Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.09, the SlowMist security team received the Bitlayer team's security audit application for peg-Token, 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

The project includes token contract and TokenManager contract. The TokenManager contract is an entry contract used to manage the creation of the PegToken contract and the setting of other parameters.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/bitlayer-org/peg-tokens-contract

commit: 645e4ba161072dadc5492f51e18fa6040add91ce

(Focus on code changes from **version**: f5e9c4dd5a1b77cbd00404ba622c6070dabe194d to version:

645e4ba161072dadc5492f51e18fa6040add91ce)

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:

PegToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyManager
decimals	Public	-	-
setBlacklist	External	Can Modify State	onlyManager
setMinter	External	Can Modify State	onlyManager
pause	External	Can Modify State	onlyManager whenNotPaused
unpause	External	Can Modify State	onlyManager whenPaused
mint	External	Can Modify	onlyMinter notBlacklisted

PegToken			
		State	
recall	External	Can Modify State	onlyManager
freeze	External	Can Modify State	onlyManager
unfreeze	External	Can Modify State	onlyManager
transfer	Public	Can Modify State	notBlacklisted notBlacklisted whenNotPaused worldNotStopped
transferFrom	Public	Can Modify State	notBlacklisted notBlacklisted whenNotPaused worldNotStopped
burn	External	Can Modify State	notBlacklisted whenNotPaused worldNotStopped

TokenManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyRole
setStopTheWorld	External	Can Modify State	onlyRole
setBlackList	External	Can Modify State	onlyRole
setMinter	External	Can Modify State	onlyRole
getTokenAddress	Public	-	-
createToken	External	Can Modify State	onlyRole
upgradeToken	External	Can Modify State	onlyRole
pauseToken	External	Can Modify State	onlyRole
unpauseToken	External	Can Modify State	onlyRole
freezeToken	External	Can Modify State	onlyRole

TokenManager			
unfreezeToken	External	Can Modify State	onlyRole
recall	External	Can Modify State	onlyRole
getDeployedTokenOrRevert	Internal	-	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.The PegToken contracts and the TokenManager contract are implemented using the OpenZeppelin upgradeable model, allowing the `AdminRole` role to perform contract upgrades. In the TokenManager contract, the `Operator` role can upgrade the specified PegToken contract through the `upgradeToken` function.However, this design introduces an excessive privilege risk.

- TokenManager.sol#L123-L131

```
    function upgradeToken(string memory symbol, address newImpl, bytes memory
callData)
        external
        onlyRole(Operator)
    {
        require(newImpl != address(0), "invalid new impl");

        PegToken peg = getDeployedTokenOrRevert(symbol);
        peg.upgradeToAndCall(newImpl, callData);
    }
```

2.In the TokenManager contract, the `Operator` role can call the `setBlacklist` function of the specified PegToken contract through the `setBlackList` function to add a blacklist address.

- TokenManager.sol#L79-L87

```
    function setBlackList(string memory symbol, address account, bool toBlacklist)
        external
```

```
        onlyRole(Operator)
    {
        require(account != address(0), "invalid account");

        PegToken peg = getDeployedTokenOrRevert(symbol);
        peg.setBlacklist(account, toBlacklist);
    }
```

- PegToken.sol#L86-L92

```
    function setBlacklist(address account, bool toBlacklist)
        external
        onlyManager
    {
        isBlacklist[account] = toBlacklist;
        emit BlacklistAdded(account, toBlacklist);
    }
```

3.In the TokenManager contract, the `Operator` role can add the `Minter` role by calling the `setMinter` function of the specified PegToken contract through the `setMinter` function.

- TokenManager.sol#L89-L97

```
    function setMinter(string memory symbol, address account, bool asMinter)
        external
        onlyRole(Operator)
    {
        require(account != address(0), "invalid account");

        PegToken peg = getDeployedTokenOrRevert(symbol);
        peg.setMinter(account, asMinter);
    }
```

- PegToken.sol#L94-L100

```
    function setMinter(address account, bool asMinter)
        external
        onlyManager
    {
        minters[account] = asMinter;
        emit MinterSet(account, asMinter);
    }
```

4.In the PegToken contract, the `Minter` role can mint any number of tokens by calling the `mint` function through the `mint` function.

- PegToken.sol#L112-L118

```
    function mint(address to, uint256 amount)
        external
        onlyMinter
        notBlacklisted(to)
    {
        _mint(to, amount);
    }
```

5.In the TokenManager contract, the `FreezeRole` role can perform transfer operations by calling the `recall` function of the specified PegToken contract through the `recall` function.

- TokenManager.sol#L159-L165

```
    function recall(string memory symbol, address from, address to, uint256 value)
        external
        onlyRole(FreezeRole)
    {
        PegToken peg = getDeployedTokenOrRevert(symbol);
        peg.recall(from, to, value);
    }
```

- PegToken.sol#L120-L126

```
    function recall(address from, address to, uint256 value)
        external
        onlyManager
    {
        _transfer(from, to, value);
        emit TokenRecalled(from, to, value);
    }
```

6.In the TokenManager contract, the `FreezeRole` role can freeze the specified token at the specified address by calling the `freeze` function of the specified PegToken contract through the `freezeToken` function.

- TokenManager.sol#L143-L149

```
function freezeToken(string memory symbol, address account, uint256 value)
    external
    onlyRole(FreezeRole)
{
    PegToken peg = getDeployedTokenOrRevert(symbol);
    peg.freeze(account, value);
}
```

- PegToken.sol#L128-L136

```
function freeze(address account, uint256 value)
    external
    onlyManager
{
    _transfer(account, address(this), value);
    freezedToken[account] += value;

    emit TokenFreezed(account, value);
}
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404110001	SlowMist Security Team	2024.04.09 - 2024.04.11	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 1 medium risk. All the findings were 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