// Smart Contract Security Assessment 10.02.2024 - 10.25.2024

Tagus SC Audit (Reaudit + New Features)

InceptionLRT

HALBORN

Tagus SC Audit (Reaudit + New Features) - InceptionLRT

Prepared by: HALBORN

Last Updated 10/30/2024

Date of Engagement by: October 2nd, 2024 - October 25th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
10	1	1	3	4	1

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
   - 7.1 Lack of access control on critical facet management functions
   - 7.2 Inaccurate total deposits calculation in mellowhandler
   - 7.3 Missing emergency withdrawal capability in imellowrestaker
   - 7.4 Denial of service (dos) attack on withdrawal system
   - 7.5 Slippage protection in delegate function
   - 7.6 Setter function for trustee manager address
   - 7.7 Hardcoded slippage and deadline in withdrawmellow function
   - 7.8 Single-step ownership transfer in inceptionvault
   - 7.9 Lack of input validation in settargetflashcapacity
   - 7.10 Optimization opportunities in imellowrestaker
8. Automated Testing

1. Introduction

`InceptionLRT` engaged Halborn to conduct a security assessment on smart contracts beginning on *10/02/2024* and ending on *10/25/2024*. The security assessment was scoped to the smart contracts provided to the Halborn team.

2. Assessment Summary

The team at Halborn dedicated 18 working days for the engagement and assigned one full-time security engineer to evaluate the security of the smart contract.

The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols

The purpose of this assessment is to:

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

In summary, Halborn identified some minor improvements to reduce the likelihood and impact of risks, which were successfully addressed by the `InceptionLRT team`. The main ones were the following:

- Access control for configuring critical variables
- Correct calculations for depositing and withdrawing
- Inception-Mellow integration: emergency withdrawal
- Potential DoS attack on withdraws
- Slippage protection when delegating to mellow vaults

3. Test Approach And Methodology

Halborn performed a combination of manual, semi-automated and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this assessment. 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 security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walk-through.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any vulnerability classes
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions. (`Slither`)
- Local deployment and testing (`Hardhat`)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: smart-contracts

(b) Assessed Commit ID: abdff6e

(c) Items in scope:

- projects/vaults/contracts/vaults/Symbiotic/InceptionVault_S.sol
- projects/vaults/contracts/restakers/IMellowRestaker.sol
- projects/vaults/contracts/mellow-handler/MellowHandler.sol
- projects/vaults/contracts/vaults/EigenLayer (Diamond Proxy)

Out-of-Scope:

REMEDIATION COMMIT ID:

- 13fbcd4
- 1e86e14
- 8f79381
- 4342e7d

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	3	4	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF ACCESS CONTROL ON CRITICAL FACET MANAGEMENT FUNCTIONS	CRITICAL	SOLVED - 10/04/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INACCURATE TOTAL DEPOSITS CALCULATION IN MELLOWHANDLER	HIGH	SOLVED - 10/27/2024
MISSING EMERGENCY WITHDRAWAL CAPABILITY IN IMELLOWRESTAKER	MEDIUM	SOLVED - 10/21/2024
DENIAL OF SERVICE (DOS) ATTACK ON WITHDRAWAL SYSTEM	MEDIUM	SOLVED - 10/21/2024
SLIPPAGE PROTECTION IN DELEGATE FUNCTION	MEDIUM	SOLVED - 10/27/2024
SETTER FUNCTION FOR TRUSTEE MANAGER ADDRESS	LOW	SOLVED - 10/29/2024
HARDCODED SLIPPAGE AND DEADLINE IN WITHDRAWMELLOW FUNCTION	LOW	SOLVED - 10/27/2024
SINGLE-STEP OWNERSHIP TRANSFER IN INCEPTIONVAULT	LOW	SOLVED - 10/27/2024
LACK OF INPUT VALIDATION IN SETTARGETFLASHCAPACITY	LOW	SOLVED - 10/29/2024
OPTIMIZATION OPPORTUNITIES IN IMELLOWRESTAKER	INFORMATIONAL	SOLVED - 10/29/2024

