Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.09.13, the SlowMist security team received the team's security audit application for Unemeta, 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:

contracts.zip

6bce4bc6f4b7f06ba3683f073493990e9312d75a53c6425db8d372b88ebfdb6b

Fixed version:

contracts9_17.zip

f21ec3ffdcaf84b70e64c8d3cd9a1ed03ced98f87bc09195f91248b31c54bfcb

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Business logic issue	Design Logic Audit	Medium	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability	Low	Fixed
N3	Risk of replay attack	Replay Vulnerability	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:

UnemetaMarket			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
matchSellerOrdersWETH	External	Payable	nonReentrant
matchSellerOrders	External	Can Modify State	nonReentrant
matchesBuyerOrder	External	Can Modify State	nonReentrant
cancelAllOrdersForSender	External	Can Modify State	-
cancelMultipleMakerOrders	External	Can Modify State	-
isUserOrderNonceExecutedOrCancelled	External	-	-
_transferFeesAndFunds	Internal	Can Modify State	-
_transferFeesAndFundsWithWETH	Internal	Can Modify State	-
_transferNonFungibleToken	Internal	Can Modify State	-
_calculateProtocolFee	Internal	-	-
_validateOrder	Internal	-	-
updateCurrencyManager	External	Can Modify State	onlyOwner
updateExecutionManager	External	Can Modify State	onlyOwner
updateProtocolFeeRecipient	External	Can Modify State	onlyOwner
updateRoyaltyFeeManager	External	Can Modify State	onlyOwner
updateTransferSelectorNFT	External	Can Modify State	onlyOwner

OrderModel			
Function Name	Visibility	Mutability	Modifiers
hash	Internal	-	-

CurrencyManager			
Function Name	Visibility	Mutability	Modifiers
addCurrency	External	Can Modify State	onlyOwner
removeCurrency	External	Can Modify State	onlyOwner
isCurrencyWhitelisted	External	-	-
viewCountWhitelistedCurrencies	External	-	-
viewWhitelistedCurrencies	External	-	-

ExecutionManager			
Function Name	Visibility	Mutability	Modifiers
addStrategy	External	Can Modify State	onlyOwner
removeStrategy	External	Can Modify State	onlyOwner
isStrategyWhitelisted	External	-	-
viewCountWhitelistedStrategies	External	-	-
viewWhitelistedStrategies	External	-	-

RoyaltyFeeManager			
Function Name	Visibility	Mutability	Modifiers

RoyaltyFeeManager			
<Constructor>	Public	Can Modify State	-
calculateRoyaltyFeeAndGetRecipient	External	-	-

TransferSelectorNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addCollectionTransferManager	External	Can Modify State	onlyOwner
removeCollectionTransferManager	External	Can Modify State	onlyOwner
checkTransferManagerForToken	External	-	-

StrategyFixedPrice			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
canExecuteBuy	External	-	-
canExecuteSell	External	-	-
viewProtocolFee	External	-	-

RoyaltyFeeRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateRoyaltyFeeLimit	External	Can Modify State	onlyOwner

RoyaltyFeeRegistry			
updateRoyaltyInfoForCollection	External	Can Modify State	onlyOwner
royaltyInfo	External	-	-
royaltyFeeInfoCollection	External	-	-

RoyaltyFeeSetter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateRoyaltyInfoForCollectionIfAdmin	External	Can Modify State	-
updateRoyaltyInfoForCollectionIfOwner	External	Can Modify State	-
updateRoyaltyInfoForCollectionIfSetter	External	Can Modify State	-
updateRoyaltyInfoForCollection	External	Can Modify State	onlyOwner
updateOwnerOfRoyaltyFeeRegistry	External	Can Modify State	onlyOwner
updateRoyaltyFeeLimit	External	Can Modify State	onlyOwner
checkForCollectionSetter	External	-	-
_updateRoyaltyInfoForCollectionIfOwnerOrAdmin	Internal	Can Modify State	-

TransferManagerERC721			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

TransferManagerERC721			
transferNonFungibleToken	External	Can Modify State	-

TransferManagerERC1155			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
transferNonFungibleToken	External	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Medium] Business logic issue

Category: Design Logic Audit

Content

1.In the UnemetaMarket contract, the matchSellerOrdersWETH and matchSellerOrders use the strategy to match orders, but the canExecuteTakerBid function is not in the StrategyFixedPrice contract.

