# GammaSwap Labs - Core, Strategies and Periphery

## Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 19th, 2022 - January 6th, 2023**

Visit: **Halborn.com**

# DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Document Creation | 12/30/2022 | Luis Arroyo |
| 0.2 | Document Updates | 01/11/2023 | Pawel Bartunek |
| 0.3 | Document Updates | 01/16/2023 | Ataberk Yavuzer |
| 0.4 | Draft Review | 01/17/2023 | Roberto Reigada |
| 0.5 | Draft Review | 01/17/2023 | Piotr Cielas |
| 0.6 | Draft Review | 01/17/2023 | Gabi Urrutia |
| 1.0 | Remediation Plan | 01/27/2023 | Ataberk Yavuzer |
| 1.1 | Remediation Plan Edits | 02/15/2023 | Ataberk Yavuzer |
| 1.2 | Remediation Plan Review | 02/16/2023 | Piotr Cielas |
| 1.3 | Remediation Plan Review | 02/16/2023 | Gabi Urrutia |

# CONTACTS

| CONTACT | COMPANY | EMAIL |
| --- | --- | --- |
| Rob Behnke | Halborn | Rob.Behnke@halborn.com |
| Steven Walbroehl | Halborn | Steven.Walbroehl@halborn.com |
| Gabi Urrutia | Halborn | Gabi.Urrutia@halborn.com |
| Piotr Cielas | Halborn | Piotr.Cielas@halborn.com |
| Roberto Reigada | Halborn | Roberto.Reigada@halborn.com |
| Ataberk Yavuzer | Halborn | Ataberk.Yavuzer@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

GammaSwap allows users to provide liquidity into an existing AMM (e.g. Uniswap, Sushiswap, Balancer). At the same time, it allows other users to short LP tokens (borrow the Uniswap LP Token and retrieve the reserve tokens it represents) from users that have provided liquidity to an existing AMM though GammaSwap and therefore profit from the change in price. What is normally impermanent loss to liquidity providers becomes impermanent gains to liquidity shorters.

GammaSwap Labs engaged Halborn to conduct a security audit on their smart contracts beginning on December 19th, 2022 and ending on January 6th, 2023 . The security assessment was scoped to the smart contracts provided in the V1-core, V1-strategies and V1-periphery GitHub repositories. Commit hashes and further details can be found in the Scope section of this report.

## 1.2 AUDIT SUMMARY

The team at Halborn was provided 3 weeks for the engagement and assigned 3 full-time security engineer/engineers to audit the security of the smart contracts in scope. The security engineers are blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by GammaSwap Labs . The main ones are the following:

1. All functions with mathematical operations should have sanity checks against division by zero findings
2. Token decimals should be considered during price/fee calculations
3. Other chain specifications should be considered

## 1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Foundry)

### RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities.

The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

**RISK SCALE - LIKELIHOOD**

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

**RISK SCALE - IMPACT**

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

| | | | | |
|---|---|---|---|---|
| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |

**10** - CRITICAL
**9** - **8** - HIGH
**7** - **6** - MEDIUM
**5** - **4** - LOW
**3** - **1** - VERY LOW AND INFORMATIONAL

## 1.4 SCOPE

The security assessment was scoped to the following smart contracts on these repositories:

1. v1-core Repository: v1-core main branch
2. Commit ID: dbda72a563622afda678e8373f4a3e0cd091bb43
3. Smart contracts in scope:
   - contracts/pools/CPMMGammaPool.sol
   - contracts/libraries/LibStorage.sol
   - contracts/libraries/AddressCalculator.sol
   - contracts/storage/AppStorage.sol
   - contracts/GammaPoolFactory.sol
   - contracts/base/AbstractGammaPoolFactory.sol
   - contracts/base/GammaPoolERC4626.sol
   - contracts/base/GammaPool.sol
   - contracts/base/GammaPoolERC20.sol

EXECUTIVE OVERVIEW

1. v1-strategies Repository: v1-strategies main branch
2. Commit ID: f60285582b2b1178e36a8ebc8dddc6deaeb5c7f8
3. Smart contracts in scope:
    - contracts/strategies/cpmm/CPMMBaseLongStrategy.sol
    - contracts/strategies/cpmm/CPMMLongStrategy.sol
    - contracts/strategies/cpmm/CPMMShortStrategy.sol
    - contracts/strategies/cpmm/CPMMBaseStrategy.sol
    - contracts/strategies/cpmm/CPMMLiquidationStrategy.sol
    - contracts/strategies/base/LongStrategy.sol
    - contracts/strategies/base/BaseLongStrategy.sol
    - contracts/strategies/base/BaseStrategy.sol
    - contracts/strategies/base/LiquidationStrategy.sol
    - contracts/strategies/base/ShortStrategy.sol
    - contracts/strategies/base/ShortStrategyERC4626.sol
    - contracts/rates/LogDerivativeRateModel.sol
    - contracts/rates/LinearKinkedRateModel.sol

1. v1-periphery Repository: v1-periphery main branch
2. Commit ID: 1e28654c4ed123f543ef22927e26630929ca4680
3. Smart contracts in scope:
    - contracts/base/GammaPoolERC721.sol
    - contracts/base/Transfers.sol
    - contracts/PositionManager.sol
    - contracts/storage/AppStorage.sol
    - contracts/interfaces/ITransfers.sol
    - contracts/interfaces/ISendTokensCallback.sol
    - contracts/interfaces/IPositionManager.sol

The minor code differences mentioned in the following commits and implemented during the audit were also reviewed:

- v1-strategies changes::9a84053eb14de1ace6512a399956bdc638c469d7
- v1-periphery changes::6fef148010ded823650694d4f0dbb7836e73d9c0
- v1-core changes::b9a96e6fb76709aa89e30814eca0faa41a101baf

Out-of-Scope Changes:

- Balancer Contracts

# 2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 2 | 3 | 5 | 6 |

LIKELIHOOD

IMPACT

| | | | | |
|---|---|---|---|---|
| | | (HAL-02) | | |
| | | (HAL-03) | | |
| (HAL-10) | (HAL-06)<br>(HAL-07) | (HAL-04)<br>(HAL-05) | | (HAL-01) |
| (HAL-11) | (HAL-09) | (HAL-08) | | |
| (HAL-12)<br>(HAL-13)<br>(HAL-14)<br>(HAL-15)<br>(HAL-16) | | | | |

# EXECUTIVE OVERVIEW

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| HAL01 - INITIAL LP DEPOSIT IS IMPOSSIBLE DUE TO MISCALCULATION | High | SOLVED - 01/19/2023 |
| HAL02 - FIRST LIQUIDITY PROVIDER CAN LOSE FUNDS DUE TO ROUNDING ISSUE | High | SOLVED - 02/09/2023 |
| HAL03 - INCORRECT INVARIANT FACTOR CALCULATION MAY LEAD TO A LOSS OF ACCRUED FUNDS | Medium | SOLVED - 01/19/2023 |
| HAL04 - CALLING THE BATCHLIQUIDATIONS FUNCTION WITH TOKENID 0 ALWAYS REVERTS | Medium | NOT APPLICABLE |
| HAL05 - INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS | Medium | NOT APPLICABLE |
| HAL06 - CENTRALIZATION RISK | Low | RISK ACCEPTED |
| HAL07 - LOAN CALCULATIONS CAN BE MISLEADING FOR DIFFERENT CHAINS DUE TO HARDCODED VARIABLES | Low | SOLVED - 01/19/2023 |
| HAL08 - TOKEN SWAPPING CAN FAIL DUE TO DIVISION BY ZERO | Low | NOT APPLICABLE |
| HAL09 - LACK OF ZERO AMOUNT CHECK CAN LEAD TO DIVISION BY ZERO | Low | SOLVED - 01/19/2023 |
| HAL10 - MISSING ZERO ADDRESS CHECKS | Low | SOLVED - 01/23/2023 |
| HAL11 - MISSING UPPER/LOWER BOUND CHECKS | Informational | SOLVED - 01/23/2023 |
| HAL12 - FOR LOOP OPTIMIZATIONS | Informational | SOLVED - 01/23/2023 |
| HAL13 - THE IMMUTABLE KEYWORD COSTS LESS GAS FOR CONSTANT VARIABLES | Informational | SOLVED - 01/23/2023 |
| HAL14 - UNUSED IMPORTS | Informational | SOLVED - 01/23/2023 |
| HAL15 - MISSING NATSPEC DOCUMENTATION | Informational | SOLVED - 02/14/2023 |
| HAL16 - OPEN TODOs | Informational | SOLVED - 01/23/2023 |

# FINDINGS & TECH DETAILS

## 3.1 (HAL-01) INITIAL LP DEPOSIT IS IMPOSSIBLE DUE TO MISCALCULATION - HIGH

Description:

The _depositNoPull() function in the BaseStrategy contract does not work properly. The first deposit is reverted with the ZeroAmount() error.

In the _depositNoPull() function, the updateIndex() internal function is invoked with accFeeIndex, lastFeeIndex and lastCFMMFeeIndex call parameters. If the value of the s.BORROWED_INVARIANT variable is positive, then some LP tokens are minted to developers, since the protocol charges a handling fee. It has been observed however that the contract also tries to mint LP tokens to developers even if the s.BORROWED_INVARIANT is zero. In this case, the devShares variable returns zero since there are no deposits in the contract and s.totalSupply is equal to zero, which prevents users from depositing funds in the contract.

Proof of Concept:

**Listing 1: First Deposit Fails - PoC**

```
function test_depositNoPullPoC() public {
        vm.roll(16392000 + 1); // deployment block + 1
        _addLiquidityWithUniRouter(deployer, address(usdt), address(weth), 50 * 1e6, 10 * 1e18);

        vm.startPrank(deployer);

        usdt.transfer(user1, 5 * 1e6);
        weth.transfer(user1, 2 * 1e18);

        usdt.transfer(user2, 6 * 1e6);
        weth.transfer(user2, 10 * 1e18);


        uint256 loanId = positionManager.createLoan(1, address(cfmm_usdt_weth), deployer, type(uint).max);
        uint256 loanId2 = positionManager.createLoan(1, address(cfmm_usdt_weth), user2, type(uint).max);
        positionManager.transferFrom(deployer, user1, loanId);

        IPositionManager.DepositWithdrawParams memory depositData = IPositionManager.DepositWithdrawParams({
            protocolId: 1,
            cfmm: address(cfmm_usdt_weth),
            to: user1,
            lpTokens: cfmm_usdt_weth.balanceOf(deployer),
            deadline: 99999
        });

        cfmm_usdt_weth.approve(address(positionManager), cfmm_usdt_weth.balanceOf(deployer));

        positionManager.depositNoPull(depositData);

        vm.stopPrank();
    }
```

Screenshot:

```
├─ [101046] PositionManager::depositNoPull((1, 0xB9C9478b61e8e393797112BfafeA0dbc38EEbF50, 0xc12e5f6E8567AEF9b3180CDB7E63f37D58407e02, 22360679773997, 99999))
│   ├─ [618] GammaPoolFactory::getProtocol(1) [staticcall]
│   │   └─ ← CPMMGammaPool: [0x8FDd6cCf62E04dF2A1719E991b829903FE191061]
│   ├─ [20749] CFMM_USDT/WETH::transferFrom(Contract Owner: [0x7F82C1936932C39f3956fd377D042C639b71703D], 0x00350E66005aD02dEB304734A2EAd7C5665E9b1f, 22360679773997)
│   │   ├─ emit Transfer(from: Contract Owner: [0x7F82C1936932C39f3956fd377D042C639b71703D], to: 0x00350E66005aD02dEB304734A2EAd7C5665E9b1f, value: 22360679773997)
│   │   └─ ← true
│   ├─ [95836] 0x00350E66005aD02dEB304734A2EAd7C5665E9b1f::depositNoPull(Alice: [0xc12e5f6E8567AEF9b3180CDB7E63f37D58407e02])
│   │   ├─ [95668] CPMMGammaPool::depositNoPull(Alice: [0xc12e5f6E8567AEF9b3180CDB7E63f37D58407e02]) [delegatecall]
│   │   │   ├─ [91919] CPMMShortStrategy::_depositNoPull(Alice: [0xc12e5f6E8567AEF9b3180CDB7E63f37D58407e02]) [delegatecall]
│   │   │   │   ├─ [492] CFMM_USDT/WETH::balanceOf(0x00350E66005aD02dEB304734A2EAd7C5665E9b1f) [staticcall]
│   │   │   │   │   └─ ← 22360679773997
│   │   │   │   ├─ [517] CFMM_USDT/WETH::getReserves() [staticcall]
│   │   │   │   │   └─ ← 0x0000000000000000000000000000000000000000000000008ac7230489e80000000000000000000000000000000000000000000000000000000000000002faf0800000000000000000000000000000000000000000000000000000000
000000001
│   │   │   │   ├─ [432] CFMM_USDT/WETH::totalSupply() [staticcall]
│   │   │   │   │   └─ ← 22360679774997
│   │   │   │   ├─ [2356] GammaPoolFactory::feeInfo() [staticcall]
│   │   │   │   │   └─ ← FeeReceiver: [0x0eB22e32Bb66b357eaf6368CbeA17f4ea98884AB], 10000
│   │   │   │   └─ ← "ZeroAmount()"
│   │   │   └─ ← "EvmError: Revert"
│   │   └─ ← "EvmError: Revert"
│   └─ ← "EvmError: Revert"
└─ ← "EvmError: Revert"

Test result: FAILED. 0 passed; 1 failed; finished in 8.20ms
```

Code Location:

BaseStrategy

**Listing 2: BaseStrategy (Lines 94,95)**

```
90 function updateIndex() internal virtual returns(uint256 accFeeIndex, uint256 lastFeeIndex, uint256 lastCFMMFeeIndex) {
91     lastCFMMFeeIndex = updateCFMMIndex();
92     lastFeeIndex = updateFeeIndex(lastCFMMFeeIndex);
93     accFeeIndex = updateStore(lastFeeIndex);
94     if(s.BORROWED_INVARIANT >= 0) {
95         mintToDevs(lastFeeIndex, lastCFMMFeeIndex);
96     }
97 }
```

BaseStrategy

**Listing 3: BaseStrategy (Line 126)**

```
124 function _mint(address account, uint256 amount) internal virtual {
125     if(amount == 0) {
126         revert ZeroAmount();
127     }
128     s.totalSupply += amount;
129     s.balanceOf[account] += amount;
130     emit Transfer(address(0), account, amount);
131 }
```

Risk Level:

**Likelihood - 5**
**Impact - 3**

Recommendation:

Replace the greater than or equal to (>=) symbol with greater (>) to prevent calling the mintToDevs function for the first deposit.

Remediation Plan:

**SOLVED:** This finding was identified in a live code walkthrough jointly by the GammaSwap team and the Halborn team, and the existence of the finding was confirmed by the Halborn team.

The greater than or equal to (>=) relation was replaced with greater than (>) relation.

Commit ID: v1-strategies::6f6a7ba1f0fe8b9d7a7cb9b756ef5b3e6bfa55ab

## 3.2 (HAL-02) FIRST LIQUIDITY PROVIDER LOSES FUNDS DUE TO ROUNDING ISSUE - HIGH

Description:

During the initial asset deposit for ERC4626 Vaults, first liquidity provider can lose funds due to rounding issues.

The risk above was already explained in EIP4626 standard:

> Finally, ERC-4626 Vault implementers should be aware of the need for specific, opposing rounding directions across the different mutable and view methods, as it is considered most secure to favor the Vault itself during calculations over its users:

> If (1) it's calculating how many shares to issue to a user for a certain amount of the underlying tokens they provide or (2) it's determining the amount of the underlying tokens to transfer to them for returning a certain amount of shares, it should **round down**.

> If (1) it's calculating the amount of shares a user has to supply to receive a given amount of the underlying tokens or (2) it's calculating the amount of underlying tokens a user has to provide to receive a certain amount of shares, it should **round up**.

The current ShortStrategyERC4626 contract does not have any rounding validations for the security consideration above. Thus, the contract is vulnerable to the front-running attack.

In this case, any attacker can front-run the first deposit operation to claim more assets during the _redeem call.

Proof of Concept:

Listing 4: PoC Code - ERC4626 Vulnerability

```
function test_erc4626vulnerability() public {
        vm.roll(16392000 + 1); // deployment block + 1
        _addLiquidityWithUniRouter(deployer, address(usdt),
↳ address(weth), 20 * 1e6, 20 * 1e18);

        vm.startPrank(deployer);
        weth.transfer(user1, 10e18);
        usdt.transfer(user1, 10e6);
        vm.stopPrank();

        _addLiquidityWithUniRouter(user1, address(usdt), address(
↳ weth), 10 * 1e6, 10 * 1e18);

        vm.startPrank(user1);
        cfmm_usdt_weth.approve(address(univ2_usdt_weth_pool), 1);
        uint256 balance = cfmm_usdt_weth.balanceOf(user1);
        cfmm_usdt_weth.transfer(address(univ2_usdt_weth_pool),
↳ balance / 2);
        univ2_usdt_weth_pool.deposit(1, user1);

        vm.stopPrank();

        vm.startPrank(deployer);
        cfmm_usdt_weth.approve(address(univ2_usdt_weth_pool),
↳ balance);
        univ2_usdt_weth_pool.deposit(balance, deployer);
        vm.stopPrank();

        vm.roll(16392000 + 2);
        vm.prank(user1);

        univ2_usdt_weth_pool.redeem(1, user1, user1);
        uint256 balance_final = cfmm_usdt_weth.balanceOf(user1);

        console.log("Initial balance:", balance);
        console.log("Final balance:", balance_final);
        console.log("Profit:", balance_final - balance);
    }
```

Screenshot:

```
Running 1 test for test/MainTest.t.sol:GammaSwapMainTest
[PASS] test_erc4626() (gas: 692868)
Logs:
  Initial balance: 100000000000000
  Final balance: 124999999999999
  Profit: 24999999999999

Test result: ok. 1 passed; 0 failed; finished in 8.42ms
```

Code Location:

Listing 5: GammaPoolERC4626.sol (Line 43)

```
39 function convertToShares(uint256 assets) public view virtual
 ↳ returns (uint256) {
40     uint256 supply = totalSupply();
41     uint256 _totalAssets = totalAssets();
42
43     return supply == 0 || _totalAssets == 0 ? assets : (assets *
 ↳ supply) / _totalAssets;
44 }
```

Risk Level:

**Likelihood - 3**
**Impact - 5**

Recommendation:

The contract should do **INITIAL DEPOSIT** to any address to prevent this attack to occur. For example, some amounts should be deposited for zero address for the initial deposit.

Remediation Plan:

**SOLVED:** The GammaSwap team fixed the vulnerability by making the first deposit to the zero address. Furthermore, there is a minimum amount requirement to prevent rounding issues to occur.

Commit IDs:

- v1-strategies::5744f386f49d20fabb8760d16088a6f66631e335
- v1-core::59dab5059a8214aec4f92fac9feeb11d8eaf9f4f

## 3.3 (HAL-03) INCORRECT INVARIANT FACTOR CALCULATION MAY LEAD TO A LOSS OF ACCRUED FUNDS - MEDIUM

Description:

The invariant factor formula makes use of the number of decimals of the first token in token pair. The result of this calculation is further used in multiple places across the contracts. For example, when you borrow liquidity from a GammaSwap pool, the protocol calculates that factor to update some storage variables such as lastFeeIndex and BORROWED_INVARIANT . Instead of focusing on s.decimal[0] only, both decimals should be considered.

As a result, calculating this variable based on a wrong number of decimals affects borrowed/repaid liquidity and many storage variables in the protocol.

Proof of Concept:

**Listing 6: Invariant Factor PoC**

```
// replace the s.decimals[0] variable in the getInvariantFactor() with 1e18 and 1e6 to see difference.
function test_BorrowAndRepayLifeCycleTask01() public {
    vm.roll(16392000 + 1200);
    _addLiquidityWithUniRouter(deployer, address(usdt), address(weth), 10 * 1e6, 5 * 1e18);

    vm.startPrank(deployer);

    usdt.transfer(user1, 5 * 1e6);
    weth.transfer(user1, 2 * 1e18);

    usdt.transfer(user2, 6 * 1e6);
    weth.transfer(user2, 10 * 1e18);


```

```
    uint256 loanId = positionManager.createLoan(1, address(cfmm_usdt_weth), deployer, type(uint).max);
    uint256 loanId2 = positionManager.createLoan(1, address(cfmm_usdt_weth), user2, type(uint).max);
    positionManager.transferFrom(deployer, user1, loanId);

    IPositionManager.DepositWithdrawParams memory depositData = IPositionManager.DepositWithdrawParams({
        protocolId: 1,
        cfmm: address(cfmm_usdt_weth),
        to: user1,
        lpTokens: cfmm_usdt_weth.balanceOf(deployer),
        deadline: 99999
    });

    cfmm_usdt_weth.approve(address(positionManager), cfmm_usdt_weth.balanceOf(deployer));

    positionManager.depositNoPull(depositData);

    vm.stopPrank();

    _addLiquidityWithUniRouter(user1, address(usdt), address(weth), 1e6, 1e18);

    vm.startPrank(user1);

    uint256[] memory amountsDesired = new uint256[](2);
    amountsDesired[0] = 1e18;
    amountsDesired[1] = 1e6;

    IPositionManager.AddRemoveCollateralParams memory increaseCollData = IPositionManager.AddRemoveCollateralParams({
        protocolId: 1,
        cfmm: address(cfmm_usdt_weth),
        to: user1,
        tokenId: loanId,
        deadline: 99999,
        amounts: amountsDesired
    });

    usdt.approve(address(positionManager), amountsDesired[1]);
    weth.approve(address(positionManager), amountsDesired[0]);

