Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.05.12, the SlowMist security team received the SwellNetwork team's security audit application for Swell Balancer Vault, 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.

Level	Description
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

Serial Number	Audit Class	Audit Subclass
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
		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/SwellNetwork/vault/blob/main/contracts/implementations/SwellBalancerVault.sol

commit: db212449126d46da68c8b49d03980382b6d86b36

Fixed version:

https://github.com/SwellNetwork/vault/blob/main/contracts/implementations/SwellBalancerVault.sol

commit: ede6bcc4954f64a1734088b3d039f01b30064023

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Malleable attack risk	Replay Vulnerability	Suggestion	Fixed
N2	Potential Compatibility Risk	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

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:

SwellBalancerVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC4626
totalAssets	Public	-	-
beforeWithdraw	Internal	Can Modify State	-
afterDeposit	Internal	Can Modify State	-

ERC4626			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
deposit	Public	Can Modify State	-
mint	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
redeem	Public	Can Modify State	-
totalAssets	Public	-	-
convertToShares	Public	-	-
convertToAssets	Public	-	-
previewDeposit	Public	-	-
previewMint	Public	-	-
previewWithdraw	Public	-	-

ERC4626

previewRedeem	Public	-	-
maxDeposit	Public	-	-
maxMint	Public	-	-
maxWithdraw	Public	-	-
maxRedeem	Public	-	-
beforeWithdraw	Internal	Can Modify State	-
afterDeposit	Internal	Can Modify State	-

ERC20

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
approve	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
permit	Public	Can Modify State	-
DOMAIN_SEPARATOR	Public	-	-
computeDomainSeparator	Internal	-	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-

WeightedMath

WeightedMath			
Function Name	Visibility	Mutability	Modifiers
_calculateInvariant	Internal	-	-
_calcOutGivenIn	Internal	-	-
_calcInGivenOut	Internal	-	-
_calcBptOutGivenExactTokensIn	Internal	-	-
_calcTokenInGivenExactBptOut	Internal	-	-
_calcAllTokensInGivenExactBptOut	Internal	-	-
_calcBptInGivenExactTokensOut	Internal	-	-
_calcTokenOutGivenExactBptIn	Internal	-	-
_calcTokensOutGivenExactBptIn	Internal	-	-
_calcDueTokenProtocolSwapFeeAmount	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Malleable attack risk

Category: Replay Vulnerability

Content

In the ERC20 contract, the permit function restores the address of the signer through the ecrecover function, but does not check the value of v and s. Since EIP2 still allows the malleability for ecrecover, this will lead to the risk of transaction malleability attacks.

Code location:

contracts/vendor/rari-capital/ERC20.sol#129-176

```
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual {
        require(deadline >= block.timestamp, "PERMIT_DEADLINE_EXPIRED");

        // Unchecked because the only math done is incrementing
        // the owner's nonce which cannot realistically overflow.
        unchecked {
            address recoveredAddress = ecrecover(
                keccak256(
                    abi.encodePacked(
                        "\x19\x01",
                        DOMAIN_SEPARATOR(),
                        keccak256(
                            abi.encode(
                                keccak256(
                                    "Permit(address owner,address spender,uint256
value,uint256 nonce,uint256 deadline)"
                                ),
                                owner,
                                spender,
                                value,
                                nonces[owner]++,
                                deadline
                            )
                        )
                    )
                ),
                v,
                r,
                s
            );

            require(
                recoveredAddress != address(0) && recoveredAddress == owner,
                "INVALID_SIGNER"
            );
```

```
            allowance[recoveredAddress][spender] = value;
        }

        emit Approval(owner, spender, value);
    }
```

Solution

It is recommended to use the ECDSA library of openzeppelin to check the signature.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol

Status

Fixed

[N2] [Suggestion] Potential Compatibility Risk

Category: Others

Content

In the ERC4624 contract, users can call the deposit tokens through the deposit function. If the deposited tokens are deflationary tokens, the actual number of tokens received by the contract is inconsistent with the number recorded by tokenAmounts.

Code location:

contracts/vendor/rari-capital/ERC4626.sol#L51-67

```
    function deposit(uint256 assets, address receiver)
        public
        virtual
        returns (uint256 shares)
    {
        // Check for rounding error since we round down in previewDeposit.
        require((shares = previewDeposit(assets)) != 0, "ZERO_SHARES");

        // Need to transfer before minting or ERC777s could reenter.
        asset.safeTransferFrom(msg.sender, address(this), assets);

        _mint(receiver, shares);
```

```
        emit Deposit(msg.sender, receiver, assets, shares);

        afterDeposit(assets, shares);
    }

    function afterDeposit(uint256 assets, uint256) internal override {
        (IERC20[] memory tokensFromPool, , ) = balancerVault.getPoolTokens(
            poolId
        );

        IAsset[] memory tokens = new IAsset[](2);
        uint256[] memory tokenAmounts = new uint256[](2);

        /* find the index of the asset token in the tokens from pool array and
assign the deposit amount to that index in the tokens amount array*/
        for (uint256 i = 0; i < tokensFromPool.length; i++) {
            tokens[i] = IAsset(address(tokensFromPool[i]));
            if (address(asset) == address(tokensFromPool[i])) {
                tokenAmounts[i] = assets;
            }
        }

        balancerVault.joinPool(
            poolId,
            address(this),
            address(this),
            JoinPoolRequest(
                tokens,
                tokenAmounts,
                abi.encode(JoinKind.EXACT_TOKENS_IN_FOR_BPT_OUT, tokenAmounts),
                false
            )
        );
    }
```

Solution

It is recommended to use the balance difference of the contract before and after the user transfer as the amount transferred by the user. And pay attention to token compatibility issues when docking with strategies.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002205190003	SlowMist Security Team	2022.05.12 - 2022.05.19	Passed

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 suggestion vulnerabilities. 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