Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.09.11, the SlowMist security team received the Puffer Finance team's security audit application for Rewards Contract, 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

This is the RewardManager module for the puffer protocol.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Inaccurate error information	Others	Suggestion	Fixed
N2	Ignore return value	Others	Suggestion	Fixed

NO	Title	Category	Level	Status
N3	Risk of excessive privilege	Authority Control Vulnerability Audit	Low	Acknowledged
N4	Lack of complete parameter validation	Design Logic Audit	Low	Acknowledged
N5	Missing zero address check	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/PufferFinance/puffer-contracts/tree/feat/new-bridge

Initial audit version: 7c8762c294648085cf111bc964402abe678b2d56

Final audit version: 1e4d8ac4fd9805092b028926d8a128f0a6865aaa

Audit Scope:

```
- puffer-contracts/mainnet-contracts/src/L1RewardManager.sol
- puffer-contracts/l2-contracts/src/L2RewardManager.sol
```

The main network address of the contract is as follows:

L1RewardManager address:

0x157788cc028Ac6405bD406f2D1e0A8A22b3cf17b (Ethereum mainnet)

L2RewardManager address:

0xF9Dd335bF363b2E4ecFe3c94A86EBD7Dd3Dcf0e7 (Base mainnet)

4.2 Visibility Description

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

L1RewardManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setL2RewardClaimer	External	Can Modify State	-
mintAndBridgeRewards	External	Can Modify State	restricted
lzCompose	External	Can Modify State	restricted
setAllowedRewardMintAmount	External	Can Modify State	restricted
setAllowedRewardMintFrequency	External	Can Modify State	restricted
setDestinationEID	External	Can Modify State	restricted
getDestinationEID	External	View	-
_setAllowedRewardMintFrequency	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	restricted

L2RewardManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
claimRewards	External	Can Modify State	restricted
lzCompose	External	Can Modify State	restricted
freezeAndRevertInterval	External	Can Modify State	restricted
freezeClaimingForInterval	Public	Can Modify State	restricted
revertInterval	External	Can Modify State	restricted
setDelayPeriod	External	Can Modify State	restricted

L2RewardManager			
setPufETHOFT	External	Can Modify State	restricted
setDestinationEID	External	Can Modify State	restricted
getIntervalId	Public	-	-
isClaimingLocked	External	-	-
isClaimed	Public	-	-
getEpochRecord	External	-	-
getRewardsClaimer	Public	-	-
getClaimingDelay	External	-	-
getPufETHOFT	External	-	-
getDestinationEID	External	-	-
_handleMintAndBridge	Internal	Can Modify State	-
_handleSetClaimer	Internal	Can Modify State	-
_setClaimingDelay	Internal	Can Modify State	-
_isClaimingLocked	Internal	-	-
_freezeClaimingForInterval	Internal	Can Modify State	-
_revertInterval	Internal	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	restricted

4.3 Vulnerability Summary

[N1] [Suggestion] Inaccurate error information

Category: Others

Content

In the `claimRewards` function of the L2RewardManager contract, when a reward interval has been reverted

(`timeBridged = 0`), the `ClaimingLocked` error displays a misleading unlock time (`lockedUntil = claimingDelay`), which confuses two completely different states: "rewards temporarily locked awaiting unlock" and "rewards permanently reverted".

- l2-contracts/src/L2RewardManager.sol#L62-L129

```
    function claimRewards(ClaimOrder[] calldata claimOrders) external restricted {
        //...
            if (_isClaimingLocked(claimOrders[i].intervalId)) {
                revert ClaimingLocked({
                    intervalId: claimOrders[i].intervalId,
                    account: claimOrders[i].account,
                    lockedUntil: epochRecord.timeBridged + $.claimingDelay
                });
            }
        //...
    }
```

Solution

It is recommended to set lockedUntil of ClaimingLocked error to 0 when the reward has been reverted. This will avoid confusion with the time lock status and provide accurate contract status feedback.

