Smart Contract Security Audit Report

Table Of Contents

6 Statement

1 Executive Summary

On 2025.12.2, the SlowMist security team received the SpaceAI security audit application for Account Abstraction, developed the audit plan according to the agreement of both parties and the characteristics of the project, and fi nally 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

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Arbitrary transfer issues	Design Logic Audit	Low	Fixed
N2	Issue of the return value	Design Logic Audit	Suggestion	Acknowledged
N3	Preemptive initialization	Race Conditions Vulnerability	Suggestion	Fixed
N4	Conditional judgement issue	Others	Suggestion	Fixed
N5	Redundant codes	Others	Suggestion	Acknowledged
N6	The issue of the transfer amount	Design Logic Audit	High	Fixed
N7	Shadow variable issue	Design Logic Audit	High	Fixed
N8	Value subject matter	Design Logic Audit	Low	Acknowledged

NO	Title	Category	Level	Status
	issue			
N9	The DoS issue	Denial of Service Vulnerability	Medium	Fixed
N10	Gas optimization	Gas Optimization Audit	Suggestion	Acknowledged
N11	Flag Bit Suggestions for Initialisation Functions	Design Logic Audit	Low	Acknowledged
N12	Missing event record	Malicious Event Log Audit	Suggestion	Acknowledged
N13	call() should be used instead of transfer()	Others	Suggestion	Acknowledged
N14	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged
N15	PayMaster signature check is missing	Design Logic Audit	High	Fixed
N16	Risk of Stale Price Data from Oracle	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

Commit: 9287577b95e805c918d8dc9d6a505202092ebc87

Review Commit:2f3982738fa4706b4f5806ac3657a8e36a90a6b0

The main network address of the contract is as follows:

Deployed Addresses

Deployed on Solana, POLYGON, OP, AVAX, ARBITRUM

Core Contracts

Contracts	Address
EntryPoint	0xdc5319815CdAaC2d113f7F275bc893ed7D9cA469
TokenPaymaster	0xd348FB9D8a421f5B3CB077e819dE38c9Cd7fe6F2
FreeGasPaymaster	0xd4cA5B29f8E222aAEEF944F445D1aC368a5d7694
DefaultCallbackHandler	0xc9b02677ebFa3f4dA43EBEfC6fc38e11148b664D
SmartAccount	0x3DbeB76d9d9444D7Db9DcF3799e17ACd247f8fac
SmartAccountProxyFactory	0x81E11c4701C5189b0122ef42DaF1fF3d453D968E
UserOperationHelper	0x9A998225AB0A872665B35a8dC615aAbd5e73Cd12
BundlerDepositHelper	0x71C9F21517F85D36A0FCDB8E31Ba8a8e28622cFa

Contracts deployed on ETH, OKTC, BNB, POLYGON, OP, AVAX, ARBITRUM have the same address.

Oracle Adapters

Networks	Contracts	Address
ETH	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A
OKTC	EXOracleAdapter	0x9857f966529cb205689B7D698f495eA423E48d9c
BNB	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A
POLYGON	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A
OP	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A
AVAX	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A
ARBITRUM	ChainlinkOracleAdapter	0x7bB8FF337C5172E004C0dEca560c1c1bB7f7FF0A

Swap Helpers

Networks	Contracts	Address
ETH	UniSwapV3Adapter	0x1C821cD745924f2E008e2B6759c272a1736c6d8b

Networks	Contracts	Address
OKTC	OKCSwapAdapter	0x03e70e92dC6ED4b65B7ace9b44b85Bb2b55400f2
BNB	UniSwapV2Adapter	0x7f4D0B7ee0a9a75D947419F8fDfB78d5aB91E57e
POLYGON	UniSwapV3Adapter	0x1C821cD745924f2E008e2B6759c272a1736c6d8b
OP	UniSwapV3Adapter	0x1C821cD745924f2E008e2B6759c272a1736c6d8b
AVAX	TradeJoeV2Adapter	0xE92E3568087D2999227c7a289eAf3c4a29c4CB90
ARBITRUM	UniSwapV3Adapter	0x1C821cD745924f2E008e2B6759c272a1736c6d8b

4.2 Visibility Description

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

Executor			
Function Name	Visibility	Mutability	Modifiers
execute	Internal	Can Modify State	-

