Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.06.27, the SlowMist security team received the Puffer Finance team's security audit application for Puffer L2 Staking, 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

Audit Version

https://github.com/PufferFinance/puffer-contracts

commit: d3e318f3c45d744bfa2dbadfa1abe998fa49d4b5

Fixed Version

https://github.com/PufferFinance/puffer-contracts

commit: 5334d007eb1da7ce377600ab9104db25ebbf51d3

Audit scope:

- mainnet-contracts/src/PufLocker.sol

- mainnet-contracts/src/PufLockerStorage.sol
- mainnet-contracts/src/PufToken.sol
- mainnet-contracts/src/PufferL2Depositor.sol
- mainnet-contracts/src/interface/IPufLocker.sol
- mainnet-contracts/src/interface/IPufStakingPool.sol
- mainnet-contracts/src/interface/IPufferL2Depositor.sol

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Token compatibility issues	Design Logic Audit	Low	Acknowledged
N2	Gas Optimization of the permit operation	Gas Optimization Audit	Suggestion	Fixed
N3	Missing chainID check in the signature verification	Replay Vulnerability	Low	Fixed

4 Code Overview

4.1 Contracts Description

This audit is about three staking-related contracts of puffer finance, namely PufLocker, PufToken and PufferL2Depositor. In the PufferL2Depositor contract, the DAO will create a wrapper token pufToken corresponding to the staked token, and then you can transfer the token into the PufToken contract by calling the deposit function to complete the staking operation. The PufLocker contract is mainly about the token locking mechanism. Users lock the token through the deposit function and can only withdraw tokens after the locking time is over.

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:

PufferL2Depositor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManaged
deposit	External	Can Modify State	onlySupportedTokens restricted
depositETH	External	Payable	restricted
addNewToken	External	Can Modify State	restricted
setMigrator	External	Can Modify State	restricted
setDepositCap	External	Can Modify State	onlySupportedTokens restricted
revertIfPaused	External	Can Modify State	restricted
_deposit	Internal	Can Modify State	-
_addNewToken	Internal	Can Modify State	-

PufToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit
deposit	External	Can Modify State	whenNotPaused
withdraw	External	Can Modify State	validateAddressAndAmount
migrate	External	Can Modify State	onlyAllowedMigratorContract whenNotPaused

PufToken			
migrateWithSignature	External	Can Modify State	onlyAllowedMigratorContract whenNotPaused
setDepositCap	External	Can Modify State	onlyPufferFactory
_deposit	Internal	Can Modify State	validateAddressAndAmount
_migrate	Internal	Can Modify State	validateAddressAndAmount
decimals	Public	-	-

PufLocker			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
deposit	External	Can Modify State	isAllowedToken restricted
withdraw	External	Can Modify State	-
setIsAllowedToken	External	Can Modify State	restricted
setLockPeriods	External	Can Modify State	restricted
getDeposits	External	-	-
getAllDeposits	External	-	-
getLockPeriods	External	-	-

4.3 Vulnerability Summary

[N1] [Low] Token compatibility issues

Category: Design Logic Audit

Content

When calling the deposit function to transfer tokens to the contract, the deposit amount is not calculated based

on the difference in the contract balance before and after the transfer. Instead, the amount parameter passed in is directly used. Therefore, if the transferred tokens are reflective tokens(deflation/inflation type tokens), the actual deposit amount will not be equal to the number of tokens transferred into the contract, resulting in a calculation error.

Code Location:

mainnet-contracts/src/PufferL2Depositor.sol#L47-72

```
    function deposit(address token, address account, Permit calldata permitData,
uint256 referralCode)
        external
        onlySupportedTokens(token)
        restricted
    {
        ...

        IERC20(token).safeTransferFrom(msg.sender, address(this), permitData.amount);

        _deposit({
            token: token,
            depositor: msg.sender,
            account: account,
            amount: permitData.amount,
            referralCode: referralCode
        });
    }
```

mainnet-contracts/src/PufToken.sol#L188-207

```
    function _deposit(address depositor, address account, uint256 amount) internal {
        TOKEN.safeTransferFrom(msg.sender, address(this), amount);

        uint256 normalizedAmount = _normalizeAmount(amount);

        if (totalSupply() + normalizedAmount > totalDepositCap) {
            revert TotalDepositCapReached();
        }

        // Mint puffToken to the account
        _mint(account, normalizedAmount);
```

```
        ...
    }
```

mainnet-contracts/src/PufLocker.sol#L45-77

```
    function deposit(address token, uint128 lockPeriod, Permit calldata permitData)
        external
        isAllowedToken(token)
        restricted
    {
        ...

        IERC20(token).safeTransferFrom(msg.sender, address(this), permitData.amount);

        uint128 releaseTime = uint128(block.timestamp) + lockPeriod;

        $.deposits[msg.sender][token].push(Deposit(uint128(permitData.amount),
releaseTime));

        emit Deposited(msg.sender, token, uint128(permitData.amount), releaseTime);
    }
```

Solution

It is recommended to use the difference in the token balance in the contract before and after the user's transfer as the actual deposit amount of the user, instead of directly using the amount parameter passed in.

Status

Acknowledged; The project team responded: They will not support any deflation/inflation type tokens. So using the amount should be okay and more gas efficient.

[N2] [Suggestion] Gas Optimization of the permit operation

Category: Gas Optimization Audit

Content

In the PufferL2Depositor and PufLocker contract, When calling the deposit function, the ERC20Permit(token).permit function is first called to perform the token authorization operation. However, even if the deposited token (e.g. weth) does not support the permit function, the ERC20Permit(token).permit function is still called for authorization, which leads to additional gas consumption.

Code Location:

mainnet-contracts/src/PufferL2Depositor.sol#L53-61

```
    function deposit(address token, address account, Permit calldata permitData,
uint256 referralCode)
        external
        onlySupportedTokens(token)
        restricted
    {

        try ERC20Permit(token).permit({
            owner: msg.sender,
            spender: address(this),
            value: permitData.amount,
            deadline: permitData.deadline,
            v: permitData.v,
            s: permitData.s,
            r: permitData.r
        }) { } catch { }

        ...
    }
```

mainnet-contracts/src/PufLocker.sol#L60-68

```
    function deposit(address token, uint128 lockPeriod, Permit calldata permitData)
        external
        isAllowedToken(token)
        restricted
    {
       ...

        try ERC20Permit(token).permit({
            owner: msg.sender,
            spender: address(this),
            value: permitData.amount,
            deadline: permitData.deadline,
            v: permitData.v,
            s: permitData.s,
            r: permitData.r
        }) { } catch { }

        ...
    }
```

Solution

It is recommended to skip the approval operation for tokens that do not support the permit function, but to perform proper checks on the allowance amount.

Status

Fixed; The project team responded: They will use the front-end to check whether the permit data can be passed and the on-chain contract to check whether the permit data signature exists to solve this optimization.

[N3] [Low] Missing chainID check in the signature verification

Category: Replay Vulnerability

Content

In the PufToken contract, the users can call the migrateWithSignature function to perform token migration operations through signed transactions. However, when hashing the data that needs to be checked for the signature, the chainId is not included in the hash. This means that if the project is deployed on multiple chains, the signature may be maliciously replayed by an attacker on another chain, potentially resulting in the user's funds being compromised.

Code Location:

mainnet-contracts/src/PufToken.sol#L143-154

```
    function migrateWithSignature(
        address depositor,
        address migratorContract,
        address destination,
        uint256 amount,
        uint256 signatureExpiry,
        bytes memory stakerSignature
    ) external onlyAllowedMigratorContract(migratorContract) whenNotPaused {
        ...

        bytes32 structHash = keccak256(
            abi.encode(
                _MIGRATE_TYPEHASH,
                depositor,
                migratorContract,
                destination,
                address(TOKEN),
                amount,
```

```
                signatureExpiry,
                _useNonce(depositor)
            )
        );

        if (!SignatureChecker.isValidSignatureNow(depositor,
_hashTypedDataV4(structHash), stakerSignature)) {
            revert InvalidSignature();
        }

        ...
    }
```

Solution

It's recommended to add the chainId in the hash calculation of the signature message.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406280004	SlowMist Security Team	2024.06.27 - 2024.06.28	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 2 low risk and 1 suggestion vulnerabilities. All the 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