Code location:

Manager/UnemetaMarket.sol#161-162, 223-224

```
        (bool isExecutionValid, uint256 tokenId, uint256 amount) =
TheExStrategy(makerAsk.strategy)
        .canExecuteTakerBid(takerBid, makerAsk);

        (bool isExecutionValid, uint256 tokenId, uint256 amount) =
TheExStrategy(makerAsk.strategy)
        .canExecuteTakerBid(takerBid, makerAsk);
```

2.In the UnemetaMarket contract, the updateCurrencyManager function will update the currencyManager, but the ICurrencyManager interface can not be found in this contract or import parts.

Code location:

```
Manager/UnemetaMarket.sol#9,585
    import {Currency} from "./interface/Currency.sol";

    function updateCurrencyManager(address _currencyManager) external onlyOwner {
        require(_currencyManager != address(0), "Cannot update to a null address");
        currencyManager = ICurrencyManager(_currencyManager);
        emit NewCurrencyManager(_currencyManager);
    }
```

3.In the TransferManagerERC721 and TransferManagerERC1155 contracts, the transferNonFungibleToken has been overridden but this function can not be found in the inherent contract TheTransferSelector.

Code location:

Manager/trans/TransferManagerERC721.sol#6,12,33

Manager/trans/TransferManagerERC1155.sol#6,9,27

```
import {TheTransferSelector} from "./interface/TheTransFerSelector.sol";

contract TransferManagerERC721 is TheTransferSelector {
    function transferNonFungibleToken(
        address collection,
        address from,
        address to,
        uint256 tokenId,
        uint256
    ) external override {
        require(msg.sender == UNEMETA_MARKET, "Only Unemeta Market can call this
function");
        // https://docs.openzeppelin.com/contracts/2.x/api/token/erc721#IERC721-
safeTransferFrom
        IERC721(collection).safeTransferFrom(from, to, tokenId);
    }
}
```

4.In the UnemetaMarket contract, the OrderModel contract is using for the OrderModel.MakerOrder and OrderModel.TakerOrder type as struct, but the type of the OrderModel should be a library.

Code location:

Manager/UnemetaMarket.sol#25,26

```
    using OrderModel for OrderModel.MakerOrder;
    using OrderModel for OrderModel.TakerOrder;
```

Solution

It is recommended to clarify the logic implementation and make sure the contract can be compiled and run normally.

Status

Fixed

[N2] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The owner role of the RoyaltyFeeRegistry and RoyaltyFeeSetter contract can update the royaltyFeeLimit and the FeeInfo of the NFT collection.

Code location:

Manager/royalty/RoyaltyFeeRegistry.sol#33-60

```
    function updateRoyaltyFeeLimit(uint256 _royaltyFeeLimit) external override
onlyOwner {
        //不得高于上限
        require(_royaltyFeeLimit <= 9500, "Royalty fee limit too high");
        royaltyFeeLimit = _royaltyFeeLimit;

        emit NewRoyaltyFeeLimit(_royaltyFeeLimit);
    }

    function updateRoyaltyInfoForCollection(
        address collection,
```

```
        address setter,
        address receiver,
        uint256 fee
    ) external override onlyOwner {
        require(fee <= royaltyFeeLimit, " Royalty fee set too high");
        _royaltyFeeInfoCollection[collection] = FeeInfo({setter : setter, receiver :
receiver, fee : fee});

        emit RoyaltyFeeUpdate(collection, setter, receiver, fee);
    }
```

Manager/royalty/RoyaltyFeeSetter.sol#102-109,127-129

```
    function updateRoyaltyInfoForCollection(
        address collection,
        address setter,
        address receiver,
        uint256 fee
    ) external onlyOwner {

IRoyaltyFeeRegistry(royaltyFeeRegistry).updateRoyaltyInfoForCollection(collection,
setter, receiver, fee);
    }

    function updateRoyaltyFeeLimit(uint256 _royaltyFeeLimit) external onlyOwner {
IRoyaltyFeeRegistry(royaltyFeeRegistry).updateRoyaltyFeeLimit(_royaltyFeeLimit);
    }
```

Solution

It is recommended to use a time lock mechanism or community governance to restrict.

