Eigen Labs

EigenLayer – Strategy Factory

Security Assessment Report

Version: 1.0

August, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Eigen Labs smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Eigen Labs smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Eigen Labs smart contracts.

Overview

EigenLayer is a restaking protocol on Ethereum, designed to enable stakers to contribute to the cryptoeconomic security of various protocols beyond Ethereum, in return for rewards.

This review focused on an incremental change to how strategies are deployed and whitelisted on EigenLayer. This was done by introducing a new `StrategyFactory` contract that would allow for permissionless deployments and whitelisting of strategies.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the eigenlayer-contracts repository.

The scope of this time-boxed review was strictly limited to changes in PR#522 at commit aa05081.

This included changes to the following files:

- `interfaces/`
  - `IStrategy.sol`
  - `IStrategyFactory.sol`
  - `IStrategyManager.sol`
- `strategies/`
  - `StrategyBase.sol`
  - `StrategyFactory.sol`
  - `StrategyFactoryStorage.sol`

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.
- Low: 1 issue.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Eigen Labs smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
EGN6-01	Failed Token Transfer In Withdrawal Can Result In Frozen Funds	Medium	Open
EGN6-02	`blacklistTokens()` Can Be Frontrun	Low	Open
EGN6-03	`tokenStrategy` Mapping Has Insufficient Documentation	Informational	Open
EGN6-04	Miscellaneous General Comments	Informational	Open

EGN6-01	Failed Token Transfer In Withdrawal Can Result In Frozen Funds		
Asset	`StrategyBase.sol`, `DelegationManager.sol` *(out-of-scope)*		
Status	**Open**		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Failed token transfers of tokens with blocklists may cause integrators' funds to be frozen.

`StrategyFactory` allows for the permissionless deployment and whitelisting of any token, including tokens that can block transfers for blacklisted addresses, such as `cbETH`.

If any token transfer in a withdrawal fails due to the withdrawer being blacklisted, then the entire withdrawal (which may also include other strategies) will fail. This is currently possible with strategies such as `cbETH`, which implement blacklists, and will become more common with the introduction of permissionless whitelisted strategies.

It is possible to rescue the funds by calling `DelegationManager::completeQueuedWithdrawal()` with `receiveAsTokens = false`, and then queuing another withdrawal without the affected strategy. However, integrators such as liquid restaking protocols that hardcode the `receiveAsToken` parameter to `true` have no way of recovering the frozen funds without upgrading their contracts.

Note, this issue has been deemed to have a low likelihood of occurring as it requires the withdrawer to have shares in the affected strategy and also to queue a withdrawal for that strategy alongside other strategies.

Recommendations

Implement any/both of the following recommendations:

1. Instead of reverting, catch failed transfers and refund the strategy shares to the withdrawer.
2. Add comments and documentation to spread awareness of this issue to integrators, ensuring that there are ways to recover frozen funds by including the option to set `receiveAsTokens = false` when completing queued withdrawals.

EGN6-02	`blacklistTokens()` Can Be Frontrun		
Asset	`StrategyFactory.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

A user can frontrun the call to `blacklistTokens()` to deploy strategies for tokens that are about to be blacklisted.

The `blacklistTokens()` function is used to prevent users from deploying strategies for tokens that have pre-existing whitelisted strategies, or for tokens that will have custom strategies such as `EIGEN`.

The blacklisting of tokens for pre-existing whitelisted strategies cannot be frontrun, as the EigenLayer team will perform this operation during the 10-day timelock of setting the `strategyWhitelister` role in `StrategyManager`, such that new strategies cannot be deployed during the timelock.

However, after the `strategyWhitelister` role has been set to `StrategyFactory`, users can frontrun the `blacklistTokens()` call to deploy strategies for tokens that are about to be blacklisted.

This issue has a low impact because the owner can always call `removeStrategiesFromWhitelist()` to unwhitelist the new strategy.

Recommendations

Though `removeStrategiesFromWhitelist()` can already be called to unwhitelist strategies belonging to blacklisted tokens, consider adding logic to unwhitelist deployed strategies for tokens that are being blacklisted in `blacklistTokens()` in order to minimise the chance of human error.