```

```
    vm.roll(16392000 + 1252);
    positionManager.increaseCollateral(increaseCollData);

    vm.stopPrank();

    vm.startPrank(user2);

    increaseCollData.tokenId = loanId2;
    increaseCollData.to = user2;

    amountsDesired[0] = 1e18;
    amountsDesired[1] = 1e6;

    increaseCollData.amounts = amountsDesired;

    usdt.approve(address(positionManager), amountsDesired[1]);
    weth.approve(address(positionManager), amountsDesired[0]);

    vm.roll(16392000 + 1304);
    positionManager.increaseCollateral(increaseCollData);

    vm.stopPrank();

    vm.startPrank(user1);

    IPositionManager.BorrowLiquidityParams memory borrowLqtyData = IPositionManager.BorrowLiquidityParams({
        protocolId: 1,
        cfmm: address(cfmm_usdt_weth),
        to: user1,
        tokenId: loanId,
        lpTokens: uint256(1e12) * 800 / 1000,
        deadline: 99999,
        minBorrowed: new uint256[](2)
    });

    positionManager.borrowLiquidity(borrowLqtyData);

    vm.roll(16392000 + 1340);

    amountsDesired[0] = 1e18;
    amountsDesired[1] = 1e6;

    increaseCollData.amounts = amountsDesired;
```

```
    vm.stopPrank();

    vm.prank(user2);
    positionManager.decreaseCollateral(increaseCollData);


    IPositionManager.RepayLiquidityParams memory repayData = IPositionManager.RepayLiquidityParams({
        protocolId: 1,
        cfmm: address(cfmm_usdt_weth),
        to: user1,
        tokenId: loanId,
        liquidity: 1000000 / 2,
        deadline: 99999,
        minRepaid: new uint256[](2)
    });

    vm.stopPrank();

    vm.startPrank(deployer);
    usdt.transfer(address(cfmm_usdt_weth), 10 * 1e6);
    weth.transfer(address(cfmm_usdt_weth), 5 * 1e18);
    vm.stopPrank();
}
```

Screenshot:

```
➜  v1-core-foundry git:(main) ✗ forge test --match-contract GammaSwapMainTest --match-test test_BorrowAndRepayLifeCycleTask01 -vvv --silent

Running 1 test for test/MainTest.t.sol:GammaSwapMainTest
[PASS] test_BorrowAndRepayLifeCycleTask01() (gas: 1596976)
Logs:
  Borrow rate: 10000000000000000
  Borrowed Invariant: 0
  Borrow rate: 10000000000000000
  Borrowed Invariant: 0
  Borrow rate: 10000000000000000
  Borrowed Invariant: 800000172999

Test result: ok. 1 passed; 0 failed; finished in 9.32ms
➜  v1-core-foundry git:(main) ✗ forge test --match-contract GammaSwapMainTest --match-test test_BorrowAndRepayLifeCycleTask02 -vvv --silent

Running 1 test for test/MainTest.t.sol:GammaSwapMainTest
[PASS] test_BorrowAndRepayLifeCycleTask02() (gas: 1596996)
Logs:
  Borrow rate: 10000000000000000
  Borrowed Invariant: 0
  Borrow rate: 10000000000000000
  Borrowed Invariant: 0
  Borrow rate: 10518638573892587
  Borrowed Invariant: 800000179630

Test result: ok. 1 passed; 0 failed; finished in 9.06ms
➜  v1-core-foundry git:(main) ✗
```

s.decimals[0] was calculated as 6 for the first borrowing scenario

s.decimals[0] was calculated as 18 for the first second borrowing scenario

The loss will be 6631 wei for just one borrowing operation, this gap will be increase with every other operations in the protocol

Code Location:

BaseStrategy

Listing 7: BaseStrategy.sol (Line 33)

```
32 function getInvariantFactor() internal virtual override view returns(uint256) {
33     return 10 ** s.decimals[0];
34 }
```

AbstractRateModel

Listing 8: AbstractRateModel.sol (Line 11)

```
6 function calcUtilizationRate(uint256 lpInvariant, uint256 borrowedInvariant) internal virtual view returns(uint256) {
7     uint256 totalInvariant = lpInvariant + borrowedInvariant;
8     if(totalInvariant == 0)
9         return 0;
10
11     return borrowedInvariant * getInvariantFactor() / totalInvariant;
12 }
```

Risk Level:

**Likelihood - 3**
**Impact - 4**

Recommendation:

Consider changing the invariant factor formula to include the number of decimals of both tokens in a pair. If the invariant factor is less than 1e18, loss of precision may occur.

Remediation Plan:

**SOLVED:** This finding was identified in a code walkthrough jointly by the GammaSwap team and the Halborn team, and the existence of the finding was confirmed by the Halborn team.
The invariant factor is now calculated as 10**18.

Commit ID: v1-strategies::85864193924b7d1299dd99a27ded752c807ae2cb

# 3.4 (HAL-04) CALLING THE BATCHLIQUIDATIONS FUNCTION WITH TOKENID 0 ALWAYS REVERTS - MEDIUM

Description:

The `_batchLiquidations` function implemented in the `LiquidationStrategy` contract is designed to perform more than one liquidation in a go. The `tokenId > 0` check on the `payLoanAndRefundLiquidator` function assures that users can use `0` as `tokenId` for batch liquidation operation. However, it is not possible to use `0` as `tokenId` in the `_batchLiquidations` function.

The `_batchLiquidations` function makes an internal call to the `sumLiquidity` function. During the calculation of the `liquidity` variable, the execution is reverted with the **Division or modulo by 0** error since the `_loan.rateIndex` is also zero for `tokenId` 0.

Code Location:

**Listing 9: LiquidationStrategy.sol (Line 164)**

```
156 function sumLiquidity(uint256[] calldata tokenIds) internal virtual returns(uint256 liquidityTotal, uint256 collateralTotal, uint256 lpTokensPrincipalTotal, uint128[] memory tokensHeldTotal) {
157     address[] memory tokens = s.tokens;
158     uint128[] memory tokensHeld;
159     address cfmm = s.cfmm;
160     tokensHeldTotal = new uint128[](tokens.length);
161     (uint256 accFeeIndex,,) = updateIndex();
162     for(uint256 i = 0; i < tokenIds.length; i++) {
163         LibStorage.Loan storage _loan = s.loans[tokenIds[i]];
164         uint256 liquidity = uint128((_loan.liquidity * accFeeIndex) / _loan.rateIndex);
165         tokensHeld = _loan.tokensHeld;
166         lpTokensPrincipalTotal = lpTokensPrincipalTotal + _loan.lpTokens;
167         _loan.liquidity = 0;
```

```
        _loan.initLiquidity = 0;
        _loan.rateIndex = 0;
        _loan.lpTokens = 0;
        uint256 collateral = calcInvariant(cfmm, tokensHeld);
        canLiquidate(collateral, liquidity, 950);
        collateralTotal = collateralTotal + collateral;
        liquidityTotal = liquidityTotal + liquidity;
        for(uint256 j = 0; j < tokens.length; j++) {
            tokensHeldTotal[j] = tokensHeldTotal[j] + tokensHeld[j];
            _loan.tokensHeld[j] = 0;
        }
    }
}
```

Risk Level:

**Likelihood - 3**
**Impact - 3**

Recommendation:

It is recommended to add a sanity check to prevent division by zero in the sumLiquidity function. The contract should not continue dividing if denominator of division operations are zero.

Remediation Plan:

**NOT APPLICABLE:** The GammaSwap team confirmed that throwing the “Division or modulo by zero” error is the intended behavior.

# 3.5 (HAL-05) INCOMPATIBILITY WITH FEE-ON-TRANSFER OR DEFLATIONARY TOKENS - MEDIUM

## Description:

When depositing reserves, it was identified that the preDepositToCFMM() function assumes that the deposited amount of tokens is the same as sent in the parameter plus the balance before the deposit. This could block any deposit if a deflationary token is used, as the calculated amount could be lower than the deposited amount.

The comparison with the not equal (!=) sign does not work with any of Fee-On-Transfer tokens.

## Code Location:

ShortStrategy

**Listing 10: ShortStrategy.sol (Line 61)**

```
52     function preDepositToCFMM(uint256[] memory amounts, address to, bytes memory data) internal virtual {
53         address[] storage tokens = s.tokens;
54         uint256[] memory balances = new uint256[](tokens.length);
55         for(uint256 i = 0; i < tokens.length; i++) {
56             balances[i] = GammaSwapLibrary.balanceOf(IERC20(tokens[i]), to);
57         }
58         ISendTokensCallback(msg.sender).sendTokensCallback(tokens, amounts, to, data); // TODO: Risky. Should set sender to PosMgr
59         for(uint256 i = 0; i < tokens.length; i++) {
60             if(amounts[i] > 0) {
61                 if(balances[i] + amounts[i] != GammaSwapLibrary.balanceOf(IERC20(tokens[i]), to)) {
62                     revert WrongTokenBalance(tokens[i]);
63                 }
64             }
```

Risk Level:

**Likelihood - 3**
**Impact - 3**

Recommendation:

The last if statement of preDepositToCFMM() function should be replaced to include preBalance and postBalance variables and verification of the balance before/after the transfer.

Remediation Plan:

**NOT APPLICABLE:** The GammaSwap team confirmed this if block is added intentionally.
The GammaSwap team added some features to contracts to prevent potential problems with fee-on-transfer tokens.

Commit IDs:

- v1-core::59dab5059a8214aec4f92fac9feeb11d8eaf9f4f
- v1-periphery::43eab8008f052671b9f091b0834b7dcc0d15c9fc

# 3.6 (HAL-06) CENTRALIZATION RISK - LOW

## Description:

The extra condition in the if statement described in the **Code Location** section poses a centralization risk. The isRestricted function is designed to check if a protocolId is restricted. However, the _owner of the contract is excluded from this control. This increases the centralization of the contract.

## Code Location:

GammaPoolFactory

Listing 11: GammaPoolFactory.sol (Line 26)

```
25 function isRestricted(uint16 protocolId, address _owner) internal virtual view {
26     if(isProtocolRestricted[protocolId] == true && msg.sender != _owner) {
27         revert ProtocolRestricted();
28     }
29 }
```

## Risk Level:

**Likelihood - 2**
**Impact - 3**

## Recommendation:

Consider removing _owner != msg.sender control from the if block or decentralizing protocol governance.

Remediation Plan:

**RISK ACCEPTED:** The GammaSwap team accepted the risk, and they confirmed a multisig wallet will be the owner of the contract.

# 3.7 (HAL-07) LOAN CALCULATION CAN BE MISLEADING FOR DIFFERENT CHAINS DUE TO HARDCODED VARIABLES - LOW

Description:

The calcFeeIndex function in the BaseStrategy contract uses one year as a constant, with its value equal to the number of blocks per year (2252571). This assumes a block is mined every 14 seconds on average. However, average block time may vary across EVM-compatible chains. In this case, the results can be misleading or inaccurate for those other chains.

Code Location:

BaseStrategy

**Listing 12: BaseStrategy.sol (Lines 47,49)**

```
45 function calcFeeIndex(uint256 lastCFMMFeeIndex, uint256 borrowRate, uint256 lastBlockNum) internal virtual view returns(uint256) {
46     uint256 blockDiff = block.number - lastBlockNum;
47     uint256 adjBorrowRate = (blockDiff * borrowRate) / 2252571;//2252571 year block count
48     uint256 ONE = 10**18;
49     uint256 apy1k = ONE + (blockDiff * 10 * ONE) / 2252571;
50     return Math.min(apy1k, lastCFMMFeeIndex + adjBorrowRate);
51 }
```

Risk Level:

**Likelihood - 2**
**Impact - 3**

Recommendation:

Instead of hardcoding the number of blocks per year, consider adding a constructor argument to be able to deploy the protocol into other chains than Ethereum network.

Remediation Plan:

**SOLVED:** The GammaSwap team replaced the hardcoded variable with a constructor parameter. With this update, it is possible to change the number of blocks per year.

Commit ID: v1-strategies::1d2c35c5324209bb74133338bee991be4b4378b7

# 3.8 (HAL-08) TOKEN SWAPPING CAN FAIL DUE TO DIVISION BY ZERO - LOW

## Description:

There is an edge case scenario of token swapping operation which leads to the division by zero error. The beforeSwapTokens function in the CPMMBaseLongStrategy contract is a function which calculates the exact in and out amounts of the swap operation. If amountIn and reserveIn parameters of calcAmtOut() function are equal, then the denominator is equal to zero. As a result, the swapping operation fails in this case.

## Code Location:

Listing 13: CPMMBaseLongStrategy.sol (Line 109)

```
105 function calcAmtOut(uint256 amountIn, uint256 reserveOut, uint256 reserveIn) internal view returns (uint256) {
106     if(reserveOut == 0 || reserveIn == 0) {
107         revert ZeroReserves();
108     }
109     uint256 denominator = (reserveIn - amountIn) * tradingFee1;
110     return (reserveOut * amountIn * tradingFee2 / denominator) + 1;
111 }
```

## Risk Level:

**Likelihood - 3**
**Impact - 2**

## Recommendation:

Consider implementing an additional condition to prevent the denominator from being equal to zero.

Remediation Plan:

**NOT APPLICABLE:** The GammaSwap team confirmed the delta of reserveIn and amountIn will not be equal to 0 for Uniswap swaps. Therefore, reaching the “Division by zero” revert is impossible in this edge case.

# 3.9 (HAL-09) LACK OF ZERO AMOUNT CHECK MAY LEAD DIVISION BY ZERO - LOW

## Description:

In Solidity, transactions are reverted with the “division by zero” error message when a division by zero is attempted. In the BaseStrategy.sol contract, some possible divisions by 0 can be performed when calling calcCFMMFeeIndex() function.

This division operation should be preceded by sanity checks to prevent dividing a number by zero.

## Code Location:

Listing 14: BaseStrategy.sol (Line 39)

```
36 function calcCFMMFeeIndex(uint256 borrowedInvariant, uint256 lastCFMMInvariant, uint256 lastCFMMTotalSupply, uint256 prevCFMMInvariant, uint256 prevCFMMTotalSupply) internal virtual view returns(uint256) {
37         if(lastCFMMInvariant > 0 && lastCFMMTotalSupply > 0 && prevCFMMInvariant > 0 && prevCFMMTotalSupply > 0) {
38             uint256 prevInvariant = borrowedInvariant > prevCFMMInvariant ? borrowedInvariant : prevCFMMInvariant; // deleverage CFMM Yield
39             uint256 denominator = (prevInvariant * lastCFMMTotalSupply) / 10**18;
40             return ((lastCFMMInvariant * prevCFMMTotalSupply + lastCFMMTotalSupply * (prevInvariant - prevCFMMInvariant)) / denominator);
41         }
42         return 10**18;
43     }
```

Risk Level:

**Likelihood - 2**
**Impact - 2**

Recommendation:

It is recommended to add a sanity check to control whether the borrowed amount is zero or not.

Remediation Plan:

**SOLVED:** This finding was resolved by the GammaSwap team after changing the calculation to remove the possibility of dividing by 0.

v1-strategies::d5f3cedf864d7def66ea9e2ea72273704d1e4992

## 3.10 (HAL-10) MISSING ZERO ADDRESS CHECKS - LOW

Description:

Contracts in-scope are missing address validation in constructors and setter functions. It is possible to configure the address(0), which may cause issues during execution.

For instance, if address(0) is passed to setFeeToSetter function, it will not be possible to change this address in the future.

Code Location:

Following functions are not validating, that given address is different from zero:

- PositionManager.sol, #55 - payee
- GammaPoolFactory.sol, #95 - setFeeToSetter
- ShortStrategy.sol, #36 - to
- ShortStrategy.sol, #68 - to
- BaseLongStrategy.sol, #34 - to

Risk Level:

**Likelihood - 1**
**Impact - 3**

Recommendation:

Consider adding proper address validation when every state variable assignment done from user supplied input.

Remediation Plan:

**SOLVED:** This finding was solved by the GammaSwap team by requiring the payee and setFeeToSetter variables not to be equal to the zero address in the PositionManager and GammaPoolFactory contracts.

- PositionManager - payee, fix:v1-periphery:f273ae. . . bfc50
- GammaPoolFactory - setFeeToSetter, fix:v1-core:0a735. . . 5d623

This finding is not applicable to the (to) variables because the GammaSwap team will allow users to send their funds to address(0) in order to burn assets.

# 3.11 (HAL-11) MISSING UPPER/LOWER BOUND CHECKS - INFORMATIONAL

Description:

The _baseRate, _factor, _maxApy parameters of the LogDerivativeRateModel contract and _baseRate, _slope1, _slope2, _optimalUtilRate of the LinearKinkedRateModel contract are not checked to fall in the expected ranges. Some function calls might be therefore reverted due to possible arithmetic overflow issues, since no sanity checks are performed.

Code Location:

LogDerivativeRateModel

**Listing 15: LogDerivativeRateModel (Lines 15,16,17)**

```
14 constructor(uint64 _baseRate, uint80 _factor, uint80 _maxApy) {
15     baseRate = _baseRate;
16     factor = _factor;
17     maxApy = _maxApy;
18 }
```

LinearKinkedRateModel

**Listing 16: LinearKinkedRateModel (Lines 15,16,17,18)**

```
14 constructor(uint64 _baseRate, uint64 _optimalUtilRate, uint64
   ↳ _slope1, uint64 _slope2) {
15     baseRate = _baseRate;
16     optimalUtilRate = _optimalUtilRate;
17     slope1 = _slope1;
18     slope2 = _slope2;
19 }
```

Risk Level:

**Likelihood - 1**
**Impact - 2**

Recommendation:

Consider implementing lower and upper bounds for these variables.

Remediation Plan:

**SOLVED:** The GammaSwap team fixed this issue by adding sanity checks in the above contracts.

Commit ID: v1-strategies::db000bdedb35ce0c7c6e93f892f552c21be286d4

# 3.12 (HAL-12) FOR LOOP OPTIMIZATIONS - INFORMATIONAL

Description:

In for loops, the variable i is incremented using i++. It is known that in loops, using ++i costs less gas per iteration than i++.

It has also been detected that some uint256 variables are initialized to 0. These variables are already initialized to 0 by default, so uint256 i = 0 would reassign the 0 to i which wastes gas.

In addition, starting from pragma 0.8.0, adding unchecked keyword for arithmetical operations can reduce gas usage on contracts where underflow/underflow is unrealistic.

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

It is recommended to apply the following pattern for Solidity pragma version 0.8.0 and later. All for loops in contracts should be replaced with the following pattern.

**Listing 17: Possible Suggestion**

```
for (uint256 i; i < arrayLength; ) {
    . . .
    unchecked {
     ++i
    }
```

Remediation Plan:

**SOLVED:** The GammaSwap team solved this issue by optimizing the **for** loops in contracts.

Commit IDs:

- v1-strategies::aba9e566ec55c61d349575b0c445ccb91d5fab39
- v1-core::f642d03fe55aeead926ccc1c3eb36c31d52d454d
- v1-periphery::fbbf9befe45a0818b35c1916d7eb6a7e5bceb606

## 3.13 (HAL-13) THE IMMUTABLE KEYWORD COSTS LESS GAS FOR CONSTANT VARIABLES - INFORMATIONAL

### Description:

The _name and _symbol variables in the GammaPoolERC721 contract were designed to be set only once. The immutable keyword consumes less gas. It might be useful to declare these variables as immutable to optimize gas usage in the protocol.

### Code Location:

**Listing 18: GammaPoolERC721.sol (Lines 34,37)**

```
34 string private _name;
35
36 // Token symbol
37 string private _symbol;
```

### Risk Level:

**Likelihood - 1**
**Impact - 1**

### Recommendation:

It is suggested to immutable keyword for this constant variable to optimize gas usage.

### Remediation Plan:

**SOLVED:** This finding was solved by the GammaSwap team by moving the _name and _symbol variables to the PositionManager contract and declaring them constant.

Commit ID: v1-periphery::4e2b377de6888c1f7845cbd9485605ecfca053f1

## 3.14 (HAL-14) UNUSED IMPORTS - INFORMATIONAL

Description:

There are a few unused imports on the code base. These imports should be cleaned up from the code if they have no purpose. Clearing these imports will increase the readability of contracts.

Code Location:

- LiquidationStrategy.sol#L6
- LiquidationStrategy.sol#L7
- CPMMBaseStrategy.sol#L7

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Consider removing imports from the code.

Remediation Plan:

**SOLVED:** This finding was solved by the GammaSwap team by removing unused imports from the above contracts.

Commit ID: v1-strategies::8e9aa10f9a6ece1c4fb0b8b1b25a1f8e7644bcc0

# 3.15 (HAL-15) MISSING NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Consider adding documentation in Natspec format.

Remediation Plan:

**SOLVED:** This finding was fixed by the GammaSwap team by adding natspecs to all contracts.

v1-core::5dd28bb25f77bf7b3360b7420507e16688692f60

## 3.16 (HAL-16) OPEN TODOs - INFORMATIONAL

Description:

Open TO-DOs can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

References:

- LiquidationStrategy.sol, #75
- ShortStrategy.sol, #58

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Consider resolving the TO-DOs before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

**SOLVED:** This issue was solved by the GammaSwap team by closing open TODOs.

Commit IDs:

- v1-strategies::c1d8b61f3c956aba31eaa23febc50d451da4a7a8
- v1-strategies::6caa88c36e3750e816fbb49168156ae7c1c342ef
- v1-strategies::58dd22e5d56ea43c4cf4dcfbda9a746a58ed0f72

# AUTOMATED TESTING

# 4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

Core contracts:

```
FaucetERC20.allowedToWithdraw(address) (contracts/test/FaucetERC20.sol#31-38) uses a dangerous strict equality:
	- lastAccessTime[_address] == 0 (contracts/test/FaucetERC20.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in GammaPoolFactory.createPool(uint16,address,address[]) (contracts/GammaPoolFactory.sol#58-76):
	External calls:
	- IGammaPool(pool).initialize(cfmm,_tokens,_decimals) (contracts/GammaPoolFactory.sol#71)
	State variables written after the call(s):
	- getPool[key] = pool (contracts/GammaPoolFactory.sol#73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

GammaPoolERC4626.callStrategy(address,bytes).result_scope_0 (contracts/base/GammaPoolERC4626.sol#93) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

GammaPoolFactory.setIsProtocolRestricted(uint16,bool).isRestricted (contracts/GammaPoolFactory.sol#53) shadows:
	- GammaPoolFactory.isRestricted(uint16,address) (contracts/GammaPoolFactory.sol#25-29) (function)
GammaPool.initialize(address,address[],uint8[]).cfmm (contracts/base/GammaPool.sol#29) shadows:
	- GammaPool.cfmm() (contracts/base/GammaPool.sol#36-38) (function)
	- IGammaPool.cfmm() (contracts/interfaces/IGammaPool.sol#47) (function)
GammaPool.initialize(address,address[],uint8[]).tokens (contracts/base/GammaPool.sol#29) shadows:
	- GammaPool.tokens() (contracts/base/GammaPool.sol#40-42) (function)
	- IGammaPool.tokens() (contracts/interfaces/IGammaPool.sol#49) (function)
GammaPool.initialize(address,address[],uint8[]).decimals (contracts/base/GammaPool.sol#29) shadows:
	- GammaPoolERC20.decimals (contracts/base/GammaPoolERC20.sol#16) (state variable)
IGammaPool.initialize(address,address[],uint8[]).cfmm (contracts/interfaces/IGammaPool.sol#45) shadows:
	- IGammaPool.cfmm() (contracts/interfaces/IGammaPool.sol#47) (function)
IGammaPool.initialize(address,address[],uint8[]).tokens (contracts/interfaces/IGammaPool.sol#45) shadows:
	- IGammaPool.tokens() (contracts/interfaces/IGammaPool.sol#49) (function)
IGammaPool.validateCFMM(address[],address).tokens (contracts/interfaces/IGammaPool.sol#61) shadows:
	- IGammaPool.tokens() (contracts/interfaces/IGammaPool.sol#49) (function)
CPMMGammaPool.validateCFMM(address[],address).tokens (contracts/pools/CPMMGammaPool.sol#32) shadows:
	- GammaPool.tokens() (contracts/base/GammaPool.sol#40-42) (function)
	- IGammaPool.tokens() (contracts/interfaces/IGammaPool.sol#49) (function)
CPMMGammaPool.validateCFMM(address[],address).decimals (contracts/pools/CPMMGammaPool.sol#32) shadows:
	- GammaPoolERC20.decimals (contracts/base/GammaPoolERC20.sol#16) (state variable)
TestGammaPool.validateCFMM(address[],address).tokens (contracts/test/TestGammaPool.sol#12) shadows:
	- GammaPool.tokens() (contracts/base/GammaPool.sol#40-42) (function)
	- IGammaPool.tokens() (contracts/interfaces/IGammaPool.sol#49) (function)
TestGammaPool.validateCFMM(address[],address).decimals (contracts/test/TestGammaPool.sol#12) shadows:
	- GammaPoolERC20.decimals (contracts/base/GammaPoolERC20.sol#16) (state variable)
TestGammaPoolFactory.createPool(uint16,address,address[]).protocolId (contracts/test/TestGammaPoolFactory.sol#38) shadows:
	- TestGammaPoolFactory.protocolId (contracts/test/TestGammaPoolFactory.sol#12) (state variable)
TestGammaPoolFactory.createPool(uint16,address,address[]).cfmm (contracts/test/TestGammaPoolFactory.sol#38) shadows:
	- TestGammaPoolFactory.cfmm (contracts/test/TestGammaPoolFactory.sol#11) (state variable)
TestGammaPoolFactory.createPool(uint16,address,address[]).tokens (contracts/test/TestGammaPoolFactory.sol#38) shadows:
	- TestGammaPoolFactory.tokens (contracts/test/TestGammaPoolFactory.sol#13) (state variable)
TestGammaPoolFactory.isProtocolRestricted(uint16).protocolId (contracts/test/TestGammaPoolFactory.sol#41) shadows:
	- TestGammaPoolFactory.protocolId (contracts/test/TestGammaPoolFactory.sol#12) (state variable)
TestGammaPoolFactory.setIsProtocolRestricted(uint16,bool).protocolId (contracts/test/TestGammaPoolFactory.sol#45) shadows:
	- TestGammaPoolFactory.protocolId (contracts/test/TestGammaPoolFactory.sol#12) (state variable)
TestGammaPoolFactory.removeProtocol(uint16).protocolId (contracts/test/TestGammaPoolFactory.sol#52) shadows:
	- TestGammaPoolFactory.protocolId (contracts/test/TestGammaPoolFactory.sol#12) (state variable)
TestGammaPoolFactory.getProtocol(uint16).protocolId (contracts/test/TestGammaPoolFactory.sol#56) shadows:
	- TestGammaPoolFactory.protocolId (contracts/test/TestGammaPoolFactory.sol#12) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

Figure 1: Slither Result - core 1

```
GammaPoolFactory.setFee(uint16) (contracts/GammaPoolFactory.sol#83-86) should emit an event for:
	- fee = _fee (contracts/GammaPoolFactory.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

GammaPoolFactory.constructor(address)._feeToSetter (contracts/GammaPoolFactory.sol#15) lacks a zero-check on :
		- feeToSetter = _feeToSetter (contracts/GammaPoolFactory.sol#16)
		- feeTo = _feeToSetter (contracts/GammaPoolFactory.sol#17)
GammaPoolFactory.setFeeTo(address)._feeTo (contracts/GammaPoolFactory.sol#88) lacks a zero-check on :
		- feeTo = _feeTo (contracts/GammaPoolFactory.sol#90)
GammaPoolFactory.setFeeToSetter(address)._feeToSetter (contracts/GammaPoolFactory.sol#93) lacks a zero-check on :
		- feeToSetter = _feeToSetter (contracts/GammaPoolFactory.sol#95)
CPMMGammaPool.constructor(uint16,address,address,address,address,address,bytes32)._cfmmFactory (contracts/pools/CPMMGammaPool.sol#21) lacks a zero-check on :
		- cfmmFactory = _cfmmFactory (contracts/pools/CPMMGammaPool.sol#23)
TestGammaPoolFactory.constructor(address,uint16,address[])._cfmm (contracts/test/TestGammaPoolFactory.sol#17) lacks a zero-check on :
		- cfmm = _cfmm (contracts/test/TestGammaPoolFactory.sol#18)
TestGammaPoolFactory.addProtocol(address)._protocol (contracts/test/TestGammaPoolFactory.sol#48) lacks a zero-check on :
		- protocol = _protocol (contracts/test/TestGammaPoolFactory.sol#49)
TestERC20Strategy.totalAssets(address,uint256,uint256,uint256,uint256,uint256).cfmm (contracts/test/strategies/TestERC20Strategy.sol#31) lacks a zero-check on :
		- (success,data) = address(cfmm).staticcall(abi.encodeWithSelector(BALANCE_OF,msg.sender)) (contracts/test/strategies/TestERC20Strategy.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in GammaPoolFactory.createPool(uint16,address,address[]) (contracts/GammaPoolFactory.sol#58-76):
	External calls:
	- IGammaPool(pool).initialize(cfmm,_tokens,_decimals) (contracts/GammaPoolFactory.sol#71)
	State variables written after the call(s):
	- allPools.push(pool) (contracts/GammaPoolFactory.sol#74)
Reentrancy in TestGammaPoolFactory.createPool2() (contracts/test/TestGammaPoolFactory.sol#27-36):
	External calls:
	- IGammaPool(pool).initialize(cfmm,tokens,decimals) (contracts/test/TestGammaPoolFactory.sol#33)
	State variables written after the call(s):
	- getPool[key] = pool (contracts/test/TestGammaPoolFactory.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in GammaPoolFactory.createPool(uint16,address,address[]) (contracts/GammaPoolFactory.sol#58-76):
	External calls:
	- IGammaPool(pool).initialize(cfmm,_tokens,_decimals) (contracts/GammaPoolFactory.sol#71)
	Event emitted after the call(s):
	- PoolCreated(pool,cfmm,protocolId,implementation,allPools.length) (contracts/GammaPoolFactory.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

FaucetERC20.allowedToWithdraw(address) (contracts/test/FaucetERC20.sol#31-38) uses timestamp for comparisons
	Dangerous comparisons:
	- lastAccessTime[_address] == 0 (contracts/test/FaucetERC20.sol#32)
	- block.timestamp >= lastAccessTime[_address] (contracts/test/FaucetERC20.sol#34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AbstractGammaPoolFactory.cloneDeterministic(address,bytes32) (contracts/base/AbstractGammaPoolFactory.sol#42-56) uses assembly
	- INLINE ASM (contracts/base/AbstractGammaPoolFactory.sol#44-51)
CPMMGammaPool.isContract(address) (contracts/pools/CPMMGammaPool.sol#54-63) uses assembly
	- INLINE ASM (contracts/pools/CPMMGammaPool.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

GammaPoolFactory.isRestricted(uint16,address) (contracts/GammaPoolFactory.sol#25-29) compares to a boolean constant:
	-isProtocolRestricted[protocolId] == true && msg.sender != _owner (contracts/GammaPoolFactory.sol#26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

TestERC20Strategy._burn(address,uint256) (contracts/test/strategies/TestERC20Strategy.sol#74-79) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

Figure 2: Slither Result - core 2

```
Low level call in GammaPoolERC4626.callStrategy(address,bytes) (contracts/base/GammaPoolERC4626.sol#92-96):
	- (success,result) = strategy.delegatecall(data) (contracts/base/GammaPoolERC4626.sol#93)
Low level call in CPMMGammaPool.tokenDecimals(address) (contracts/pools/CPMMGammaPool.sol#47-52):
	- (success,data) = token.staticcall(abi.encodeWithSelector(DECIMALS)) (contracts/pools/CPMMGammaPool.sol#48-49)
Low level call in TestERC20Strategy._depositNoPull(address) (contracts/test/strategies/TestERC20Strategy.sol#12-17):
	- (success,data) = s.cfmm.staticcall(abi.encodeWithSelector(BALANCE_OF,msg.sender)) (contracts/test/strategies/TestERC20Strategy.sol#13)
Low level call in TestERC20Strategy.totalAssets(address,uint256,uint256,uint256,uint256,uint256) (contracts/test/strategies/TestERC20Strategy.sol#31-35):
	- (success,data) = address(cfmm).staticcall(abi.encodeWithSelector(BALANCE_OF,msg.sender)) (contracts/test/strategies/TestERC20Strategy.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

TestGammaPoolFactory.deployer (contracts/test/TestGammaPoolFactory.sol#10) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
. analyzed (27 contracts with 76 detectors), 42 result(s) found
```

Figure 3: Slither Result - core 3

Periphery contracts:

```
Transfers.send(address,address,address,uint256) (contracts/base/Transfers.sol#52-63) uses arbitrary from in transferFrom: TransferHelper.safeTransferFrom(IERC20(token),sender,to,amount) (contracts/base/Transfers.sol#61)
Reference: https://github.com/trailofbits/slither/wiki/Detector-Documentation#arbitrary-send-erc20

Transfers.send(address,address,address,uint256) (contracts/base/Transfers.sol#52-63) sends eth to arbitrary user
	Dangerous calls:
	- IWETH(WETH).deposit{value: amount}() (contracts/base/Transfers.sol#54)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes) (contracts/base/GammaPoolERC721.sol#420-443) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (contracts/base/GammaPoolERC721.sol#427-439)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

TestGammaPool.constructor(uint16,address,address,address,address)._factory (contracts/test/TestGammaPool.sol#24) lacks a zero-check on :
		- factory = _factory (contracts/test/TestGammaPool.sol#26)
TestGammaPool.constructor(uint16,address,address,address,address)._longStrategy (contracts/test/TestGammaPool.sol#24) lacks a zero-check on :
		- longStrategy = _longStrategy (contracts/test/TestGammaPool.sol#27)
TestGammaPool.constructor(uint16,address,address,address,address)._shortStrategy (contracts/test/TestGammaPool.sol#24) lacks a zero-check on :
		- shortStrategy = _shortStrategy (contracts/test/TestGammaPool.sol#28)
TestGammaPool.constructor(uint16,address,address,address,address)._liquidationStrategy (contracts/test/TestGammaPool.sol#24) lacks a zero-check on :
		- liquidationStrategy = _liquidationStrategy (contracts/test/TestGammaPool.sol#29)
TestGammaPool.initialize(address,address[],uint8[])._cfmm (contracts/test/TestGammaPool.sol#32) lacks a zero-check on :
		- cfmm = _cfmm (contracts/test/TestGammaPool.sol#33)
TestGammaPoolFactory.constructor(address)._feeToSetter (contracts/test/TestGammaPoolFactory.sol#17) lacks a zero-check on :
		- feeToSetter = _feeToSetter (contracts/test/TestGammaPoolFactory.sol#18)
		- feeTo = _feeToSetter (contracts/test/TestGammaPoolFactory.sol#19)
TestGammaPoolFactory.addProtocol(address)._protocol (contracts/test/TestGammaPoolFactory.sol#35) lacks a zero-check on :
		- implementation = _protocol (contracts/test/TestGammaPoolFactory.sol#36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes).retval (contracts/base/GammaPoolERC721.sol#427)' in GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes) (contracts/base/GammaPoolERC721.sol#420-443) potentially used before declaration: retval == IERC721Receiver.onERC721
Received.selector (contracts/base/GammaPoolERC721.sol#428)
Variable 'GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes).reason (contracts/base/GammaPoolERC721.sol#429)' in GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes) (contracts/base/GammaPoolERC721.sol#420-443) potentially used before declaration: reason.length == 0 (contracts/base
/GammaPoolERC721.sol#430)
Variable 'GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes).reason (contracts/base/GammaPoolERC721.sol#429)' in GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes) (contracts/base/GammaPoolERC721.sol#420-443) potentially used before declaration: revert(uint256,uint256)(32 + reaso
n,mload(uint256)(reason)) (contracts/base/GammaPoolERC721.sol#436)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```

Figure 4: Slither Result - periphery 1

```
Reentrancy in TestGammaPoolFactory.createPool(uint16,address,address[]) (contracts/test/TestGammaPoolFactory.sol#43-57):
	External calls:
	- IGammaPool(pool).initialize(cfmm,tokens,decimals) (contracts/test/TestGammaPoolFactory.sol#50)
	State variables written after the call(s):
	- allPools.push(pool) (contracts/test/TestGammaPoolFactory.sol#54)
	- getPool[key] = pool (contracts/test/TestGammaPoolFactory.sol#52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in PositionManager.borrowLiquidity(address,uint256,uint256,uint256[]) (contracts/PositionManager.sol#136-140):
	External calls:
	- amounts = IGammaPool(gammaPool).borrowLiquidity(tokenId,lpTokens) (contracts/PositionManager.sol#137)
	Event emitted after the call(s):
	- BorrowLiquidity(gammaPool,tokenId,amounts.length) (contracts/PositionManager.sol#139)
Reentrancy in PositionManager.borrowLiquidity(IPositionManager.BorrowLiquidityParams) (contracts/PositionManager.sol#167-171):
	External calls:
	- amounts = borrowLiquidity(gammaPool,params.tokenId,params.lpTokens,params.minBorrowed) (contracts/PositionManager.sol#169)
		- amounts = IGammaPool(gammaPool).borrowLiquidity(tokenId,lpTokens) (contracts/PositionManager.sol#137)
	Event emitted after the call(s):
	- LoanUpdate(tokenId,gammaPool,owner,tokensHeld,liquidity,lpTokens,initLiquidity,IGammaPool(gammaPool).getCFMMPrice()) (contracts/PositionManager.sol#103)
		- logLoan(gammaPool,params.tokenId,msg.sender) (contracts/PositionManager.sol#170)
Reentrancy in PositionManager.createLoan(address,address) (contracts/PositionManager.sol#124-128):
	External calls:
	- tokenId = IGammaPool(gammaPool).createLoan() (contracts/PositionManager.sol#125)
	- _safeMint(to,tokenId) (contracts/PositionManager.sol#126)
		- IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (contracts/base/GammaPoolERC721.sol#427-439)
	Event emitted after the call(s):
	- CreateLoan(gammaPool,to,tokenId) (contracts/PositionManager.sol#127)
	- Transfer(address(0),to,tokenId) (contracts/base/GammaPoolERC721.sol#316)
		- _safeMint(to,tokenId) (contracts/PositionManager.sol#126)
Reentrancy in PositionManager.createLoan(uint16,address,address,uint256) (contracts/PositionManager.sol#161-165):
	External calls:
	- tokenId = createLoan(gammaPool,to) (contracts/PositionManager.sol#163)
		- tokenId = IGammaPool(gammaPool).createLoan() (contracts/PositionManager.sol#125)
		- IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (contracts/base/GammaPoolERC721.sol#427-439)
	Event emitted after the call(s):
	- LoanUpdate(tokenId,gammaPool,owner,tokensHeld,liquidity,lpTokens,initLiquidity,IGammaPool(gammaPool).getCFMMPrice()) (contracts/PositionManager.sol#103)
		- logLoan(gammaPool,tokenId,to) (contracts/PositionManager.sol#164)
Reentrancy in TestGammaPoolFactory.createPool(uint16,address,address[]) (contracts/test/TestGammaPoolFactory.sol#43-57):
	External calls:
	- IGammaPool(pool).initialize(cfmm,tokens,decimals) (contracts/test/TestGammaPoolFactory.sol#50)
	Event emitted after the call(s):
	- PoolCreated(pool,cfmm,protocolId,address(0),allPools.length) (contracts/test/TestGammaPoolFactory.sol#56)
Reentrancy in PositionManager.decreaseCollateral(address,address,uint256,uint256[]) (contracts/PositionManager.sol#154-157):
	External calls:
	- tokensHeld = IGammaPool(gammaPool).decreaseCollateral(tokenId,amounts,to) (contracts/PositionManager.sol#155)
	Event emitted after the call(s):
	- DecreaseCollateral(gammaPool,tokenId,tokensHeld.length) (contracts/PositionManager.sol#156)
Reentrancy in PositionManager.decreaseCollateral(IPositionManager.AddRemoveCollateralParams) (contracts/PositionManager.sol#185-189):
	External calls:
	- tokensHeld = decreaseCollateral(gammaPool,params.to,params.tokenId,params.amounts) (contracts/PositionManager.sol#187)
		- tokensHeld = IGammaPool(gammaPool).decreaseCollateral(tokenId,amounts,to) (contracts/PositionManager.sol#155)
	Event emitted after the call(s):
	- LoanUpdate(tokenId,gammaPool,owner,tokensHeld,liquidity,lpTokens,initLiquidity,IGammaPool(gammaPool).getCFMMPrice()) (contracts/PositionManager.sol#103)
		- logLoan(gammaPool,params.tokenId,msg.sender) (contracts/PositionManager.sol#188)
Reentrancy in PositionManager.depositReserves(IPositionManager.DepositReservesParams) (contracts/PositionManager.sol#80-86):
	External calls:
	- (reserves,shares) = IGammaPool(gammaPool).depositReserves(params.to,params.amountsDesired,params.amountsMin,abi.encode(SendTokensCallbackData(params.cfmm,params.protocolId,msg.sender))) (contracts/PositionManager.sol#82-84)
	Event emitted after the call(s):
	- DepositReserve(gammaPool,reserves.length,shares) (contracts/PositionManager.sol#85)
Reentrancy in PositionManager.rebalanceCollateral(address,uint256,int256[],uint128[]) (contracts/PositionManager.sol#142-146):
	External calls:
	- tokensHeld = IGammaPool(gammaPool).rebalanceCollateral(tokenId,deltas) (contracts/PositionManager.sol#143)
	Event emitted after the call(s):
	- RebalanceCollateral(gammaPool,tokenId,tokensHeld.length) (contracts/PositionManager.sol#145)
```

Figure 5: Slither Result - periphery 2

```
Reentrancy in PositionManager.rebalanceCollateral(IPositionManager.RebalanceCollateralParams) (contracts/PositionManager.sol#191-195):
	External calls:
	- tokensHeld = rebalanceCollateral(gammaPool,params.tokenId,params.deltas,params.minCollateral) (contracts/PositionManager.sol#193)
		- tokensHeld = IGammaPool(gammaPool).rebalanceCollateral(tokenId,deltas) (contracts/PositionManager.sol#143)
	Event emitted after the call(s):
	- LoanUpdate(tokenId,gammaPool,owner,tokensHeld,liquidity,lpTokens,initLiquidity,IGammaPool(gammaPool).getCFMMPrice()) (contracts/PositionManager.sol#103)
		- logLoan(gammaPool,params.tokenId,msg.sender) (contracts/PositionManager.sol#194)
Reentrancy in PositionManager.repayLiquidity(address,uint256,uint256,uint256[]) (contracts/PositionManager.sol#148-152):
	External calls:
	- (liquidityPaid,amounts) = IGammaPool(gammaPool).repayLiquidity(tokenId,liquidity) (contracts/PositionManager.sol#149)
	Event emitted after the call(s):
	- RepayLiquidity(gammaPool,tokenId,liquidityPaid,amounts.length) (contracts/PositionManager.sol#151)
Reentrancy in PositionManager.repayLiquidity(IPositionManager.RepayLiquidityParams) (contracts/PositionManager.sol#173-177):
	External calls:
	- (liquidityPaid,amounts) = repayLiquidity(gammaPool,params.tokenId,params.liquidity,params.minRepaid) (contracts/PositionManager.sol#175)
		- (liquidityPaid,amounts) = IGammaPool(gammaPool).repayLiquidity(tokenId,liquidity) (contracts/PositionManager.sol#149)
	Event emitted after the call(s):
	- LoanUpdate(tokenId,gammaPool,owner,tokensHeld,liquidity,lpTokens,initLiquidity,IGammaPool(gammaPool).getCFMMPrice()) (contracts/PositionManager.sol#103)
		- logLoan(gammaPool,params.tokenId,msg.sender) (contracts/PositionManager.sol#176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

PositionManager.checkDeadline(uint256) (contracts/PositionManager.sol#36-40) uses timestamp for comparisons
	Dangerous comparisons:
	- deadline < block.timestamp (contracts/PositionManager.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

GammaPoolERC721._checkOnERC721Received(address,address,uint256,bytes) (contracts/base/GammaPoolERC721.sol#420-443) uses assembly
	- INLINE ASM (contracts/base/GammaPoolERC721.sol#435-437)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Redundant expression "a (contracts/test/ERC721ReceiverMock.sol#37)" inERC721ReceiverMock (contracts/test/ERC721ReceiverMock.sol#7-43)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Reentrancy in WETH9.withdraw(uint256) (contracts/test/WETH9.sol#23-28):
	External calls:
	- address(msg.sender).transfer(amount) (contracts/test/WETH9.sol#26)
	Event emitted after the call(s):
	- Withdrawal(msg.sender,amount) (contracts/test/WETH9.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

TestGammaPoolFactory.protocol (contracts/test/TestGammaPoolFactory.sol#15) is never used in TestGammaPoolFactory (contracts/test/TestGammaPoolFactory.sol#9-66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

TestGammaPoolFactory.protocol (contracts/test/TestGammaPoolFactory.sol#15) should be constant
WETH9.decimals (contracts/test/WETH9.sol#9) should be constant
WETH9.name (contracts/test/WETH9.sol#7) should be constant
WETH9.symbol (contracts/test/WETH9.sol#8) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
. analyzed (33 contracts with 76 detectors), 35 result(s) found
```

Figure 6: Slither Result - periphery 3

Strategies contracts:

```
TestPositionManager.send(address,address,address,uint256) (contracts/test/TestPositionManager.sol#49-57) uses arbitrary from in transferFrom: TransferHelper.safeTransferFrom(IERC20(token),sender,to,amount) (contracts/test/TestPositionManager.sol#55)
Reference: https://github.com/trailofbits/slither/wiki/Detector-Documentation#arbitrary-send-erc20

TestCFMM2.withdrawReserves(uint256) (contracts/test/TestCFMM2.sol#10-18) ignores return value by IERC20(token0).transfer(msg.sender,reserves[0]) (contracts/test/TestCFMM2.sol#15)
TestCFMM2.withdrawReserves(uint256) (contracts/test/TestCFMM2.sol#10-18) ignores return value by IERC20(token1).transfer(msg.sender,reserves[1]) (contracts/test/TestCFMM2.sol#16)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

TestLiquidationStrategy.incBorrowedInvariant(uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#130-136) performs a multiplication on the result of a division:
        - feeGrowth = borrowedInvariant * (10 ** 18) / s.BORROWED_INVARIANT (contracts/test/strategies/base/TestLiquidationStrategy.sol#132)
        - s.accFeeIndex = uint96(s.accFeeIndex * feeGrowth / (10 ** 18)) (contracts/test/strategies/base/TestLiquidationStrategy.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

BaseStrategy._mint(address,uint256) (contracts/strategies/base/BaseStrategy.sol#124-131) uses a dangerous strict equality:
        - amount == 0 (contracts/strategies/base/BaseStrategy.sol#125)
CPMMBaseLongStrategy.calcAmtIn(uint256,uint256,uint256) (contracts/strategies/cpmm/CPMMBaseLongStrategy.sol#95-102) uses a dangerous strict equality:
        - reserveOut == 0 || reserveIn == 0 (contracts/strategies/cpmm/CPMMBaseLongStrategy.sol#96)
CPMMBaseLongStrategy.calcAmtOut(uint256,uint256,uint256) (contracts/strategies/cpmm/CPMMBaseLongStrategy.sol#105-111) uses a dangerous strict equality:
        - reserveOut == 0 || reserveIn == 0 (contracts/strategies/cpmm/CPMMBaseLongStrategy.sol#106)
BaseStrategy.calcLPInvariant(uint256,uint256,uint256) (contracts/strategies/base/BaseStrategy.sol#75-77) uses a dangerous strict equality:
        - lastCFMMTotalSupply == 0 (contracts/strategies/base/BaseStrategy.sol#76)
BaseStrategy.calcLPTokenBorrowedPlusInterest(uint256,uint256,uint256) (contracts/strategies/base/BaseStrategy.sol#71-73) uses a dangerous strict equality:
        - lastCFMMInvariant == 0 (contracts/strategies/base/BaseStrategy.sol#72)
AbstractRateModel.calcUtilizationRate(uint256,uint256) (contracts/interfaces/rates/AbstractRateModel.sol#6-12) uses a dangerous strict equality:
        - totalInvariant == 0 (contracts/interfaces/rates/AbstractRateModel.sol#8)
LiquidationStrategy.payLoanAndRefundLiquidator(uint256,uint128[],uint256,uint256,bool) (contracts/strategies/base/LiquidationStrategy.sol#60-99) uses a dangerous strict equality:
        - lpDeposit == 0 (contracts/strategies/base/LiquidationStrategy.sol#71)
BaseLongStrategy.payLoanLiquidity(uint256,uint256,LibStorage.Loan) (contracts/strategies/base/BaseLongStrategy.sol#116-128) uses a dangerous strict equality:
        - remainingLiquidity == 0 (contracts/strategies/base/BaseLongStrategy.sol#125)
ShortStrategy._convertToAssets(uint256) (contracts/strategies/base/ShortStrategy.sol#183-187) uses a dangerous strict equality:
        - supply == 0 (contracts/strategies/base/ShortStrategy.sol#186)
ShortStrategy._convertToShares(uint256) (contracts/strategies/base/ShortStrategy.sol#177-181) uses a dangerous strict equality:
        - supply == 0 || _totalAssets == 0 (contracts/strategies/base/ShortStrategy.sol#180)
ShortStrategy._depositAssetsNoPull(address) (contracts/strategies/base/ShortStrategy.sol#36-46) uses a dangerous strict equality:
        - shares == 0 (contracts/strategies/base/ShortStrategy.sol#42)
ShortStrategy._withdrawAssetsNoPull(address,bool) (contracts/strategies/base/ShortStrategy.sol#83-97) uses a dangerous strict equality:
        - assets == 0 (contracts/strategies/base/ShortStrategy.sol#89)
TestBaseShortStrategy.calcBorrowRate(uint256,uint256) (contracts/test/strategies/base/TestBaseShortStrategy.sol#133-136) uses a dangerous strict equality:
        - totalInvariant == 0 (contracts/test/strategies/base/TestBaseShortStrategy.sol#135)
ShortStrategyERC4626._deposit(uint256,address) (contracts/strategies/base/ShortStrategyERC4626.sol#8-17) uses a dangerous strict equality:
        - shares == 0 (contracts/strategies/base/ShortStrategyERC4626.sol#13)
ShortStrategyERC4626._mint(uint256,address) (contracts/strategies/base/ShortStrategyERC4626.sol#19-28) uses a dangerous strict equality:
        - assets == 0 (contracts/strategies/base/ShortStrategyERC4626.sol#24)
ShortStrategyERC4626._redeem(uint256,address,address) (contracts/strategies/base/ShortStrategyERC4626.sol#44-54) uses a dangerous strict equality:
        - assets == 0 (contracts/strategies/base/ShortStrategyERC4626.sol#47)
ShortStrategyERC4626._withdraw(uint256,address,address) (contracts/strategies/base/ShortStrategyERC4626.sol#30-42) uses a dangerous strict equality:
        - shares == 0 (contracts/strategies/base/ShortStrategyERC4626.sol#38)
CPMMShortStrategy.calcDepositAmounts(uint256[],uint256[]) (contracts/strategies/cpmm/CPMMShortStrategy.sol#18-48) uses a dangerous strict equality:
        - reserve0 == 0 && reserve1 == 0 (contracts/strategies/cpmm/CPMMShortStrategy.sol#27)
CPMMShortStrategy.calcDepositAmounts(uint256[],uint256[]) (contracts/strategies/cpmm/CPMMShortStrategy.sol#18-48) uses a dangerous strict equality:
        - reserve0 == 0 || reserve1 == 0 (contracts/strategies/cpmm/CPMMShortStrategy.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

Figure 7: Slither Result - strategies 1

```
Reentrancy in ShortStrategyERC4626._depositAssetsFrom(address,address,uint256,uint256) (contracts/strategies/base/ShortStrategyERC4626.sol#56-64):
        External calls:
        - GammaSwapLibrary.safeTransferFrom(IERC20(s.cfmm),caller,address(this),assets) (contracts/strategies/base/ShortStrategyERC4626.sol#62)
        State variables written after the call(s):
        - _depositAssets(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategyERC4626.sol#63)
                - s.LP_TOKEN_BALANCE = lpTokenBalance (contracts/strategies/base/ShortStrategy.sol#110)
                - s.totalSupply += amount (contracts/strategies/base/BaseStrategy.sol#128)
                - s.balanceOf[account] += amount (contracts/strategies/base/BaseStrategy.sol#129)
                - s.LP_INVARIANT = lpInvariant (contracts/strategies/base/ShortStrategy.sol#111)
Reentrancy in ShortStrategy._depositReserves(address,uint256[],uint256[],bytes) (contracts/strategies/base/ShortStrategy.sol#68-77):
        External calls:
        - preDepositToCFMM(reserves,payee,data) (contracts/strategies/base/ShortStrategy.sol#72)
                - ISendTokensCallback(msg.sender).sendTokensCallback(tokens,amounts,to,data) (contracts/strategies/base/ShortStrategy.sol#58)
        State variables written after the call(s):
        - shares = _depositAssetsNoPull(to) (contracts/strategies/base/ShortStrategy.sol#76)
                - s.BORROWED_INVARIANT = uint128(accrueBorrowedInvariant(s.BORROWED_INVARIANT,lastFeeIndex)) (contracts/strategies/base/BaseStrategy.sol#80)
                - s.LP_TOKEN_BORROWED_PLUS_INTEREST = calcLPTokenBorrowedPlusInterest(s.BORROWED_INVARIANT,s.lastCFMMTotalSupply,s.lastCFMMInvariant) (contracts/strategies/base/BaseStrategy.sol#82)
                - s.LP_INVARIANT = uint128(calcLPInvariant(s.LP_TOKEN_BALANCE,s.lastCFMMInvariant,s.lastCFMMTotalSupply)) (contracts/strategies/base/BaseStrategy.sol#83)
                - s.LP_TOKEN_BALANCE = lpTokenBalance (contracts/strategies/base/ShortStrategy.sol#110)
                - s.totalSupply += amount (contracts/strategies/base/BaseStrategy.sol#128)
                - s.accFeeIndex = uint96(accFeeIndex) (contracts/strategies/base/BaseStrategy.sol#86)
                - s.balanceOf[account] += amount (contracts/strategies/base/BaseStrategy.sol#129)
                - s.LP_INVARIANT = lpInvariant (contracts/strategies/base/ShortStrategy.sol#111)
                - s.lastCFMMInvariant = uint128(lastCFMMInvariant) (contracts/strategies/base/BaseStrategy.sol#59)
                - s.LAST_BLOCK_NUMBER = uint48(block.number) (contracts/strategies/base/BaseStrategy.sol#87)
                - s.lastCFMMTotalSupply = lastCFMMTotalSupply (contracts/strategies/base/BaseStrategy.sol#60)
Reentrancy in ShortStrategy._withdrawAssets(address,address,address,uint256,uint256,bool) (contracts/strategies/base/ShortStrategy.sol#119-157):
        External calls:
        - GammaSwapLibrary.safeTransfer(IERC20(cfmm),receiver,assets) (contracts/strategies/base/ShortStrategy.sol#147)
        State variables written after the call(s):
        - s.LP_INVARIANT = lpInvariant (contracts/strategies/base/ShortStrategy.sol#151)
        - s.LP_TOKEN_BALANCE = lpTokenBalance (contracts/strategies/base/ShortStrategy.sol#152)
Reentrancy in TestBaseShortStrategy.borrowLPTokens(uint256) (contracts/test/strategies/base/TestBaseShortStrategy.sol#94-100):
        External calls:
        - TestCFMM(s.cfmm).burn(lpTokens,address(this)) (contracts/test/strategies/base/TestBaseShortStrategy.sol#96)
        State variables written after the call(s):
        - s.BORROWED_INVARIANT += uint128(TestCFMM(s.cfmm).convertSharesToInvariant(lpTokens)) (contracts/test/strategies/base/TestBaseShortStrategy.sol#97)
        - s.LP_TOKEN_BORROWED += lpTokens (contracts/test/strategies/base/TestBaseShortStrategy.sol#98)
        - s.LP_TOKEN_BALANCE = IERC20(s.cfmm).balanceOf(address(this)) (contracts/test/strategies/base/TestBaseShortStrategy.sol#99)
Reentrancy in TestLiquidationStrategy.createLoan(uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#63-78):
        External calls:
        - TestCFMM2(s.cfmm).withdrawReserves(lpTokens) (contracts/test/strategies/base/TestLiquidationStrategy.sol#68)
        State variables written after the call(s):
        - tokensHeld = updateCollateral(_loan) (contracts/test/strategies/base/TestLiquidationStrategy.sol#70)
                - s.TOKEN_BALANCE = tokenBalance (contracts/strategies/base/BaseLongStrategy.sol#166)
        - liquidity = openLoan(_loan,lpTokens) (contracts/test/strategies/base/TestLiquidationStrategy.sol#72)
                - s.BORROWED_INVARIANT = uint128(borrowedInvariant) (contracts/strategies/base/BaseLongStrategy.sol#55)
                - s.LP_TOKEN_BORROWED = s.LP_TOKEN_BORROWED + lpTokens (contracts/strategies/base/BaseLongStrategy.sol#56)
                - s.LP_TOKEN_BALANCE = lpTokenBalance (contracts/strategies/base/BaseLongStrategy.sol#59)
                - s.LP_INVARIANT = uint128(lpInvariant) (contracts/strategies/base/BaseLongStrategy.sol#63)
                - s.lastCFMMInvariant = uint128(lastCFMMInvariant) (contracts/strategies/base/BaseLongStrategy.sol#64)
                - s.lastCFMMTotalSupply = lastCFMMTotalSupply (contracts/strategies/base/BaseLongStrategy.sol#65)
                - s.LP_TOKEN_BORROWED_PLUS_INTEREST = s.LP_TOKEN_BORROWED_PLUS_INTEREST + lpTokensPlusOrigFee (contracts/strategies/base/BaseLongStrategy.sol#66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

BaseStrategy.updateIndex() (contracts/strategies/base/BaseStrategy.sol#90-97) contains a tautology or contradiction:
        - s.BORROWED_INVARIANT >= 0 (contracts/strategies/base/BaseStrategy.sol#94)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction

TestLiquidationStrategy.createLoan(uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#63-78) ignores return value by TestCFMM2(s.cfmm).withdrawReserves(lpTokens) (contracts/test/strategies/base/TestLiquidationStrategy.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Figure 8: Slither Result - strategies 2

```
TestLongStrategy.initialize(address,uint16,address[],uint8[]).tokens (contracts/test/strategies/base/TestLongStrategy.sol#23) shadows:
	- TestLongStrategy.tokens() (contracts/test/strategies/base/TestLongStrategy.sol#28-30) (function)
TestCPMMLongStrategy.initialize(address,address[],uint8[]).cfmm (contracts/test/strategies/cpmm/TestCPMMLongStrategy.sol#19) shadows:
	- TestCPMMLongStrategy.cfmm() (contracts/test/strategies/cpmm/TestCPMMLongStrategy.sol#23-25) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

TestPositionManager.constructor(address,address,uint16)._pool (contracts/test/TestPositionManager.sol#24) lacks a zero-check on :
		- pool = _pool (contracts/test/TestPositionManager.sol#25)
TestPositionManager.constructor(address,address,uint16)._cfmm (contracts/test/TestPositionManager.sol#24) lacks a zero-check on :
		- cfmm = _cfmm (contracts/test/TestPositionManager.sol#26)
TestBaseStrategy.constructor(address,uint16).factory (contracts/test/strategies/base/TestBaseStrategy.sol#21) lacks a zero-check on :
		- _factory = factory (contracts/test/strategies/base/TestBaseStrategy.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in ShortStrategyERC4626._depositAssetsFrom(address,address,uint256,uint256) (contracts/strategies/base/ShortStrategyERC4626.sol#56-64):
	External calls:
	- GammaSwapLibrary.safeTransferFrom(IERC20(s.cfmm),caller,address(this),assets) (contracts/strategies/base/ShortStrategyERC4626.sol#62)
	Event emitted after the call(s):
	- Deposit(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategy.sol#112)
		- _depositAssets(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategyERC4626.sol#63)
	- PoolUpdated(lpTokenBalance,s.LP_TOKEN_BORROWED,s.LAST_BLOCK_NUMBER,s.accFeeIndex,s.LP_TOKEN_BORROWED_PLUS_INTEREST,lpInvariant,s.BORROWED_INVARIANT) (contracts/strategies/base/ShortStrategy.sol#113-114)
		- _depositAssets(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategyERC4626.sol#63)
	- Transfer(address(0),account,amount) (contracts/strategies/base/BaseStrategy.sol#130)
		- _depositAssets(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategyERC4626.sol#63)
Reentrancy in ShortStrategy._depositReserves(address,uint256[],uint256[],bytes) (contracts/strategies/base/ShortStrategy.sol#68-77):
	External calls:
	- preDepositToCFMM(reserves,payee,data) (contracts/strategies/base/ShortStrategy.sol#72)
		- ISendTokensCallback(msg.sender).sendTokensCallback(tokens,amounts,to,data) (contracts/strategies/base/ShortStrategy.sol#58)
	Event emitted after the call(s):
	- Deposit(caller,receiver,assets,shares) (contracts/strategies/base/ShortStrategy.sol#112)
		- shares = _depositAssetsNoPull(to) (contracts/strategies/base/ShortStrategy.sol#76)
	- PoolUpdated(lpTokenBalance,s.LP_TOKEN_BORROWED,s.LAST_BLOCK_NUMBER,s.accFeeIndex,s.LP_TOKEN_BORROWED_PLUS_INTEREST,lpInvariant,s.BORROWED_INVARIANT) (contracts/strategies/base/ShortStrategy.sol#113-114)
		- shares = _depositAssetsNoPull(to) (contracts/strategies/base/ShortStrategy.sol#76)
	- Transfer(address(0),account,amount) (contracts/strategies/base/BaseStrategy.sol#130)
		- shares = _depositAssetsNoPull(to) (contracts/strategies/base/ShortStrategy.sol#76)
Reentrancy in ShortStrategy._withdrawAssets(address,address,address,uint256,uint256,bool) (contracts/strategies/base/ShortStrategy.sol#119-157):
	External calls:
	- GammaSwapLibrary.safeTransfer(IERC20(cfmm),receiver,assets) (contracts/strategies/base/ShortStrategy.sol#147)
	Event emitted after the call(s):
	- PoolUpdated(lpTokenBalance,s.LP_TOKEN_BORROWED,s.LAST_BLOCK_NUMBER,s.accFeeIndex,s.LP_TOKEN_BORROWED_PLUS_INTEREST,lpInvariant,s.BORROWED_INVARIANT) (contracts/strategies/base/ShortStrategy.sol#155-156)
	- Withdraw(caller,receiver,owner,assets,shares) (contracts/strategies/base/ShortStrategy.sol#154)
Reentrancy in TestLiquidationStrategy.createLoan(uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#63-78):
	External calls:
	- TestCFMM2(s.cfmm).withdrawReserves(lpTokens) (contracts/test/strategies/base/TestLiquidationStrategy.sol#68)
	Event emitted after the call(s):
	- LoanCreated(msg.sender,tokenId) (contracts/test/strategies/base/TestLiquidationStrategy.sol#77)
Reentrancy in TestPositionManager.depositReserves(address,uint256[],uint256[]) (contracts/test/TestPositionManager.sol#43-47):
	External calls:
	- (reserves,shares) = TestShortStrategy(pool)._depositReserves(to,amountsDesired,amountsMin,abi.encode(SendTokensCallbackData(cfmm,protocolId,msg.sender))) (contracts/test/TestPositionManager.sol#44-45)
	Event emitted after the call(s):
	- DepositReserve(pool,reserves.length,reserves,shares) (contracts/test/TestPositionManager.sol#46)
Reentrancy in TestCPMMBaseStrategy.testDepositToCFMM(address,uint256[],address) (contracts/test/strategies/cpmm/TestCPMMBaseStrategy.sol#33-36):
	External calls:
	- liquidity = depositToCFMM(cfmm,amounts,to) (contracts/test/strategies/cpmm/TestCPMMBaseStrategy.sol#34)
		- ICPMM(cfmm).mint(to) (contracts/strategies/cpmm/CPMMBaseStrategy.sol#21)
	Event emitted after the call(s):
	- DepositToCFMM(cfmm,to,liquidity) (contracts/test/strategies/cpmm/TestCPMMBaseStrategy.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

Figure 9: Slither Result - strategies 3

```
CPMMBaseLongStrategy.originationFee() (contracts/strategies/cpmm/CPMMBaseLongStrategy.sol#22-24) is never used and should be removed
TestBaseStrategy.depositToCFMM(address,uint256[],address) (contracts/test/strategies/base/TestBaseStrategy.sol#220) is never used and should be removed
TestBaseStrategy.withdrawFromCFMM(address,address,uint256) (contracts/test/strategies/base/TestBaseStrategy.sol#222) is never used and should be removed
TestERC20WithFee._burn(address,uint256) (contracts/test/TestERC20WithFee.sol#47-62) is never used and should be removed
TestLiquidationStrategy.calcBorrowRate(uint256,uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#152-154) is never used and should be removed
TestLiquidationStrategy.calcCFMMFeeIndex(uint256,uint256,uint256,uint256,uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#157-159) is never used and should be removed
TestLiquidationStrategy.calcFeeIndex(uint256,uint256,uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#161-163) is never used and should be removed
TestLiquidationStrategy.payLoan(LibStorage.Loan,uint256,uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#148-149) is never used and should be removed
TestLiquidationStrategy.updateCFMMIndex() (contracts/test/strategies/base/TestLiquidationStrategy.sol#165-166) is never used and should be removed
TestLiquidationStrategy.updateReserves(address) (contracts/test/strategies/base/TestLiquidationStrategy.sol#169-170) is never used and should be removed
TestLiquidationStrategy.withdrawFromCFMM(address,address,uint256) (contracts/test/strategies/base/TestLiquidationStrategy.sol#179-180) is never used and should be removed
TestLongStrategy.calcBorrowRate(uint256,uint256) (contracts/test/strategies/base/TestLongStrategy.sol#78-80) is never used and should be removed
TestLongStrategy.updateReserves(address) (contracts/test/strategies/base/TestLongStrategy.sol#129-130) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
. analyzed (52 contracts with 76 detectors), 54 result(s) found
```

Figure 10: Slither Result - strategies 4

AUTOMATED TESTING

## 4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

Report for src/v1-core/GammaPoolFactory.sol
https://dashboard.mythx.io/#/console/analyses/ef777ab6-bc2a-4853-baa3-20c94db4d9fc

| Line | SWC Title | Severity | Short Description |
|---|---|---|---|
| 8 | (SWC-123) Requirement Violation | Low | Requirement violation. |
| 39 | (SWC-123) Requirement Violation | Low | Requirement violation. |

Report for src/v1-core/libraries/LibStorage.sol
https://dashboard.mythx.io/#/console/analyses/ef777ab6-bc2a-4853-baa3-20c94db4d9fc

| Line | SWC Title | Severity | Short Description |
|---|---|---|---|
| 72 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |

Report for src/v1-strategies/strategies/base/BaseStrategy.sol
https://dashboard.mythx.io/#/console/analyses/ef777ab6-bc2a-4853-baa3-20c94db4d9fc

| Line | SWC Title | Severity | Short Description |
|---|---|---|---|
| 47 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
| 88 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |

Figure 11: MythX Result

The findings obtained as a result of the MythX scan were examined and the findings were not included in the report because they were false positive.

THANK YOU FOR CHOOSING

HALBORN