Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

InceptionLRT

Veridise Inc.
January 12, 2024

► **Prepared For:**

InceptionLRT
https://www.inceptionlrt.com/

► **Prepared By:**

Timothy Hoffman
Bryan Tan

► **Contact Us:** contact@veridise.com

► **Version History:**

Jan. 12, 2024	V1.02 - updated fix statuses
Jan. 3, 2024	V1.01 - fixed typos and updated fix statuses
Dec. 28, 2023	V1

Contents

1 Executive Summary

From Dec.15, 2023 to Dec. 19, 2023, InceptionLRT engaged Veridise to review the security of their InceptionLRT smart contracts. The review covered a token pool smart contract that acts as a staker in an EigenLayer* strategy smart contract. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `11af9fb`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing. Due to the heavy use of third-party protocols, Veridise engineers also investigated some of the third-party EigenLayer smart contracts (which are not in the scope of the audit) invoked by the InceptionLRT smart contracts.

Project summary. The security assessment covered the source files making up the `InceptionVault` smart contract as well as several of its concrete implementations. The vault contract acts as a "pool" of assets, where users can deposit a specific token ("asset") into the vault. In turn, the vault will deposit those assets into a specific EigenLayer strategy contract and then grant vault-specific ERC20 tokens ("Inception tokens") to the user. Users may exchange their Inception tokens with the vault to queue a withdrawal. A special account called the "operator" can then send a withdrawal request to the EigenLayer smart contracts. Once a withdrawal has been processed, users can then "redeem" their withdrawal to obtain assets from the vault.

Code assessment. The InceptionLRT developers provided the source code of the InceptionLRT contracts for review. The source code appears to be mostly original code written by the InceptionLRT developers. No documentation was provided to the Veridise auditors for the audit.

The source code contained a test suite, which the Veridise auditors noted provide partial coverage of the deposit, withdraw, and redeem workflows of the protocol in both successful and unsuccessful scenarios.

Summary of issues detected. The audit uncovered 14 issues, 2 of which are assessed to be of medium severity by the Veridise auditors. Specifically, users that attempt to withdraw may suffer from slippage issues due to multiple conversions involving spot-prices (V-INCP-VUL-001), and a missing check may allow users to redeem withdrawals earlier than intended (V-INCP-VUL-002). The Veridise auditors also identified several low-severity issues including multiple centralization risks (V-INCP-VUL-004) and a missing check on approval success status (V-INCP-VUL-003), as well as 6 warnings and 4 informational findings. The InceptionLRT developers have acknowledged all 14 of the reported issues, 9 of which they have fixed.

* https://www.eigenlayer.xyz/

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
InceptionLRT	`11af9fb`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Dec.15 - Dec. 19, 2023	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	2	1	2
Low-Severity Issues	2	0	2
Warning-Severity Issues	6	5	6
Informational-Severity Issues	4	3	4
TOTAL	14	9	14

Table 2.4: Category Breakdown.

Name	Number
Logic Error	5
Data Validation	3
Maintainability	3
Authorization	1
Reentrancy	1
Missing/Incorrect Events	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of InceptionLRT's smart contracts. In our audit, we sought to answer questions such as:

- Are there any assumptions that the vault makes about how the EigenLayer strategy will manage deposited assets?
- Does the rebalancing/settlement logic work correctly?
- Does the vault perform correct bookkeeping for deposits and withdrawals?
- Does the vault correctly use EigenLayer's withdrawal protocol?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy, misuse of upgradeable contracts, and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the `contracts` folder of the source code provided by the InceptionLRT developers, which contains the smart contract implementation of the InceptionLRT. The smart contracts interact heavily with third-party smart contracts from EigenLayer. During the audit, the Veridise auditors referred to these third-party contracts, but the third-party contracts were not in the scope of the audit and were assumed to be correctly implemented.

Methodology. Veridise auditors first inspected the provided tests. They then began a manual audit of the code assisted by both static analyzers and automated testing. During the audit, the Veridise auditors regularly referred to the EigenLayer smart contracts and documentation to understand how the InceptionLRT code interacts with EigenLayer.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-INCP-VUL-001	Asset value slippage may cause inconsistent boo. .	Medium	Acknowledged
V-INCP-VUL-002	_withdrawFromEL does not check for pending wit	Medium	Fixed
V-INCP-VUL-003	Unchecked ERC20 approve status	Low	Partially Fixed
V-INCP-VUL-004	Centralization risks	Low	Open
V-INCP-VUL-005	previewDeposit and previewWithdraw implement	Warning	Fixed
V-INCP-VUL-006	Not all InceptionVault superclass initializers . . .	Warning	Fixed
V-INCP-VUL-007	Deposit fee functionality not used	Warning	Fixed
V-INCP-VUL-008	depositExta fails silently if totalAmountToWith. . .	Warning	Open
V-INCP-VUL-009	Potential reentrancy risk in claimCompletedWith. .	Warning	Fixed
V-INCP-VUL-010	setDepositFee percentage not validated	Warning	Fixed
V-INCP-VUL-011	Strategy deposit return value ignored	Info	Partially Fixed
V-INCP-VUL-012	Typo in depositExta function name	Info	Fixed
V-INCP-VUL-013	InceptionAssetsHandler does not extend its inte. . .	Info	Fixed
V-INCP-VUL-014	redeem() parameter should be named ‘receiver’	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-INCP-VUL-001: Asset value slippage may cause inconsistent bookkeeping

Severity	Medium	**Commit**	11af9fb
Type	Logic Error	**Status**	Acknowledged
File(s)	`InceptionVault.sol`		
Location(s)	withdraw(), redeem()		
Confirmed Fix At			

The `InceptionVault` contract uses the storage variable `totalAmountToWithdraw` to track amounts of assets being actively withdrawn by users from the vault. Tracking the ***asset*** quantities directly—but not the shares in the workflow—may result in slippage problems. In particular, there are three different points in the withdraw workflow where assets need to be converted between shares, resulting in spot prices being used in calculations:

1. To withdraw assets, a user must invoke `withdraw()` with the number of `InceptionTokens` that they wish to exchange back to assets. The spot price of the assets (as determined by the underlying EigenLayer strategy) will be added to `totalAmountToWithdraw` as well as the `_claimerWithdrawals[receiver].amount` entry for the receiver. These amounts will be **queued** for withdrawal and not withdrawn immediately.

```
function withdraw(uint256 iShares, address receiver) external nonReentrant {
    // ...
    uint256 amount = Convert.multiplyAndDivideFloor(iShares, 1e18, ratio());
    require(
        amount >= minAmount,
        "InceptionVault: amount is less than the minimum withdrawal"
    );
    // burn Inception token in view of the current ratio
    inceptionToken.burn(claimer, iShares);

    // update global state and claimer's state
    totalAmountToWithdraw += amount;
    Withdrawal storage request = _claimerWithdrawals[receiver];
    request.amount += _getAssetReceivedAmount(amount);
    request.receiver = receiver;
    request.epoch = epoch;

    emit Withdraw(claimer, receiver, claimer, amount, iShares);
}
```

Snippet 4.1: Relevant code in `withdraw()`. Note that `ratio()` is based on the spot price of the vault's assets as determined by the strategy.

2. The `operator` of the vault can then trigger a withdraw request in EigenLayer by invoking the vault's `withdrawFromEL()` method. This will convert the `totalAmountToWithdraw` (minus some other quantities) to strategy shares based on the current spot price (as determined by the EigenLayer strategy) and then request that those shares be withdrawn from the strategy.
3. After some delay in time, the withdraw request can then be completed by calling `claimCompletedWithdrawals()`, at which point the EigenLayer strategy will transfer the

```
sharesToWithdraw[0] = strategy.underlyingToSharesView(amount);

uint256 totalAssetSharesInEL = strategyManager.stakerStrategyShares(
    address(this),
    strategy
);
// we need to withdraw the remaining dust from EigenLayer
if (
    totalAssetSharesInEL < sharesToWithdraw[0] + 5 ||
    sharesToWithdraw[0] > totalAssetSharesInEL
) {
    sharesToWithdraw[0] = totalAssetSharesInEL;
}
```

Snippet 4.2: Relevant code in `_generateELWithdrawal()` that generates the withdraw request to be sent to EigenLayer. Again, note that `underlyingToSharesView()` will be based on a spot price.

assets corresponding to the shares in the previous step back to the vault. The strategy may or may not use a spot price to perform the conversion.

Impact Since spot prices are used, the total amount of assets withdrawn in `claimCompletedWithdrawals()` may not match the total amount of assets requested for withdrawal in `withdraw()`. For example, if the strategy has negative returns, then the strategy will transfer less assets than what was originally calculated in `withdraw()` (and worse, also less than what was originally deposited). This may cause the vault to not have enough assets to transfer back to users in `redeem()`.

Recommendation The slippage issue can be avoided by refactoring the workflow to only perform a single spot price conversion (that ideally is done in `claimCompletedWithdrawals()` when the withdraw from the strategy has been finalized). However, this may require a significant change in the way the bookkeeping is implemented, such as splitting `totalAmountToWithdraw` into multiple variables representing different units (`InceptionTokens`, strategy shares, etc.). For example, instead of directly tracking asset quantities to withdraw, the vault could track `InceptionTokens` and/or strategy shares that are being withdrawn; these shares are only converted to assets once a withdraw from the strategy is finalized.

Developer Response The developers have acknowledged the issue but do not intend to fix the issue at this time:

> [it is too complicated to fix in] terms of transaction costs(updating the storage for each withdrawal request) and refactoring as well. Moreover, I tend to think that the suggested solution has the same problem and doesn't completely solve the issue

The developers noted that they assume that the strategy used by the vault will always have positive returns. They explicitly indicated that they will "trust" the strategy contract, and they do not consider negative events, such as the strategy contract being hacked, as part of their threat model.

4.1.2 V-INCP-VUL-002: _withdrawFromEL does not check for pending withdrawal

Severity	Medium	**Commit**	11af9fb
Type	Data Validation	**Status**	Fixed
File(s)	`EigenLayerHandler.sol`		
Location(s)	_withdrawFromEL()		
Confirmed Fix At	1f9738b		

The `InceptionVault` has a notion of "epochs" (numbered with monotonically increasing integer values) that it uses to keep track of rebalancing periods. The rebalancing period is the time between the `InceptionVault` operator calling `withdrawFromEL()` to request assets from the EigenLayer and claiming the completed withdraw with `claimCompletedWithdrawals()`. Both of these functions increment the epoch value.

During a rebalancing period, the `InceptionVault` ensures that it has the necessary funds to cover all pending withdraw requests made prior to this rebalancing. Once a user initiates a withdraw request, they must wait until after the next rebalancing phase is done to complete their withdraw via the `redeem()` function. The `redeem()` function uses `isAbleToRedeem()` to ensure that the withdraw request was made prior to the last rebalancing period so that the necessary funds are available in the `InceptionVault`.

```
function isAbleToRedeem(address claimer) public view returns (bool) {
    Withdrawal storage request = _claimerWithdrawals[claimer];
    // the request is empty
    if (request.amount == 0) {
        return false;
    }
    if (request.amount > totalAssets()) {
        return false;
    }
    // a claimer withdrew during the rebalance
    if (request.epoch % 2 != 0) {
        if (epoch - request.epoch < 3) {
            return false;
        }
    } else {
        if (epoch - request.epoch < 2) {
            return false;
        }
    }

    return true;
}
```

Snippet 4.3: Definition of `isAbleToRedeem()`

Within `isAbleToRedeem()`, the check for `request.epoch % 2 != 0` assumes that odd numbered epochs are always the rebalancing periods. Accordingly, it requires that the epoch has advanced 3 times for a request made during a rebalancing period and 2 times otherwise, ensuring that at least one rebalancing period has occurred between the user calling `withdraw()` and `redeem()`.

However, this scenario assumes that `withdrawFromEL()` and `claimCompletedWithdrawals()` transaction are always alternating. There is no check to ensure the operator does not complete two

`withdrawFromEL()` transactions in row without a `claimCompletedWithdrawals()` between. Even for a well-behaved operator that always alternates calls between the two functions, it's possible for the operator to accidentally call `withdrawFromEL()` twice in a row. This can be successful as long as a user has called `withdraw()` in between the two `withdrawFromEL()` calls, so that `_generateELWithdrawal()` does not revert when checking the `totalAmountToWithdraw()` against `totalAssets() + _pendingWithdrawalAmount()`.

Impact The checks within `isAbleToRedeem()` could pass when the vault does not have enough assets to cover the pending withdraw requests. Before the corresponding `claimCompletedWithdrawals()` transactions are complete, this would cause the call to `redeem()` to revert unexpectedly due to insufficient funds.

Recommendation Add `require(request.epoch % 2 == 0)` to `withdrawFromEL()` to ensure it cannot succeed during a rebalancing period, i.e. twice in a row.

4.1.3 V-INCP-VUL-003: Unchecked ERC20 approve status

Severity	Low	**Commit**	11af9fb
Type	Data Validation	**Status**	Partially Fixed
File(s)	`EigenLayerHandler.sol`		
Location(s)	__EigenLayerHandler_init()		
Confirmed Fix At	9997778		

Part of an `InceptionVault`'s initialization logic will call `__EigenLayerHandler_init()`, which approves the EigenLayer `StrategyManager` to arbitrarily transfer `_asset` ERC20 tokens from the vault. Although the ERC20 `approve()` method returns a boolean value indicating whether the approval was successful, this success status is not checked.

```
function __EigenLayerHandler_init(
    IStrategyManager _strategyManager,
    IStrategy _assetStrategy
) internal onlyInitializing {
    strategyManager = _strategyManager;
    strategy = _assetStrategy;

    __InceptionAssetsHandler_init(_assetStrategy.underlyingToken());
    // approve spending by stategyManager
    _asset.approve(address(strategyManager), type(uint256).max);
```

Snippet 4.4: Relevant code in `__EigenLayerHandler_init()`

Impact If approval is not actually granted by the ERC20 token, then deposits into EigenLayer will fail.

Recommendation We suggest the following improvements:

- Use OpenZeppelin's `SafeERC20.safeApprove()` library function instead of directly calling `approve()`. This has the benefit of also handling non-compliant ERC20 tokens that have an `approve()` method that does not return anything.
- Instead of granting infinite approval, call `approve()` only before depositing into EigenLayer and revoke the approval immediately after the deposit finishes. This can help mitigate damage in the event of the `StrategyManager` suffering from a bug or an attack.

Developer Response The developers added a `require()` to check the result of the `approve()` call. However, this will revert if the asset is a non-compliant ERC20 token that implements `approve()` without a return value.

4.1.4 V-INCP-VUL-004: Centralization risks

Severity	Low	**Commit**	11af9fb
Type	Authorization	**Status**	Open
File(s)	`InceptionVault.sol`		
Location(s)	N/A		
Confirmed Fix At			

An `InceptionVault` has special roles such as `owner` and `operator` that can exercise control over core functionalities of the vault. In particular:

- The contract is upgradeable, so the `owner` of the vault may add to, modify, and remove from the contract bytecode.
- The `owner` is allowed to change the `operator`, the deposit fee percentage, and the minimum deposit amount.
- The `operator` is the only address that may call `withdrawFromEL()`, the method that causes the vault to request a withdrawal from the EigenLayer strategy used by the vault.
- The `operator` may call `depositExta()` to cause the vault to deposit any unused, unredeemable assets into the vault's EigenLayer strategy.

Impact These privileges may introduce the following risks for users of the vault:

- If the `owner`'s private key is leaked to an attacker, then the attacker can upgrade the vault contract to withdraw from EigenLayer and then drain funds.
- If the `operator` does not regularly call `withdrawFromEL()`, then there is no way for users to withdraw their deposited funds. This may result in a locked funds issue.
- The owner is capable of setting the minimum deposit amount to be a very high value (such as the max `uint256` value) that will always cause the `deposit()` function to revert.

Recommendation The developers should introduce mitigations against the centralization risks, such as:

- Secure the `owner` account using technology such as multi-signature wallets.
- Add a way for ordinary users to send a withdrawal request to the EigenLayer strategy. For example, instead of requiring the privileged `operator` to send the request, allow such a request to be sent by any user when no other withdraw request is pending.

Developer Response The developers noted that the `owner` is kept in a multi-signature wallet, while the `operator` is kept with an ordinary private key.

4.1.5 V-INCP-VUL-005: previewDeposit and previewWithdraw implementations are no-ops

Severity	Warning	**Commit**	11af9fb
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionVault.sol`		
Location(s)	previewDeposit(), previewWithdraw()		
Confirmed Fix At	9997778		

The `InceptionVault` contract defines two view methods `previewDeposit()` and `previewWithdraw()`, both of which take a single integer as a parameter and return a single integer. However, the methods do not actually do anything. This is indicative of some unimplemented or missing functionality.

```
function previewDeposit(uint256 assets) public view returns (uint256) {}

function previewWithdraw(uint256 assets) public view returns (uint256) {}
```

Snippet 4.5: Definitions of `previewDeposit()` and `previewWithdraw()`

Impact Both `previewDeposit()` and `previewWithdraw()` always return zero, which can have unintended consequences for the callers.