7. FINDINGS & TECH DETAILS

7.1 LACK OF ACCESS CONTROL ON CRITICAL FACET MANAGEMENT FUNCTIONS

// CRITICAL

Description

The smart contract lacks proper access control mechanisms on several critical facet management functions, specifically:

- `setEigenLayerFacet`
- `setERC4626Facet`
- `setSetterFacet`
- `setSignature`

These functions control the setting or updating of key contract facets, which are crucial components of the Diamond Proxy architecture. Without access control, any actor can invoke these functions and manipulate the diamond's internal facets, potentially redirecting proxy calls to malicious implementations.
This vulnerability represents a significant security risk and could lead to the complete compromise of funds, data, and contract operations.

Proof of Concept

Following is the Foundry test case, which calls the setter functions with a random user.

```
    function test_setFacets_randomUser() public {
        evmCallWithUser(
            alice,
            address(inceptionVault),
            InceptionVault_EL.setEigenLayerFacet.selector,
            abi.encode(address(eigenLayerImpl))
        );
        evmCallWithUser(
            alice,
            address(inceptionVault),
            InceptionVault_EL.setERC4626Facet.selector,
            abi.encode(address(el_erc4626Impl))
        );
        evmCallWithUser(
            alice,
            address(inceptionVault),
            InceptionVault_EL.setSetterFacet.selector,
            abi.encode(address(eigenSetterImpl))
```

```
        );
    }
```

```
Ran 1 test for test/EigenLayer/EL.t.sol:ELDiamondTest
[PASS] test_setFacets_randomUser() (gas: 108766)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 9.59ms (1.01ms CPU time)

Ran 1 test suite in 356.18ms (9.59ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Given that this test case executes successfully without fail, proves the presence of the vulnerability.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:C/I:C/D:C/Y:C (10.0)

Recommendation

Apply `onlyOwner` or a similar modifier to ensure that only the contract owner or authorized accounts can modify these facets.

Remediation

SOLVED: The suggested mitigation was implemented.

Remediation Hash

13fbcd4f3d59a0bd4e07196b907 18b91a0018e43

7.2 INACCURATE TOTAL DEPOSITS CALCULATION IN MELLOWHANDLER

// HIGH

Description

In the `MellowHandler` contract, the `getTotalDeposited()` function fails to subtract the `redeemReservedAmount` from its calculation. This results in an overstatement of the total deposits managed by the contract.
The current implementation is as follows:

```
function getTotalDeposited() public view returns (uint256) {
    return getTotalDelegated() + totalAssets() + getPendingWithdrawalAmount
}
```

This function is missing the subtraction of `redeemReservedAmount`, which represents funds that are reserved for pending redemptions and should not be considered as part of the total deposits.
The impact of this vulnerability includes:
1. Inaccurate reporting of total deposits, potentially misleading users and other contracts about the actual amount of funds under management.
2. Potential cascading errors in other parts of the system that rely on the `getTotalDeposited()` function, as this function is used in various other calculations throughout the contract.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (7.5)

Recommendation

To address this vulnerability, the `getTotalDeposited()` function should be updated to include the subtraction of `redeemReservedAmount`.

```
function getTotalDeposited() public view returns (uint256) {
    return getTotalDelegated() + totalAssets() + getPendingWithdrawalAmount
}
```

Remediation

SOLVED: The suggested mitigation was implemented by the **InceptionLRT team.**

Remediation Hash

`1e86e14565d905b9e9d7c12ec4eb60fddeaf1706`

7.3 MISSING EMERGENCY WITHDRAWAL CAPABILITY IN IMELLOWRESTAKER

// MEDIUM

Description

The `IMellowRestaker` contract currently lacks the ability to initiate or handle emergency withdrawals from Mellow vaults.

- In the Mellow protocol, a regular withdrawal process involves a user registering a withdrawal request, followed by a series of operator-driven actions to ensure sufficient `underlyingTokens` are available in the vault. The operator then calls the `processWithdrawals` function to complete the withdrawal.
- Crucially, if a user's withdrawal request is not processed within the `emergencyWithdrawalDelay` period, Mellow vaults provide an emergency withdrawal mechanism that allows users to retrieve their funds directly.

However, the current implementation of `IMellowRestaker` does not include this failsafe functionality.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:M/Y:L (6.9)

Recommendation

Add an emergency withdrawal function to `IMellowRestaker` which calls `emergencyWithdraw` on the Mellow vault