Status

Fixed

[N3] [Suggestion] Risk of replay attack

Category: Replay Vulnerability

Content

The chainid is defined when the contract is initialized, but it is not reimplemented when DOMAIN(domainSeparator) is used in the verify function. So the domainSeparator 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:

Manager/UnemetaMarket.sol#101-126

```
    constructor(
        address _currencyManager, //货币管理者
        address _executionManager, //执行管理者
        address _royaltyFeeManager, //版税支付管理者
        address _WETH, //WETH地址
        address _protocolFeeRecipient//手续费接收者
    ) {
        DOMAIN = keccak256(
            abi.encode(
                0x8b73c3c69bb8fe3d512ecc4cf759cc79239f7b179b0ffacaa9a75d522b39400f,
            // keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)")
                0x2e3445393f211d11d7f88d325bc26ce78976b4decd39029feb202d9b409fc3c5,
            // keccak256("UnemetaMarket")
                0xc89efdaa54c0f20c7adf612882df0950f5a951637e0307cdcb4c672f298b8bc6,
            // keccak256(bytes("1")) for versionId = 1
                block.chainid,
                address(this)
            )
        );

        currencyManager = Currency(_currencyManager);
        executionManager = TheExManager(_executionManager);
        royaltyFeeManager = TheRoyaltyManager(_royaltyFeeManager);
        WETH = _WETH;
        protocolFeeRecipient = _protocolFeeRecipient;
    }

    function _validateOrder(OrderModel.MakerOrder calldata Make, bytes32 Hash)
internal view {
        // Verify whether order nonce has expired
        require(
```

```
        //确定订单是否取消或者超时
            (!_theUserOrderExecutedOrCancelled[Make.signer][Make.nonce]) &&
            (Make.nonce >= userMinOrderNonce[Make.signer]),
            "Order: Matching order expired"
        );

        //签名订单不能为空
        require(Make.signer != address(0), "The Order signer cannot be the zero
address");

        //确定提供的数量是否大于0
        require(Make.amount > 0, "The order amount should be greater than 0");

        //确定签名有效性
        //因为使用的是服务器中储存的eip712 签名。必须按照同样的结构复原
        //保证对应的签名参数是有效的
        require(
            SignatureChecker.
            verify(
                Hash, //hash
                Make.signer, // 挂单签名者
                Make.v, //签名参数 来自于eip712标准
                Make.r,
                Make.s,
                DOMAIN
            ),
            "Signature: Invalid"
        );

        //确定指定的货币是否在白名单中
        require(currencyManager.isCurrencyWhitelisted(Make.currency), " Not in
Currency whitelist");

        //确定交易策略是否在白名单中切是否能够正确执行
        require(executionManager.isStrategyWhitelisted(Make.strategy), " Not in
Strategy whitelist");
    }
```

lib/SignatureChecker.sol#55-76

```
    function verify(
        bytes32 hash,
```

```
        address signer,
        uint8 v,
        bytes32 r,
        bytes32 s,
        bytes32 domainSeparator
    ) internal view returns (bool) {
        // \x19\x01 是标准化的编码前缀
        // https://eips.ethereum.org/EIPS/eip-712#specification
        //对传入的domain 和hash 进行编码校验
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator,
hash));
        //如果签名地址是合约地址
        if (Address.isContract(signer)) {
            // 0x1626ba7e 是签名合约的 interfaceId(see IERC1271)
            // 1271的标准接口
            return IERC1271(signer).isValidSignature(digest, abi.encodePacked(r, s,
v)) == 0x1626ba7e;
        } else {
            //确定签名的地址是否和传入的地址一致
            return recover(digest, v, r, s) == signer;
        }
    }
```

Solution

It is recommended to redefine when using DOMAINSEPARATOR.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002209160002	SlowMist Security Team	2022.09.13 - 2022.09.16	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 1 medium risk, 1 low risk, 1 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist