Quantstamp

Puffer Finance

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	DeFi
Timeline	2024-02-01 through 2024-02-07
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	docs
Source Code	• PufferFinance/pufETH #6802ef7
Auditors	• Hytham Farah Auditing Engineer • Roman Rohleder Senior Auditing Engineer • Jeffrey Kam Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	11 Fixed: 1 Acknowledged: 9 Mitigated: 1
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	6 Acknowledged: 5 Mitigated: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	3 Acknowledged: 3

Summary of Findings

Puffer Finance is a DeFi protocol leveraging Eigenlayer to offer users an interface for pooling funds and earning yield on their Ethereum by participating in staking strategies. At the moment, the protocol makes use of stETH, but this will be gradually phased out in later iterations.

Critical administrative functionalities are managed primarily by various multi-sig wallets controlled by administrators or the community.

For this audit, Quantstamp focused primarily on functionality around depositing, interactions with Lido and Eigenlayer, timelocked access control as well as the deployment of those contracts.

Our findings pertain to interactions with an external protocol e.g. an unchecked call to a 1Inch router or the use of a deprecated contract. We also uncovered that the deployment script does not follow the least privilege principle, given that there is reasonable usage for this, we followed up with the dev team to see if the access control rights were revoked. Note that this is reasonable given that developers will want higher privilege for flexibility right after deployment in case arguments need to be adjusted, then revoking the access controls before making the system public to normal users.

Furthermore, we recommend more extensive testing to increase coverage across the protocol.

Update: We note that all issues that were pointed out related to deployment scripts have been effectively mitigated by the team by follow-up on-chain transactions ensuring all contracts are correctly configured. There is no risk of incorrect parameters or deployment issues for the current running system of the protocol. The team accepts that certain risks may be present to users who interact directly with the contract, however, these risks are only limited to those users themselves. Regular users will not be exposed to the risk as Puffer intends to craft well-made transactions on the user's behalf using their front end. We encourage the team to continue enhancing the test suite to increase code coverage.

ID	DESCRIPTION	SEVERITY	STATUS
PUFF-1	**Calling to-be-Deprecated Functions From Strategy Manager**	• Low	Acknowledged
PUFF-2	**Unsanitized Parameters in `swapAndDeposit1Inch()` May Lead to Inflated Vault Balances**	• Low	Acknowledged
PUFF-3	**Residual Funds of a Swap Are Not Returned to the User and Can Be Claimed by Others**	• Low	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
PUFF-4	Depositing with Permit Uses `msg.sender` Funds Instead of the Permit Owner's Funds	Medium	Fixed
PUFF-5	Missing Input Validation	Low	Mitigated
PUFF-6	Eigenlayer View Functions Should Be Used in `getELBackingEthAmount()`	Low	Acknowledged
PUFF-7	Unlocked Pragma	Informational	Acknowledged
PUFF-8	Privileged Roles	Low	Acknowledged
PUFF-9	`restricted` Should only Be Used on `external` or `public` Functions	Informational	Acknowledged
PUFF-10	No Event Emission in `Timelock.executeTransaction()` for Calls Through `COMMUNITY_MULTISIG`	Informational	Acknowledged
PUFF-11	Eigenlayer Slashing Can Lead to Incorrect Vault Accounting	Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

This audit is restricted to the following files ONLY:

- PufferDepositor.sol
- PufferVault.sol
- Timelock.sol
- DeployPuffETH.s.sol

Files Included

Repo:

https://github.com/PufferFinance/pufETH/tree/main/(6802ef7b9f9db80723c9dce15a451a0e40f0ee73)

Files:

- https://github.com/PufferFinance/pufETH/blob/audit-suggestions/src/PufferDepositor.sol,
- https://github.com/PufferFinance/pufETH/blob/audit-suggestions/src/PufferVault.sol,
- https://github.com/PufferFinance/pufETH/blob/audit-suggestions/src/Timelock.sol,
- https://github.com/PufferFinance/pufETH/blob/audit-suggestions/script/DeployPuffETH.s.sol

Findings

PUFF-1

Calling to-be-Deprecated Functions From Strategy Manager

• Low Acknowledged

> **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > We checked with the Eigenlayer team and their mainnet addresses are being used currently by PufferVault. The PufferVault contract will be updated prior to Puffer's mainnet, so this will be upgraded then.

File(s) affected: `PufferVault`

Description: Note that Eigenlayer has a to-be-deprecated strategy manager, which was likely deprecated due to the problematic functions `queueWithdrawl()` and `completeQueuedWithdrawl()` following the issue outlined here. These deprecated functions are called in the `PufferVault` contract.

Note that there is currently a function `migrateQueuedWithdrawl()` for migrating queued withdrawals in the new strategy manager but will likely be removed with the next contract.

Recommendation: Do not make use of the functions in the deprecated strategy manager contract.

PUFF-2

Unsanitized Parameters in `swapAndDeposit1Inch()` May Lead to Inflated Vault Balances

• Low Acknowledged

> **Update**
>
> **To Be Fixed**. The client has outlined the intended fix below:
>
> 1. Acknowledged that the issue exists if a user manually modifies the transaction calldata generated by the UI
> 2. The swapping functionality will be removed shortly from the code as it is no longer in the roadmap to accept other tokens (to be swapped for stETH)
>   - The depositor contract will no longer swap and use 1Inch or any other DEX

File(s) affected: `PufferDepositor`

Description: We first would like to point out that there is minimal impact based on the assumption that this contract does not hold funds.

The function `swapAndDeposit1Inch()` transfers a caller-supplied amount of a caller-supplied token from the caller to the depositor contract and increases the allowance of said token to the `_1INCH_ROUTER` contract accordingly. Followingly, the `_1INCH_ROUTER` contract is called with caller-supplied calldata, where the return value is assumed to be the `amountOut` of a token swap.

However, since the input token may be an arbitrary address a malicious actor may deploy an ERC20-conforming but useless contract and use that as the input token. Next, instead of calling `_1INCH_ROUTER` with a swap route, the malicious actor may call an arbitrary non-reverting function, where the return value is non-zero, leading him to be awarded a non-zero amount in the vault.

Recommendation: We recommend the following:

1. Implement and enforce a whitelist of tokens that are accepted by the depositor contract.
2. Implement and enforce a whitelist of calldata/function signatures that can be made to the `_1INCH_ROUTER` contract or hardcoding the route as is done in `swapAndDeposit()`.

PUFF-3

Residual Funds of a Swap Are Not Returned to the User and Can Be Claimed by Others

Low Acknowledged

Update

To Be Fixed: The client has outlined the intended fix below:

1. Similar to Puff-2, the swap functionality will be fully removed in the coming update.
2. Similarly, this issue only exists if the user manually modifies the calldata provided by the UI.
    - Given that the contracts are upgradable, any residual funds will not be locked and can be withdrawn with planned upgrades.

File(s) affected: `PufferDepositor`

Description: The functions `swapAndDeposit()` and `swapAndDeposit1Inch()` may leave some residual amount of tokens approved but unused within the contract. This occurs when these functions do not fully utilize the `amount` provided during the swap. For instance, in the following snippet in `swapAndDeposit()`, it is possible that `_SUSHI_ROUTER.processRoute()` does not use up all of the `_amountIn`, leaving some residual amount of the input tokens locked in the contract. As the contract is upgradeable, it is possible to rescue these funds, albeit in a manual manner.

```
if (tokenIn != _NATIVE_ETH) {
    SafeERC20.safeTransferFrom(IERC20(tokenIn), msg.sender, address(this), amountIn);
    SafeERC20.safeIncreaseAllowance(IERC20(tokenIn), address(_SUSHI_ROUTER), amountIn);
}

uint256 stETHAmountOut = _SUSHI_ROUTER.processRoute{ value: msg.value }( ... _amountIn ... )
```

Similarly, `_1INCH_ROUTER.call{ value: msg.value }(callData)` in `swapAndDeposit1Inch()` does not necessarily enforce the router to use all of the `_amountIn` input tokens (e.g. the specified amount to swap can be less than `_amountIn`), potentially leading to leftovers remaining in the contract. For example, in the generic router inherited by the `1INCH` router, we have the following function. It shows that the unspent input tokens can be sent back to the `msg.sender`, which is the `PufferDepositor` contract in this case.

```
function swap() {
    ...
    spentAmount = desc.amount;
    ...
    uint256 unspentAmount = srcToken.uniBalanceOf(address(this));
    srcToken.uniTransfer(payable(msg.sender), unspentAmount);
    ...
}
```

However, there's another attack vector stemming from this issue. Since we increase the allowance for `1INCH_ROUTER` by `_amountIn`, the residual funds could potentially be misappropriated by other users. This is possible due to the lack of constraints on the `callData` for `swapAndDeposit1Inch()`.

Exploit Scenario: In particular, we illustrate with the following example.

1. Suppose Alice decides to swap and deposit `100 token A` into the vault using `swapAndDeposit1Inch()`.
2. During the swap, not all of the token A are used, so `10 token A` remains in the contract. However, in `swapAndDeposit1Inch()`, we increased the allowance of token A by `100` to be used by `_1INCH_ROUTER`. This means, after the swap, the `_1INCH_ROUTER` can still use `10 token A`.
3. Now, an attacker Bob can "steal" the `10 token A` by crafting a custom `callData` to `swapAndDeposit1Inch()`.
    - First, Bob calls `swapAndDeposit1Inch()` with `tokenIn` being Native ETH and `_amountIn` being zero (see related issue PUFF-3).
    - The calldata is crafted such that it will try to swap `10 token A` to `stETH`.
    - Since `_1INCH_ROUTER` still has an approval to use `10 token A`, this will succeed and so the funds will be deposited into the vault under Bob's address.

Recommendation: To remove the manual process of rescuing funds or preventing users from accidentally sending more funds than needed for the swap, we recommend checking the actual amount used during the swap and returning the unspent funds to the user. Furthermore, we also suggest removing the approval (corresponding to the allowance increase) after the swap to avoid the potential exploit above.

PUFF-4
Depositing with Permit Uses `msg.sender` Funds Instead of the Permit Owner's Funds

Medium | Fixed

> **Update**
> We note that in the currently deployed version of this contract, with proxy address 0×4aA799C5dfc01ee7d790e3bf1a7C2257CE1DcefF and implementation address 0×7276925e42F9c4054afA2fad80fA79520C453D6A the problem has been fixed.

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> For this, we followed the OZ recommendation
> https://docs.openzeppelin.com/contracts/5.x/api/token/erc20#security_consideration

File(s) affected: `PufferDepositor`

Description: The function `swapAndDepositWithPermit()` uses funds from `msg.sender` instead of the permit owner's funds. This discrepancy arises because the permit mechanism allows a user (the permit owner) to authorize another user (the spender) to spend tokens on their behalf. However, in the current implementation, the function executes transactions using `msg.sender`'s funds, potentially leading to a mismatch between the user who granted the permit and the actual funds being used. The primary concern here occurs when `msg.sender` is not the same as the permit owner, potentially leading to unexpected reverts if the contract does not have approval from `msg.sender` to use his funds, or worse yet, swap and deposit `msg.sender`'s funds into the `PufferVault` instead.

Similar issue exists for `swapAndDeposit1InchWithPermit()`, `depositWstETH()`, and `depositStETH()`.

Recommendation: Revise `swapAndDeposit()` to consider the case where it is called by `swapAndDepositWithPermit()` so that funds are transferred from the permit owner and the deposits are attributed to the permit owner instead of `msg.sender`. Apply similar changes to `swapAndDeposit1InchWithPermit()`, `depositWstETH()`, and `depositStETH()` as well.