Recommendation Remove or implement the two methods.

Developer Response The developers removed the two methods.

4.1.6 V-INCP-VUL-006: Not all InceptionVault superclass initializers are called

Severity	Warning	**Commit**	11af9fb
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionVault.sol`		
Location(s)	initialize()		
Confirmed Fix At	9997778		

The `InceptionVault` contract has the inheritance hierarchy shown in the diagram below.

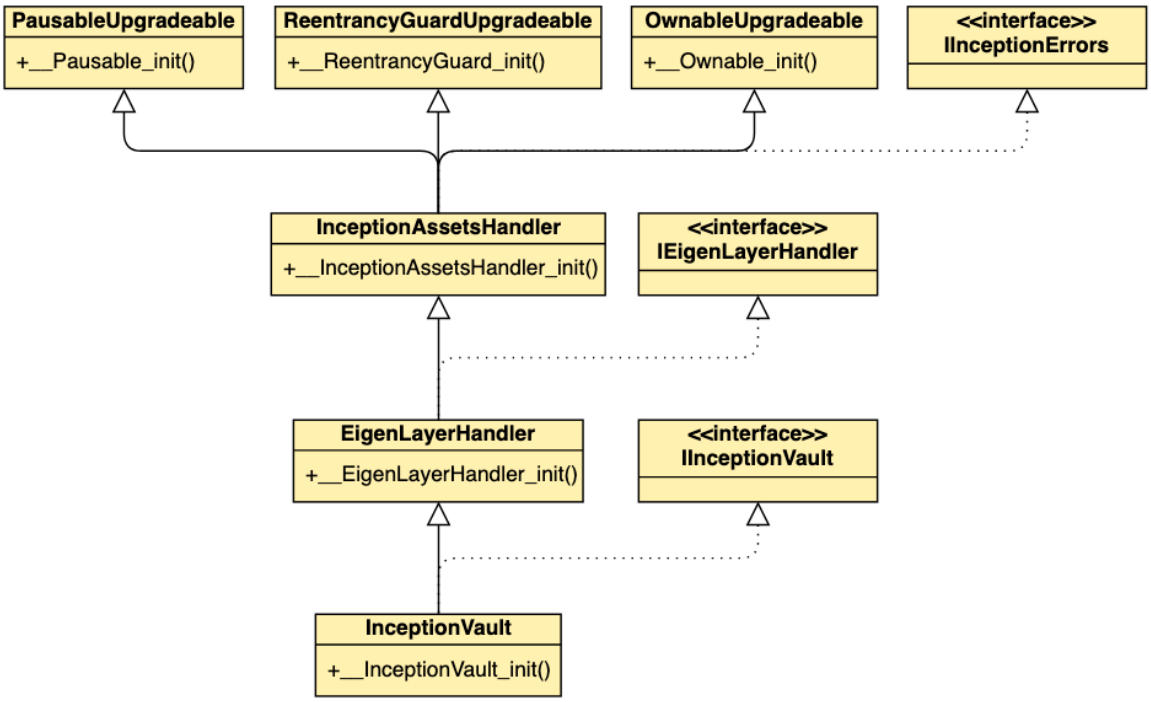

Figure 4.6: `InceptionVault` inheritance hierarchy

Because the `InceptionVault` uses OpenZeppelin's `Initializable` pattern, it should call all of the superclass `*_init()` functions when `__InceptionVault_init()` is called, either directly or via a chain of calls. However, the implementation of `__InceptionVault_init()` will not end up calling `__Pausable_init()` or `__ReentrancyGuard_init()`.

Impact The internal state of the `PausableUpgradeable` and `ReentrancyGuardUpgradeable` will not be properly initialized.