EGN6-03	`tokenStrategy` Mapping Has Insufficient Documentation
Asset	`StrategyFactory.sol, StrategyFactoryStorage.sol`
Status	Open
Rating	Informational

Description

There is insufficient documentation on the `tokenStrategy` mapping, which may potentially result in incorrect integration implementations by external apps, such as liquid restaking protocols.

The `tokenStrategy` mapping represents all strategies that have been deployed by the `StrategyFactory` contract.

However, it is important to note that this mapping should not be used as a registry for whitelisted strategies, as it comes with the following caveats:

- the strategies in the mapping are only those deployed by `StrategyFactory`
- `StrategyFactory` can only deploy one strategy per token address
- the strategies in the mapping may not be whitelisted in `StrategyManager`
- the strategies in the mapping may correspond to tokens that are blacklisted in `StrategyFactory`
- the strategies in the mapping may not be the only strategy for the underlying token, whether whitelisted or not

The current documentation for the `tokenStrategy` mapping does not inform integrators on the caveats above, as seen below:

```
// @notice Mapping token => Strategy contract for the token
mapping(IERC20 => IStrategy) public tokenStrategy;
```

Recommendations

Add documentation and comments to inform integrators about the properties and caveats of the `tokenStrategy` mapping, as described above.

EGN6-04	Miscellaneous General Comments
Asset	All contracts
Status	Open
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Misleading Natspec**

   Related Asset(s): StrategyBase.sol, StrategyFactory.sol

   There are multiple instances in the codebase where the Natspec comments of a function do not match its actual behaviour:

   - The `StrategyBase::_emitExchangeRate()` Natspec comment in line [**295**] states that the exchange rate is returned, when it is emitted.
   - The `StrategyBase::_emitExchangeRate()` Natspec comment in line [**296**] states that the function assumes the token will have no more than 18 decimals, but there is no logic that depends on this assumption.
   - The `StrategyFactory::whitelistStrategies()` Natspec comment in line [**87**] states that the function also adds the strategy to the `tokenStrategy` mapping when it doesn't.

   Change the Natspec comments or function logic to ensure that the comments match with the behavior of the code.

2. **Misleading Error String**

   Related Asset(s): StrategyFactory.sol

   The error string in line [**79**] of `StrategyBase::blacklistTokens()` does not match the condition inside the `require()` function:

   ```
   require(!isBlacklisted[tokens[i]], "StrategyFactory.blacklistTokens: Cannot blacklist deployed strategy");
   ```

   Change the line to:

   ```
   require(!isBlacklisted[tokens[i]], "StrategyFactory.blacklistTokens: Token already blacklisted");
   ```

3. **Use Of Magic Numbers**

   Related Asset(s): StrategyBase.sol

   The `_emitExchangeRate()` function uses the magic number `1e18` to scale the exchange rate to 18 decimals.

   To increase code readability, set this value as a named constant.

4. **Missing Sanity Checks**

   Related Asset(s): StrategyFactory.sol

   The `initialize()` function is missing sanity checks to ensure that arguments are non-zero.

   Add sanity checks to ensure that the following arguments are non-zero:

   - `_initialOwner`
   - `_strategyBeacon`

`_pauserRegistry` is already sanity checked in the `Pausable::_initializePauser()` function.

5. **Potentially Outdated Token Decimals**

   Related Asset(s): StrategyBase.sol

   The underlying token's decimals is emitted when initializing a new strategy:

```
/// @notice Sets the `underlyingToken` and `pauserRegistry` for the strategy.
function _initializeStrategyBase(
    IERC20 _underlyingToken,
    IPauserRegistry _pauserRegistry
) internal onlyInitializing {
    underlyingToken = _underlyingToken;
    _initializePauser(_pauserRegistry, UNPAUSE_ALL);
    emit StrategyTokenSet(underlyingToken, IERC20Metadata(address(_underlyingToken)).decimals());
}
```

   This assumes that the token's decimals is constant. Although extremely uncommon, it is possible for upgradeable tokens to change their decimals.