```
function emergencyWithdraw(address _mellowVault) external onlyTrustee whenN
    // Call emergencyWithdraw on _mellowVault
    // Handle the received funds appropriately
    // This may involve transferring them to InceptionVault_S
}
```

Remediation

SOLVED: A new function was added in the `IMellowRestaker` contract - `withdrawEmergencyMellow` to support emergency withdrawals.

Remediation Hash

8f79381bd91daac193b8fb5908b459efed8d9847

7.4 DENIAL OF SERVICE (DOS) ATTACK ON WITHDRAWAL SYSTEM

// MEDIUM

Description

The `InceptionVault's` withdrawal system is vulnerable to a Denial of Service (DoS) attack. The vulnerability lies in the `_updateEpoch` function, which iterates over the `claimerWithdrawalsQueue` array:

```
function _updateEpoch(uint256 availableBalance) internal {
    uint256 withdrawalsNum = claimerWithdrawalsQueue.length;
    for (uint256 i = epoch; i < withdrawalNum;) {
        uint256 amount = claimerWithdrawalsQueue[i].amount;
        unchecked {
            if (amount > availableBalance) {
                break;
            }
            redeemReservedAmount += amount;
            availableBalance -= amount;
            ++epoch;
            ++i;
        }
    }
}
```

A malicious user can exploit this by creating thousands of small withdrawal requests, each of which adds a new entry to the `claimerWithdrawalsQueue`.

```
function withdraw(uint256 iShares, address receiver) external whenNotPaused
    // ... (other code)
    claimerWithdrawalsQueue.push(
        Withdrawal({
            epoch: claimerWithdrawalsQueue.length,
            receiver: receiver,
            amount: _getAssetReceivedAmount(amount)
        })
    );
    // ... (other code)
}
```

When `_updateEpoch` is called, it attempts to process all requests in a single transaction. If the queue becomes excessively large, the gas required to process the loop could exceed the block gas limit, causing the transaction to fail. This can prevent the `epoch` from updating, effectively blocking all withdrawal requests, as the `redeem` function depends on an up-to-date `epoch`.

BVSS

AO:A/AC:M/AX:L/R:N/S:U/C:N/A:H/I:N/D:N/Y:N (5.0)

Recommendation

Increase the minimum withdrawal amount so that the cost of attack increases and renders the attack unfeasible.

Remediation

SOLVED: The `minAmount` was increased to `10000000000000` to increase the cost of attack.

Remediation Hash

8f79381bd91daac193b8fb5908b459efed8d9847

7.5 SLIPPAGE PROTECTION IN DELEGATE FUNCTION

// MEDIUM

Description

The `IMellowRestaker::delegate` function lacks slippage protection when calling the `deposit`function of the `MellowDepositWrapper`. The relevant part of the function is as follows:

```
function delegate(uint256 deadline) external onlyTrustee whenNotPaused retu
    // ... (other code)
    for (uint8 i = 0; i < mellowVaults.length; i++) {
        // ... (other code)
        if (IMellowHandler(_vault).getFreeBalance() >= bal && bal > 0) {
            _asset.safeTransferFrom(_vault, address(this), bal);
            IMellowDepositWrapper wrapper = mellowDepositWrappers[address(m
            IERC20(_asset).safeIncreaseAllowance(address(wrapper), bal);
            lpAmount += wrapper.deposit(address(this), address(_asset), bal
            amount += bal;
        }
    }
}
```

The issue lies in the `wrapper.deposit` call, where the `minLPAmount` parameter is set to 0. This means there's no minimum limit set for the amount of LP tokens received in return for the deposited assets. This oversight can lead to the following:

1. Front-running Vulnerability: Malicious actors could exploit this by front-running the transaction and manipulating the pool's composition, resulting in the user receiving fewer LP tokens than expected.
2. Unexpected Slippage: In volatile market conditions, the lack of a minimum LP amount could result in significant slippage, leading to substantial losses for the user.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:M/Y:N (5.0)

Recommendation

To address this vulnerability, implement proper slippage protection by setting a non-zero minimum LP amount according to a configurable threshold.

Remediation

SOLVED: Two new slippage parameters were introduced - `depositSlippage` and `withdrawSlippage`. These can be configured by the owner using the function `slippages`.

Remediation Hash

`1e86e14565d905b9e9d7c12ec4eb60fddeaf1706`

7.6 SETTER FUNCTION FOR TRUSTEE MANAGER ADDRESS

// LOW

Description

The `IMellowRestaker` includes an internal address variable for the trustee manager:

```
address internal _trusteeManager;
```

However, there is no setter function provided to update this address. This oversight presents several security and operational risks:
1. Account Compromise: If the trustee manager's account is compromised, there's no way to transfer control to a new, secure address.
2. Key Loss: In the event of private key loss, the trustee manager role becomes permanently inaccessible.

BVSS

AO:A/AC:M/AX:M/R:N/S:U/C:N/A:M/I:N/D:H/Y:N (3.9)

Recommendation

To address this vulnerability, implement a setter function for the `_trusteeManager` address with appropriate access controls.

Remediation

SOLVED: The function `setTrusteeManager` was implemented to allow modifying the `_trusteeManager`.

Remediation Hash

4342e7d99ce207e7db33ddd7ca42800c00525008

7.7 HARDCODED SLIPPAGE AND DEADLINE IN WITHDRAWMELLOW FUNCTION

// LOW

Description

The `IMellowRestaker::withdrawMellow` function in the contract contains two issues:
1. Hardcoded Slippage: The minimum amount for withdrawal is calculated with a fixed "dust" value.
2. Hardcoded Deadline: The deadline for the withdrawal request is set to a fixed 15 days from the current timestamp.
Here's the relevant part of the function:

```
function withdrawMellow(address _mellowVault, uint256 amount, bool closePrev
    external
    override
    onlyTrustee
    whenNotPaused
    returns (uint256)
{
    // ... (other code)
    uint256[] memory minAmounts = new uint256[](1);
    minAmounts[0] = amount - 5; // dust
    // ... (other code)
    mellowVault.registerWithdrawal(
        address(this),
        lpAmount,
        minAmounts,
        block.timestamp + 15 days,
        block.timestamp + 15 days,
        closePrevious
    );
    // ... (rest of the function)
}
```

The contract owner has no ability to adjust these parameters based on their risk tolerance or market expectations.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:L/I:N/D:M/Y:N (3.8)

Recommendation

To address these issues, it is recommended to implement the following changes:

- Dynamic Slippage Calculation

```
uint256 slippageBps = 50; // 0.5% slippage, configurable
uint256 minAmount = amount * (10000 - slippageBps) / 10000;
```

- Configurable Withdrawal Period

```
uint256 public withdrawalPeriod = 15 days; // Default to 15 days, but confi

function setWithdrawalPeriod(uint256 _newPeriod) external onlyOwner {
    require(_newPeriod > 0 && _newPeriod <= 100 days, "Invalid withdrawal p
    withdrawalPeriod = _newPeriod;
    emit WithdrawalPeriodChanged(_newPeriod);
}
```

Remediation

SOLVED: The variables `requestDeadline` and `withdrawSlippage` were added to make the values configurable.

Remediation Hash

`1e86e14565d905b9e9d7c12ec4eb60fddeaf1706`

7.8 SINGLE-STEP OWNERSHIP TRANSFER IN INCEPTIONVAULT

// LOW

Description

The `InceptionVault_S` inherits from OpenZeppelin's `OwnableUpgradeable` and use its single-step ownership transfer mechanism. This is risky because if the owner accidentally transfers ownership to an incorrect address, the contract becomes permanently locked with no way to recover.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:H/I:H/D:N/Y:L (2.0)

Recommendation

Replace `OwnableUpgradeable` with `Ownable2StepUpgradeable` from OpenZeppelin, which implements a two-step ownership transfer:
1. The current owner calls `transferOwnership(address newOwner)` to initiate the transfer
2. The new owner must call `acceptOwnership()` to complete the transfer

Remediation

SOLVED: The suggested mitigation was implemented by the **InceptionLRT team.**

Remediation Hash

`1e86e14565d905b9e9d7c12ec4eb60fddeaf1706`

7.9 LACK OF INPUT VALIDATION IN SETTARGETFLASHCAPACITY

// LOW

Description

The `MellowHandler::setTargetFlashCapacity` function in the contract allows the owner to set a new target flash capacity without validating the input value. The function is implemented as follows:

```
function setTargetFlashCapacity(uint256 newTargetCapacity) external onlyOwn
    emit TargetCapacityChanged(targetCapacity, newTargetCapacity);
    targetCapacity = newTargetCapacity;
}
```

This implementation lacks a check for a non-zero value, which will lead to disruption of the normal functioning of the contract, especially when other functions of the system depend on this value being non-zero.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:H/I:H/D:N/Y:L (2.0)

Recommendation

To address this vulnerability, implement a zero-value check before setting the new target capacity.

Remediation

SOLVED: The suggested mitigation was implemented by the **InceptionLRT team.**

Remediation Hash

4342e7d99ce207e7db33ddd7ca42800c00525008

7.10 OPTIMIZATION OPPORTUNITIES IN IMELLOWRESTAKER

// INFORMATIONAL

Description

In the `IMellowRestaker` contract, two functions, `getDeposited` and `getTotalDeposited`, perform unnecessary computations when balances are zero, leading to potential gas wastage. Here are the current implementations:

```
function getDeposited(address _mellowVault) public view returns (uint256) {
    IMellowVault mellowVault = IMellowVault(_mellowVault);
    uint256 balance = mellowVault.balanceOf(address(this));
    return lpAmountToAmount(balance, mellowVault);
}

function getTotalDeposited() public view returns (uint256) {
    uint256 total;
    for (uint256 i = 0; i < mellowVaults.length; i++) {
        uint256 balance = mellowVaults[i].balanceOf(address(this));
        total += lpAmountToAmount(balance, mellowVaults[i]);
    }
    return total;
}
```

In both functions, `lpAmountToAmount` is called regardless of whether the balance is zero. This leads to unnecessary gas consumption, especially in `getTotalDeposited` where it happens in a loop.

- Increased Gas Costs: Users or the contract itself may spend more gas than necessary when checking deposits, especially for addresses or vaults with no balance.
- Reduced Efficiency: In scenarios where these functions are called frequently or as part of larger transactions, the cumulative effect of this inefficiency could be significant.
- Potential for DOS: In extreme cases, if the `mellowVaults` array becomes very large, the `getTotalDeposited` function might consume so much gas that it becomes unusable in transactions or even reverts due to out-of-gas errors.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (1.7)

Recommendation

To address these issues, we can add checks to skip the `lpAmountToAmount` conversion when the balance is zero. Here are the optimized versions of both functions:

```
function getDeposited(address _mellowVault) public view returns (uint256) {
    IMellowVault mellowVault = IMellowVault(_mellowVault);
```

```
    uint256 balance = mellowVault.balanceOf(address(this));
    if (balance == 0) {
        return 0;
    }
    return lpAmountToAmount(balance, mellowVault);
}

function getTotalDeposited() public view returns (uint256) {
    uint256 total;
    for (uint256 i = 0; i < mellowVaults.length; i++) {
        uint256 balance = mellowVaults[i].balanceOf(address(this));
        if (balance > 0) {
            total += lpAmountToAmount(balance, mellowVaults[i]);
        }
    }
    return total;
}
```

Remediation

SOLVED: The suggested mitigation was implemented by the **InceptionLRT team.**

Remediation Hash

4342e7d99ce207e7db33ddd7ca42800c00525008

8. AUTOMATED TESTING

Introduction

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.
The security team conducted a comprehensive review of all findings generated by the Slither static analysis tool.

```
INFO:Detectors:
IMellowRestaker.delegateMellow(uint256,uint256,uint256,address) (src/restakers/IMellowRestaker.sol#82-95) uses arbitrary from in transferFrom: _asset.safe
TransferFrom(_vault,address(this),amount) (src/restakers/IMellowRestaker.sol#91)
IMellowRestaker.delegate(uint256) (src/restakers/IMellowRestaker.sol#98-117) uses arbitrary from in transferFrom: _asset.safeTransferFrom(_vault,address(t
his),bal) (src/restakers/IMellowRestaker.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
```

```
INFO:Detectors:
Reentrancy in MellowHandler.claimCompletedWithdrawals() (src/mellow-handler/MellowHandler.sol#77-90):
        External calls:
        - withdrawnAmount = mellowRestaker.claimMellowWithdrawalCallback() (src/mellow-handler/MellowHandler.sol#85)
        State variables written after the call(s):
        - _updateEpoch(availableBalance + withdrawnAmount) (src/mellow-handler/MellowHandler.sol#89)
                - redeemReservedAmount += amount (src/mellow-handler/MellowHandler.sol#116)
        MellowHandler.redeemReservedAmount (src/mellow-handler/MellowHandler.sol#27) can be used in cross function reentrancies:
        - MellowHandler._updateEpoch(uint256) (src/mellow-handler/MellowHandler.sol#108-122)
        - MellowHandler.getFlashCapacity() (src/mellow-handler/MellowHandler.sol#148-150)
        - MellowHandler.redeemReservedAmount (src/mellow-handler/MellowHandler.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
InceptionVault_S._deposit(uint256,address,address).depositBonus (src/vaults/Symbiotic/InceptionVault_S.sol#121) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
MellowHandler._depositAssetIntoMellow(uint256,address) (src/mellow-handler/MellowHandler.sol#59-62) ignores return value by _asset.approve(address(mellowR
estaker),amount) (src/mellow-handler/MellowHandler.sol#60)
MellowHandler._depositAssetIntoMellow(uint256,address) (src/mellow-handler/MellowHandler.sol#59-62) ignores return value by mellowRestaker.delegateMellow(
amount,0,block.timestamp,mellowVault) (src/mellow-handler/MellowHandler.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Reentrancy in MellowHandler.claimCompletedWithdrawals() (src/mellow-handler/MellowHandler.sol#77-90):
        External calls:
        - withdrawnAmount = mellowRestaker.claimMellowWithdrawalCallback() (src/mellow-handler/MellowHandler.sol#85)
        State variables written after the call(s):
        - _updateEpoch(availableBalance + withdrawnAmount) (src/mellow-handler/MellowHandler.sol#89)
                - ++ epoch (src/mellow-handler/MellowHandler.sol#118)
Reentrancy in InceptionVault_S.flashWithdraw(uint256,address) (src/vaults/Symbiotic/InceptionVault_S.sol#257-281):
        External calls:
        - inceptionToken.burn(claimer,iShares) (src/vaults/Symbiotic/InceptionVault_S.sol#266)
        State variables written after the call(s):
        - depositBonusAmount += (fee - protocolWithdrawalFee) (src/vaults/Symbiotic/InceptionVault_S.sol#273)
Reentrancy in InceptionVault_S.withdraw(uint256,address) (src/vaults/Symbiotic/InceptionVault_S.sol#192-215):
        External calls:
        - inceptionToken.burn(claimer,iShares) (src/vaults/Symbiotic/InceptionVault_S.sol#199)
        State variables written after the call(s):
        - genRequest.amount += _getAssetReceivedAmount(amount) (src/vaults/Symbiotic/InceptionVault_S.sol#204)
        - claimerWithdrawalsQueue.push(Withdrawal({epoch:claimerWithdrawalsQueue.length,receiver:receiver,amount:_getAssetReceivedAmount(amount)})) (src/v
aults/Symbiotic/InceptionVault_S.sol#206-212)
        - totalAmountToWithdraw += amount (src/vaults/Symbiotic/InceptionVault_S.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function OwnableUpgradeable.__Ownable_init() (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#33-35) is not in mi
xedCase
Variable OwnableUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (lib/openzeppelin-contracts-upgradeable/contracts/security/PausableUpgradeable.sol#34-36) is not in mixedCa
se
Function PausableUpgradeable.__Pausable_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/security/PausableUpgradeable.sol#38-40) is not
in mixedCase
Variable PausableUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/PausableUpgradeable.sol#116) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#40-
42) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeab
le.sol#44-46) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/security/ReentrancyGuardUpgradeable.sol#80) is not in mixedCas
e
Function ContextUpgradeable.__Context_init() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#21-22) is not in mix
edCase
Variable ContextUpgradeable.__gap (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#59) is not in mixedCase
Function InceptionAssetsHandler.__InceptionAssetsHandler_init(IERC20) (src/assets-handler/InceptionAssetsHandler.sol#26-33) is not in mixedCase
Variable InceptionAssetsHandler.__reserver (src/assets-handler/InceptionAssetsHandler.sol#24) is not in mixedCase
Contract IInceptionVault_EL (src/interfaces/eigenlayer-vault/IInceptionVault_EL.sol#7-138) is not in CapWords
Contract IInceptionVault_S (src/interfaces/symbiotic-vault/IInceptionVault_S.sol#7-103) is not in CapWords
Function IMellowVault.Q96() (src/interfaces/symbiotic-vault/mellow-core/IMellowVault.sol#90) is not in mixedCase
Function IMellowVault.D9() (src/interfaces/symbiotic-vault/mellow-core/IMellowVault.sol#93) is not in mixedCase
Function IMellowVaultConfigurator.MAX_DELAY() (src/interfaces/symbiotic-vault/mellow-core/IMellowVaultConfigurator.sol#48) is not in mixedCase
Function IMellowVaultConfigurator.MAX_WITHDRAWAL_FEE() (src/interfaces/symbiotic-vault/mellow-core/IMellowVaultConfigurator.sol#52) is not in mixedCase
Function MellowHandler.__MellowHandler_init(IERC20,IIMellowRestaker) (src/mellow-handler/MellowHandler.sol#43-47) is not in mixedCase
Parameter MellowHandler.__MellowHandler_init(IERC20,IIMellowRestaker)._mellowRestaker (src/mellow-handler/MellowHandler.sol#43) is not in mixedCase
Variable MellowHandler.__gap (src/mellow-handler/MellowHandler.sol#36) is not in mixedCase
Contract InceptionVault_S (src/vaults/Symbiotic/InceptionVault_S.sol#16-467) is not in CapWords
Function InceptionVault_S.__InceptionVault_init(string,address,IERC20,IInceptionToken,IIMellowRestaker) (src/vaults/Symbiotic/InceptionVault_S.sol#51-80)
is not in mixedCase
Parameter InceptionVault_S.__InceptionVault_init(string,address,IERC20,IInceptionToken,IIMellowRestaker)._inceptionToken (src/vaults/Symbiotic/InceptionVa
ult_S.sol#55) is not in mixedCase
Parameter InceptionVault_S.__InceptionVault_init(string,address,IERC20,IInceptionToken,IIMellowRestaker)._mellowRestaker (src/vaults/Symbiotic/InceptionVa
ult_S.sol#56) is not in mixedCase
Function InceptionVault_S.__beforeDeposit(address,uint256) (src/vaults/Symbiotic/InceptionVault_S.sol#86-91) is not in mixedCase
Function InceptionVault_S.__afterDeposit(uint256) (src/vaults/Symbiotic/InceptionVault_S.sol#93-95) is not in mixedCase
Function InceptionVault_S.__beforeWithdraw(address,uint256) (src/vaults/Symbiotic/InceptionVault_S.sol#181-187) is not in mixedCase
Contract InVault_S_E2 (src/vaults/Symbiotic/vault_e2/InVault_S_E2.sol#8-41) is not in CapWords
Parameter InVault_S_E2.initialize(string,address,IERC20,IInceptionToken,IIMellowRestaker)._inceptionToken (src/vaults/Symbiotic/vault_e2/InVault_S_E2.sol#
18) is not in mixedCase
```

After careful examination and consideration of the flagged issues, it was determined that within the project's specific context and scope, all were false positives. Including the reentrancy, as the call is being made to trusted contracts.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project’s integrity and addressing potential vulnerabilities introduced by code modifications.