Recommendation Add calls to `__Pausable_init()` and `__ReentrancyGuard_init()` within `__InceptionAssetsHandler_init()`.

```
function __InceptionVault_init(
    address operatorAddress,
    IStrategyManager _strategyManager,
    IInceptionToken _inceptionToken,
    IStrategy _assetStrategy
) internal {
    __Ownable_init();
    __EigenLayerHandler_init(_strategyManager, _assetStrategy);

    _operator = operatorAddress;
    inceptionToken = _inceptionToken;

    depositFee = 100000000000000000; // 10.00%
    minAmount = 100;
}
```

Snippet 4.7: Definition of `InceptionVault.__InceptionVault_init()`

```
function __EigenLayerHandler_init(
        IStrategyManager _strategyManager,
        IStrategy _assetStrategy
    ) internal onlyInitializing {
        strategyManager = _strategyManager;
        strategy = _assetStrategy;

        __InceptionAssetsHandler_init(_assetStrategy.underlyingToken());
        // approve spending by stategyManager
        _asset.approve(address(strategyManager), type(uint256).max);
    }
```

Snippet 4.8: Definition of `EigenLayerHandler.__EigenLayerHandler_init()`

```
function __InceptionAssetsHandler_init(
        IERC20 assetAddress
    ) internal onlyInitializing {
        _asset = assetAddress;
    }
```

Snippet 4.9: Definition of `InceptionAssetsHandler.__InceptionAssetsHandler_init()`

4.1.7 V-INCP-VUL-007: Deposit fee functionality not used

Severity	Warning	**Commit**	11af9fb
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionVault.sol`		
Location(s)	See description		
Confirmed Fix At	9997778		

The `InceptionVault` contract defines a `depositFee` storage variable that only appears to be used in the `setDepositFee()`. Since this method only changes the value of `depositFee`, the `depositFee` variable is never actually used in practice.

```
uint256 public depositFee;

function __InceptionVault_init(
    /* ... */
) internal {
  // ...
  depositFee = 100000000000000000; // 10.00%
  // ...
}

function setDepositFee(uint256 newDepositFee) external onlyOwner {
    emit DepositFeeChanged(depositFee, newDepositFee);
    depositFee = newDepositFee;
}
```

Snippet 4.10: Relevant definitions in `InceptionVault`

Impact The `depositFee` functionality is either dead code, or it corresponds to unimplemented functionality that may be important to the protocol.

Recommendation Implement or remove the corresponding functionality.

Developer Response The developers plan to implement the `depositFee` feature in the future, but it has not been prioritized yet. They have removed the `depositFee` and related code from the current version.

4.1.8 V-INCP-VUL-008: depositExta fails silently if totalAmountToWithdraw exceeds vault asset balance

Severity	Warning	Commit	11af9fb
Type	Logic Error	Status	Open
File(s)	EigenLayerHandler.sol		
Location(s)	depositExta()		
Confirmed Fix At			

The `depositExta()` method can be called by the `operator` to deposit any asset tokens held by the vault into the vault's EigenLayer strategy contract. However, if the `totalAmountToWithdraw` is greater than or equal to the vault's asset token balance, then the `depositExta()` method will return without doing anything. There is no way for the `operator` to determine whether the "extra" asset tokens were deposited or not.

```
/// @dev deposits extra assets into strategy
function depositExta() external onlyOperator {
    uint256 vaultBalance = totalAssets();
    uint256 toWithdrawAmount = totalAmountToWithdraw;
    if (vaultBalance <= toWithdrawAmount) {
        return;
    }

    strategyManager.depositIntoStrategy(
        strategy,
        _asset,
        vaultBalance - toWithdrawAmount
    );

    emit DepositedToEL(vaultBalance - toWithdrawAmount);
}
```

Snippet 4.11: Definition of `depositExta()`

Impact The `operator` may expect `depositExta()` to perform some sort of action with EigenLayer, but `depositExta()` itself will not provide any way to know whether it is successful. Instead, the `operator` will have to compare the asset token balance of the vault before and after calling `depositExta()`.

Recommendation Add a way for the operator to determine if `depositExta()` actually took an action. For example, `depositExta()` could be modified to return the number of EigenLayer strategy shares that were created as a result of the deposit (this would be the return value of `strategyManager.depositIntoStrategy()`). A return value of `0` would indicate that no funds were deposited.

4.1.9 V-INCP-VUL-009: Potential reentrancy risk in claimCompletedWithdrawals

Severity	Warning	**Commit**	11af9fb
Type	Reentrancy	**Status**	Fixed
File(s)	`EigenLayerHandler.sol`		
Location(s)	claimCompletedWithdrawals()		
Confirmed Fix At	9997778		

The `InceptionVault` contract has several methods that use the OpenZeppelin `ReentrancyGuard` and `nonReentrant` modifiers to mitigate reentrancy attacks. These methods include `withdraw()`, `deposit()`, and `withdrawFromEL()`. However, the `claimCompletedWithdrawals()` method does not have a `nonReentrant` modifier.

Impact Based on the implementations of the methods marked with `nonReentrant` modifiers mentioned above, it appears that the `nonReentrant` modifiers are meant to mitigate against reentrancy attacks that may launched by callbacks when invoking EigenLayer code. Since `claimCompletedWithdrawals()` does not have a `nonReentrant` modifier, it may be possible for a reentrancy attack launched from the methods mentioned above to invoke `claimCompletedWithdrawals()`. However, we have not identified any concrete reentrancy attacks involving `claimCompletedWithdrawals()` during the audit.

Recommendation For additional guarantees against reentrancy attacks, add the `nonReentrant` modifier to `claimCompletedWithdrawals()`.

4.1.10 V-INCP-VUL-010: setDepositFee percentage not validated

Severity	Warning	**Commit**	11af9fb
Type	Data Validation	**Status**	Fixed
File(s)	`InceptionVault.sol`		
Location(s)	setDepositFee()		
Confirmed Fix At	9997778		

The `setDepositFee()` method can be called by the `owner` of an `InceptionVault` to change the deposit fee percentage. However, the new deposit fee is not validated to be less than 100%.

```
function setDepositFee(uint256 newDepositFee) external onlyOwner {
    emit DepositFeeChanged(depositFee, newDepositFee);
    depositFee = newDepositFee;
}
```

Snippet 4.12: Definition of `setDepositFee()`

Impact The `depositFee` functionality is currently not used in the code but may be used in the future (see V-INCP-VUL-007). However, if it is implemented in the future, a deposit fee of 100% or above could cause `deposit()` to always revert.

Recommendation Change `setDepositFee()` to revert if the `newDepositFee` is 100% or higher.

Developer Response The developers removed the `depositFee` variable and the `setDepositFee()` method.

4.1.11 V-INCP-VUL-011: Strategy deposit return value ignored

Severity	Info	**Commit**	11af9fb
Type	Missing/Incorrect Eve	**Status**	Partially Fixed
File(s)	`EigenLayerHandler.sol`		
Location(s)	_depositAssetToEL(), depositExta()		
Confirmed Fix At	935c00f		

The `EigenLayerHandler` contract uses `IStrategyManager.depositIntoStrategy()` to transfer ERC20 tokens to an EigenLayer `IStrategy` contract in exchange for shares. The function returns the number of shares that were created and credited to the vault. There are two uses of `IStrategyManager.depositIntoStrategy()` within the `EigenLayerHandler`, and both uses ignore the return value of the function.

```
/// @dev deposits asset to the corresponding strategy
function _depositAssetToEL(uint256 amount) internal {
    // deposit the asset to the appropriate strategy
    strategyManager.depositIntoStrategy(strategy, _asset, amount);

    emit DepositedToEL(amount);
}
```

Snippet 4.13: Definition of `EigenLayerHandler._depositAssetToEL()`

```
/// @dev deposits extra assets into strategy
function depositExta() external onlyOperator {
    uint256 vaultBalance = totalAssets();
    uint256 toWithdrawAmount = totalAmountToWithdraw;
    if (vaultBalance <= toWithdrawAmount) {
        return;
    }

    strategyManager.depositIntoStrategy(
        strategy,
        _asset,
        vaultBalance - toWithdrawAmount
    );

    emit DepositedToEL(vaultBalance - toWithdrawAmount);
}
```

Snippet 4.14: Definition of `EigenLayerHandler.depositExta()`

Impact While should currently have no security impact, it may be useful to track the obtained strategy shares for debugging or bookkeeping purposes.

Recommendation For debugging purposes, the return value of `strategyManager.depositIntoStrategy()` should be logged in the `DepositedToEL` event in both locations.

Developer Response The developers changed the `emit` statement so that it logs the asset value of the shares instead of `vaultBalance - toWithdrawAmount`, though the auditors would recommend that both quantities be logged.

4.1.12 V-INCP-VUL-012: Typo in depositExta function name

Severity	Info	**Commit**	11af9fb
Type	Maintainability	**Status**	Fixed
File(s)	`EigenLayerHandler.sol`		
Location(s)	depositExta()		
Confirmed Fix At	9997778		

The name of the `depositExta()` method is missing an "r" and should probably be named `depositExtra()`.

Impact The `depositExta()` method can only be called by the `operator` of the vault. It may be likely for the `operator` to attempt to call `depositExtra()` by mistake, but this will simply cause a reverting transaction.

Recommendation Fix the typo.

4.1.13 V-INCP-VUL-013: InceptionAssetsHandler does not extend its interface

Severity	Info	Commit	11af9fb
Type	Maintainability	Status	Fixed
File(s)	`InceptionAssetsHandler.sol`		
Location(s)	InceptionAssetsHandler		
Confirmed Fix At	9997778		

The `InceptionAssetsHandler` contract does not extend its corresponding interface `IInceptionAssetHandler`.

Impact There is currently no security impact resulting from this issue, but it may cause issues in other contracts if `IInceptionAssetHandler` deviates from the `InceptionAssetsHandler` contract.

Recommendation Make `InceptionAssetsHandler` extend from `IInceptionAssetsHandler`.

4.1.14 V-INCP-VUL-014: redeem() parameter should be named 'receiver'

Severity	Info	**Commit**	11af9fb
Type	Maintainability	**Status**	Fixed
File(s)	`InceptionVault.sol`		
Location(s)	redeem()		
Confirmed Fix At	9997778		

From the end-user perspective, withdrawing assets from the `InceptionVault` is a three-part process:

1. First, a user that holds some quantity of `InceptionToken` (i.e., shares) calls the `withdraw()` function, specifying the number of shares to withdraw and the `recipient` address that should receive the equivalent amount of the ERC20 asset when the withdraw completes. The `InceptionVault` stores information about the pending withdraw in the `_claimerWithdrawals` mapping, keyed on the recipient.
2. Second, the user must wait for the funds to be withdrawn from EigenLayer to the `InceptionVault`.
3. Finally, the user (or any arbitrary address on their behalf) calls the `redeem()` function, specifying the recipient address provided during the `withdraw()` call. The `InceptionVault` uses that address to lookup the pending withdraw from `_claimerWithdrawals` and completes the withdraw by transferring the appropriate amount of ERC20 assets to the recipient.

```
function withdraw(uint256 iShares, address receiver) external nonReentrant {
    if (iShares == 0) {
        revert NullParams();
    }
    if (receiver == address(0)) {
        revert NullParams();
    }
    address claimer = msg.sender;
    uint256 amount = Convert.multiplyAndDivideFloor(iShares, 1e18, ratio());
    require(
        amount >= minAmount,
        "InceptionVault: amount is less than the minimum withdrawal"
    );
    // burn Inception token in view of the current ratio
    inceptionToken.burn(claimer, iShares);

    // update global state and claimer's state
    totalAmountToWithdraw += amount;
    Withdrawal storage request = _claimerWithdrawals[receiver];
    request.amount += _getAssetReceivedAmount(amount);
    request.receiver = receiver;
    request.epoch = epoch;

    emit Withdraw(claimer, receiver, claimer, amount, iShares);
}
```

Snippet 4.15: Definition of `InceptionVault.withdraw()`

However, the recipient address parameter of the `redeem()` function is actually named `claimer`.

```
function redeem(address claimer) public nonReentrant {
    require(
        isAbleToRedeem(claimer),
        "InceptionVault: claimer is not able to claim"
    );
    Withdrawal storage request = _claimerWithdrawals[claimer];
    uint256 amount = request.amount;
    totalAmountToWithdraw -= amount;

    delete _claimerWithdrawals[claimer];
    _transferAssetTo(claimer, amount);

    emit Redeem(msg.sender, claimer, amount);
}
```

Snippet 4.16: Definition of `InceptionVault.redeem()`

It would be better to rename the `claimer` parameter in `redeem()` to `receiver`, as this would be more consistent with the usage of `claimer` and `receiver` in the `withdraw()` function.

Impact Inconsistency in naming can cause confusion, leading to introduction of bugs when making code modifications or misuse of the API by end-users.

Recommendation The parameter of the `redeem()` function should be named `receiver`. Documentation should be updated accordingly.