FallbackManager			
Function Name	Visibility	Mutability	Modifiers
getFallbackHandler	Public	-	-
setFallbackHandler	External	Can Modify State	authorized
setFallbackHandler	Internal	Can Modify State	-
initializeFallbackHandler	Internal	Can Modify State	-
<Fallback>	External	Can Modify State	-

GuardManager			
Function Name	Visibility	Mutability	Modifiers

GuardManager			
getGuard	Public	-	-
setGuard	External	Can Modify State	authorized
initializeGuard	Internal	Can Modify State	-
execTransactionBatch	External	Can Modify State	authorized
executeWithGuard	Internal	Can Modify State	-
executeWithGuardBatch	Internal	Can Modify State	-

ModuleManager			
Function Name	Visibility	Mutability	Modifiers
initializeModules	Internal	Can Modify State	-
enableModule	Public	Can Modify State	authorized
disableModule	Public	Can Modify State	authorized
isModuleEnabled	Public	-	-
execTransactionFromModule	Public	Can Modify State	-
execTransactionFromModuleReturnData	Public	Can Modify State	-

OwnerManager			
Function Name	Visibility	Mutability	Modifiers
initializeOwners	Internal	Can Modify State	-
isOwner	Public	-	-
getOwner	Public	-	-

SignatureManager			
Function Name	Visibility	Mutability	Modifiers

SignatureManager			
<Constructor>	Public	Can Modify State	-
getUOPHash	Public	-	-
getUOPSignedHash	Public	-	-
validateUserOp	Public	Can Modify State	-
validateUserOpWithoutSig	Public	Can Modify State	-
isValidSignature	External	-	-

SecuredTokenTransfer			
Function Name	Visibility	Mutability	Modifiers
transferToken	Internal	Can Modify State	-

SelfAuthorized			
Function Name	Visibility	Mutability	Modifiers
requireSelfCall	Private	-	-

SignatureDecoder			
Function Name	Visibility	Mutability	Modifiers
signatureSplit	Internal	-	-

Singleton			
Function Name	Visibility	Mutability	Modifiers
updateImplement	External	Can Modify State	authorized

DefaultCallbackHandler			
Function Name	Visibility	Mutability	Modifiers

DefaultCallbackHandler			
onERC1155Received	External	-	-
onERC1155BatchReceived	External	-	-
onERC721Received	External	-	-
tokensReceived	External	-	-
supportsInterface	External	-	-

SimulateTxAccessor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
simulate	External	Can Modify State	onlyDelegateCall

SmartAccountInitCode			
Function Name	Visibility	Mutability	Modifiers
getInitCode	Public	-	-

SmartAccount			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	SignatureManager
Initialize	External	Can Modify State	-
validateUserOp	Public	Can Modify State	onlyEntryPoint
validateUserOpWithoutSig	Public	Can Modify State	onlyEntryPoint
execTransactionFromEntrypoint	Public	Can Modify State	onlyEntryPoint
execTransactionFromEntrypointBatch	External	Can Modify State	onlyEntryPoint
execTransactionFromModule	Public	Can Modify State	-

SmartAccountProxy			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	-
masterCopy	External	-	-
<Fallback>	External	Payable	-

SmartAccountProxyFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setSafeSingleton	Public	Can Modify State	onlyOwner
proxyRuntimeCode	Public	-	-
proxyCreationCode	Public	-	-
deployProxyWithNonce	Internal	Can Modify State	-
createProxyWithNonce	Internal	Can Modify State	-
createAccount	Public	Can Modify State	-
getAddress	Public	-	-

ChainlinkOracleAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	PriceOracle
exchangePrice	Public	-	-

EXOracleAdapter			
Function Name	Visibility	Mutability	Modifiers
setExOraclePriceData	Public	Can Modify State	onlyOwner

EXOracleAdapter			
setPriceType	Public	Can Modify State	onlyOwner
setOracleDecimals	Public	Can Modify State	onlyOwner
<Constructor>	Public	Can Modify State	PriceOracle
exchangePrice	Public	-	-

FreeGasPaymaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addToWhitelist	External	Can Modify State	onlyOwner
removeFromWhitelist	External	Can Modify State	onlyOwner
withdrawDepositNativeToken	Public	Can Modify State	onlyOwner onlyWhitelisted
getHash	Public	-	-
validatePaymasterUserOp	External	-	-
validatePaymasterUserOpWithoutSig	External	-	-
postOp	External	Can Modify State	-

TokenPaymaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
postOp	External	Can Modify State	onlyEntryPoint
getHash	Public	-	-
validatePaymasterUserOp	External	-	-
validatePaymasterUserOpWithoutSig	External	-	-

TokenPaymaster			
withdrawERC20	External	Can Modify State	onlyOwner onlyWhitelisted
withdrawDepositNativeToken	Public	Can Modify State	onlyOwner onlyWhitelisted
swapToNative	External	Can Modify State	onlyOwner
setSwapHelper	External	Can Modify State	onlyOwner
setPriceOracle	External	Can Modify State	onlyOwner
addToWhitelist	External	Can Modify State	onlyOwner
removeFromWhitelist	External	Can Modify State	onlyOwner

SwapHelper			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
swapToNative	External	Can Modify State	-
swapToNativeViaUniV2	Internal	Can Modify State	-
slippageOf	Public	-	-
setSlippage	External	Can Modify State	onlyOwner

PriceOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setPriceFeed	External	Can Modify State	onlyOwner
exchangePrice	Public	-	-
exchangeRate	External	-	-
getValueOf	External	-	-
tokenDecimals	Public	-	-
setDecimals	External	Can Modify State	onlyOwner

Storage			
Function Name	Visibility	Mutability	Modifiers
setWalletWhitelistControl	Public	Can Modify State	onlyOwner
setUnrestrictedBundler	Public	Can Modify State	onlyOwner
setModuleWhitelistControl	Public	Can Modify State	onlyOwner
setBundlerOfficialWhitelist	Public	Can Modify State	onlyOwner
setWalletProxyFactoryWhitelist	Public	Can Modify State	onlyOwner
setModuleWhitelist	Public	Can Modify State	onlyOwner
validateModuleWhitelist	Public	-	-
validateWalletWhitelist	Public	-	-

BundlerDepositHelper			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setValidEntryPoint	Public	Can Modify State	onlyOwner
batchDepositForBundler	Public	Payable	-

OKXEntryPoint			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OKXEntryPointLogic
simulateValidationWithWalletWhitelistValidate	External	Can Modify State	-
simulateHandleOpWithoutSig	External	Can Modify State	-
_validatePrepaymentWithoutSig	Internal	Can Modify State	-

OKXEntryPoint			
_validateAccountPrepaymentWithoutSig	Internal	Can Modify State	-
_validatePaymasterPrepaymentWithoutSig	Internal	Can Modify State	-
_executeUserOpWithResult	Internal	Can Modify State	-
innerHandleOpWithResult	External	Can Modify State	-

OKXEntryPointLogic			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
handleOps	Public	Can Modify State	-
handleOps	Public	Can Modify State	-
handleOp	External	Can Modify State	-
handleAggregatedOps	Public	-	-

StakeManager			
Function Name	Visibility	Mutability	Modifiers
getDepositInfo	Public	-	-
getStakeInfo	Internal	-	-
balanceOf	Public	-	-
<Receive Ether>	External	Payable	-
internalIncrementDeposit	Internal	Can Modify State	-
depositTo	Public	Payable	-
addStake	Public	Payable	-
unlockStake	External	Can Modify State	-

StakeManager			
withdrawStake	External	Can Modify State	-
withdrawTo	External	Can Modify State	-

SenderCreator			
Function Name	Visibility	Mutability	Modifiers
createSender	External	Can Modify State	-

EntryPoint			
Function Name	Visibility	Mutability	Modifiers
_compensate	Internal	Can Modify State	-
_executeUserOp	Internal	Can Modify State	-
handleOps	Public	Can Modify State	-
handleAggregatedOps	Public	Can Modify State	-
simulateHandleOp	External	Can Modify State	-
innerHandleOp	External	Can Modify State	-
getUserOpHash	Public	-	-
_copyUserOpToMemory	Internal	-	-
simulateValidation	External	Can Modify State	-
_getRequiredPrefund	Internal	-	-
_createSenderIfNeeded	Internal	Can Modify State	-
getSenderAddress	Public	Can Modify State	-
_validateAccountPrepayment	Internal	Can Modify State	-
_validatePaymasterPrepayment	Internal	Can Modify State	-

