# Puffer Institutional Security Review


**Auditors**

Noah Marconi, Lead Security Researcher

Hyh, Lead Security Researcher

Ladboy233, Security Researcher

**Report prepared by:** Lucas Goiriz



March 17, 2025

# Contents

# 1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

# 2 Introduction

Puffer is a decentralized native liquid restaking protocol (nLRP) built on Eigenlayer. It makes native restaking on Eigenlayer more accessible, allowing anyone to run an Ethereum Proof of Stake (PoS) validator while supercharging their rewards.

*Disclaimer*: This security review does not guarantee against a hack. It is a snapshot in time of Puffer Institutional according to the specific commit. Any modifications to the code will require a new security review.

# 3 Risk classification

| Severity level | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| **Likelihood: high** | Critical | High | Medium |
| **Likelihood: medium** | High | Medium | Low |
| **Likelihood: low** | Medium | Low | Low |

## 3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

## 3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

## 3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

# 4 Executive Summary

Over the course of 2 days in total, Puffer Finance engaged with Spearbit to review the puffer-institutional protocol. In this period of time a total of **13** issues were found.

**Summary**

| | |
|---|---|
| **Project Name** | Puffer Finance |
| **Repository** | puffer-institutional |
| **Commit** | 918531ec |
| **Type of Project** | DeFi, Staking |
| **Audit Timeline** | Feb 28th to Mar 2nd |

**Issues Found**

| Severity | Count | Fixed | Acknowledged |
|---|---|---|---|
| Critical Risk | 0 | 0 | 0 |
| High Risk | 0 | 0 | 0 |
| Medium Risk | 0 | 0 | 0 |
| Low Risk | 5 | 3 | 2 |
| Gas Optimizations | 0 | 0 | 0 |
| Informational | 8 | 3 | 5 |
| **Total** | **13** | **6** | **7** |

# 5 Findings

## 5.1 Low Risk

### 5.1.1 `depositETH` does not validate the `maxDeposit` cap

**Severity:** Low Risk

**Context:** InstitutionalVault.sol#L86-L92

**Description:**

```
function deposit(uint256 assets, address receiver) public virtual returns (uint256) {
uint256 maxAssets = maxDeposit(receiver);
if (assets > maxAssets) {
    revert ERC4626ExceededMaxDeposit(receiver, assets, maxAssets);
}
```

In the ERC4626 implementation, when an asset is deposited to mint a share, the `maxDeposit` is called to validate if the minted share exceeds the `maxMint`. `depositETH` skips such check:

```
function depositETH(address receiver) public payable virtual restricted {
    uint256 shares = previewDeposit(msg.value);

    _mint(receiver, shares);
```

**Recommendation:**

```
function depositETH(address receiver) public payable virtual restricted {

        uint256 maxAssets = maxDeposit(receiver);

        if (msg.value > maxAssets) {  revert ERC4626ExceededMaxDeposit }

    uint256 shares = previewDeposit(msg.value);

    _mint(receiver, shares);

    emit Deposit(_msgSender(), receiver, msg.value, shares);
}
```

**Puffer Finance:** Fixed in PR 7.

**Cantina Managed:** Fix verified.

### 5.1.2 Unchecked math may silently overflow

**Severity:** Low Risk

**Context:** InstitutionalVault.sol#L236

**Description:** Given `restakedValidatorsETH` and `nonRestakedValidatorsETH` may be set to arbitrary values by the vault owner, an erroneously set value for these variables will result in a silent overflow.

**Recommendation:** Include a separate overflow / bounds check or revert back to solidity checked math.

**Puffer Finance:** Fixed in PR 7.

**Cantina Managed:** Fix verified.

### 5.1.3 Overridden `totalAssets` function leads to ERC4626 non compliance

**Severity:** Low Risk

**Context:** InstitutionalVault.sol#L26

**Description:** The `totalAssets` function includes references to currently staked assets. These assets held outside of the vault inflate the value returned by `maxWithdraw` and `maxRedeem`. Similarly `previewWithdraw` and `previewRedeem` return with values that do not conform to the requirement described in the spec: `return as close to and no fewer than the exact amount of Vault shares that would be burned in a withdraw call in the same transaction..`

A related issue occurs due to pausing or access control. Using `restricted` as a proxy for pausing (as described by the project team), means that these functions do not conform to spec during a paused state.

**Recommendation:** Highlight in the documentation where/when the vault deviates from the spec or modify the getters to account for these edge cases.

**Puffer Finance:** Fixed in PR 8.

**Cantina Managed:** Acknowledged.

#### 5.1.4 Withdrawn stake can be miscounted in total assets right after the Eigen Layer withdrawal

**Severity:** Low Risk

**Context:** InstitutionalVault.sol#L314-L318

**Description:** The total asset value becomes incorrect after the `completeQueuedWithdrawals()` call concludes, since it doesn't reduce `restakedValidatorsETH`, but increases ETH balance by the corresponding amount withdrawn:

- EigenPod.sol#L412-L420:

```
function withdrawRestakedBeaconChainETH(address recipient, uint256 amountWei) external
↪   onlyEigenPodManager {
    uint64 amountGwei = uint64(amountWei / GWEI_TO_WEI);
    amountWei = amountGwei * GWEI_TO_WEI;
    require(amountGwei <= restakedExecutionLayerGwei, InsufficientWithdrawableBalance());
    restakedExecutionLayerGwei -= amountGwei;
    emit RestakedBeaconChainETHWithdrawn(recipient, amountWei);
    // transfer ETH from pod to `recipient` directly
    Address.sendValue(payable(recipient), amountWei); // <<<
}
```

Also, total assets can be miscalculated and value removed from the Vault shares owners if `staker` in the structure is set to any address other than Vault's:

- DelegationManager.sol#L600:

```
shareManager.withdrawSharesAsTokens({
    staker: withdrawal.staker, // <<<
    strategy: withdrawal.strategies[i],
    token: tokens[i],
    shares: sharesToWithdraw
});
```

**Recommendation:** Consider adding the initial value of ETH supplied for restaking corresponding to this exact withdrawal directly to function arguments, e.g. making it `function completeQueuedWithdrawals(withdrawal, receiveAsTokens, restakedETHRemoved)` and performing the update atomically:

- InstitutionalVault.sol#L314-L318:

```
  EIGEN_DELEGATION_MANAGER.completeQueuedWithdrawals({
      withdrawals: withdrawals,
      tokens: tokens,
      receiveAsTokens: receiveAsTokens
  });
+ $.restakedValidatorsETH -= restakedETHRemoved;
```

Also, as `staker` in `Withdrawal` structure has to be the Vault address to order to keep the accounting consistent, consider controlling for this.

**Puffer Finance:** Introducing a code like:

```
// If we are dealing with ETH withdrawal that will transfer ETH to the vault
// We need to update the non restaked validators ETH
if (receiveAsTokens[0]) {
    _getVaultStorage().restakedValidatorsETH -= SafeCast.toUint128(withdrawals[0].scaledShares[0]);
    emit RestakedValidatorsETHUpdated(_getVaultStorage().restakedValidatorsETH);
}
```

This would introduce another issue (DoS):

- Assume one restaking validator is started. (This means that the `restakedValidatorsETH = 32 ETH`).
- They earn 1 ETH of rewards and those rewards accumulate in the `EigenPod`.
- They shut down the validator & queue 33 ETH withdrawal.

Now the `completeQueuedWithdrawal` can't be finished because of the overflow/underflow issue.

Since this is a permissioned, closed system, the issue of inaccurate exchange rates has already been documented and will be communicated to our clients and partners. Given this, along with the DoS attack scenario, we have decided not to implement the suggested change. Additionally, the system includes the `setValidatorsETH()` function, which allows the admin to centrally update the number of restaked and non-restaked validators' ETH.

**Cantina Managed:** Acknowledged.

#### 5.1.5 Vault's `completeQueuedWithdrawals()` supplies incorrect token to `EIGEN_DELEGATION_MANAGER.completeQueuedWithdrawals()`

**Severity:** Low Risk

**Context:** InstitutionalVault.sol#L312

**Description:** Having the deposit contract address, `IERC20(address(BEACON_DEPOSIT_CONTRACT))`, as a preferred output token, `tokens[0][0]`, isn't correct as this value has to specify the output token to receive, which is not represented by the deposit contract. For now the `tokens` value looks to be ignored for the beacon chain case:

- DelegationManager.sol#L595-L605:

```
if (receiveAsTokens) {
    // Withdraws `shares` in `strategy` to `withdrawer`. If the shares are virtual beaconChainETH
    ↪  shares,
    // then a call is ultimately forwarded to the `staker`s EigenPod; otherwise a call is
    ↪  ultimately forwarded
    // to the `strategy` with info on the `token`.
    shareManager.withdrawSharesAsTokens({
        staker: withdrawal.staker,
        strategy: withdrawal.strategies[i],
        token: tokens[i], // <<<
        shares: sharesToWithdraw
    });
} else {
```

- EigenPodManager.sol#L183-L188:

```
function withdrawSharesAsTokens(
    address staker,
    IStrategy strategy,
    IERC20, // <<<
    uint256 shares
) external onlyDelegationManager nonReentrant {
```

However, this might change in the future Eigen Layer versions, which can lead to permanent unavailability of the `completeQueuedWithdrawals()` (while restaked withdrawals can be still completed via `customExternalCall()`).