PUFF-5 Missing Input Validation

Low | Mitigated

> **Update**
> The issue has been mitigated in the deployed version of the code.
>
> The contracts are already deployed on mainnet. Here are our contracts: PufferVault (pufETH token): 0xD9A442856C234a39a81a089C06451EBAa4306a72 PufferDepositor: 0×4aA799C5dfc01ee7d790e3bf1a7C2257CE1DcefF AccessManager: 0×8c1686069474410E6243425f4a10177a94EBEE11 Timelock: 0×3C28B7c7Ba1A1f55c9Ce66b263B33B204f2126eA Here is an article describing the multisigs https://blocksec.com/blog/demystify-the-access-control-mechanism-in-puffer-protocol
>
> Quantstamp has checked the deployed contracts and validated that correct access control parameters have been set, but note that the risk persists in case a redeployment is ever needed.

File(s) affected: `DeployPuffETH`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

1. `DeployPuffETH.s.run()`: `_broadcaster`, `operationsMultisig`, `pauserMultisig` and `communityMultisig` not checked to be different from `address(0)`. It would also be beneficial to check that the addresses of the multisigs are contracts to ensure that they are not Ethereum EOAs.

Note that the severity of this missing input validation is heightened due to the use of the `envOr` function which will create a dummy address using `makeAddr()` in case there is a missing environment variable for the addresses in question.

Recommendation: We recommend adding the relevant checks. In particular, ensure the address environment variables are loaded properly or do not run the script.

PUFF-6
Eigenlayer View Functions Should Be Used in `getELBackingEthAmount()`

Low | Acknowledged

Update

To Be Fixed

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Acknowledged. We will fix this in the next version of the contracts

File(s) affected: `PufferVault`

Description: The function `getELBackingEthAmount()` in `PufferVault` is intended to be a view function, in which it uses the function `_EIGEN_STETH_STRATEGY.userUnderlying()`. However, based on EigenLayer's code documentation, `userUnderlying()` may modify states. One should use `userUnderlyingView()` for the non-state-modifying version. While this is not a problem now because `userUnderlying()` simply calls `userUnderlyingView()`, it is nonetheless best practice to use `userUnderlyingView()` instead to mitigate against unexpected future changes to `userUnderlying()`.

Recommendation: Consider using `userUnderlyingView()` instead of `userUnderlying()`.

PUFF-7 Unlocked Pragma

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client.

File(s) affected: `Timelock`, `PufferVault`, `PufferDepositor`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*` or `pragma solidity (>=)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

PUFF-8 Privileged Roles

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The documentation is In progress. We've queued the removal of the ROLE_ID_UPGRADER from the operations multi-sig so only the community has the power to do an instant upgrade, operations needs to queue the tx through the Timelock On the Timelock we've increased the MINIMUM_DELAY to 7 days. See this report for more info https://blocksec.com/blog/demystify-the-access-control-mechanism-in-puffer-protocol

File(s) affected: `PufferDepositor`, `PufferVault`, `Timelock`, `DeployPuffETH`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

The `PufferDepositor.sol` contract contains the following privileged roles (use of `restricted`-modifier from `AccessManaged.sol`):

1. `PUBLIC_ROLE` (any address), as initialized during deployment can call the following listed functions **as long as not paused via** `Timelock.sol`:
    1. `swapAndDeposit1Inch()`: Swap **arbitrary** tokens via 1inch and generate vault rewards.
    2. `swapAndDepositWithPermit1Inch()`: As above, but preceeded by a permit.
    3. `swapAndDeposit()`: Swap an **arbitrary** input token to `stETH` using sushi router and generate vault rewards.
    4. `swapAndDepositWithPermit()`: As above, but preceeded by a permit.
    5. `depositWstETH()`: Transfer wrapped `stETH` and generate according vault rewards (1:1).
    6. `depositStETH()`: Transfer `stETH` and generate vault rewards (1:1).
2. `ADMIN_ROLE`, as initialized during deployment to the `Timelock.sol` contract:
    1. `_authorizeUpgrade()`: **Perform contract upgrades**.

The `PufferVault.sol` contract contains the following privileged roles (use of `restricted`-modifier from `AccessManaged.sol`):

1. `PUBLIC_ROLE` (any address), as initialized during deployment and can call the following listed functions **as long as not paused via** `Timelock.sol`:

1. `deposit()` : Deposit `stETH` directly and generate vault rewards.
2. `mint()` : Deposit `stETH` directly and generate vault rewards.
3. `ADMIN_ROLE` , as initialized during deployment to the `Timelock.sol` contract:
4. `_authorizeUpgrade()` : **Perform contract upgrades**.

2. `ROLE_ID_OPERATORS` , as initialized during deployment to address `operationsMultisig` can call the following listed functions **as long as not paused via** `Timelock.sol` :
   1. `depositToEigenLayer()`
   2. `initiateStETHWithdrawalFromEigenLayer()`
   3. `initiateETHWithdrawalsFromLido()`

The `Timelock.sol` contract contains the following privileged roles:

1. `OPERATIONS_MULTISIG` , as initialized during the constructor call to parameter `operationsMultisig` :
   1. `queueTransaction()` : Queue **arbitrary calls** for timelocked (**minimum delay of 2 days**) execution.
   2. `cancelTransaction()` : Cancel queued timelocked transactions.
   3. `executeTransaction()` : Execute queues timelocked transactions that have surpassed their lock time.
2. `COMMUNITY_MULTISIG` , as initialized during the constructor call to parameter `communityMultisig` :
   1. `executeTransaction()` : **Execute arbitrary calls immediately, bypassing the timelock.**
3. `pauserMultisig` , as initialized during the constructor call to parameter `pauser` :
   1. `pause()` : Pause (prevent further execution) of arbitrary functions that use the `restricted` -modifier (see above).

Note: Functions that may be executed through the `Timelock.executeTransaction()` function include **any external function**, including:

1. Pause arbitrary `restricted` -modifier protected functions (see above).
2. Unpausing, paused functions.
3. Changing the `pauserMultisig` pauser role address.
4. Changing the timelock delay to an arbitrary number higher or equal to 2 days.
5. Perform contract upgrades.
6. Add/Remove any addresses to/from any role.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

PUFF-9

`restricted` Should only Be Used on `external` or `public` Functions

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.

File(s) affected: `PufferVault` , `DeployPuffETH`

Description: The use of the `restricted` modifier in `_authorizeUpgrade()` is not recommended per OpenZepplin's recommendation. OpenZeppelin's documentation recommends applying the restricted modifier only to external or public functions. While this does not pose a security issue because the restrictions are set correctly to the `upgradeToAndCall()` selector in the deployment script, this may not be the best practice when using the `restricted` modifier.

Recommendation: Consider whether this is acceptable or override `upgradeToAndCall()` with the modifier instead.

PUFF-10

No Event Emission in `Timelock.executeTransaction()` for Calls Through `COMMUNITY_MULTISIG`

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> We are utilizing BlockSec and Hexagate monitors and alerts to monitor the Community multisig and the owners.

File(s) affected: `Timelock`

Description: Function `executeTransaction()` executes queued operations when the corresponding delay has been reached and when called by `OPERATIONS_MULTISIG` . However, calls made by `COMMUNITY_MULTISIG` may execute arbitrary operations immediately, bypassing otherwise additionally applied checks and operations. One of these being the emission of the event `TransactionExecuted()` , which may lead to inconsistent tracking and interaction with off-chain components.

Recommendation: We recommend restructuring the call to `_executeTransaction()` in `Timelock.sol#L197` to be split into two parts, as is done in `Timelock.sol#L218` and emit said event before returning `(success, returnData)` .

PUFF-11

Eigenlayer Slashing Can Lead to Incorrect Vault Accounting

• **Undetermined** Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The current version of our contracts is for our pre-mainent launch of the staking/restaking product. This means that we will not be participating in any restaking/delegation in this version of the contracts.

File(s) affected: `PufferVault`

Description: There is a concern regarding the potential impact of Eigenlayer slashing on the internal accounting of the PufferVault. Slashing can happen in Eigenlayer StrategyManager, which can penalize a staker by reducing their shares in a particular strategy. If this happens, the accounting of assets in the vault can be out-of-sync because we keep track of the variable `eigenLayerPendingWithdrawalSharesAmount` internally, but the pending shares can potentially be slashed as well.

Recommendation: It is unclear what the impact of this is concerning the overall system behavior. It likely affects `getELBackingEthAmount()` and thus `totalAssets()`, but these are view functions and are likely used by off-chain components. We recommend the team confirm the impact of this issue and apply any accounting adjustments as necessary.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `003...49e ./src/PufferVaultMainnet.sol`
- `2e3...079 ./src/NoImplementation.sol`
- `b75...285 ./src/Timelock.sol`
- `fc7...f89 ./src/PufferOracle.sol`
- `bea...ff9 ./src/PufferDepositorStorage.sol`
- `5ab...b62 ./src/PufferVault.sol`
- `b4a...d90 ./src/PufferDepositor.sol`
- `1f3...b53 ./src/PufferVaultStorage.sol`
- `b59...75d ./src/structs/PufferDeployment.sol`
- `f1b...3ab ./src/interface/IPufferDepositor.sol`

- `c22...8e8 ./src/interface/IPufferVault.sol`
- `86c...ae8 ./src/interface/Other/ISushiRouter.sol`
- `ef7...ba0 ./src/interface/Other/IWETH.sol`
- `01c...987 ./src/interface/Lido/IStETH.sol`
- `36d...b1f ./src/interface/Lido/IWstETH.sol`
- `d6f...7ef ./src/interface/Lido/ILidoWithdrawalQueue.sol`
- `df0...5fa ./src/interface/EigenLayer/IEigenLayer.sol`
- `6d9...a23 ./src/interface/EigenLayer/IStrategy.sol`

Tests

- `fda...1d8 ./test/Integration/PufferTest.integration.t.sol`
- `64b...aa2 ./test/unit/PufETH.t.sol`
- `e44...d98 ./test/unit/Timelock.t.sol`
- `a62...3ca ./test/mocks/EigenLayerManagerMock.sol`
- `8ba...310 ./test/mocks/stETHStrategyMock.sol`
- `2f4...bd4 ./test/mocks/LidoWithdrawalQueueMock.sol`
- `5b4...8f7 ./test/mocks/stETHMock.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.8.3

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

All results were either included in the report as findings or discarded as false positives.

Test Suite Results

The test suit would benefit from more focused unit testing on the `PufferVault` and `PufferDepositor` contracts in general. The assessment of the test quality is medium at the time of the initial audit.

Update: Tests have been improved significantly since the initial audit. The team has added 2 new test suites, for `PufferDepositor` and `PufferVault` contracts, as well as 11 new tests. The suite now features a healthy blend of integration and unit tests.

```
Ran 13 tests for test/unit/Timelock.t.sol:TimelockTest
[PASS] test_cancel_reverts_if_caller_unauthorized(address) (runs: 256, μ: 11272, ~: 11272)
[PASS] test_cancel_transaction() (gas: 38801)
[PASS] test_change_pauser() (gas: 23527)
[PASS] test_execute_reverts_if_caller_unauthorized(address) (runs: 256, μ: 13166, ~: 13166)
[PASS] test_initial_access_manager_setup(address) (runs: 256, μ: 68521, ~: 68521)
[PASS] test_pause_depositor(address) (runs: 256, μ: 59282, ~: 59282)
[PASS] test_pause_should_revert_if_bad_caller(address) (runs: 256, μ: 15709, ~: 15709)
[PASS] test_queue_should_revert_if_operations_is_not_the_caller(address) (runs: 256, μ: 11273, ~: 11273)
[PASS] test_queueing_duplicate_transaction_different_operation_id() (gas: 79063)
[PASS] test_setDelay_reverts_if_caller_unauthorized(address) (runs: 256, μ: 9506, ~: 9506)
[PASS] test_setPauser_reverts_if_caller_unauthorized(address) (runs: 256, μ: 9545, ~: 9545)
[PASS] test_set_delay_queued() (gas: 43164)
[PASS] test_update_delay_from_community_without_timelock() (gas: 31482)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 5.61s
```

```
Ran 6 tests for test/Integration/PufferVaultMainnet.fork.t.sol:PufferVaultMainnetForkTest
[PASS] test_deposit() (gas: 254761)
[PASS] test_eth_weth_stETH_deposits() (gas: 426745)
[PASS] test_max_deposit() (gas: 52161)
[PASS] test_max_withdrawal() (gas: 252761)
[PASS] test_mint() (gas: 274898)
[PASS] test_sanity() (gas: 122645)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 19.43s

Ran 3 tests for test/Integration/PufferDepositorMainnet.fork.t.sol:PufferDepositorMainnetForkTest
[PASS] test_stETH_permit_deposit() (gas: 352305)
[PASS] test_stETH_permit_deposit_to_bob() (gas: 314961)
[PASS] test_wstETH_permit_deposit() (gas: 291746)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 102.70s

Ran 23 tests for test/Integration/PufferTest.integration.t.sol:PufferTest
[PASS] test_1inch_complex_swap() (gas: 21829891)
[PASS] test_ape_to_pufETH() (gas: 468864)
[PASS] test_conversions_and_deposit_to_el() (gas: 1754220)
[PASS] test_deposit_stETH_permit() (gas: 315987)
[PASS] test_deposit_wstETH() (gas: 324761)
[PASS] test_deposit_wstETH_permit() (gas: 324163)
[PASS] test_depositingStETH_and_withdrawal() (gas: 1502959)
[PASS] test_eigenlayer_cap_reached() (gas: 232344)
[PASS] test_eth_1inch_swap() (gas: 393599)
[PASS] test_eth_sushi_swap() (gas: 367518)
[PASS] test_failed_swap_1inch() (gas: 179425)
[PASS] test_lido_withdrawal_dos() (gas: 696274)
[PASS] test_minting_and_lido_rebasing() (gas: 1039391)
[PASS] test_swap_1inch() (gas: 439801)
[PASS] test_swap_1inch_permit() (gas: 431084)
[PASS] test_upgrade_to_mainnet() (gas: 7050172)
[PASS] test_usdc_permit_upgrade() (gas: 21626461)
[PASS] test_usdc_to_pufETH() (gas: 513797)
[PASS] test_usdc_to_pufETH_permit() (gas: 516169)
[PASS] test_usdt_to_pufETH() (gas: 511286)
[PASS] test_withdraw_from_eigenLayer() (gas: 659458)
[PASS] test_withdraw_from_eigenLayer_dos() (gas: 888388)
[PASS] test_zero_stETH_deposit() (gas: 225443)
Test result: ok. 23 passed; 0 failed; 0 skipped; finished in 102.71s

Ran 28 tests for test/unit/PufETH.t.sol:PufETHTest
[PASS] testFail_redeem((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 627499, ~:
629987)
[PASS] testFail_withdraw((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 634292, ~:
635763)
[PASS] test_RT_deposit_redeem((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2369, ~:
2369)
[PASS] test_RT_deposit_withdraw((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2391,
~: 2391)
[PASS] test_RT_mint_redeem((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2347, ~:
2347)
[PASS] test_RT_mint_withdraw((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2391, ~:
2391)
[PASS] test_RT_redeem_deposit((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2392, ~:
2392)
[PASS] test_RT_redeem_mint((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2346, ~:
2346)
[PASS] test_RT_withdraw_deposit((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2389,
~: 2389)
[PASS] test_RT_withdraw_mint((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2347, ~:
2347)
```

```
[PASS] test_asset((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 480461, ~: 483491)
[PASS] test_convertToAssets((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 488560, ~:
492152)
[PASS] test_convertToShares((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 488869, ~:
491876)
[PASS] test_deposit((address[4],uint256[4],uint256[4],int256),uint256,uint256) (runs: 256, μ: 531754, ~:
536020)
[PASS] test_erc4626_interface() (gas: 238648)
[PASS] test_maxDeposit((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 480856, ~: 483508)
[PASS] test_maxMint((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 478986, ~: 483374)
[PASS] test_maxRedeem((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 478995, ~: 483611)
[PASS] test_maxWithdraw((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 483840, ~: 486802)
[PASS] test_mint((address[4],uint256[4],uint256[4],int256),uint256,uint256) (runs: 256, μ: 539872, ~:
542187)
[PASS] test_previewDeposit((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 529915, ~:
532818)
[PASS] test_previewMint((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 537106, ~:
539228)
[PASS] test_previewRedeem((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2390, ~:
2390)
[PASS] test_previewWithdraw((address[4],uint256[4],uint256[4],int256),uint256) (runs: 256, μ: 2346, ~:
2346)
[PASS] test_redeem((address[4],uint256[4],uint256[4],int256),uint256,uint256) (runs: 256, μ: 2345, ~:
2345)
[PASS] test_roles_setup() (gas: 99794)
[PASS] test_totalAssets((address[4],uint256[4],uint256[4],int256)) (runs: 256, μ: 482385, ~: 485648)
[PASS] test_withdraw((address[4],uint256[4],uint256[4],int256),uint256,uint256) (runs: 256, μ: 2369, ~:
2369)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 102.71s

Ran 5 test suites in 333.17s: 73 tests passed, 0 failed, 0 skipped (73 total tests)
```

Code Coverage

We recommend getting code coverage above 90% for the `PufferDepositor` as well as the `Timelock` contracts.

Update: Coverage has improved since the initial audit. `PufferVault` has increased its function coverage by 6.25%, `PufferDepositor` has increased function coverage by 19.64%, and `PufferVaultMainnet` has increased line coverage by 20.42% to give a few examples. Given that the team plans to remove certain functions, we expect the coverage to improve with the future upgrade. We continue to encourage the team to increase coverage on the `PufferDepositor` contract.

File	% Lines	% Statements	% Branches	% Funcs
script/DeployPuffETH.s.sol	100.00% (**78/**78)	100.00% (**97/**97)	100.00% (**2/**2)	100.00% (**5/**5)
script/GenerateAccessManagerCallData.sol	100.00% (**25/**25)	100.00% (**35/**35)	100.00% (**0/**0)	100.00% (**1/**1)
src/NoImplementation.sol	100.00% (**1/**1)	100.00% (**1/**1)	50.00% (**1/**2)	100.00% (**1/**1)
src/PufferDepositor.sol	50.00% (**15/**30)	47.50% (**19/**40)	30.00% (**3/**10)	62.50% (**5/**8)
src/PufferDepositorMainnet.sol	100.00% (**7/**7)	100.00% (**10/**10)	100.00% (**0/**0)	66.67% (**2/**3)
src/PufferVault.sol	98.18% (**54/**55)	98.80% (**82/**83)	100.00% (**4/**4)	81.25% (**13/**16)
src/PufferVaultMainnet.sol	70.42%	76.47%	33.33% (**6/**18)	64.71% (**11/**17)

File	% Lines	% Statements	% Branches	% Funcs
	(**50/**71)	(**78/**102)		
src/PufferVaultStorage.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
src/Timelock.sol	82.46% (**47/**57)	82.28% (**65/**79)	68.75% (**22/**32)	90.00% (**9/**10)
test/mocks/EigenLayerManagerMock.sol	100.00% (**0/**0)	100.00% (**0/**0)	100.00% (**0/**0)	0.00% (**0/**5)
test/mocks/LidoWithdrawalQueueMock.sol	100.00% (**0/**0)	100.00% (**0/**0)	100.00% (**0/**0)	0.00% (**0/**2)

Changelog

- 2024-02-07 - Initial report
- 2024-02-14 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.