EntryPoint			
_validateDeadline	Internal	-	-
_validatePrepayment	Internal	Can Modify State	-
_handlePostOp	Internal	Can Modify State	-
getUserOpGasPrice	Internal	-	-
min	Internal	-	-
getOffsetOfMemoryBytes	Internal	-	-
getMemoryBytesFromOffset	Internal	-	-
numberMarker	Internal	-	-

4.3 Vulnerability Summary

[N1] [Low] Arbitrary transfer issues

Category: Design Logic Audit

Content

- contracts/wallet/base/SignatureManager.sol

Since the function can be executed in its entirety even with the signature data of a non-owner, all the nativeTokens in the contract can be transferred by simply constructing a signature data.

```
function
    validateUserOp( UserOperation
    calldata userOp, bytes32,
    address,
    uint256 missingAccountFunds
) public virtual returns (uint256)
    { if (missingAccountFunds != 0)
    {
        payable(msg.sender).call{
            value: missingAccountFunds,
            gas: type(uint256).max
        }("");
```

```
    }
```

```
        unchecked {
            if (userOp.nonce != nonce++)
                { return  SIG_VALIDATION_FAILED;
            }
        }

        if
            ( ECDSA.recove
            r(
                getUOPSignedHash( SignatureType(uint8(bytes1(userOp.signature[0:1]
                    ))), msg.sender,
                    userOp
                ),
                userOp.signature[33:]
            ) != owner
        ) {
            return SIG_VALIDATION_FAILED;
        } else {
            return uint256(bytes32(userOp.signature[1:33]));
        }
    }

function
        validateUserOpWithoutSig( UserO
        peration calldata userOp,
        bytes32,
        address,
        uint256 missingAccountFunds
    ) public virtual returns (uint256)
        { if (missingAccountFunds != 0)
        {
            payable(msg.sender).call{  value:
                missingAccountFunds, gas:
                type(uint256).max
            }("");
        }
        unchecked {
            if (userOp.nonce != nonce++)
                { return  SIG_VALIDATION_FAILED;
            }
        }
        if
            ( ECDSA.recove
            r(
                getUOPSignedHash(
                    SignatureType(uint8(bytes1(userOp.signature[0:1]))),
                    msg.sender,
                    userOp
```

```
            ),
            userOp.signature[33:]
        ) != owner
```

```
        ) {
            return uint256(bytes32(userOp.signature[1:33]));
        } else {
            return uint256(bytes32(userOp.signature[1:33]));
        }
    }
```

Solution

Can verify that the owner of the signature is the owner before proceeding further.

Status

Fixed; In SmartAccount.sol, the functions of `validateUserOp` and `validateUserOpWithoutSig` are rewritten, and the function can only be called by `onlyEntryPoint`.

[N2] [Suggestion] Issue of the return value

Category: Design Logic Audit

Content

- contracts/wallet/base/SignatureManager.sol

Regardless of whether the signer is the owner, `uint256(bytes32(userOp.signature[1:33]))` will be returned. In other words, if the data passed in includes a signature from someone other than the owner, it will still be accepted and returned.

```
    function
        validateUserOpWithoutSig(  User
        Operation    calldata    userOp,
        bytes32,
        address,
        uint256 missingAccountFunds
    ) public virtual returns (uint256) {
        ...
        if
            ( ECDSA.recove
            r(
                getUOPSignedHash( SignatureType(uint8(bytes1(userOp.signature[0:1]
                    ))), msg.sender,
                    userOp
                ),
                userOp.signature[33:]
            ) != owner
        ) {
```

```
            return uint256(bytes32(userOp.signature[1:33]));
        } else {
            return uint256(bytes32(userOp.signature[1:33]));
        }
    }
```

Solution

Data that is not signed by the owner should not be used.

Status

Acknowledged; This function is used to predict gas.

[N3] [Suggestion] Preemptive initialization

Category: Race Conditions Vulnerability

Content

- contracts/wallet/SmartAccount.sol

This function has the problem of being preempted.

```
    function Initialize(address _owner) external
        { require(getOwner() == address(0), "account: have set
        up"); initializeOwners(_owner);
        initializeFallbackHandler(FallbackHandler);
        initializeModules();
    }
```

Solution

It is suggested that the initialize operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Fixed; This is initialised when the contract is deployed.

