SLOWMIST

Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.19, the SlowMist security team received the team's security audit application for Particle Vault WrapMintV2, 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/Particle-Platforms/vault-contract

commit: 9214216ae8d378d18448de6427d423943e3bcca8

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	High	Fixed
N2	Receive can lock users' native tokens	Others	Low	Acknowledged
N3	Unable to get back principal after burning tokens by mistake	Design Logic Audit	Suggestion	Acknowledged
N4	Missing the 0 address check	Others	Suggestion	Acknowledged
N5	Risk of replay attack	Replay Vulnerability	Information	Acknowledged
N6	Malleable attack risk	Replay Vulnerability	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Particle Vault WrapMintV2 is an iterative audit of Particle Protocol. In WrapMintV2, the calculation method of fixedRate is modified, so that fixed users lock the rate lower limit instead of the current rate. The more fixedRate the user purchases, the lower the locked rate should be. At the same time, DuoAssetToken is added to allow users to earn yield while owning corresponding tradable tokens.

The main network address of the contract is as follows:

Particle Protocol (BLAST)		
Contract Name	Contract Address (WETH)	Contract Address (USDB)
Contract	WETH 90D	USDB 90D
Vault Core	0x16A0bc7336072e7891a06830acf969e315341645	0x19ed350cfFB9cc1AfaD808b4f81991f42B7eC28B
Yield Manager Core	0x0a14658c3EB6A5Aa16e1d51a99c434C5aEa5B371	0x04f04e28bf8B44495604FE68046C57e435b5C22B

Particle Protocol (BLAST)		
Wrap Mint Core	0x18140C0a3D0D0153E131b78eC6c3EcFcFEBaD080	0xC16f16d6033EAa96d2B8440C2Fc908dA868914D0
Duo Asset Token Core	0x4230da5d79D7503a080684564BA30dD9f0F74020	0xe96b3e15384EA4D41f7eBFdDD0001C6458094492
Fixed Rate Implementation	0x558cE86d3c721F3C2D05905b1BCb1c6606aAFEBf	0x1cEaE01e9Bf34e48004753d2350D0b065696Dca9
Variable Rate Implementation	0x5b3C0725efa8278b1B5536c6708a7385173BEbD0	0x481aec4c0DB0f8e3FD24d99e63Bf4631b640602e
Vault	0x504f7ab83F5fD8B3842323dc62A68421DFC57Eff	0x74BE79460A2499813570dd9dCf3A5005e72Ab93e
Yield Manager	0xabF868bbe80550DC2EE4F8dFfcb06A5eb9B0760E	0xBc65c01680E09FE14F466B00Cc2A9248a022F101
Wrap Mint V2	0x5ea5EdF0F1B22C527C3c42833cA671678bD819ED	0xf711e8587992464Aa9ebc96002aCc0C817BB0303
Duo Asset Token	0x1Da40C742F32bBEe81694051c0eE07485fC630f6	0x1A3D9B2fa5c6522c8c071dC07125cE55dF90b253

4.2 Visibility Description

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

DuoAssetToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 CoreRef
mint	External	Can Modify State	onlyMinter
burn	Public	Can Modify State	-
burnFrom	Public	Can Modify State	onlyBurner
permit	External	Can Modify State	-

WrapMintV2			
Function Name	Visibility	Mutability	Modifiers

WrapMintV2			
<Constructor>	Public	Can Modify State	CoreRef
_swap	Internal	Can Modify State	-
mintFixedRate	External	Can Modify State	nonReentrant
mintFixedRateEth	External	Payable	nonReentrant
_mintFixedRate	Internal	Can Modify State	-
burnFixedRate	External	Can Modify State	nonReentrant
withdrawFixedRate	External	Can Modify State	nonReentrant
mintVariableRate	External	Can Modify State	nonReentrant
mintVariableRateEth	External	Payable	nonReentrant
_mintVariableRate	Internal	Can Modify State	-
burnVariableRate	External	Can Modify State	nonReentrant
withdrawVariableRate	External	Can Modify State	nonReentrant
transferInFixedRateNft	External	Can Modify State	nonReentrant
transferOutFixedRateNft	External	Can Modify State	nonReentrant
transferInVariableRateNft	External	Can Modify State	nonReentrant
transferOutVariableRateNft	External	Can Modify State	nonReentrant
addExchange	External	Can Modify State	onlyGovernor
removeExchange	External	Can Modify State	onlyGovernor
setFixedRateNft	External	Can Modify State	onlyGovernor
setVariableRateNft	External	Can Modify State	onlyGovernor
setDuoAssetToken	External	Can Modify State	onlyGovernor
onERC721Received	External	-	-

WrapMintV2			
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [High] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the Permissions contract, the Governor role can grant or revoke the Minter and Burner roles. The Minter and Burner roles can mint or burn users' tokens through the mint and burnFrom functions in the DuoAssetToken.

Code location:

periphery/DuoAssetToken.sol#39-42,58-61

core/Permissions.sol#44-52, 68-76

```
    function mint(address account, uint256 amount) external override onlyMinter {
        _mint(account, amount);
        emit Minting(account, msg.sender, amount);
    }

    function burnFrom(address account, uint256 amount) public
override(IDuoAssetToken, ERC20Burnable) onlyBurner {
        _burn(account, amount);
        emit Burning(account, msg.sender, amount);
    }
```

2.In the WrapMintV2 contract, the owner role can add or remove the whitelistedExchanges, modify the fixedRateNft, the variableRateNft and the duoAssetToken address.

Code location:

periphery/WrapMintV2.sol#533-556

```
    function addExchange(address exchange) external onlyGovernor {
        whitelistedExchanges[exchange] = true;
        emit UpdateExchange(exchange, true);
    }
    function removeExchange(address exchange) external onlyGovernor {
        whitelistedExchanges[exchange] = false;
        emit UpdateExchange(exchange, false);
    }

    function setFixedRateNft(address nft) external onlyGovernor {
        fixedRateNft = nft;
        emit UpdateFixedRateNft(nft);
    }

    function setVariableRateNft(address nft) external onlyGovernor {
        variableRateNft = nft;
        emit UpdateVariableRateNft(nft);
    }

    function setDuoAssetToken(address token) external onlyGovernor {
        duoAssetToken = token;
        emit UpdateDuoAssetToken(token);
    }
```

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

Fixed; After communicating with the project team, they expressed that they granted the governor to be Ring DAO together with Timelock. Reference: https://www.tally.xyz/gov/ring/proposal/5

The Governor's role transferred to a 2-day Timelock contract and a multi-sig contract owned by 2 EOA addresses.

Timelock: 0x337c4F3054f091D0E2239ce09c0b112D874dEBf7

GnosisSafeProxy: 0x7c8b9E2De6FfA465c6f717f349B3Ab13AB46481d

[N2] [Low] Receive can lock users’ native tokens

Category: Others

Content

There is a receive function in the WrapMintV2 contract so that the contracts can receive native tokens from the WETH contract. However, the receive function can lock users' native tokens when users transfer the native token in these contracts by mistake.

Code location:

periphery/WrapMintV2.sol#566

```
    receive() external payable {}
```

Solution

It's recommended to add addresses that can send the native tokens in the contract in the require check logic in the receive() function.

Status

Acknowledged

[N3] [Suggestion] Unable to get back principal after burning tokens by mistake

Category: Design Logic Audit

Content

In the WrapMintV2, users can deposit their assets to choose the two strategies for yield and the contract will mint them the same amount of the principal and locked yield of the duoAssetToken. If they want to get their principal back and gain the yield, they need to burn the duoAssetToken. But the DuoAssetToken can let the users burn their tokens through the burn function. If the users burn their DuoAssetToken by mistake, they can not withdraw the same principal as they deposit, which can lead to the principal being lost.

Code location:

periphery/WrapMintV2.sol#244, 258, 399, 424, 471, 520

periphery/DuoAssetToken.sol#48-51

```
    DuoAssetToken(duoAssetToken).burnFrom(msg.sender, amount);

    function burn(uint256 amount) public override(IDuoAssetToken, ERC20Burnable) {
        super.burn(amount);
        emit Burning(msg.sender, msg.sender, amount);
    }
```

Solution

It’s recommended to remove the public burn function for users. Or just let the burn function in the refund function.

Status

Acknowledged

[N4] [Suggestion] Missing the 0 address check

Category: Others

Content

All the addresses are missing the 0 address check when the Governor role arbitrarily modifies the address.

Code location:

periphery/WrapMintV2.sol#533-556

```
    function addExchange(address exchange) external onlyGovernor {
        whitelistedExchanges[exchange] = true;
        emit UpdateExchange(exchange, true);
    }

    function removeExchange(address exchange) external onlyGovernor {
        whitelistedExchanges[exchange] = false;
        emit UpdateExchange(exchange, false);
    }

    function setFixedRateNft(address nft) external onlyGovernor {
        fixedRateNft = nft;
        emit UpdateFixedRateNft(nft);
    }

    function setVariableRateNft(address nft) external onlyGovernor {
        variableRateNft = nft;
        emit UpdateVariableRateNft(nft);
    }

    function setDuoAssetToken(address token) external onlyGovernor {
        duoAssetToken = token;
        emit UpdateDuoAssetToken(token);
    }
```

Solution

It is recommended to add the 0 address check.

Status

Acknowledged

[N5] [Information] Risk of replay attack

Category: Replay Vulnerability

Content

DOMAIN_SEPARATOR is defined when the contract is initialized, but it is not reimplemented when DOMAIN_SEPARATOR is used in the permit function. So the DOMAIN_SEPARATOR contains the chainId and is defined at contract deployment instead of reconstructed for every signature, there is a risk of possible replay attacks between chains in the event of a future chain split.

Code location:

periphery/DuoAssetToken.sol#17-32

```
    constructor(string memory name_, string memory symbol_, address core_)
ERC20(name_, symbol_) CoreRef(core_) {
        uint256 chainId;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            chainId := chainid()
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)"),
                keccak256(bytes(name())),
                keccak256(bytes("1")),
                chainId,
                address(this)
            )
        );
    }
```

Solution

It is recommended to redefine when using DOMAIN_SEPARATOR.

Status

Acknowledged; After communicating with the project team, they expressed that the they will not deploy the same contract multiple chains,

[N6] [Information] Malleable attack risk

Category: Replay Vulnerability

Content

In the permit function of the DuoAssetToken contract, it 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:

periphery/DuoAssetToken.sol#73-93

```
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external override {
        require(deadline >= block.timestamp, "Duo: EXPIRED");
        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, "Duo:
INVALID_SIGNATURE");
        _approve(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

Acknowledged; After communicating with the project team, they expressed that the they will not deploy the same contract multiple chains,

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404220001	SlowMist Security Team	2024.04.19 - 2024.04.22	Low Risk

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 high risk, 1 low risk, 2 suggestions, and 2 information.

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