Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.09, the SlowMist security team received the Puffer Finance team's security audit application for Puffer Finance Phase2, 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. This audit consisted of two main projects: pufETH and pufferPool.

Users will deposit wsteth or steth via pufETH's PufferDepositorV2 contract and receive a deposit voucher(the pufETH token).

The pufferPool is mainly responsible for validator and node operations, including validator registration, activation of nodes, skip activation and withdrawal exit. Validator registration need cost the pufETH and validator ticket tokens. Potential gains are made by creating and activating nodes in the eigenlayer protocol. Most of these operations need to be verified by the guardian module, which uses a multi-signature model to ensure that the operations are reasonable.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Unexpected amount of deposits	Design Logic Audit	Information	Acknowledged
N2	Redundant configurations in the initialization operation	Gas Optimization Audit	Suggestion	Acknowledged
N3	Missing last withdrawal time check	Design Logic Audit	Medium	Acknowledged
N4	Missing scope limit when setting the vtPenalty	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/PufferFinance/pufETH

commit: 5db7863db529e007a43bacd88aa7809332027fae

https://github.com/PufferFinance/PufferPool

commit: 6055908dbd51462007fd71c94a32ae564f70f786

Fixed Version:

https://github.com/PufferFinance/pufETH

commit: 3d41da2d8e4eb7e89a817fa8b5848bca839954a1

https://github.com/PufferFinance/PufferPool

commit: 376c17769b384b698a52565ee95ab7ea6c5fcd33

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:

PufferDepositorV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	-
depositWstETH	External	Can Modify State	restricted
depositStETH	External	Can Modify State	restricted
_authorizeUpgrade	Internal	Can Modify State	restricted

PufferVaultV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	PufferVault
<Receive Ether>	External	Payable	-
initialize	Public	Can Modify State	reinitializer
totalAssets	Public	-	-
withdraw	Public	Can Modify State	revertIfDeposited restricted
redeem	Public	Can Modify State	revertIfDeposited restricted
depositETH	Public	Payable	markDeposit restricted
depositStETH	Public	Can Modify State	markDeposit restricted
deposit	Public	Can Modify State	markDeposit restricted

PufferVaultV2			
mint	Public	Can Modify State	markDeposit restricted
initiateETHWithdrawalsFromLido	External	Can Modify State	restricted
claimWithdrawalsFromLido	External	Can Modify State	restricted
transferETH	External	Can Modify State	restricted
burn	Public	Can Modify State	restricted
convertToSharesUp	Public	-	-
setDailyWithdrawalLimit	External	Can Modify State	restricted
setExitFeeBasisPoints	External	Can Modify State	restricted
getRemainingAssetsDailyWithdrawalLimit	Public	-	-
maxWithdraw	Public	-	-
maxRedeem	Public	-	-
previewWithdraw	Public	-	-
previewRedeem	Public	-	-
getExitFeeBasisPoints	Public	-	-
initiateStETHWithdrawalFromEigenLayer	External	Can Modify State	restricted
claimWithdrawalFromEigenLayerM2	External	Can Modify State	restricted
claimWithdrawalFromEigenLayer	External	Can Modify State	-
depositToEigenLayer	External	Can Modify State	-
_feeOnRaw	Internal	-	-
_feeOnTotal	Internal	-	-

PufferVaultV2			
_wrapETH	Internal	Can Modify State	-
_updateDailyWithdrawals	Internal	Can Modify State	-
_setDailyWithdrawalLimit	Internal	Can Modify State	-
_setExitFeeBasisPoints	Internal	Can Modify State	-
_resetDailyWithdrawals	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	restricted
_getERC4626StorageInternal	Private	-	-

BalancerRateProvider			
Function Name	Visibility	Mutability	Modifiers
getRate	External	-	-

EnclaveVerifier			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManaged
getIntelRootCAPubKey	External	-	-
addLeafX509	External	Can Modify State	-
removeLeafX509	External	Can Modify State	restricted
verifyEvidence	External	-	-

GuardianModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	AccessManaged

GuardianModule			
<Receive Ether>	External	Payable	-
splitGuardianFunds	Public	Can Modify State	-
validateSkipProvisioning	External	-	-
validateProvisionNode	External	-	-
validateBatchWithdrawals	External	-	-
validateTotalNumberOfValidators	External	-	-
validateGuardiansEOASignatures	Public	-	-
validateGuardiansEnclaveSignatures	Public	-	-
setEjectionThreshold	External	Can Modify State	restricted
setGuardianEnclaveMeasurements	External	Can Modify State	restricted
addGuardian	External	Can Modify State	restricted
removeGuardian	External	Can Modify State	restricted
setThreshold	External	Can Modify State	restricted
getThreshold	External	-	-
getGuardians	External	-	-
rotateGuardianKey	External	Can Modify State	-
getEjectionThreshold	External	-	-
getGuardiansEnclaveAddress	External	-	-
getGuardiansEnclaveAddresses	Public	-	-
getGuardiansEnclavePubkeys	External	-	-
getMrenclave	External	-	-
getMrsigner	External	-	-

GuardianModule			
isGuardian	External	-	-
_addGuardian	Internal	Can Modify State	-
_setThreshold	Internal	Can Modify State	-
_setEjectionThreshold	Internal	Can Modify State	-
_validateSignatures	Internal	-	-

PufferModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	-
initialize	External	Can Modify State	initializer
<Receive Ether>	External	Payable	-
callStake	External	Payable	onlyPufferProtocol
queueWithdrawals	External	Can Modify State	onlyPufferModuleManager
completeQueuedWithdrawals	External	Can Modify State	whenNotPaused onlyPufferModuleManager
verifyAndProcessWithdrawals	External	Can Modify State	whenNotPaused onlyPufferModuleManager
verifyWithdrawalCredentials	External	Can Modify State	onlyPufferModuleManager
withdrawNonBeaconChainETHBalanceWei	External	Can Modify State	onlyPufferModuleManager
call	External	Can Modify State	onlyPufferProtocol
collectRewards	External	Can Modify State	whenNotPaused
postRewardsRoot	External	Can Modify State	whenNotPaused
callDelegateTo	External	Can Modify	onlyPufferModuleManager

PufferModule			
		State	
callUndelegate	External	Can Modify State	onlyPufferModuleManager
getLastProofOfRewardsBlock	External	-	-
getWithdrawalCredentials	Public	-	-
getEigenPod	External	-	-
NAME	External	-	-
_getPufferModuleStorage	Internal	-	-

PufferModuleManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
callVerifyWithdrawalCredentials	External	Can Modify State	restricted
callCompleteQueuedWithdrawals	External	Can Modify State	restricted
createNewPufferModule	External	Can Modify State	onlyPufferProtocol
callVerifyAndProcessWithdrawals	External	Can Modify State	restricted
callWithdrawNonBeaconChainETHBalanceWei	External	Can Modify State	restricted
callQueueWithdrawals	External	Can Modify State	restricted
createNewRestakingOperator	External	Can Modify State	restricted
callModifyOperatorDetails	External	Can Modify State	restricted
callUpdateMetadataURI	External	Can Modify	restricted

PufferModuleManager			
		State	
callOptIntoSlashing	External	Can Modify State	restricted
callDelegateTo	External	Can Modify State	restricted
callUndelegate	External	Can Modify State	restricted
updateAVSRegistrationSignatureProof	External	Can Modify State	restricted
_authorizeUpgrade	Internal	Can Modify State	restricted

PufferOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManaged
getLockedEthAmount	External	-	-
isOverBurstThreshold	External	-	-
getValidatorTicketPrice	External	-	-
setMintPrice	External	Can Modify State	restricted
_setMintPrice	Internal	Can Modify State	-

PufferOracleV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManaged
exitValidators	Public	Can Modify State	restricted
provisionNode	External	Can Modify State	restricted
setMintPrice	External	Can Modify State	restricted
setTotalNumberOfValidators	External	Can Modify State	restricted

PufferOracleV2			
getLockedEthAmount	External	-	-
getTotalNumberOfValidators	External	-	-
isOverBurstThreshold	External	-	-
getValidatorTicketPrice	External	-	-
_setMintPrice	Internal	Can Modify State	-

PufferProtocol			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
depositValidatorTickets	External	Can Modify State	restricted
withdrawValidatorTickets	External	Can Modify State	restricted
registerValidatorKey	External	Payable	restricted
provisionNode	External	Can Modify State	restricted
batchHandleWithdrawals	External	Can Modify State	restricted
skipProvisioning	External	Can Modify State	restricted
changeMinimumVTAmount	External	Can Modify State	restricted
createPufferModule	External	Can Modify State	restricted
setModuleWeights	External	Can Modify State	restricted
setValidatorLimitPerModule	External	Can Modify State	restricted
setVTPenalty	External	Can Modify State	restricted
getVTPenalty	External	-	-
getDepositDataRoot	External	-	-

PufferProtocol			
getValidators	External	-	-
getNextValidatorToProvision	Public	-	-
getNextValidatorToBeProvisionedIndex	External	-	-
getPendingValidatorIndex	External	-	-
getValidatorInfo	External	-	-
getNodeInfo	External	-	-
getModuleAddress	External	-	-
getWithdrawalCredentials	Public	-	-
getModuleWeights	External	-	-
getModuleSelectIndex	External	-	-
getValidatorTicketsBalance	Public	-	-
getMinimumVtAmount	Public	-	-
getPayload	External	-	-
getModuleLimitInformation	External	-	-
revertIfPaused	External	Can Modify State	restricted
_storeValidatorInformation	Internal	Can Modify State	-
_setValidatorLimitPerModule	Internal	Can Modify State	-
_setVTPenalty	Internal	Can Modify State	-
_setModuleWeights	Internal	Can Modify State	-
_createPufferModule	Internal	Can Modify State	-
_checkValidatorRegistrationInputs	Internal	-	-
_changeMinimumVTAmount	Internal	Can Modify State	-

PufferProtocol			
_getBondBurnAmount	Internal	-	-
_validateSignaturesAndProvisionValidator	Internal	Can Modify State	-
_getVTBurnAmount	Internal	-	-
_callPermit	Internal	Can Modify State	-
_decreaseNumberOfRegisteredValidators	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	restricted

PufferProtocolStorage			
Function Name	Visibility	Mutability	Modifiers
_getPufferProtocolStorage	Internal	-	-

RestakingOperator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
optIntoSlashing	External	Can Modify State	onlyPufferModuleManager
modifyOperatorDetails	External	Can Modify State	onlyPufferModuleManager
updateOperatorMetadataURI	External	Can Modify State	onlyPufferModuleManager
updateSignatureProof	External	Can Modify State	onlyPufferModuleManager
isValidSignature	External	-	-
_getRestakingOperatorStorage	Internal	-	-

ValidatorTicket			
Function Name	Visibility	Mutability	Modifiers

ValidatorTicket			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
purchaseValidatorTicket	External	Payable	restricted
burn	External	Can Modify State	restricted
setProtocolFeeRate	External	Can Modify State	restricted
setGuardiansFeeRate	External	Can Modify State	restricted
getProtocolFeeRate	External	-	-
getGuardiansFeeRate	External	-	-
_sendETH	Internal	Can Modify State	-
_setProtocolFeeRate	Internal	Can Modify State	-
_setGuardiansFeeRate	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	restricted

ValidatorTicketStorage			
Function Name	Visibility	Mutability	Modifiers
_getValidatorTicketStorage	Internal	-	-

4.3 Vulnerability Summary

[N1] [Information] Unexpected amount of deposits

Category: Design Logic Audit

Content

In the PufferDepositorV2 contract, When a user makes a deposit to a puffer vault contract, the amount of tokens transferred is the balance of all stETH in the PufferDepositorV2 contract. However, if another user transfers

stETH directly to the PufferDepositorV2 contract by mistake before making the deposit, then this part of the mistakenly transferred funds will also be deposited into the vault during the deposit operation.

Code Location:

pufETH/src/PufferDepositorV2.sol

```
    function depositWstETH(Permit calldata permitData, address recipient)
        external
        restricted
        returns (uint256 pufETHAmount)
    {
        ...

        return PUFFER_VAULT.depositStETH(_ST_ETH.sharesOf(address(this)), recipient);
    }

        function depositStETH(Permit calldata permitData, address recipient)
        external
        restricted
        returns (uint256 pufETHAmount)
    {
        ...
        return PUFFER_VAULT.depositStETH(_ST_ETH.sharesOf(address(this)), recipient);
    }
```

Solution

N/A

Status

Acknowledged; Response from the project team: This is a known behaviour and discussed on previous audits. If a user transfers stETH to the smart contract by mistake (without calling the proper functions), the first proper call to deposit after that deposit will consume all the deposited stETH for the caller.

[N2] [Suggestion] Redundant configurations in the initialization operation

Category: Gas Optimization Audit

Content

1.In the PufferVaultV2 contract, The daily withdrawal limit, total number of withdrawals for the day and the exit fee base points are set in the constructor when the contract is created. However, in the initialize function, these

values are set again repeatedly with the same values once more.

This is not in line with smart contract development best practices as it would consume additional gas.

Code Location:

pufETH/src/PufferVaultV2.sol#L47-79

```
    constructor(
        IStETH stETH,
        IWETH weth,
        ILidoWithdrawalQueue lidoWithdrawalQueue,
        IStrategy stETHStrategy,
        IEigenLayer eigenStrategyManager,
        IPufferOracle oracle,
        IDelegationManager delegationManager
    ) PufferVault(stETH, lidoWithdrawalQueue, stETHStrategy, eigenStrategyManager) {
        _WETH = weth;
        PUFFER_ORACLE = oracle;
        _DELEGATION_MANAGER = delegationManager;
        ERC4626Storage storage erc4626Storage = _getERC4626StorageInternal();
        erc4626Storage._asset = _WETH;
        _setDailyWithdrawalLimit(100 ether);
        _updateDailyWithdrawals(0);
        _setExitFeeBasisPoints(100); // 1%
        _disableInitializers();
    }

    ...

    function initialize() public reinitializer(2) {
        // In this initialization, we swap out the underlying stETH with WETH
        ERC4626Storage storage erc4626Storage = _getERC4626StorageInternal();
        erc4626Storage._asset = _WETH;
        _setDailyWithdrawalLimit(100 ether);
        _updateDailyWithdrawals(0);
        _setExitFeeBasisPoints(100); // 1%
    }
```

2.In the PufferProtocol contract, the initialize function will create a module named "_PUFFER_MODULE_0" by calling the _createPufferModule function. In the _createPufferModule function, the module name will be added to the moduleWeights array. Therefore, it is redundant to repeatedly call the _setModuleWeights function in the initialize function to set the weights.

Code Location:

PufferPool/src/PufferProtocol.sol#L131

```
    function initialize(address accessManager) external initializer {
        if (address(accessManager) == address(0)) {
            revert InvalidAddress();
        }
        __AccessManaged_init(accessManager);
        _createPufferModule(_PUFFER_MODULE_0);
        _setValidatorLimitPerModule(_PUFFER_MODULE_0, type(uint128).max);
        bytes32[] memory weights = new bytes32[](1);
        weights[0] = _PUFFER_MODULE_0;
        _setModuleWeights(weights);
        _changeMinimumVTAmount(28 ether); // 28 Validator Tickets
        _setVTPenalty(10 ether); // 10 Validator Tickets
    }

    ...

    function _createPufferModule(bytes32 moduleName) internal returns (address) {
        ...

        $.moduleWeights.push(moduleName);

        ...
    }
```

Solution

It is recommended that the redundant configuration in the constructor should be removed.

Status

Acknowledged; Response from the project team: This is due to the way the implemented Fuzzer (Echidna) functions and requires the constructor to set these values on the implementation. We will add inline documentation to explain why these parameters are set here.

[N3] [Medium] Missing last withdrawal time check

Category: Design Logic Audit

Content

In the PufferVaultV2 contract, the privileged role can set a new daily withdrawal limit by calling the setDailyWithdrawalLimit function. However, the time of the last withdrawal is not checked in this function. This

will cause the total withdrawals for the day to be unexpectedly reset to zero and the total withdrawals for the day to eventually exceed the upper limit unexpectedly.

Consider the following examples:

1.Assume that the current withdrawal limit for the day is 100 ETH, and 50 ETH have been withdrawn today.

2.At this time, the privileged role (dao) just wants to increase the daily withdrawal limit to 150 ETH. After calling setDailyWithdrawalLimit at this time, the upper limit of withdrawals for the day is set to 150 ETH, and the total withdrawals for the day are set to 0.

3.Then you can withdraw 150 ETH again that day, and the total amount of ETH withdrawn that day is 50 + 150 = 200.

Code Location:

pufETH/src/PufferVaultV2.sol#L356-359

```
    function setDailyWithdrawalLimit(uint96 newLimit) external restricted {
        _setDailyWithdrawalLimit(newLimit);
        _resetDailyWithdrawals();
    }
```

Solution

It is recommended that the value of lastWithdrawalDay should be checked before resetting the withdrawal amount for the day ($.lastWithdrawalDay < block.timestamp / 1 days).

Status

Acknowledged; Response from the project team: This is expected behavior, as we are aiming to reset the withdrawal limit on the time of changing withdrawal limit.

[N4] [Low] Missing scope limit when setting the vtPenalty

Category: Design Logic Audit

Content

1.In the PufferProtocol contract, privileged role can set the amount of validator tickets burned when skipping validator activation by calling the setVTPenalty function. However, there is no scope limit here. If the set

vtPenalty exceeds the amount of validator tickets in the validator account waiting for activation, the operation of skipping the activation will fail.

Code Location:

PufferPool/src/PufferProtocol.sol#L684-687

```
    function setVTPenalty(uint256 newPenaltyAmount) external restricted {
        _setVTPenalty(newPenaltyAmount);
    }

    ...

    function _setVTPenalty(uint256 newPenaltyAmount) internal {
        ProtocolStorage storage $ = _getPufferProtocolStorage();
        emit VTPenaltyChanged($.vtPenalty, newPenaltyAmount);
        $.vtPenalty = newPenaltyAmount;
    }
```

2.In the PufferOracleV2 contract, the privileged role can call the setMintPrice function to set the price of a validator ticket and there is no range limit when setting the price. If the permission is lost, it may cause the price of the validator's ticket to deviate from a reasonable range, thus making the proper functioning of the contract compromised.

Code Location:

PufferPool/src/PufferOracleV2.sol#L141-144

```
    function _setMintPrice(uint256 newPrice) internal {
        emit ValidatorTicketMintPriceUpdated(_validatorTicketPrice, newPrice);
        _validatorTicketPrice = newPrice;
    }
```

Solution

It is recommended to perform a cap check when setting the amount of vtPenalty and setting the price of the validator ticket.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404220003	SlowMist Security Team	2024.04.09 - 2024.04.22	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 vulnerabilities and 1 info. All findings were fixed and acknowledged. 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist