Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2024.01.24, the SlowMist security team received the Puffer Finance team's security audit application for pufETH, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project 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.

This audit contains PufferDepositor, PufferVault, Timelock and DeployPuffETH. DeployPuffETH is used for contract deployment with the foundry suite. The PufferDepositor contract helps users swap tokens and deposit into the PufferVault contract. The PufferVault contract interacts with the Eigenlayer protocol. Timelock is used for delayed transaction execution.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risk of unintended claim operations	Design Logic Audit	Medium	Fixed
N2	Potential issues when `permitData.owner` is inconsistent with `msg.sender`	Design Logic Audit	Low	Fixed
N3	Authority transfer enhancement	Authority Control Vulnerability Audit	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/PufferFinance/pufETH/tree/feature/simple-timelock

commit: 9a2a470bd276b850daf66b15463d0a9ad9b38a0f

Fixed Version:

https://github.com/PufferFinance/pufETH

commit: c46d4f1de6e22b2b8ff33111a7852225aef443e6

Audit Scope:

- src/PufferDepositor.sol
- src/PufferVault.sol
- src/Timelock.sol
- script/DeployPuffETH.s.sol

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
queueTransaction	Public	Can Modify State	-
pause	Public	Can Modify State	-
cancelTransaction	Public	Can Modify State	-
executeTransaction	External	Can Modify State	-
setDelay	Public	Can Modify State	-
setPauser	Public	Can Modify State	-
_setPauser	Internal	Can Modify State	-
_setDelay	Internal	Can Modify State	-
_executeTransaction	Internal	Can Modify State	-
_validateAddresses	Internal	-	-

PufferDepositor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	-
initialize	External	Can Modify State	initializer
swapAndDeposit1Inch	Public	Can Modify State	restricted
swapAndDepositWithPermit1Inch	Public	Can Modify State	restricted
swapAndDeposit	Public	Can Modify State	restricted

PufferDepositor			
swapAndDepositWithPermit	Public	Can Modify State	restricted
depositWstETH	External	Can Modify State	restricted
_authorizeUpgrade	Internal	Can Modify State	restricted

PufferVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	-
initialize	External	Can Modify State	initializer
<Receive Ether>	External	Payable	-
claimWithdrawalsFromLido	External	Can Modify State	-
redeem	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
totalAssets	Public	-	-
getELBackingEthAmount	Public	-	-
getPendingLidoETHAmount	Public	-	-
depositToEigenLayer	External	Can Modify State	restricted
initiateStETHWithdrawalFromEigenLayer	External	Can Modify State	restricted
claimWithdrawalFromEigenLayer	External	Can Modify State	-
initiateETHWithdrawalsFromLido	External	Can Modify State	restricted
onERC721Received	External	Can Modify State	-
decimals	Public	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

4.3 Vulnerability Summary

[N1] [Medium] Risk of unintended claim operations

Category: Design Logic Audit

Content

In the PufferVault contract, Users can collect ETH tokens to be claimed in the lido withdrawal process by calling the claimWithdrawalsFromLido function. The normal expectation of this function is that the incoming requestIds parameter should be created by the initiateETHWithdrawalsFromLido function, and only then the lidoLockedETH variable will be deducted correctly.

However, a malicious user can directly call requestWithdrawals function in lido to generate requestIds, and then call claimWithdrawalsFromLido function, lidoLockedETH will be deducted additionally, resulting in a normal claim operation failing due to insufficient lidoLockedETH.

The following scenarios can be used as a reference:

1. The contract has a total of 10 ETH total deposited in Lido, at which point a normal user calls the initiateETHWithdrawalsFromLido function to submit a request to withdraw 10 ETH($.lidoLockedETH = 10).
2. A malicious user calls requestWithdrawals function directly on Lido to generate a withdrawal request to withdraw 1 ETH(need to specify WithdrawalRequest._owner as pufferVault).
3. The malicious user calls the claimWithdrawalsFromLido function and passes in the requestIds generated in step 2, the value of $.lidoLockedETH is equal to 9.
4. A normal user calls the claimWithdrawalsFromLido function and passes in the requestIds generated in the first step, at which point the amount of ETH to be fetched is 10, while the value of $.lidoLockedETH is 9, which causes $.lidoLockedETH -= msg.value to overflow and the entire transaction fails.
5. So the final result is that 10 ETH cannot be successfully withdrawn through the pufferVault contract. And this security risk also exists when withdrawing ETH from EigenLayer(Specify withdrawer as the pufferVault address to achieve the same effect).

Code location: src/PufferVault.sol

```
    receive() external payable virtual {
        VaultStorage storage $ = _getPufferVaultStorage();
        if ($.isLidoWithdrawal) {
            $.lidoLockedETH -= msg.value;
        }
    }
    ...
    function claimWithdrawalsFromLido(uint256[] calldata requestIds) external virtual
{
        ...
    }
    ...
    function claimWithdrawalFromEigenLayer(
        IEigenLayer.QueuedWithdrawal calldata queuedWithdrawal,
        IERC20[] calldata tokens,
        uint256 middlewareTimesIndex
    ) external virtual {
        ...

        $.eigenLayerPendingWithdrawalSharesAmount -= queuedWithdrawal.shares[0];

        ...
    }
```

Solution

It is recommended that when withdrawing from lido, the contract need to check whether the requestIds passed in during the claim operation are created by the initiateETHWithdrawalsFromLido function; and when withdrawing from eigenlayer, the claimWithdrawalFromEigenLayer function need to check whether the depositor passed in the queuedWithdrawal parameter is the address of the pufferVault contract (ie address (this)).

Status

Fixed

[N2] [Low] Potential issues when `permitData.owner` is inconsistent with `msg.sender`

Category: Design Logic Audit

Content

In the PufferDepositor contract, the swapAndDepositWithPermit1Inch/swapAndDepositWithPermit functions are used to verify a user's permit and call swapAndDeposit to help users swap tokens and deposit to the vault in one transaction. However, note that the owner for permit verification is the `permitData.owner` passed in by the user, while the `from` address for token transfers in `swapAndDeposit` uses `msg.sender`. When the `permitData.owner` passed in by the user is not `msg.sender`, this can lead to unexpected token transfers.

Here is a simple example:

Address A previously approved tokens to the PufferDepositor contract. Now address B signs a permit, but address B does not have enough native tokens to pay transaction fees. So it delegates address A to call swapAndDepositWithPermit1Inch/swapAndDepositWithPermit, to transfer tokens from address B into the PufferDepositor contract. But actually `swapAndDeposit` transfers from address A instead of address B. This causes unnecessary trouble for users.

Code location: src/PufferDepositor.sol

```
    function swapAndDepositWithPermit1Inch(
        ...
    ) public virtual restricted returns (uint256 pufETHAmount) {
        try ERC20Permit(address(tokenIn)).permit({
            owner: permitData.owner,
            ...
        }) { } catch { }

        return swapAndDeposit1Inch(tokenIn, permitData.amount, callData);
    }

    function swapAndDepositWithPermit(
        ...
    ) public virtual restricted returns (uint256 pufETHAmount) {
        try ERC20Permit(address(tokenIn)).permit({
            owner: permitData.owner,
            ...
        }) { } catch { }

        return swapAndDeposit(tokenIn, permitData.amount, amountOutMin, routeCode);
    }

    function swapAndDeposit(address tokenIn, uint256 amountIn, uint256 amountOutMin,
bytes calldata routeCode)
        public
        virtual
        restricted
        returns (uint256 pufETHAmount)
    {
        SafeERC20.safeTransferFrom(IERC20(tokenIn), msg.sender, address(this),
amountIn);
        SafeERC20.safeIncreaseAllowance(IERC20(tokenIn), address(_SUSHI_ROUTER),
amountIn);

        ...
    }
```

Solution

It is recommended to change `permitData.owner` to `msg.sender` to avoid this issue.

Status

Fixed

[N3] [Suggestion] Authority transfer enhancement

Category: Authority Control Vulnerability Audit

Content

The pauserMultisig role does not adopt the pending and access processes. If the pauserMultisig is incorrectly set, the owner permission will be lost.

Code location: src/Timelock.sol

```
    function _setPauser(address newPauser) internal {
        emit PauserChanged(pauserMultisig, newPauser);
        pauserMultisig = newPauser;
    }
```

Solution

It is recommended to adopt the pending and access processes. Only the new pauserMultisig role can accept the permissions to transfer.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002401290002	SlowMist Security Team	2024.01.24 - 2024.01.29	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 1 low risk, 1 suggestion. All the findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist