Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

InceptionLRT

Veridise Inc.

Mar. 25, 2025

► **Prepared For:**

Inceptive Labs
https://www.inceptivelabs.xyz/

► **Prepared By:**

Bryan Tan
Benjamin Mariano

► **Contact Us:**

contact@veridise.com

► **Version History:**

Mar. 25, 2025	V3
Mar. 24, 2025	V2
Mar. 14, 2025	V1
Mar. 10, 2025	Initial Draft

Contents

1 Executive Summary

From Feb. 17, 2025 to Mar. 7, 2025, Inceptive Labs engaged Veridise to conduct a security assessment of their InceptionLRT project. The security assessment covered an implementation of an ERC-4626 vault smart contract that restakes deposited funds in various third-party protocols. Veridise conducted the assessment over 6 person-weeks, with 2 security analysts reviewing the project over 3 weeks on commit `0ebd568`. The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. The security assessment covered a vault smart contract, which will be referred to as the "Inception Vault" in this report, with the following major features:

- Any user may deposit a specific type of ERC-20 token called the *asset* token into the vault. In return, they will be granted an equivalent amount of *share* tokens (specific to the vault) that represent ownership over a portion of the total assets deposited in the vault.
- A trusted role called the *vault operator* may *delegate* a portion of the vault's asset balance to various third-party staking protocols, with the expectation that the delegated assets will increase in value over time. The operator can later *claim* back the delegated assets from the third-party protocols and make them available for withdrawal by the vault users.
- Users may burn their shares to withdraw asset tokens from the vault. This is achieved in one of two ways: (1) through a *flash withdrawal*, where the requested assets are immediately transferred to the user in the same transaction, minus a fee; or (2) through an asynchronous withdrawal, where the user must request a withdrawal in one transaction, wait for the vault operator to claim assets back from third-party protocols, and finally complete the withdrawal in a later transaction. In the case of asynchronous withdrawal, no fee is charged.
- The vault implements the ERC-4626 vault token standard.

The assessment also covered implementations of *adapter* smart contracts which implement interactions between the Inception Vault and the third-party protocols, primarily the Eigen Layer, Symbiotic, and Mellow protocols.

Code Assessment. The InceptionLRT developers provided the source code of the InceptionLRT contracts for the code review. The source code appears to be mostly original code written by the InceptionLRT developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise security analysts understanding of the code, the InceptionLRT developers shared high-level documentation for their protocol along with descriptions of the intended user flows.

The source code contained a test suite, which the Veridise security analysts noted covered most of the intended behaviors of the protocol.

Summary of Issues Detected. The security assessment uncovered 27 issues, 5 of which are assessed to be of medium severity by the Veridise analysts. These issues included a potential denial-of-service issue on redeems (V-IVC-APP-VUL-001), missing unit conversions (V-IVC-VUL-003), incorrect deposit amount conversion that can lead to locked funds (V-IVC-VUL-005), and incorrect calculation of pending withdrawals from Eigen Layer (V-IVC-VUL-003). The Veridise analysts also identified 8 low-severity issues, including lack of compliance with the relevant ERC standards (V-IVC-VUL-006), potentially locked funds (V-IVC-VUL-011), and unused token approvals (V-IVC-VUL-012) as well as 10 warnings and 4 informational findings.

Among the 27 issues reported, all have been acknowledged by the Inceptive Labs. Of theses 27 acknowledged issues, Inceptive Labs has fixed 22 issues and indicated the remaining issues are either intended behavior or are handled by logic outside the scope of this audit. Their fixes include full fixes for 5 medium-severity issues and 7 low-severity issues.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the security of InceptionLRT.

Additional review for out-of-scope code. A number of security-critical portions of the codebase were out-of-scope for this review. First, the `AdapterHandler` contract is out-of-scope but contains much of the important logic for maintaining the Inception Vault state and interacting with adapters. Second, the ratio feed is out-of-scope but is used to determine the ratio of Inception tokens to the underlying asset; determining an appropriate ratio is critical to the ability of the vault to behave as intended and issues with this process could cause large losses by both the protocol and users. Finally, there is an intention to have some sort of referral program, but no code related to this was in scope. As mentioned in V-IVC-VUL-016, failure to properly handle referrals could cause issues with the protocol.

Ratio feed behavior. While the ratio feed logic was out-of-scope for this audit, the developers did share the implementation of the feed with the team to help facilitate the analysts' understanding of the protocol. While reading the code, Veridise analysts noticed a few issues that could lead to broader issues in the protocol. First, the ratio feed allows a trusted user to set the ratio for a given vault, modulo a few small checks. This gives that trusted user immense power to impact the proper functioning of the protocol. Second, one of the small checks on ratio updates is that the value of inception tokens always increases over time. However, this may not be true in the case of slashing.

Eigen Layer rewards coordinator. The Eigen Layer adapter uses the rewards coordinator to enable some other trusted user to claim on behalf of the Eigen Layer adapter. Nowhere in the Eigen Layer adapter contract does it specify that rewards claimed on behalf of the Eigen Layer adapter should go to the Inception Vault. If rewards go somewhere else, the rewards will be lost to depositors in the vault. Even if rewards go back to the Eigen Layer adapter, due to issues like V-IVC-VUL-011, those rewards may be stuck and will not be returned to depositors in the vault.

Potential lack of interoperability with Mellow protocol. A number of the external calls to Mellow vaults in the Mellow adapter assume that the Mellow vaults implement interfaces that are inconsistent with what is described in the Mellow documentation. It appears that the vaults are in the process of being updated, so it is possible they will support the interfaces used by InceptionLRT in the future. However, given that the interfaces are changing, special care should

be taken to ensure proper interoperability with the Mellow vaults once they reach a more fixed state.

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	`0ebd568`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Feb. 17–Mar. 7, 2025	Manual & Tools	2	6 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	5	5	5
Low-Severity Issues	8	8	7
Warning-Severity Issues	10	10	6
Informational-Severity Issues	4	4	4
TOTAL	27	27	22

Table 2.4: Category Breakdown.

Name	Number
Logic Error	10
Data Validation	4
Frontrunning	2
Denial of Service	2
Usability Issue	2
Access Control	2
Maintainability	2
Authorization	1
Reentrancy	1
Gas Optimization	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of InceptionLRT's smart contracts. During the assessment, the security analysts aimed to answer questions such as:

- Can one user steal another user's deposit?
- Is the vault vulnerable to price manipulation attacks?
- Can funds become locked in the protocol?
- Is the protocol vulnerable to Denial-of-Service attacks that can prevent expected interaction by users (especially on withdrawals).
- Are conversions between various asset types computed appropriately?
- Does the project contain common DeFi vulnerabilities such as reentrancy?
- Does the protocol have significant centralization concerns?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a combination of human experts and automated program analysis & testing tools. In particular, the security assessment was conducted with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged Veridise's 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, unused return values, and uninitialized variables.
- *Fuzzing/Property-based Testing.* Security analysts leveraged fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, the desired behavior of the protocol was formulated as [V] specifications and then tested using Veridise's fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this security assessment is limited to the following paths:

- `projects/vaults/contracts/adapters`
- `projects/vaults/contracts/interfaces/adapters`
- `projects/vaults/contracts/interfaces/eigenlayer-vault/IInceptionVault_EL.sol`
- `projects/vaults/contracts/interfaces/symbiotic-vault/ISymbioticHandler.sol`
- `projects/vaults/contracts/interfaces/symbiotic-vault/IInceptionVault_S.sol`
- `projects/vaults/contracts/lib/InceptionLibrary.sol`
- `projects/vaults/contracts/vaults/Symbiotic/InceptionVault_S.sol`
- `projects/vaults/contracts/vaults/Symbiotic/vault_e2/InVault_S_E2.sol`
- `projects/vaults/contracts/tokens/InceptionToken.sol`

Methodology. Veridise security analysts reviewed the reports of previous audits for InceptionLRT, inspected the provided tests, and read the InceptionLRT documentation. They then began a review of the code assisted by both static analyzers and automated testing.