Status

Fixed

[N2] [Suggestion] Ignore return value

Category: Others

Content

In the L2RewardManager contract, the `claimRewards` function ignores the return values of `IERC20(xPufETH).transfer` and `IPufETH($.pufETHOFT).transfer`.

- l2-contracts/src/L2RewardManager.sol#L62-L129

```
    function claimRewards(ClaimOrder[] calldata claimOrders) external restricted {
        //...
            if (xPufETHBalance > 0) {
                if (xPufETHBalance >= amountToTransfer) {
                    IERC20(xPufETH).transfer(recipient, amountToTransfer);
                } else {
```

```
                    IERC20(xPufETH).transfer(recipient, xPufETHBalance);
                    IPufETH($.pufETHOFT).transfer(recipient, amountToTransfer -
xPufETHBalance);
                }
            } else {
                IPufETH($.pufETHOFT).transfer(recipient, amountToTransfer);
            }
        //...
    }
```

Solution

It is recommended to use SafeERC20 for secure transfers.

Status

Fixed

[N3] [Low] Risk of excessive privilege

Category: Authority Control Vulnerability Audit

Content

1.In the L1RewardManager contract, the `ROLE_ID_DAO` role can modify important parameters of the contract (`allowedRewardMintAmount`, `allowedRewardMintFrequency`, `destinationEID`).

- mainnet-contracts/src/L1RewardManager.sol#L212-L218, L225-L227, L233-L237

```
    function setAllowedRewardMintAmount(uint104 newAmount) external restricted {}

    function setAllowedRewardMintFrequency(uint104 newFrequency) external restricted
{}

    function setDestinationEID(uint32 newDestinationEID) external restricted {}
```

2.In the L1RewardManager contract, the `ROLE_ID_OPERATIONS_PAYMASTER` role can mint pufETH and bridge it to the L2RewardManager contract on L2.

- mainnet-contracts/src/L1RewardManager.sol#L102-L157

```
    function mintAndBridgeRewards(MintAndBridgeParams calldata params) external
payable restricted {}
```

3.In the L2RewardManager contract, the `ROLE_ID_DAO` role can modify important parameters in the contract(`claimingDelay`, `pufETHOFT`, `destinationEID`), as well as freeze or rollback rewards for a specific `intervalId`.

- l2-contracts/src/L2RewardManager.sol#L214-L216, L222-L230, L236-L241

```
    function setDelayPeriod(uint256 delayPeriod) external restricted {}

    function setPufETHOFT(address newPufETHOFT) external restricted {}

    function setDestinationEID(uint32 newDestinationEID) external restricted {}
```

4.In the L2RewardManager contract, the `ROLE_ID_OPERATIONS_PAYMASTER` role can freeze or rollback rewards for a specific `intervalId`.

- l2-contracts/src/L2RewardManager.sol#L187-L191, L197-L199, L206-L208

```
    function freezeAndRevertInterval(uint256 startEpoch, uint256 endEpoch) external payable restricted {}

    function freezeClaimingForInterval(uint256 startEpoch, uint256 endEpoch) public restricted {}

    function revertInterval(uint256 startEpoch, uint256 endEpoch) external payable restricted {}
```

5.Both the L1RewardManager and L2RewardManager contracts use OpenZeppelin's upgradeable contract architecture, implementing upgrade functionality by inheriting UUPSUpgradeable and AccessManagedUpgradeable.

- mainnet-contracts/src/L1RewardManager.sol#L8, L258

```
    import { UUPSUpgradeable } from "@openzeppelin-contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";

    function _authorizeUpgrade(address newImplementation) internal virtual override restricted { }
```

- l2-contracts/src/L2RewardManager.sol#L10, L446

```
    import { UUPSUpgradeable } from "@openzeppelin/contracts-
upgradeable/proxy/utils/UUPSUpgradeable.sol";

    function _authorizeUpgrade(address newImplementation) internal virtual override
restricted { }
```

Solution

In the short term, to satisfy business requirements, managing the privileged role through a multi-signature scheme can effectively mitigate single-point risk. In the long term, entrusting these privileged roles to DAO governance can effectively resolve the risk of excessive privilege. During the transition period, managing through a multi-signature scheme combined with delayed transaction execution via a timelock can significantly alleviate the risk of excessive privilege.

Status

Acknowledged; According to the project's statement, the ROLE_ID_DAO role is managed via the Puffer Ops multisig. The contract upgrade mechanism utilizes an OpenZeppelin timelock contract, supporting two execution paths: the Puffer Ops multisig allows for upgrade queueing and execution within a one-week timelock, while the community multisig provides immediate queueing and execution. Furthermore, the ROLE_ID_OPERATIONS_PAYMASTER role is controlled by an off-chain reward calculation backend system, ensuring that the token minting process adheres strictly to the constraints of the reward distribution algorithm.

Puffer Ops multisig address:

0xC0896ab1A8cae8c2C1d27d011eb955Cca955580d (Ethereum mainnet)

0x37bEbCdB82E9428B89eEaB55288da70322079c46 (Base mainnet)

[N4] [Low] Lack of complete parameter validation

Category: Design Logic Audit

Content

In the L1RewardManager contract, the `mintAndBridgeRewards` function lacks complete validation of the params parameter.

1.The function lacks reproducibility verification of `params.startEpoch` and `params.endEpoch`. When an existing epoch range is used accidentally or maliciously, the same `intervalId` will be generated, directly overwriting the

existing `EpochRecord` on L2. This will cause the original `rewardRoot` to be replaced, making it impossible for users who have calculated the merkle proof to verify the legitimacy of their rewards.

2.The function lacks verification of `params.rewardsRoot` being non-empty. When bytes32(0) is passed as an empty value, users are completely unable to claim rewards because an empty `rewardRoot` cannot construct a valid merkle tree for proof verification. More seriously, this misconfiguration triggers a logical contradiction in the `_revertInterval` function in L2RewardManager. This function directly reverts when checking `epochRecord.rewardRoot == bytes32(0)`, making it impossible to fix this misconfiguration through the normal rollback mechanism. This will result in the corresponding pufETH funds being unable to be claimed by users or rolled back to L1, resulting in permanent loss of funds.

- mainnet-contracts/src/L1RewardManager.sol#L102-L157

```
    function mintAndBridgeRewards(MintAndBridgeParams calldata params) external
payable restricted {
    //...
    }
```

Solution

It is recommended to fully validate the params parameter in the mintAndBridgeRewards function of the L1RewardManager contract.

Status

Acknowledged

[N5] [Suggestion] Missing zero address check

Category: Others

Content

1.In the L1RewardManager contract, the `constructor` function lacks a zero address check for `pufETH_OFT`.

- mainnet-contracts/src/L1RewardManager.sol#L47-L55

```
    constructor(address pufETH, address l2RewardsManager, address pufETH_OFT) {
        if (pufETH == address(0) || l2RewardsManager == address(0)) {
            revert InvalidAddress();
        }
```

```
        PUFFER_VAULT = PufferVaultV5(payable(pufETH));
        PUFETH_OFT = IOFT(payable(pufETH_OFT));
        L2_REWARDS_MANAGER = l2RewardsManager;
        _disableInitializers();
    }
```

2.In the L2RewardManager contract, the `constructor` function lacks a zero address check for `xpufETH`.

- l2-contracts/src/L2RewardManager.sol#L44-L51

```
    constructor(address l1RewardManager, address xpufETH) {
        if (l1RewardManager == address(0)) {
            revert InvalidAddress();
        }
        L1_REWARD_MANAGER = l1RewardManager;
        xPufETH = xpufETH;
        _disableInitializers();
    }
```

Solution

It is recommended to add zero address check.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002509150003	SlowMist Security Team	2025.09.11 - 2025.09.15	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 low risk, 3 suggestion.

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