[N4] [Suggestion] Conditional judgement issue

Category: Others

Content

- contracts/wallet/SmartAccount.sol

SENTINEL_MODULES is a 0x0000000000000000000000000000000000000001 address, msg.sender ! =

SENTINEL_MODULES will only result in true.

```
function
    execTransactionFromModule( addr
    ess to,
    uint256  value,
    bytes calldata  data,
    Enum.Operation operation
) public virtual {
    // Only whitelisted modules are allowed.
    require(
        msg.sender != SENTINEL_MODULES && modules[msg.sender] != address(0),
        "GS104"
    );
    // Execute transaction without further confirmations.
    if (
        execute(
            ExecuteParams(false, to, value, data, ""),
            operation,
            gasleft()
        )
    ) emit ExecutionFromModuleSuccess(msg.sender);
    else emit ExecutionFromModuleFailure(msg.sender);
}
```

Solution

Conformity to design expectations.

Status

Fixed

[N5] [Suggestion] Redundant codes

Category: Others

Content

- contracts/paymaster/FreeGasPaymaster.sol

`sigValidate` is not used, if don't need to judge, can return `sigTime` directly.

```
function
    validatePaymasterUserOpWithoutSig( UserO
    peration  calldata  userOp, bytes32,
```

```
        uint256
    ) external view override returns (bytes memory, uint256) {
        uint256 sigTime = uint256(bytes32(userOp.paymasterAndData[20:52]));

        bool sigValidate = verifyingSigner !=
            getHash(userOp,
                sigTime).toEthSignedMessageHash().recover( userOp.paymasterAndData[52:]
            );

        return ("", sigTime);
    }
```

- contracts/paymaster/TokenPaymaster.sol

`sigValidate` and `requiredPreFund` is not used, if don't need to judge, this part of the validation can be removed.

```
    function
        validatePaymasterUserOpWithoutSig( User
        Operation calldata userOp,
        bytes32 userOpHash,
        uint256 requiredPreFund
    ) external view override returns (bytes memory, uint256) {
        address token  =  address(bytes20(userOp.paymasterAndData[20:40]));
        uint256 exchangeRate = uint256(bytes32(userOp.paymasterAndData[40:72]));
        uint256 sigTime =  uint256(bytes32(userOp.paymasterAndData[72:104]));
        if (exchangeRate >= tokenPriceLimitMax) {
            (uint256 price, uint256 decimals) = IPriceOracle(priceOracle)
                .exchangePrice(token);
            exchangeRate =
                (price * 10 ** IERC20Metadata(token).decimals()) /
                10 ** decimals;
        }
        bool sigValidate = verifyingSigner !=
            getHash(userOp, token, exchangeRate, sigTime)
                .toEthSignedMessageHash()
                .recover(userOp.paymasterAndData[104:]);

        return (
            abi.encode(userOpHash, userOp.sender, token, exchangeRate),
            sigTime
        );
    }
```

Solution

Unnecessary code can be deleted.

Status

Acknowledged; This function is used to predict gas.

[N6] [High] The issue of the transfer amount

Category: Design Logic Audit

Content

- contracts/paymaster/TokenPaymaster.sol

The issue arises in the function swapToNative. After the conversion of tokens to native tokens via swapToNative, the amount of tokens is not guaranteed to be 1:1. Consequently, it is incorrect to use the initial token amount as the amount for the subsequent native token transfers.

```
function swapToNative(IERC20 token, uint256 amount) external onlyOwner
    { token.safeTransfer(swapHelper, amount);
    ISwapHelper(swapHelper).swapToNative(address(token));
    IEntryPoint(supportedEntryPoint).depositTo{value: amount}(
        address(this)
    );//SLOWMIST//
}
```

Solution

After the token to native token swap, the available balance of native tokens should be checked and used as the amount for the native token transfer operation.

Status

Fixed

[N7] [High] Shadow variable issue

Category: Design Logic Audit

Content

- contracts/paymaster/swapHelper.sol

The variable `amountOut` is redeclared within the scope of the function `swapToNativeViaUniV2`, which subsequently leads to an incorrect return value of 0.

```
function
        swapToNativeViaUniV2( a
        ddress tokenIn
    ) internal returns (uint256 amountOut) {
        uint256 tokenInBalance = IERC20(tokenIn).balanceOf(address(this));

        uint256 minAmountOut =
            (priceOracle.getValueOf( tokenIn,
            WETH,
            tokenInBalance
        ) * (1e6 - slippageOf(tokenIn))) / 1e6;

        address[] memory path = new address[](2);
        path[0] = tokenIn;
        path[1] = WETH;
        IERC20(tokenIn).approve(address(uniV2Router), tokenInBalance);
        uniV2Router.swapExactTokensForETH(
            tokenInBalance,
            minAmountOut,
            path,
            address(this),
            block.timestamp
        );

        uint256 amountOut = address(this).balance;//SLOWMIST//
        require(
            amountOut >=  minAmountOut, "swapHelper:
            amountOut <  minAmountOut"
        );

        payable(msg.sender).transfer(amountOut);
    }
```

Solution

Delete duplicate statements.

Status

Fixed

[N8] [Low] Value subject matter issue

Category: Design Logic Audit

Content

-

contracts/interfaces/IPriceOracle.sol

The concern pertains to the getValueOf function in the IPriceOracle.sol contract. This function calculates the value of a given amount of tokenIn in terms of quote token. The calculation relies on prices fetched from the exchangePrice function for both tokens and takes into consideration the token decimals.

The critical point is to ensure that the priceIn for tokenIn and the priceQuote for quote token are both derived from the same base value. If not, the value calculation may lead to inaccurate results, thereby affecting the correctness of the token exchange mechanism.

```
function getValueOf(
        address tokenIn,
        address quote,
        uint256 amountIn
    ) external view virtual override returns (uint256 value)
        { (uint256 priceIn, uint8 decimalsIn) =
        exchangePrice(tokenIn);
        (uint256 priceQuote, uint8 decimalsQuote) = exchangePrice(quote);
        if (
            decimalsIn + tokenDecimals(tokenIn) >
            decimalsQuote + tokenDecimals(quote)
        ) {
            value =
                ((amountIn * priceIn) / priceQuote) *
                10 **
                    (decimalsQuote +
                        tokenDecimals(quote) -
                        (tokenDecimals(tokenIn) + decimalsIn));
        } else {
            value =
                ((amountIn * priceIn) *
                    10 **
                        (decimalsQuote +
                            tokenDecimals(quote) -
                            (tokenDecimals(tokenIn) + decimalsIn))) /
                priceQuote;
        }
    }
```

Solution

The subject matter of the guarantee value is the same.

Status

Acknowledged; Will use the value subject matter is the same as the prophecy machine.

[N9] [Medium] The DoS issue

Category: Denial of Service Vulnerability

Content

- contracts/interfaces/IPriceOracle.sol

When the condition `decimalsIn + tokenDecimals(tokenIn) >decimalsQuote + tokenDecimals(quote)` is true, `decimalsQuote + tokenDecimals(quote) - (tokenDecimals(tokenIn) + decimalsIn))` will fail.

```
function getValueOf(
        address tokenIn,
        address quote,
        uint256 amountIn
    ) external view virtual override returns (uint256 value)
        { (uint256 priceIn, uint8 decimalsIn) =
        exchangePrice(tokenIn);
        (uint256 priceQuote, uint8 decimalsQuote) = exchangePrice(quote);
        if (
            decimalsIn + tokenDecimals(tokenIn) >
            decimalsQuote + tokenDecimals(quote)
        ) {//
            value =
                ((amountIn * priceIn) / priceQuote) *
                10 **
                    (decimalsQuote +
                        tokenDecimals(quote) -
                        (tokenDecimals(tokenIn) + decimalsIn));//SLOWMIST//
        } else {
            value =
                ((amountIn * priceIn) *
                    10 **
                        (decimalsQuote +
                            tokenDecimals(quote) -
                            (tokenDecimals(tokenIn) + decimalsIn)))
                priceQuote;
        }
    }
```

Solution

Give the correct calculation formula.

Status

Fixed

[N10] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

- contracts/@eth-infinitism-v0.4/StakeManager.sol

info doesn't need to be persistent, can use `memory` to store it temporarily, you don't need to use `storage`.

```
 function depositTo(address account) public payable
        { internalIncrementDeposit(account,  msg.value);
        DepositInfo storage info = deposits[account]; //SLOWMIST//
        emit Deposited(
            msg.sender,
            address(this),
            account,
            msg.value,
            info.deposit
        );
    }

function addStake(uint32 _unstakeDelaySec) public payable
        { DepositInfo storage info = deposits[msg.sender];
        require(_unstakeDelaySec > 0, "must specify unstake delay");
        require(
            _unstakeDelaySec >= info.unstakeDelaySec,
            "cannot decrease unstake time"
        );
        uint256 stake = info.stake  +  msg.value;
        require(stake > 0, "no stake specified");
        require(stake < type(uint112).max, "stake overflow");
        deposits[msg.sender] = DepositInfo(
            info.deposit,
            true,
            uint112(stake),
            _unstakeDelaySec,
            0
        );
        emit StakeLocked(msg.sender, stake, _unstakeDelaySec);
    }
```

Solution

Using memory.

Status

Acknowledged

[N11] [Low] Flag Bit Suggestions for Initialisation Functions

Category: Design Logic Audit

Content

- contracts/wallet/SmartAccount.sol

The issue resides in the Initialize function. It uses the address(0) as the condition to check if the smart contract has been initialized. However, this is not a reliable or best practice method for initialization checks.

```
function Initialize(address _owner) external
    { require(getOwner() == address(0), "account: have set
    up"); initializeOwners(_owner);
    initializeFallbackHandler(FallbackHandler);
    initializeModules();
}
```

Solution

It is suggested to adopt the Initializable module provided by the OpenZeppelin library for initialization checks. The Initializable module provides a secure and industry standard way to handle smart contract initialization, thus preventing any possible loopholes or errors due to improper initialization checks.

Status

Acknowledged

[N12] [Suggestion] Missing eventrecord

Category: Malicious Event Log Audit

Content

Key Parameter Settings Unrecorded Events .

- contracts/core/BundlerDepositHelper.sol

The following functions do not log events `setValidEntryPoint`.

- contracts/paymaster/TokenPaymaster.sol

The following functions do not log events `addToWhitelist` , `removeFromWhitelist` , `setSwapHelper` .

- contracts/paymaster/FreeGasPaymaster.sol

The following functions do not log events `addToWhitelist` , `removeFromWhitelist` .

Solution

Recording events.

Status

Acknowledged

[N13] [Suggestion] call() should be used instead of transfer()

Category: Others

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

```
function
    swapToNative( address
    tokenIn
) external override returns (uint256 amountOut)
    { return swapToNativeViaUniV2(tokenIn);
}

function
    swapToNativeViaUniV2( addr
    ess tokenIn
) internal returns (uint256 amountOut) {
    uint256 tokenInBalance = IERC20(tokenIn).balanceOf(address(this));

    uint256 minAmountOut =
        (priceOracle.getValueOf( tokenIn,
        WETH,
        tokenInBalance
    ) * (1e6 - slippageOf(tokenIn))) / 1e6;

    address[] memory path = new address[](2);
```

```
        path[0] = tokenIn;
        path[1] = WETH;
```

```
        IERC20(tokenIn).approve(address(uniV2Router), tokenInBalance);
        uniV2Router.swapExactTokensForETH(
            tokenInBalance,
            minAmountOut,
            path,
            address(this),
            block.timestamp
        );

        uint256 amountOut = address(this).balance;
        require(
            amountOut >=  minAmountOut, "swapHelper:
            amountOut <  minAmountOut"
        );

        payable(msg.sender).transfer(amountOut);
    }
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern and/or add re-entrancy guards.

Status

Acknowledged; Here transfer is for paymaster receive() doesn't consider transfer to other address. So there is no such issue in the actual scenario.

[N14] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

The owner of the following contracts has a lot of power, and if the private key is leaked, it can cause a lot of damage.

- contracts/paymaster/oracle/ExOracleAdapter.sol

owner can setExOraclePriceData,

owner can setPriceType

owner can setOracleDecimals

- contracts/paymaster/FreeGasPaymaster.sol

owner can addToWhitelist

owner can removeFromWhitelist

owner can withdrawERC20

owner can withdrawDepositNativeToken

- contracts/paymaster/TokenPaymaster.sol

owner can withdrawERC20

owner can withdrawDepositNativeToken

owner can swapToNative

owner can setSwapHelper

owner can setPriceOracle

owner can addToWhitelist

owner can removeFromWhitelist

- contracts/core/Storage.sol

owner can setWalletWhitelistControl

owner can setUnrestrictedBundler

owner can setModuleWhitelistControl

owner can setBundlerOfficialWhitelist

owner can setWalletProxyFactoryWhitelist

owner can setModuleWhitelist

- contracts/wallet/SmartAccountProxyFactory.sol

owner can setSafeSingleton

- contracts/paymaster/swapHelper.sol

owner can setSlippage

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. And 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; Deployed on ETH, BNB, POLYGON, OP, AVAX, ARBITRUM,

Owner's multi-signature address 0x8724e70e7e608a9a06d2bf32ca17162a3c054061.

Deployed on OKTC

Owner's multi-signature address 0x41d49b4041606dfa7108111cfe1501399da8b976

[N15] [High] PayMaster signature check is missing

Category: Design Logic Audit

Content

- contracts/paymaster/TokenPaymaster.sol

In the method validatePaymasterUserOp within the file, there is an issue concerning the validation of signature data. Specifically, if exchangeRate surpasses or equals tokenPriceLimitMax, there is no validation conducted for the signature data.

This omission may permit malicious actors to exploit this behavior and utilize the funds of other paymasters to cover the gas expenses, potentially leading to losses of funds.