During the security assessment, the Veridise security analysts regularly met with the InceptionLRT developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

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

The likelihood of a vulnerability is evaluated according to the 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

The impact of a vulnerability is evaluated according to the 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 Trust Model

4.1 Privileged Roles.

Roles. This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensititivy. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive *emergency* roles may be required to perform actions quickly based on real-time monitoring, while *non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assume that the role operators perform their responsiblities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Highly-privileged roles (both emergency and non-emergency powers):
  - For each of the contracts reviewed, the contract owners have a variety of abilities which can significantly affect the proper functioning of the protocol. For the vault, these abilities include setting the ratio feed (which is responsible for determining vault token value), setting various fees and minimal amounts for deposits/withdrawals, setting the operator (discussed more in the next bullet point), and setting the trusted adapters that will manage vault funds. For the adapter contracts, these abilities include setting the trustee manager (who is trusted to facilitate interactions with third party protocols), the pointer to the inception vault (which could be abused to steal funds), as well as changing allocation strategies and pausing/unpausing the contracts.
  - For the vault contract, the operator role is in charge of delegating, undelegating, and claiming funds from the trusted set of adapters. This can affect how funds are managed – this role could significantly impact the protocol's proper functioning by choosing how funds are delegated or not anmd how frequently (or even if) funds are claimed and used to complete requested withdrawals.

Operational Recommendations. Highly-privileged, non-emergency operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-IVC-VUL-001	Frontrunning can lead to slippage on flash . . .	Medium	Fixed
V-IVC-VUL-002	DoS on longterm redeems	Medium	Fixed
V-IVC-VUL-003	Missing unit conversions in adapters	Medium	Fixed
V-IVC-VUL-004	pendingWithdrawalAmount always . . .	Medium	Fixed
V-IVC-VUL-005	Potentially incorrect amount deposited . . .	Medium	Fixed
V-IVC-VUL-006	Lack of compliance with ERC4626 and . . .	Low	Fixed
V-IVC-VUL-007	IMellowAdapter.claimPending always . . .	Low	Fixed
V-IVC-VUL-008	IMellowAdapter._delegate does not . . .	Low	Fixed
V-IVC-VUL-009	Initialization bugs in IMellowAdapter	Low	Fixed
V-IVC-VUL-010	InceptionEigenAdapterWrap does not . . .	Low	Fixed
V-IVC-VUL-011	Potential locked funds issue in adapters	Low	Acknowledged
V-IVC-VUL-012	Unused approval on delegation	Low	Fixed
V-IVC-VUL-013	Potential lost funds on claim with . . .	Low	Fixed
V-IVC-VUL-014	Deposit and withdrawal rates can be set to . . .	Warning	Fixed
V-IVC-VUL-015	DoS with large amounts of deposited funds	Warning	Acknowledged
V-IVC-VUL-016	Referral codes can be frontrun and spammed	Warning	Acknowledged
V-IVC-VUL-017	Unsafe approve in . . .	Warning	Fixed
V-IVC-VUL-018	Missing claimer validation in . . .	Warning	Fixed
V-IVC-VUL-019	No access controls on . . .	Warning	Fixed
V-IVC-VUL-020	Minor validation issues in . . .	Warning	Fixed
V-IVC-VUL-021	Dangerous blanket approval for strategy . . .	Warning	Acknowledged
V-IVC-VUL-022	Some calls to claim are not automatically . . .	Warning	Acknowledged
V-IVC-VUL-023	Theoretical reentrancy concern on deposits	Warning	Fixed
V-IVC-VUL-024	Gas Optimization	Info	Fixed
V-IVC-VUL-025	Unused program constructs	Info	Fixed
V-IVC-VUL-026	Typos, incorrect comments, and small code . . .	Info	Fixed
V-IVC-VUL-027	Unused return values	Info	Fixed

5.1 Detailed Description of Issues

5.1.1 V-IVC-VUL-001: Frontrunning can lead to slippage on flash withdrawals

Severity	Medium	**Commit**	0ebd568
Type	Frontrunning	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	flashWithdraw		
Confirmed Fix At	c931b10		

In the function `flashWithdraw`, there is no way for an EOA caller (or naively written smart contract) to check the amount of assets received. Thus, if large fluctuations occur in prices (either intentionally via frontrunning or unintentionally due to the natural behavior of the vault), large amounts of funds could potentially be lost by a user.

Impact Users potentially could lose large sums of money if the price of the asset relative to the inception token changes dramatically before a withdrawal.

Recommendation Add in some sort of slippage protection for users.

Developer Response The developers added a third parameter `minOut` for `flashWithdraw` that allows a user to specify the minimum amount of returns they will accept.

5.1.2 V-IVC-VUL-002: DoS on longterm redeems

Severity	Medium	**Commit**	0ebd568
Type	Denial of Service	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	isAbleToRedeem, redeem		
Confirmed Fix At	643e5fe		

The function `isAbleToRedeem` determines which asynchronous withdrawals by the given "claimer" are ready to be redeemed by iterating through all of the withdrawals between the earliest un-claimed epoch of the claimer and the latest epoch, as shown in the snippet below.

```
for (uint256 i = genRequest.epoch; i < epoch; ++i) {
    if (claimerWithdrawalsQueue[i].receiver == claimer) {
        able = true;
        availableWithdrawals[index] = i;
        ++index;
    }
}
```

Snippet 5.1: Snippet from `isAbleToRedeem()`

This loop includes a storage read on every iteration and could become very expensive if there is a large difference between the current epoch and the epoch of the claimer's earliest withdrawal request.

It should be noted that a similar concern exists in the `redeem` function, which iterates through all withdrawals by a given user to allow them to claim their funds. This loop is complicated with a number of reads and writes to storage. If the user has a large number of pending withdrawals, this too could become prohibitively expensive.

Impact This could make it prohibitively expensive for a user to withdraw their funds.

Recommendation Avoid prohibitive gas costs on longterm withdrawals.

Developer Response The developers acknowledged the issue and have made some efforts to reduce the cost of a user to redeem. In particular, they added a restriction that the maximum epochs between any two queued withdrawals is at most 20 and use this restriction to limit the iterations of the loop in `isAbleToRedeem`. They also introduce a new variable `withdrawalsBuffer` in the `redeem` function to save gas by avoiding multiple sload/sstore calls.

While these will help to mitigate the concerns, it is still possible that the combined loops from `redeem` and `isAbleToRedeem` could still become quite costly. The developers mentioned they have plans for an improvement that they claim will fix this issue completely.

