```

The test:

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.26;

import {Test, console} from "forge-std/Test.sol";
import {Aegis_Conversion} from "../src/Aegis_Conversion.sol";

contract Aegis_ConversionTest is Test {


uint256 nonce;
Aegis_Conversion Aegis_conversion;

function setUp() public {
Aegis_conversion = new Aegis_Conversion();
}

function test_nonceVerification() public {
//use the first 5 nonces
  for (nonce=1;nonce<6; nonce++)
 {
  Aegis_conversion.deduplicateOrder(nonce);
 }

//use the first 5 values greater than the type(uint64).max one
for (nonce=18446744073709551617; nonce<18446744073709551622; nonce++)
  {
  vm.expectRevert();
  Aegis_conversion.deduplicateOrder(nonce);

  }
  }
}
```

The result:

```
Ran 1 test for test/Aegis_Conversion.t.sol:Aegis_ConversionTest
[PASS] test_nonceVerification() (gas: 74333)
Traces:
  [74333] Aegis_ConversionTest::test_nonceVerification()
    #-- [25002] Aegis_Conversion::deduplicateOrder(1)
    #   #-- [Stop]
    #-- [1102] Aegis_Conversion::deduplicateOrder(2)
    #   #-- [Stop]
    #-- [1102] Aegis_Conversion::deduplicateOrder(3)
    #   #-- [Stop]
    #-- [1102] Aegis_Conversion::deduplicateOrder(4)
    #   #-- [Stop]
    #-- [1102] Aegis_Conversion::deduplicateOrder(5)
    #   #-- [Stop]
    #-- [0] VM::expectRevert(custom error f4844814:)
    #   #-- [Return]
    #-- [681] Aegis_Conversion::deduplicateOrder(18446744073709551617
        [1.844e19])
    #   #-- [Revert] EvmError: Revert
    #-- [0] VM::expectRevert(custom error f4844814:)
    #   #-- [Return]
    #-- [681] Aegis_Conversion::deduplicateOrder(18446744073709551618
        [1.844e19])
    #   #-- [Revert] EvmError: Revert
    #-- [0] VM::expectRevert(custom error f4844814:)
    #   #-- [Return]
    #-- [681] Aegis_Conversion::deduplicateOrder(18446744073709551619
        [1.844e19])
    #   #-- [Revert] EvmError: Revert
    #-- [0] VM::expectRevert(custom error f4844814:)
    #   #-- [Return]
    #-- [681] Aegis_Conversion::deduplicateOrder(18446744073709551620
        [1.844e19])
    #   #-- [Revert] EvmError: Revert
    #-- [0] VM::expectRevert(custom error f4844814:)
    #   #-- [Return]
    #-- [681] Aegis_Conversion::deduplicateOrder(18446744073709551621
        [1.844e19])
    #   #-- [Revert] EvmError: Revert
    ## [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 544.84ms
    (161.34ms CPU time)
```

Mitigation

```
  struct Order {
    OrderType orderType;
    address userWallet;
    address collateralAsset;
    uint256 collateralAmount;
    uint256 yusdAmount;
    uint256 slippageAdjustedAmount;
    uint256 expiry;
-     uint256 nonce;
+     uint64 nonce;
    bytes additionalData;
  }
```