```
function
        validatePaymasterUserOp( UserO
        peration calldata userOp,
        bytes32 userOpHash,
        uint256
    ) external view override returns (bytes memory, uint256) {
        address token  =  address(bytes20(userOp.paymasterAndData[20:40]));
        uint256 exchangeRate = uint256(bytes32(userOp.paymasterAndData[40:72]));
        uint256 sigTime =  uint256(bytes32(userOp.paymasterAndData[72:104]));

        if (exchangeRate >= tokenPriceLimitMax) {
            uint256 oracleExchangeRate =
                IPriceOracle(priceOracle).exchangeRate( token
            );
```

```
            return (
                abi.encode(
                    userOpHash,
                    userOp.sender,
                    token,
                    oracleExchangeRate
                ),
                sigTime
            );
        } else if (
            verifyingSigner ==
            getHash(userOp, token, exchangeRate, sigTime)
                .toEthSignedMessageHash()
                .recover(userOp.paymasterAndData[104:])
        ) {
            return (
                abi.encode(userOpHash, userOp.sender, token, exchangeRate),
                sigTime
            );
        } else {
            return ("", SIG_VALIDATION_FAILED);
        }
    }
```

Solution

To verify paymaster signature data.

Status

Fixed

[N16] [Low] Risk of Stale Price Data from Oracle

Category: Design Logic Audit

Content

- contracts/paymaster/oracle/ChainlinkOracleAdapter.sol

The method `exchangePrice` in the ChainlinkOracleAdapter contract is susceptible to potential issues due to an insufficient handling of the time data returned by `tokenPriceFeed.latestRoundData()`. This could lead to inaccuracies if the Oracle is down and the price returned is outdated.

To safeguard against such scenarios, it is essential to implement appropriate measures to handle the time returned by `tokenPriceFeed.latestRoundData()`. This could involve incorporating mechanisms to validate the freshness of the returned data or handle situations where the prediction machine might be down. These precautions would increase the reliability and accuracy of the price data used within the system.

```
    function
        exchangePrice( addre
        ss token
    ) public view virtual override returns (uint256 price, uint8 decimals)
        { AggregatorV3Interface tokenPriceFeed = AggregatorV3Interface(
            priceFeed[token]
        );
        require(tokenPriceFeed != AggregatorV3Interface(address(0)), "");

        (
            ,
            /* uint80 roundID */ int256 _price /*uint startedAt*/ /*uint timeStamp*/
/*uint80 answeredInRound*/,
            ,
            ,

        ) = tokenPriceFeed.latestRoundData();
        //  price -> uint256
        require(_price >= 0, "price is negative");
        price = uint256(_price);
        decimals = tokenPriceFeed.decimals();
    }
```

- contracts/paymaster/oracle/ExOracleAdapter.sol

`IExOraclePriceData(exOracle).get()` will also return a time, if it is the time of the record, the same also need to determine whether it is too long without updates

```
    function
        exchangePrice( addres
        s token
    ) public view virtual override returns (uint256 price, uint8 decimals)  {
        require(priceFeed[token] != address(0), "");

        (price, ) =
            IExOraclePriceData(exOracle).get( priceTy
            pe[token],
            priceFeed[token]
        );
```

```
        decimals = oracleDecimals[token];
    }
```

Solution

Not using prices that are not within the expected time frame.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002207180002	SlowMist Security Team	2025.10.20	Low 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 3 high risk, 1 medium risk, 5 low risk, 7 suggestion vulnerabilities.

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