5.1.3 V-IVC-VUL-003: Missing unit conversions in adapters

Severity	Medium	Commit	0ebd568
Type	Logic Error	Status	Fixed
File(s)	IMellowAdapter.sol, InceptionEigenAdapter.sol		
Location(s)	_delegate(), delegate()		
Confirmed Fix At	da2a145, a1da92f		

Many of the adapter functions such as `delegate()` and `withdraw()` are supposed to return amounts of assets. However, for some of the adapter implementations, the asset amount is not returned correctly or returned in the wrong units:

- `IMellowAdapter._delegate()` directly returns the value of `IEthWrapper(ethWrapper).deposit()`, which is the number of shares obtained in response to the deposit. Note that a different function on the adapter, `_delegateAuto()`, will call `lpAmountToAmount` to convert the shares back to assets.

```
// Relevant documentation on IEthWrapper.deposit:
   /**
     * @notice Deposits a specified ‘amount‘ of the ‘depositToken‘ into the provided ‘vault‘, crediting the specified ‘receiver‘ with shares.
     * ...
     * @return shares The amount of vault shares received after the deposit.
     * ...
     */

// Definition of _delegate
function _delegate(
    address mellowVault,
    uint256 amount,
    address referral
) internal returns (uint256 depositedAmount) {
    _asset.safeTransferFrom(msg.sender, address(this), amount);
    IERC20(_asset).safeIncreaseAllowance(address(ethWrapper), amount);
    return
        IEthWrapper(ethWrapper).deposit(
            address(_asset),
            amount,
            mellowVault,
            address(this),
            referral
        );
}
```

Snippet 5.2: Relevant snippets

- `InceptionEigenAdapterWrap.delegate()` returns the result of calling the EigenLayer strategy manager's `depositIntoStrategy()` function, which is the number of shares obtained rather than the number of assets deposited.

```
if (amount > 0 && operator == address(0)) {
    // transfer from the vault
    _asset.safeTransferFrom(msg.sender, address(this), amount);
    IWStethInterface(address(_asset)).unwrap(amount);
```

```
    // deposit the asset to the appropriate strategy
    return
        _strategyManager.depositIntoStrategy(
            _strategy,
            IWStethInterface(address(_asset)).stETH(),
            amount
        );
}
```

Snippet 5.3: Snippet from `InceptionEigenAdapterWrap.delegate()` where the shares are returned

- `InceptionEigenAdapterWrap.withdraw()` returns the EigenLayer strategy's underlying token, which is the stETH token. However, the asset token of the `InceptionEigenAdapterWrap` contract is the wstETH token (as can be inferred from `delegate()`).
- Similarly, `InceptionEigenAdapterWrap.getDeposited()` and `getTotalDeposited()` return stETH token amounts instead of the asset (wstETH) token amounts.

Impact The `delegate()` and `withdraw()` functions mentioned above will not correctly return the amount of deposited assets. The vault's `delegate()` method (inherited from `AdapterHandler`) will call the adapter's `delegate()` but not use its return value; thus, this has no impact on the vault. However, a "trustee manager" of the adapter that uses the return value of `delegate()` may end up with an incorrect value.

Furthermore, the vault will use the return value of `withdraw()` to determine how many assets should count towards fulfilling queued withdraw requests. If the return value is larger than the actual amount of assets transferred to the vault, then the vault's claimers may be able to claim more funds than are actually available for withdrawal. If the return value is smaller, then the asset amounts will count towards the flash capacity rather than towards fulfillment of the withdraw queue.

Recommendation Correct the locations mentioned above to return the correct amounts and/or units.

Developer Response The developers fixed the issues as suggested.

5.1.4 V-IVC-VUL-004: pendingWithdrawalAmount always returns 0

Severity	Medium	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	pendingWithdrawalAmount()		
Confirmed Fix At	da2a145		

The `pendingWithdrawalAmount()` function of `InceptionEigenAdapterWrap` is hardcoded to return 0. However, this function is supposed to return the amount of assets that the adapter currently has queued for withdrawal.

```
function pendingWithdrawalAmount()
    public
    pure
    override
    returns (uint256 total)
{
    return 0;
}
```

Snippet 5.4: Definition of `pendingWithdrawalAmount()`

Impact When the adapter has shares queued for withdrawal from an EigenLayer strategy, those assets will not counted as part of the `inactiveBalance()` of the adapter. This means that the `getTotalDeposited()` of the vault will be lower than intended, which will decrease the target capacity and therefore the deposit bonus and flash withdraw fee.

Recommendation Change `pendingWithdrawalAmount()` to return the asset value of the EigenLayer shares queued for withdrawal.

Developer Response The developers added a variable to track the pending withdrawals in terms of the shares of the underlying strategy and return the asset value of pending withdrawals by converting this saved value.

5.1.5 V-IVC-VUL-005: Potentially incorrect amount deposited into EigenLayer strategy

Severity	Medium	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	deposit()		
Confirmed Fix At	da2a145		

A trustee of the `InceptionEigenAdapterWrap` can call the `delegate()` function to deposit assets delegated to an Inception vault into an EigenLayer strategy. Based on the implementation of the adapter contract, the asset is assumed to be WstETH, whereas the underlying token of the strategy is stETH.

Specifically, the `delegate()` function will transfer `amount` of WstETH tokens from the Inception vault to the adapter. Those tokens will then be converted to stETH, after which `amount` of stETH tokens will be deposited into the strategy.

```
if (amount > 0 && operator == address(0)) {
    // transfer from the vault
    _asset.safeTransferFrom(msg.sender, address(this), amount);
    IWStethInterface(address(_asset)).unwrap(amount);
    // deposit the asset to the appropriate strategy
    return
        _strategyManager.depositIntoStrategy(
            _strategy,
            IWStethInterface(address(_asset)).stETH(),
            amount
        );
}
```

Snippet 5.5: Snippet from `delegate()` that deposits into an EigenLayer strategy.

This conversion between WstETH and stETH is not always 1:1, as indicated by the existence of the function `IWStethInterface.getStETHByWstETH()`. However, the call to `depositIntoStrategy()` uses the same `amount` variable to determine how much *stETH* to deposit into the strategy. Note that the `unwrap()` function presumably returns the stETH tokens converted, but this return value is never used by `delegate()`.

Impact The conversion from stETH will *usually* result in more funds, and thus, some funds will be left over on this adapter contract. However, with slashing, it is possible for the conversion to result in less funds, likely meaning this would revert (or possibly some other user's funds are stolen from the adapter).

Recommendation Change the call to `depositIntoStrategy()` to use the return value of `unwrap()` instead of `amount`.

Developer Response The developers used the return value from `unwrap()` as suggested.

5.1.6 V-IVC-VUL-006: Lack of compliance with ERC4626 and ERC7540 standards

Severity	Low	**Commit**	0ebd568
Type	Usability Issue	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	See issue description		
Confirmed Fix At	2c9e879, 11eb9a7, 7657cf2		

The developers have stated the intention that the Inception vault should implement the ERC4626 vault standard. However, we identified a number of ways in which the current implementation is not compliant with the expected interface and behavior of that standard:

1. ERC4626 requires the `mint()` method to guarantee that at least `shares` are minted, but there is no such check in the current implementation of `mint`.
2. ERC4626 specifies that for the function `maxMint`, it should be assumed that the caller has an infinite amount of funds. This is not the case in the current implementation.
3. The function `maxMint` is intended to compute the maximum mint for the *receiver*, not the *caller* as is computed in the current implementation.
4. `previewDeposit` is not consistent with the actual behavior of `deposit`, which will revert if the amount is below the minimum threshold for depositing. The same is true of `previewRedeem`.
5. `withdraw` is supposed to have three arguments and immediately transfer assets. Instead, the current `withdraw` implementation has only two arguments and acts more like an asynchronous withdrawal from ERC7540.
6. The vault does not extend from ERC4626.
7. The vault does not extend ERC20 and use it to represent shares.

Impact Developers writing applications integrating with the Inception Vault may falsely assume that it implements ERC4626. This could lead to confusion, user difficulties, or issues arising in code updates from misunderstanding the behavior of the code.

Recommendation Properly implement the ERC4626 standard. Furthermore, if you intend to support asynchronous withdrawals, use the predefined ERC7540 standard that supports asynchronous withdrawals from vaults.

Developer Response The developers addressed bullet points (1) through (4) as suggested. For points (5) through (7), they provided the following explanation: "We are making our vault as closely compatible to 4626 as possible. Since our contracts are deployed, we will be having hard time to extend from ERC20 or ERC4626. The current implementation resembles 7540 in-terms of withdrawals and 7575 in-terms of separate shares token. Due to Mellow being in multiple upgrade processes from past 2-3 months, it has been difficult to stick to 1 standardized approach."

While reviewing the fix for `mint` for point (1), the Veridise auditors are concerned the fix might make it difficult to interact with the contract. In particular, the call to `convertToAssets` rounds *down*, meaning that the assets deposited via `_deposit` may result in fewer shares than specified

by the user being minted, purely due to rounding. Consequently, the current fix will introduce a possibility of reverting.

This was an instance of another ERC4626 compliance concern:

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

From: `https://eips.ethereum.org/EIPS/eip-4626#security-considerations`

The second case applies to the original implementation of `mint`, which rounds down instead of up.

To address this additional concern, the developers introduced a `previewMint` function which is used in `mint` and performs a ceiling instead of a floor.

5.1.7 V-IVC-VUL-007: IMellowAdapter.claimPending always returns 0

Severity	Low	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	IMellowAdapter		
Location(s)	claimPending()		
Confirmed Fix At	d901494		

The `claimpending()` method is used to transfer claimable assets from the mellow vaults to the adapter contract. It has a single return value which is default initialized to 0 but never updated. Furthermore, each call to `claim()` will return the number of assets claimed, but the return value is never used. Based on what the code is doing, it is likely that the return value of `claimPending` is meant to be the sum of all of the `claim()`ed assets.

```
function claimPending() external returns (uint256) {
    for (uint256 i = 0; i < mellowVaults.length; i++) {
        IMellowSymbioticVault(address(mellowVaults[i])).claim(
            address(this),
            address(this),
            type(uint256).max
        );
    }
}
```

Snippet 5.6: Definition of `claimPending()`

Impact Calls to `claimPending()` will always have a return value of 0.

Recommendation Change the code to sum up the return values of `claim()` and return the sum as the result of `claimPending()`.

Developer Response The developers actually changed `claimPending()` to be a private function that is invoked by `claim()` and simply removed the return value entirely.

5.1.8 V-IVC-VUL-008: IMellowAdapter._delegate does not validate vault

Severity	Low	**Commit**	0ebd568
Type	Data Validation	**Status**	Fixed
File(s)	`IMellowAdapter.sol`		
Location(s)	_delegate()		
Confirmed Fix At	cc5855d		

The `IMellowAdapter`'s `_delegate()` function does not check that the user-provided `mellowVault` is contained in the array of `mellowVaults` in the contract's storage.

Impact A vault operator or adapter trustee manager may accidentally delegate assets to a vault that is not supported by the adapter. These assets will not be tracked by the adapter's bookkeeping, which could result in accounting issues for the protocol.

Recommendation Add a `require` statement to `_delegate` that ensures that specified `mellowVault` is contained in the `mellowVaults` array.

Developer Response The developers added a check to ensure the provided vault is contained in the `mellowVaults` array.

5.1.9 V-IVC-VUL-009: Initialization bugs in IMellowAdapter

Severity	Low	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	IMellowAdapter.sol		
Location(s)	initialize()		
Confirmed Fix At	56b7ec3		

The `initialize()` function of `IMellowAdapter` has a few minor bugs:

- The caller can provide an array of Mellow vault addresses to insert into the contract storage. Based on the behavior of the `addMellowVault()` function, these addresses must be unique; however, there is no such uniqueness check in `initialize()`.

```
// In initialize():
for (uint256 i = 0; i < _mellowVault.length; i++) {
    mellowVaults.push(_mellowVault[i]);
}

// In addMellowVault():
if (mellowVault == address(0)) revert ZeroAddress();
for (uint8 i = 0; i < mellowVaults.length; i++) {
    if (mellowVault == address(mellowVaults[i])) revert AlreadyAdded();
}
mellowVaults.push(IMellowVault(mellowVault));
```

Snippet 5.7: Comparison of the lines of code in `initialize()` and `addMellowVault()` that add vault(s). Note the lack of uniqueness check in `initialize(`

- The `allocations` mapping stores the "weight" of each vault when delegating funds in the `_delegateAuto()` function, but `initialize()` does not set the allocations of the initial vaults.

Impact

- If the deployer of the adapter accidentally specifies duplicate vaults, the view methods in the contract may double-count some of the asset amounts. Furthermore, when a trustee calls `delegate()` with auto-delegation, more assets may be delegated to duplicate vaults than have been provided by the caller, leading to a revert.
- The "auto delegation" feature in `_delegateAuto()` will not work as intended, because `_delegateAuto()` function will not deposit assets to vaults that have an allocation value of zero. Unused assets will simply be sent back to the Inception vault. Instead, the owner must manually set the allocation of each vault with the `changeAllocation()` function before `_delegateAuto()` will work.

```
function _delegateAuto(
    uint256 amount,
    address referral
) internal returns (uint256 depositedAmount) {
    uint256 allocationsTotal = totalAllocations;
    _asset.safeTransferFrom(msg.sender, address(this), amount);

```

```
8       for (uint8 i = 0; i < mellowVaults.length; i++) {
9           uint256 allocation = allocations[address(mellowVaults[i])];
10          if (allocation > 0) {
11              uint256 localBalance = (amount * allocation) / allocationsTotal;
```

Snippet 5.8: Snippet in `_delegateAuto()` that computes how the delegated amount should be distributed among the the vaults.

Recommendation

- Add a check to `initialize()` that ensures duplicate vaults cannot be added.
- Consider inserting code to initialize the allocations to have some default distribution of weights. For example, each vault could initially have the same weight.

Developer Response The developers added the duplicate check and assigned a default weight of `1` to all of the initial vaults.

5.1.10 V-IVC-VUL-010: InceptionEigenAdapterWrap does not implement pausing functionality

Severity	Low	**Commit**	0ebd568
Type	Access Control	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	delegate(), withdraw(), claim()		
Confirmed Fix At	16d6f5d		

Unlike the other adapter implementations in the project, `InceptionEigenAdapterWrap` does not implement pausing functionality on its `delegate()`, `withdraw()`, and `claim()` functions.

Impact The operator of the contract cannot pause the mentioned functions during an emergency.

Recommendation Add `whenNotPaused` to the specified functions. Note, however, that this will also increase the centralization of the protocol.

Developer Response The developers added the `whenNotPaused` modifier as suggested.

5.1.11 V-IVC-VUL-011: Potential locked funds issue in adapters

Severity	Low	**Commit**	0ebd568
Type	Logic Error	**Status**	Acknowledged
File(s)	`InceptionEigenAdapter.sol`, `ISymbioticAdapter.sol`		
Location(s)	N/A		
Confirmed Fix At	N/A		

For each of the Eigen Layer adapter and the Symbiotic adapter, there is no function that can recover any asset balance remaining on the contract and send them to the Inception vault. The `claim()` functions on those adapters will only send assets directly associated with the withdrawal to the Inception vault.

Impact At the moment, it appears to be the intention that neither the Eigen Layer adapter nor the Symbiotic adapter should have a nonzero asset balance in between transactions. However, if this invariant is not upheld or if the functionality is updated in the future, assets could get stuck in the contract with no way to extract them.

Recommendation Create `onlyTrustee` functions which can be used to send *all* funds held by the contract to the Inception vault as a fail-safe. Note that this will come at the cost of additional centralization risk.

Developer Response The developers have acknowledged this issue. They have "aimed for direct assets transfer on `claim()` avoiding any nonzero asset balance in between transactions."

5.1.12 V-IVC-VUL-012: Unused approval on delegation

Severity	Low	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	delegate()		
Confirmed Fix At	b40a6b0		

In the `delegate()` function on the `InceptionEigenAdapterWrap` contract, there are two different cases that may be executed depending on the provided value of `operator`. The first case, when `operator` is set to the zero address, is to deposit funds into the adapter's EigenLayer strategy by first transferring funds from the sender (the Inception vault, usually) and then depositing those funds into the strategy. The second case is to delegate funds that have been deposited to a particular operator through the delegation manager.

In the second case, no funds are transferred from the sender. However, when the Inception vault calls the adapter's `delegate` function from its own `AdapterHandler.delegate()` function, it always first increases the adapter's allowance by `amount`. There is no check in the second case that the `amount` is 0, meaning there may be some amount of allowance which are not used up on the call to `delegate` as expected. The allowance is not reset after the call.

```
1 _asset.safeIncreaseAllowance(address(adapter), amount);
2 IIBaseAdapter(adapter).delegate(vault, amount, _data);
3 emit DelegatedTo(adapter, vault, amount);
```

Snippet 5.9: Snippet from `AdapterHandler.delegate()` that provides an allowance to the adapter.

Impact When Inception vault's `delegate()` function is called by the vault operator to set the EigenLayer operator to delegate to in the `InceptionEigenAdapterWrap`, the adapter will be approved to use more funds than than intended. The trustee manager would then be able to make a separate call to `InceptionEigenAdapterWrap.delegate` to deposit those funds into the adapter's strategy.

Recommendation Add an explicit check that the `amount` is 0 and/or reset approvals after calling the adapter's `delegate()` function.

Developer Response The developers added the suggested check that `amount` is 0.

5.1.13 V-IVC-VUL-013: Potential lost funds on claim with unchecked asset type

Severity	Low	**Commit**	0ebd568
Type	Data Validation	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	claim()		
Confirmed Fix At	bf8d4f6		

In the `claim()` function in `InceptionEigenAdapterWrap`, the token used for the withdrawal is specified by the argument `_data[1]` of the input. However, later in the function, it is assumed that the tokens received are the `stETH()` tokens associated with the `_asset`. There is no check that ensures that `_data[1] == _asset.stETH()`.

```
IERC20 backedAsset = IWStethInterface(address(_asset)).stETH();
/* ... */
IERC20[][] memory tokens = abi.decode(_data[1], (IERC20[][]));
/* ... */
_delegationManager.completeQueuedWithdrawal(
    withdrawals,
    tokens[0],
    middlewareTimesIndexes[0],
    receiveAsTokens[0]
);
uint256 withdrawnAmount = backedAsset.balanceOf(address(this)) -
    balanceBefore;
```

Snippet 5.10: Relevant lines of code in `claim()`

Impact If the caller chooses a token type that is not the expected type, a different token type will be credited to the adapter. These tokens likely cannot be moved from the contract, essentially locking the funds.

Recommendation Either add a check that the provided token is equivalent to the expected type *OR* remove the option to provide this entirely and just provide the expected token type directly.

Developer Response The developers added a check that the provided token type is the `stEth()` associated with the `_asset`.

5.1.14 V-IVC-VUL-014: Deposit and withdrawal rates can be set to zero

Severity	Warning	**Commit**	0ebd568
Type	Data Validation	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	setFlashWithdrawFeeParams, setDepositBonusParams		
Confirmed Fix At	14ef0f0		

The function for setting parameters for computing the withdrawal fee and deposit bonus check that all the percentages used in those calculations do not exceed `100%`. However, they do not check that those percentages are not set to `0`. It is unlikely these percentages should ever be `0`, and they can even cause a revert in the call to `calculateWithdrawalFee` in `InceptionLibrary.sol` if they were.

Impact Setting fees to `0` could result in unexpected behavior or reverts.

Recommendation Disallow setting percentages to `0` unless expressly intended.

Developer Response The developers added checks that the values are not set to `0`.

5.1.15 V-IVC-VUL-015: DoS with large amounts of deposited funds

Severity	Warning	**Commit**	0ebd568
Type	Denial of Service	**Status**	Acknowledged
File(s)	`AdapterHandler.sol`		
Location(s)	_getTargetCapacity()		
Confirmed Fix At	N/A		

The following function is used to compute the "target capacity" for the vault.

```
function _getTargetCapacity() internal view returns (uint256) {
    return (targetCapacity * getTotalDeposited()) / MAX_TARGET_PERCENT;
}
```

Snippet 5.11: Implementation of `_getTargetCapacity`

The multiplication `targetCapacity * getTotalDeposited()` can overflow causing a revert if the total amount deposited is too large.

Impact The target capacity goes up to `100 * 1e18`, so the total amount deposited would need to be *quite large* (e.g., on the order of `1e57`) to trigger this overflow. However, if the overflow occurs, most of the major operations of the vault would be halted. This function is invoked in `getFreeBalance()`, which is invoked in a number of critical workflows such as delegating to and claiming funds from an adapter.

Recommendation To reduce overflow risk here, the `lib/FullMath.sol:FullMath.mulDiv()` function could be used instead of raw multiply and divide. However, note that the `mulDiv` function consumes significantly more gas (at least 15 arithmetic operations worth of gas).

Developer Response The developers have acknowledged this issue. They have decided the value that is necessary to trigger an overflow is too large to ever be encountered in practice.

5.1.16 V-IVC-VUL-016: Referral codes can be frontrun and spammed

Severity	Warning	**Commit**	0ebd568
Type	Frontrunning	**Status**	Acknowledged
File(s)	`InceptionVault_S.sol`		
Location(s)	depositWithReferral		
Confirmed Fix At	N/A		

The function `depositWithReferral` allows users to make a deposit along with a "referral code" that is simply emitted as an event and is otherwise unused. There is no checking on this referral code to check that the sender of the transaction actually owns said referral code, that the referral code is valid, or that the referral code has not already been used.

Impact The lack of validation here could lead to a number of problems depending on the intended usage of these referral codes in this application and third party applications using them. For one, because there is no validation of who owns the referral code, anyone could steal anyone else's referral code. They could do so by frontrunning, for instance. Furthermore, because there is no check for valid referral codes, someone could provide whatever spam they want, which could clog up third party applications with bogus codes. Finally, because there is no check for duplicates, they could simply spam the same referral code many times in an attempt to exploit third party applications relying on the codes.

Recommendation Add checking to avoid undesirable usage of referral codes.

Developer Response The developers indicated that all necessary information/checking occurs off-chain.

5.1.17 V-IVC-VUL-017: Unsafe approve in InceptionEigenAdapterWrap

Severity	Warning	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	initialize(), claim()		
Confirmed Fix At	898d947		

The `InceptionEigenAdapterWrap` contract makes calls to the ERC20 `approve` function in the `initialize()` and `claim()` functions, but does not check whether the approval is successful (e.g., by checking the return value of the call).

```
// in initialize()
_asset.approve(strategyManager, type(uint256).max);
IWStethInterface(address(_asset)).stETH().approve(
        strategyManager,
        type(uint256).max
    );

// in claim()
backedAsset.approve(address(_asset), withdrawnAmount);
```

Snippet 5.12: Lines of code with the ERC20 `approve` calls.

Impact Based on the project's documentation, the `_asset` token of the `InceptionEigenAdapterWrap` is meant to be the WstETH token. The verified WstETH contract on Etherscan shows that WstETH is based on OpenZeppelin's implementation, which always returns `true` in `approve`. However, the stETH token (which is also approved here) is an upgradeable contract; the future behavior of its `approve` function is unknown.

Recommendation Use OpenZeppelin's `SafeERC20.safeApprove()` library function to perform the approval.

Developer Response The developers updated the code to use the `safeApprove()` function as suggested.

5.1.18 V-IVC-VUL-018: Missing claimer validation in isAbleToRedeem

Severity	Warning	**Commit**	0ebd568
Type	Data Validation	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	isAbleToRedeem()		
Confirmed Fix At	54e8fe6		

The `isAbleToRedeem()` function is used to determine whether the given `claimer` is eligible to perform a flash withdrawal of assets. In the current implementation, the function returns true when (1) the `claimer` is recorded as having a nonzero `_claimerWithdrawals[claimer].amount` and (2) the claimer has at least one queued withdrawal between the `claimer`'s earliest withdrawal epoch and the current epoch. However, the function does not explicitly reject the case where the `claimer` is the zero address.

Impact Currently, this issue has no impact. The `withdraw()` function, where asynchronous withdraw requests are made, will revert if the claimer is the zero address. Thus, it is not possible for `_claimerWithdrawals[claimer.amount]` to ever be nonzero.

However, if future changes are made to the vault implementation such that there can be a nonzero `_claimerWithdrawals[address(0)]`, then it may be possible for `redeem()` to be successfully called with the zero address as a claimer, which would result in assets being transferred to the zero address.

Recommendation Add a check to `isAbleToRedeem` that returns `false` and an empty array if the `claimer` is the zero address.

Developer Response The developers implemented the suggestion.

5.1.19 V-IVC-VUL-019: No access controls on IMellowAdapter.claimPending

Severity	Warning	**Commit**	0ebd568
Type	Access Control	**Status**	Fixed
File(s)	`IMellowAdapter.sol`		
Location(s)	claimPending()		
Confirmed Fix At	0ad5d81		

The `claimPending()` function of `IMellowAdapter` has no access controls, meaning that anyone can call the function successfully. Note that the `claim()` function does, which indicates that this may be a bug.

Impact There currently does not seem to be any security impact on the adapter or the vault. The `claimPending()` function is only used to transfer claimable assets from the registered Mellow vaults to the adapter. A call to `claimPending()` will not have any effect on the `getTotalPendingWithdrawals()`; however, it may potentially change the value returned by `claimableWithdrawalAmount()` and therefore any third-party protocols/applications relying on the latter function.

Recommendation To be consistent with the other privileged functions in the adapter, add the `onlyTrustee` modifier to `claimPending()`.

Developer Response The developers made `claimePending()` a private function that is only invoked from `claim()`, which is already guarded by `onlyTrustee`.

5.1.20 V-IVC-VUL-020: Minor validation issues in ISymbioticAdapter.claim

Severity	Warning	**Commit**	0ebd568
Type	Logic Error	**Status**	Fixed
File(s)	`ISymbioticAdapter.sol`		
Location(s)	claim()		
Confirmed Fix At	9e93212		

A trustee of the `ISymbioticAdapter` contract can call `claim()` to complete an asynchronous withdrawal from a Symbiotic vault, which will transfer assets from the Symbiotic vault to the Inception Vault. There are several issues with the way that `claim()` validates the call:

1. The `abi.decode(...)` logic implicitly assumes that `_data.length == 1`.
2. The `sEpoch` is never validated against the epoch stored in `withdrawals`.
3. Part of the preconditions for `claim()` include a check that `IVault(vaultAddress).isWithdrawalsClaimed(sEpoch, msg.sender)` is false, which would ensure that the `msg.sender` (i.e., the trustee) does not have a claimed withdrawal request. However, based on the implementation of `withdraw()`, the entity that issues the withdrawal request to the Symbiotic vault is the adapter itself, not the trustee.

```
function claim(
    bytes[] calldata _data
) external override onlyTrustee whenNotPaused returns (uint256) {
    (address vaultAddress, uint256 sEpoch) = abi.decode(
        _data[0],
        (address, uint256)
      );
    if (!_symbioticVaults.contains(vaultAddress)) revert InvalidVault();
    if (withdrawals[vaultAddress] == 0) revert NothingToClaim();
    if (sEpoch >= IVault(vaultAddress).currentEpoch())
        revert InvalidEpoch();
    if (IVault(vaultAddress).isWithdrawalsClaimed(sEpoch, msg.sender))
        revert AlreadyClaimed();

    delete withdrawals[vaultAddress];
    return IVault(vaultAddress).claim(_inceptionVault, sEpoch);
}
```

Snippet 5.13: Definition of `claim()`

Impact The impact of the list of issues mentioned above is, respectively:

1. If data contains more than one entry, the additional entries will be silently ignored. If the caller is making incorrect assumptions about how `claim()` works, they may make a mistaken but successful call to `claim()`.
2. Based on the way that `withdraw()` and `claim()` are implemented, it seems like any current withdrawal request must be claimed before a withdrawal request can be created at a different epoch; hence there shouldn't be an effect in the current implementation. However, this assumption may not hold if there are future changes to the contract.

3. As of the time of writing, the Symbiotic vault's `claim()` implementation will validate that `isWithdrawalsClaimed(sEpoch, msg.sender)` (where `msg.sender` is the adapter) evaluates to false. Thus, the incorrect call to `isWithdrawalsClaimed` in `ISymbioticAdapter` has no effect other than prevent the claim from occurring if the trustee directly has an active withdrawal request with the Symbiotic Vault.

Recommendation

1. Insert a `require` statement that checks that `_data.length == 1`.
2. Validate `sEpoch` against `withdrawals[vaultAddress]`.
3. Remove the incorrect `isWithdrawalsClaimed()` check.

Developer Response The developers implemented the suggested fixes.

5.1.21 V-IVC-VUL-021: Dangerous blanket approval for strategy manager

Severity	Warning	**Commit**	0ebd568
Type	Authorization	**Status**	Acknowledged
File(s)	`InceptionEigenAdapter.sol`		
Location(s)	initialize()		
Confirmed Fix At	N/A		

In the `initialize` function, the `InceptionEigenAdapterWrap` contract provides an unlimited approval of both WstETH and stETH to the EigenLayer strategy manager contract.

Impact At the moment, it appears to be the case that the Eigen Layer adapter should never hold additional assets between transactions. However, if this is not the case, then a large amount of trust is placed in the given strategy manager, as it could, at any time, take all funds owned by the adapter.

Recommendation Do approvals as needed when sending funds to the strategy manager.

Developer Response The developers have acknowledged this issue but do not have a planned fix for it at this time.

5.1.22 V-IVC-VUL-022: Some calls to claim are not automatically used to fulfill queued withdrawals

Severity	Warning	**Commit**	0ebd568
Type	Logic Error	**Status**	Acknowledged
File(s)	`ISymbioticAdapter.sol`, `IMellowAdapter.sol`, `InceptionEigenAdapter.sol`		
Location(s)	claim()		
Confirmed Fix At	N/A		

When an Inception vault operator claims an adapter contract's undelegated funds by calling the vault contract's `claim()` method, the bookkeeping for the withdrawal queue will be updated with those funds in `_updateEpoch()`. This means that conceptually, the funds are immediately put towards fulfilling queued withdrawals. However, if the trustee manager directly calls `claim()` on the adapter contract, the undelegated funds will be sent back to the Inception vault *without* updating the bookkeeping related to the withdrawal queue. This means all funds recovered in this way are put towards covering flash withdrawals.

```
function claim(
    address adapter,
    bytes[] calldata _data
) public onlyOperator whenNotPaused nonReentrant {
    uint256 availableBalance = getFreeBalance();
    uint256 withdrawnAmount = IIBaseAdapter(adapter).claim(_data);
    require(/* ... some sanity check on the amounts ... */);
    _updateEpoch(availableBalance + withdrawnAmount);
}

function _updateEpoch(uint256 availableBalance) internal {
    uint256 withdrawalsNum = claimerWithdrawalsQueue.length;
    uint256 redeemReservedBuffer;
    uint256 epochBuffer;
    for (uint256 i = epoch; i < withdrawalsNum; ) {
        /* ... code to sum up the currently fulfillable withdrawal amounts ... */
    }
    redeemReservedAmount += redeemReservedBuffer;
    epoch += epochBuffer;
}

function updateEpoch() external onlyOperator whenNotPaused {
    _updateEpoch(getFreeBalance());
}
```

Snippet 5.14: Relevant code from the `AdapterHandler.claim()` function, which will invoke the adapter's `claim()` function and update the bookkeeping.

Impact If the trustee manager calls `claim()` directly on the adapter contract, the vault operator would have to manually call the function `updateEpoch()` afterwards to allocate those funds towards the withdrawal queue. This violates the expected flow and may result in unexpected delays in covering queued withdrawals.

Recommendation At the very least, the expectation that `updateEpoch()` must manually be called after the trustee calls `claim()` should be clearly documented. If automatic updating of the epoch is desired, the developers would need to add additional logic allowing the adapter to directly update the epoch for the Inception Vault.

Developer Response The developers responded as follows: "We are aware of this. The main flow for [a] claim is only through `InceptionVault`. The `onlyTrustee` [modifier] was intended for `Adapter` in which case the `updateEpoch` is needed to be called manually."

5.1.23 V-IVC-VUL-023: Theoretical reentrancy concern on deposits

Severity	Warning	**Commit**	0ebd568
Type	Reentrancy	**Status**	Fixed
File(s)	`InceptionVault_S.sol`		
Location(s)	_deposit()		
Confirmed Fix At	83269f3		

When a user deposits assets into the Inception Vault using the `deposit()` or `mint()` functions, the assets are transferred from the user to the vault. As shown, the transfer happens after the deposit bonus is calculated and before the minting occurs. Furthermore, the amount sent is determined *indirectly* by comparing the total assets held by the vault before and after the transfer.

```
uint256 depositedBefore = totalAssets();
uint256 depositBonus;
uint256 availableBonusAmount = depositBonusAmount;
if (availableBonusAmount > 0) {
    depositBonus = calculateDepositBonus(amount);
    if (depositBonus > availableBonusAmount) {
        depositBonus = availableBonusAmount;
        depositBonusAmount = 0;
    } else {
        depositBonusAmount -= depositBonus;
    }
    emit DepositBonus(depositBonus);
}
// get the amount from the sender
_transferAssetFrom(sender, amount);
amount = totalAssets() - depositedBefore;
uint256 iShares = convertToShares(amount + depositBonus);
inceptionToken.mint(receiver, iShares);
```

Snippet 5.15: Snippet from `_deposit()`, which is called by `deposit()` and `mint()`.

If the asset has a callback that transfers control flow to the sender (i.e., user), then the sender could potentially call into another part of the protocol to increase the `totalAssets()` and make it appear as if they deposited more than they actually did.

Impact If this were to happen, the impact could be quite serious, as it would allow someone to inflate the number of shares received for their deposit amount and thus ostensibly steal from other users that have deposited into the protocol. However, there are a few reasons why we believe this attack is unlikely to be pulled off in practice:

1. It requires the asset to support features that transfer control flow to the sender, such as callbacks on the sender.
2. Most of the main contracts in the protocol have reentrancy guards on their critical functions. While these *do not* prevent calls across contracts, they prevent most of the ways the `totalAssets()` can be manipulated.
3. The function calls that we could identify that impact `totalAssets()` on other contracts (and thus are still callable despite reentrancy guards) appear to be only callable through

the Inception Vault (which is prevented by reentrancy guards) or by an adapter's trustee manager (such as calling `claim()` on one of the adapters). Thus, it would appear exploiting this would require a compromised trustee manager.

Recommendation As explained above, we believe this attack to be unlikely to occur in practice. However, if the trustee manager is compromised, or if there are non-access controlled flows that impact `totalAssets()`, this could be exploited at great cost to the vault's users. Unless there is some reason to believe that the amount transferred will ever *not* just be the amount specified as an argument, we suggest keeping `amount` to be the same.

Developer Response The developers used the argument `amount` as suggested and removed the computation of total assets before and after the call.

5.1.24 V-IVC-VUL-024: Gas Optimization

Severity	Info	**Commit**	0ebd568
Type	Gas Optimization	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	ce45e88, f1ead43		

In the following locations, the auditors identified missed opportunities for potential gas savings.

- `InceptionVault_S.sol`:
  - function `adjustWithdrawals`:
    - The calculation in this loop will count the same address multiple times. For example, if my queue consists of `[a, b, a, a]` as receivers, then `withdrawals[a]` will be set to 9 instead of 3.
    - The address array `queue` will always allocate the entire queue size, including slots that have already been redeemed and zeroed out.
  - function `redeem`:
    - If `withdrawals[receiver]` is `0`, the `genRequest` could be deleted since there are no more withdrawal requests. This can prevent redundant storage reads in the future.
- `AdapterHandler.sol`:
  - function `initialize`:
    - The call `_symbioticVaults.add(vaults[i])` returns whether or not the argument is already in storage. Thus, the return value here can be used instead of the separate call to `contains` that precedes this call.
- `ISymbioticAdapter.sol`:
  - function `initialize`:
    - The calls to OpenZeppelin initialize functions are duplicated here and in `__IBaseAdapter_init`.

Impact Gas may be wasted, costing users extra funds.

Recommendation Perform the optimizations.

Developer Response The developers implemented most of the suggestions with the exception of the changes suggested for `redeem`.

5.1.25 V-IVC-VUL-025: Unused program constructs

Severity	Info	**Commit**	0ebd568
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	8b333a6		

Description The following program constructs are unused:

- `InceptionVault_S.sol`:
  - Function `redeem()`:
    - `assets = convertToAssets(shares);` is computed before the call to `_flashWithdraw` but then is overwritten by the return value of that function without ever having been used.
- `ISymbioticAdapter.sol`:
  - The `pendingRewards()` function is commented out, and it should either be implemented or removed.

Impact These constructs may become out of sync with the rest of the project, leading to errors if used in the future.

Developer Response The developers removed the unused constructs.

5.1.26 V-IVC-VUL-026: Typos, incorrect comments, and small code suggestions

Severity	Info	**Commit**	0ebd568
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	e42a190		

Description In the following locations, the auditors identified minor typos, potentially misleading comments, and other small code suggestions:

- `InceptionVault_S.sol`:
  - mapping `_claimerWithdrawals`:
    - The `receiver` field of the `Withdrawal` struct is never set nor used. This could lead to confusion/errors in the future. If it is necessary for some sort of compliance, it might make sense to rename it to something like `__unused` for clarity.
  - function `__InceptionVault_init`:
    - The values `ratioFeed` and `targetCapacity` are not initialized in this call even though they are expected to be set before the vault is able to function.
  - function `previewDeposit`:
    - The logic for calculating the ceiling on the deposit bonus is duplicated in this function and `_deposit`. It may make sense for consistency to separate these out into a single function.
  - function `maxRedeem`
    - This function references the flash withdraw version of `redeem` and not the delayed withdraw version. This is expected given the ERC4626 interface, but is confusing given the reuse of the `redeem` function name for both the delayed and flash versions. To disambiguate, we suggest renaming the delayed withdraw function to something else. The same confusion arises with `previewRedeem`.
  - function `__beforeWithdraw`:
    - If the target capacity is `0`, this reverts with the error `InceptionOnPause`. The name of this error is misleading, as there is a separate `pause` function that has nothing to do with this error.
  - function `flashWithdraw`:
    - This does not return the `assets` explicitly, while `redeem(uint256, address, address)` does.
- `InceptionToken.sol`:
  - function `setVault`:
    - This function takes a vault of type `IInceptionVault_EL` as an input. `InveptionVault_S` is not of this type, but is clearly intended to be one option for a vault implementation for the token.
- `InVault_S_E2`:

- `constructor`:
  - Should not be `payable`.

▶ `InceptionEigenAdapter.sol`:

- function `claim`:
  - There is no check that the `_data` array is the expected length. Thus, it is possible that a user might call this with a longer `_data` array that succeeds silently but doesn't do what the user intends.
  - Per explanation from the developers, the value `_data[3]` is always intended to be `true`. In that case, we suggest just hard-coding it as `true` and removing this argument.
- functions `initialize` and `setRewardsCoordinator`:
  - Per explanation from the developers, the claimer on the rewards coordinator contract is not necessarily intended to be the owner of the Eigen Layer adapter contract, as it is a separate entity which is allowed to process a claim on behalf of the adapter. However, in `initialize`, the name `ownerAddress` is given to the claimer address sent to the `_setRewardsCoordinator` function (where the argument name is also `ownerAddress`). Additionally, the function `setRewardsCoordinator` is only able to set the claimer address to the `owner()` of the Eigen Layer adapter contract. Because this is not intended to only be the owner, it would make sense to (1) change the names to something like `claimerAddress` to avoid confusion; and (2) add an additional argument to `setRewardsCoordinator` to allow setting the claimer address to an address other than the Eigen Layer adapter contract's owner.

▶ `ISymbioticAdapter.sol`:

- function `removeVault`:
  - The check `if (!Address.isContract(vaultAddress)) revert NotContract();` is not necessary, as it was already checked that the vault address is a contract when it was first added through `addVault()`. Furthermore, if someone was able to get an externally-owned account address in the vaults set, there would be no way to remove it.

▶ `IMellowAdapter.sol`:

- A number of fields are declared but unused. Per the developer comments, these are kept to maintain the expected storage layout. To clarify this further for future developers, we also suggest renaming these fields to something like `__unused`.
- function `withdraw`:
  - This function determines the amount withdrawn by computing the difference in balance before and after the call to the vault's `withdraw` function. This implies there is some chance this value could be different than the `amount` field passed to the vault's `withdraw`. It may be a good idea to record any discrepancy in these values and take action if the discrepancy is large enough.

Impact These minor errors may lead to future developer confusion.

Developer Response The developers implement many of the suggested changes.

Regarding the `IMellowAdapter.withdraw` function, the developers had the following comments:

> Yes, Mellow Finance does not give back full amount. Part of readily available assets are given back, and part are sent to pending state which will become claimable later.

5.1.27 V-IVC-VUL-027: Unused return values

Severity	Info	**Commit**	0ebd568
Type	Usability Issue	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	35af801		

In a number of locations, the return values of functions are ignored , but it may be preferable to log these values in some way:

- `InceptionEigenAdapter.sol`:
  - function `withdraw`:
    - The return value of `queueWithdrawals` returns an array of the hashes corresponding to the withdrawals made. It may be useful to log these values.
- `ISymbioticAdapter.sol`:
  - function `deposit`:
    - The return value of the call to the vault's `deposit` function returns the amount deposited (which is used) as well as the number of shares this deposit produced (which is not used). It may be useful to log the shares.
  - function `withdraw`:
    - The return value of the call to the vault's `withdraw` function returns the number of burned and minted shares. It may be useful to log these values.

Impact Failure to log relevant values can make it difficult for third parties to monitor the behavior of this protocol in its interaction with other protocols.

Recommendation Log all relevant return values.

Developer Response The developers added events logging the return values as suggested.

A Appendix

A.1 Intended Behavior: Non-Issues of Note

A.1.1 V-IVC-APP-VUL-001: Forced redemptions for users could interfere with third party applications

Severity	Low	**Commit**	0ebd568
Type	Data Validation	**Status**	Intended Behavior
File(s)	`InceptionVault_S.sol`		
Location(s)	redeem		

For `redeem`, there are no checks that prohibit another user from initializing a redeem on behalf of another user. This could allow attackers to interfere with third party applications that require control over when redemptions occur.

Impact Any third party application that requires control over when redemptions occur could be disrupted.

Recommendation Require the claimer themselves be the caller of `redeem`.

Developer Response The developers have indicated that this behavior is intended despite the potential risks for third party applications.

Glossary

ERC Ethereum Request for Comment. 46

ERC-20 The famous Ethereum fungible token standard. See `https://eips.ethereum.org/EIPS/eip-20` to learn more. 1

ERC-4626 An Ethereum Request for Comment (ERC) describing a tokenized vault representing shares of an underlying ERC20 token. See `https://eips.ethereum.org/EIPS/eip-4626` for the full ERC. 1

Ethereum Request for Comment Peer-reviewed proposals describing application-level standards and conventions. Visit `https://eips.ethereum.org/erc` to learn more. 46