**Recommendation:** Consider using `address(0)` to represent ETH, e.g.:

```
- tokens[0][0] = IERC20(address(BEACON_DEPOSIT_CONTRACT));
+ tokens[0][0] = IERC20(address(0));
```

**Puffer Finance:** The address here is wrong, but it shouldn't be `address(0)`, it should be `_BEACON_CHAIN_STRATEGY`. Here is a mainnet tx as an example:

```
https://etherscan.io/tx/0xb2a297fbfa18b1dc02548842df36b129ad085a06f57c9b07cee06ff7a70f6ab5
```

Fixed in PR 6.

**Cantina Managed:** Fix verified.

## 5.2 Informational

### 5.2.1 Malicious node validator can frontrun the non-Eigenlayer deposit to steal fund if the deposit root is not validated

**Severity:** Informational

**Context:** InstitutionalVault.sol#L260-L265

**Description:** The code deposit the 32 Ether to `BEACON_DEPOSIT_CONTRACT`:

```
BEACON_DEPOSIT_CONTRACT.deposit{ value: 32 ether }(
    pubKeys[i], withdrawalCredentials, signatures[i], depositDataRoots[i]
);
```

However, the same issue applies is specified in Immunefi's post on the LIDO frontrunning bug postmortem. A malicious user can:

1. Frontrun the valid deposit transaction (32 ETH) with prepared deposit data.
2. Malicious deposit data contains the same validator pubkey, minimal deposit for deposit contract (1ETH), and malicious user's withdraw credential.

If `pubkey` not in `validator_pubkeys` on the client side would be called first as it would never be seen before the validator's key. The second deposit, which we frontrun, would increase our deposit by 32 ETH. Those 32 ETH, as a reminder, came from the protocol deposit.

**Recommendation:** The LIDO protocol implements this fix:

> Introduce a committee that follows the events on the execution layer and off-chain collects signatures verifying that no malicious pre-deposits were made yet by active node operators at deposit contract data root XXXXX.

It is recommended to make the committee signatures and the deposit contract data root the parameters of the `startNonRestakingValidators` function. Revert the transaction unless the current deposit data root is the same as the one passed to the function AND the committee has vetted that data root. Basically the deposit root has to be validated to avoid such frontrunning either onchain or offchain.

**Puffer Finance:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 5.2.2 Malicious vault owner can DoS the total TVL computation

**Severity:** Informational

**Context:** InstitutionalFactory.sol#L93-L101

**Description:** After puffer multisig owner creates vault, the vault is managed by individual admin. If a vault admin is compromised, he can upgrade the vault and revert in external call `vault[i].totalAssets()` to DoS the function call `calculateTVL()`:

```
function totalAssets() public view virtual override(ERC4626Upgradeable, IInstitutionalVault) returns
↪  (uint256) {
    Storage storage $ = _getVaultStorage();
    uint256 callValue;
    // solhint-disable-next-line no-inline-assembly
    assembly {
        callValue := callvalue()
    }
    return WETH.balanceOf(address(this)) + (address(this).balance - callValue) + $.restakedValidatorsETH
        + $.nonRestakedValidatorsETH;
}
```

Each vault admin can also set `restakedValidatorsETH` and `nonRestakedValidatorETH` to a large number to make sure:

```
totalTVL += ERC4626Upgradeable(vaults[i]).totalAssets();
```

exceed max number of uint256 to revert the total TVL computation.

**Recommendation:** Compute the total TVL off-chain.

**Puffer Finance:** Fixed in PR 3.

**Cantina Managed:** Fix verified.

### 5.2.3 `setValidatorsETH` state changes are vulnerable to frontrunning

**Severity:** Informational

**Context:** InstitutionalVault.sol#L350-L360

**Description:** The `totalAssets` calculation is documented as having known accounting inconsistencies. The `setValidatorsETH` function is used by admin to bring internal accounting up to date. Before and after updates, shares received for deposits and assets received for withdrawls are impacted, in a way that can make a sandwich attack profitable. At present this is an informational finding only due to the admin only control of all state changing functions, including deposit and withdraw.

The code comments indicate a future version where the exchange rate will need to be accurate. When designing a version such as this, frontrunning/backrunning/sandwich attacks will need to be considered.

**Puffer Finance:** Updated readme in PR 8.

**Cantina Managed:** Acknowledged.

### 5.2.4 Transfers may be made to accounts without `restricted` permission

**Severity:** Informational

**Context:** InstitutionalVault.sol#L175, InstitutionalVault.sol#L193

**Description:** The contract permits vault share transfers to accounts without `restricted` permissions. The receiving account will not be able to `transfer` or `withdraw`/`redeem` until permission is explicitly given.

**Recommendation:** Document the behavior and exercise caution when transferring vault shares. Alternatively, a check may be added to ensure the recipient has the appropriate permission at the time of transfer (knowing the permission could be removed at a later time).

**Puffer Finance:** Acknowledged.

**Cantina Managed:** Acknowledged.

#### 5.2.5 Pausing requires a high level admin

**Severity:** Informational

**Context:** InstitutionalVault.sol#L105

**Description:** The project notes `restricted` is intended to be used as a pause capability. This means the most secure account, the `AccessManager` admin, is the only account that can pause.

**Recommendation:** The contracts may benefit from a dedicated pauser role to pause more quickly, leaving unpause to the `AccessManager` admin.

**Puffer Finance:** Acknowledged, we will discuss this with our partners to see if they want to have this functionality. And in that case the logic can be in the contract controlling the `AccessManager`.

**Cantina Managed:** Acknowledged.

#### 5.2.6 `startRestakingValidators()` uses hard coded `32 ETH` stake that will enforce validator overhead

**Severity:** Informational

**Context:** InstitutionalVault.sol#L261

**Description:** Since MAX_EFFECTIVE_BALANCE increase is scheduled as a part of Pectra soon enough (March 2025 for now) the `32 ETH` hard code posses an overhead that will be no longer necessary after the upgrade.

**Recommendation:** Consider introducing an additional stake size flexibility, e.g.:

- InstitutionalVault.sol#L222-L234.

```
  function startRestakingValidators(
+     uint256[] calldata amounts,
      bytes[] calldata pubKeys,
      bytes[] calldata signatures,
      bytes32[] calldata depositDataRoots
  ) external virtual restricted {
-     require(pubKeys.length == signatures.length && pubKeys.length == depositDataRoots.length,
↪  InvalidInput());
-     require(pubKeys.length == signatures.length && pubKeys.length == depositDataRoots.length &&
+             pubKeys.length == amounts.length && pubKeys.length > 0, InvalidInput());

-     _unwrapWETH(32 ether * pubKeys.length);

+     uint256 totalAmount;
+     for (uint256 i = 0; i < amounts.length; i++) {
+         require(amounts[i] >= 32 ether && amounts[i] <= 2048 ether, InvalidInput());
+         totalAmount += amounts[i];
+     }

+     _unwrapWETH(totalAmount);

      for (uint256 i = 0; i < pubKeys.length; i++) {
-         EIGEN_POD_MANAGER.stake{ value: 32 ether }(pubKeys[i], signatures[i],
↪  depositDataRoots[i]);
+         EIGEN_POD_MANAGER.stake{ value: amounts[i] }(pubKeys[i], signatures[i],
↪  depositDataRoots[i]);

          emit StartedRestakingValidator(pubKeys[i], depositDataRoots[i]);
      }
```

Client to be adviced to stick to `32 ether` amounts before the chain upgrades and EIP-7251 goes live.

**Puffer Finance:** EigenLayer will not support

```
EIGEN_POD_MANAGER.stake{ value: amounts[i] }(pubKeys[i], signatures[i], depositDataRoots[i]);
```

This can be observed in their Pectra EigenPod code (EigenPod.sol#L386). See it in Pectra's PR 1155.

Our `customExternalCall` can perform `0x02` deposits to the Beacon Chain Contract for Pectra Validators in EigenLayer. It can also be used for validator consolidation in the future.

This issue seems to be a suggestion for a potential future upgrade.

**Cantina Managed:** Acknowledged.

### 5.2.7 Misconfigured Vaults cannot be removed from the Factory's Vault list, which can break `getVaults()` and `calculateTVL()`

**Severity:** Informational

**Context:** InstitutionalFactory.sol#L65-L109

**Description:** Vault deployment calls initialize, that can revert, while the deployment result isn't controlled. Vault is run by a client, whom can misconfigure it. In the same time all the vaults are returned by `getVaults()` and are used in `calculateTVL()` unconditionally. Both functions can be corrupted via an operational mistake this way.

**Recommendation:** Consider controlling for Vault deployment success, e.g.:

- InstitutionalFactory.sol#L65-L81:

```
    address vault = address(
        Create2.deploy({
            // ...
        })
    );
+   require(IInstitutionalVault(vault).getEigenPod() != address(0), DeployFailed());
    vaults.push(vault);
```

In addition, consider adding a `onlyPufferMultisig` controlled functionality to remove a vault from `vaults`.

**Puffer Finance:** This update addresses two distinct issues:

- The addition of the removeVault() function, allowing faulty vaults to be removed from the system. This fix is implemented in PR 9.
- The removal of the `calculateTVL()` functionality from the `Factory`, enabling TVL calculations to be performed off-chain. This fix is implemented in PR 3.

**Cantina Managed:** Fix verified.

#### 5.2.8 Remaining informational issues summary

**Severity:** Informational

**Context:** InstitutionalVault.sol#L432-L436

**Description:**

1. Shares amount isn't returned from `depositETH()` at `InstitutionalVault.sol#L86`.
2. `mint()` is missing in the `IInstitutionalVault` interface, also language is a bit off in the description (assets are utilized, shares are minted) at `InstitutionalVault.sol#L210`.
3. `tokens` has the length of `1` this is actually limited to a single withdrawal as otherwise `completeQueuedWithdrawals()` will fail with array out of bounds:
   - DelegationManager.sol#L217-L226:

```
function completeQueuedWithdrawals(
    Withdrawal[] calldata withdrawals,
    IERC20[][] calldata tokens,
    bool[] calldata receiveAsTokens
) external onlyWhenNotPaused(PAUSED_EXIT_WITHDRAWAL_QUEUE) nonReentrant {
    uint256 n = withdrawals.length;
    for (uint256 i; i < n; ++i) {
        _completeQueuedWithdrawal(withdrawals[i], tokens[i], receiveAsTokens[i]); // <<<
    }
}
```

4. `_wrapETH()` can fail with low level WETH deposit error if there are no `assets - wethBalance` ETH present on the balance at `InstitutionalVault.sol#L435`.

**Recommendation:** Consider:

1. Returning shares from `depositETH()`, adding to the description.
2. Adding `mint()` to `IInstitutionalVault`, replacing `minted` with `consumed`.
3. To reduce operational mistakes surface (i.e. the attempts to use `completeQueuedWithdrawals()` as an array based version) consider simplifying, e.g. making it `function completeQueuedWithdrawal(IDelegationManagerTypes.Withdrawal calldata withdrawal, bool calldata receiveAsTokens)` (i.e. also removing plural by Noah's comment).
4. Applying the similar logic as in `_unwrapWETH()`, e.g.:

- InstitutionalVault.sol#L432-L436:

```
   uint256 wethBalance = WETH.balanceOf(address(this));

- if (wethBalance < assets) {
-     WETH.deposit{ value: assets - wethBalance }();
- }

+ if (wethBalance >= assets) {
+     return;
+ }

+ uint256 amountToWrap;
+ unchecked { amountToWrap = assets - balance; }

+ if (address(this).balance >= amountToWrap) {
+     WETH.deposit(amountToWrap);
+ } else {
+     revert NotEnoughETH();
+ }
```

**Puffer Finance:** Did some fixes and updated the readme to address some concerns in PR 8.

- (2) The natspec is correct if you are refering to this `minted`

```
/**
 * @notice Deposit ETH into the vault
 * Depositor receives institutionalETH shares in return
 * @param receiver The address to receive the shares
 * @return shares The amount of shares (institutionalETH) minted
 */
function depositETH(address receiver) external payable returns (uint256);

/**
 * @notice Override the deposit function to allow for the deposit of shares (institutionalETH)
 * Restricted modifier is used to pause/unpause the deposit function
 * @param assets The amount of assets (WETH) to deposit
 * @param receiver The address to receive the shares
 * @return shares The amount of shares (institutionalETH) minted
 */
function deposit(uint256 assets, address receiver) external returns (uint256);
```

3. We will skip this fix.

**Cantina Managed:** Fixed and acknowledged.