   Consider emitting the token's decimals each time `_emitExchangeRate()` is called, or add comments and documentations to inform integrators on the importance of checking a token's decimals before scaling the exchange rate.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 5 tests for test/tests-fork/StrategyBase.fork.t.sol:StrategyBaseForkTest
[PASS] test_ERC777Reentrancy_DepositThenWithdraw() (gas: 1066668)
[PASS] test_ERC777Reentrancy_WithdrawThenDeposit() (gas: 841961)
[PASS] test_deposit_FeeOnTransferToken() (gas: 595728)
[PASS] test_deposit_RevertIf_MaxUintTransferToken() (gas: 160226)
[FAIL. Reason: revert: BlocklistTokenMock: from or to is blocklisted] test_withdraw_DoSLockedFunds_Vuln() (gas: 1361990)
Suite result: FAILED. 4 passed; 1 failed; 0 skipped; finished in 947.83ms (14.38ms CPU time)

Ran 9 tests for test/tests-fork/StrategyFactory.fork.t.sol:StrategyFactoryForkTest
[PASS] testFuzz_blacklistTokens(address[]) (runs: 1001, μ: 332659, ~: 337541)
[PASS] test_blacklistTokens_RevertConditions() (gas: 67670)
[PASS] test_deployNewStrategy() (gas: 301169)
[PASS] test_deployNewStrategy_RevertIf_BlacklistedToken() (gas: 54778)
[PASS] test_deployNewStrategy_RevertIf_FakeUnderlyingToken() (gas: 406407)
[PASS] test_deployNewStrategy_RevertIf_PausedNewStrategies() (gas: 56754)
[PASS] test_deployNewStrategy_RevertIf_StrategyAlreadyDeployed() (gas: 289261)
[PASS] test_removeStrategiesFromWhitelist() (gas: 58144)
[PASS] test_whitelistStrategies() (gas: 69770)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 1.31s (374.72ms CPU time)

Ran 2 test suites in 1.31s (2.26s CPU time): 13 tests passed, 1 failed, 0 skipped (14 total tests)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

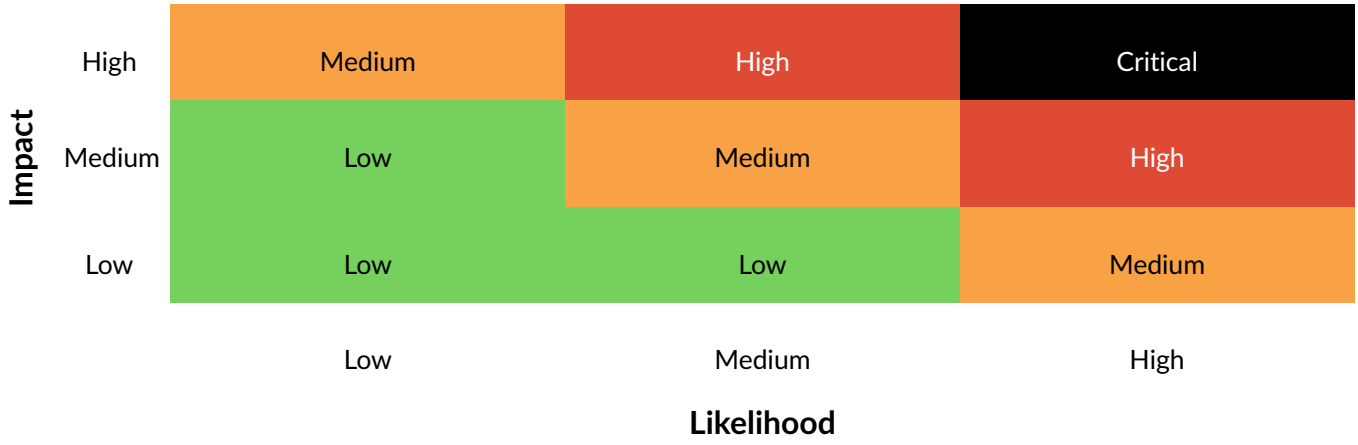

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].