SLOWMIST

Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.10.15, the SlowMist security team received the zkswap team's security audit application for zkswap V3, 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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/l2labs/zkswap-v3/tree/develop

commit: 0dede1df06ec553feec47c813b8c8c185ef8ba23

Fixed Version:

https://github.com/l2labs/zkswap-v3/tree/develop

commit: 90500b9e0d4f3a9240ba0a7dfbb39915f4f4b933

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Race conditions issue	Race Conditions Vulnerability	Low	Confirmed
N2	Missing event record	Others	Suggestion	Ignored
N3	Missing overflow/underflow check	Integer Overflow and Underflow Vulnerability	Low	Confirmed
N4	Missing permission check	Authority Control Vulnerability	Critical	Fixed
N5	Enhancement suggestions for permission checking	Authority Control Vulnerability	Suggestion	Confirmed
N6	Gas optimization	Gas Optimization Audit	Suggestion	Fixed
N7	DoS issue	Denial of Service Vulnerability	Low	Confirmed
N8	Business logic is not clear	Design Logic Audit	Suggestion	Ignored
N9	Logic contract upgrade security reminder	Others	Suggestion	Confirmed
N10	Token compatibility security reminder	Others	Suggestion	Confirmed
N11	Gas token attack issue	Gas Optimization Audit	Low	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:

DeployFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deployProxyContracts	Internal	Can Modify State	-
finalizeGovernance	Internal	Can Modify State	-

ZKBoxNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
initialize	External	Can Modify State	-
setZkSyncAddress	External	Can Modify State	-
onDeposit	External	Can Modify State	onlyZksCore
onConfirmDeposit	External	Can Modify State	onlyZksCore
mint	Internal	Can Modify State	-
toBase58	Internal	-	-
ipfsCID	Public	-	-
toAlphabet	Internal	-	-
reverse	Internal	-	-
setContractURI	Public	Can Modify State	onlyOwner

ZKBoxNFT			
contractURI	Public	-	-
packWithdrawKey	Internal	-	-
withdrawBalanceUpdate	External	Can Modify State	onlyZksCore
addWithdraw	External	Can Modify State	onlyZksCore
genWithdrawItems	External	Can Modify State	onlyZksCore
onWithdraw	External	Can Modify State	onlyZksCore
tokenURI	Public	-	-
getContentHash	External	-	-

ZkSync			
Function Name	Visibility	Mutability	Modifiers
createPair	External	Can Modify State	-
createETHPair	External	Can Modify State	-
registerCreatePair	Internal	Can Modify State	-
getNoticePeriod	External	Can Modify State	-
upgradeNoticePeriodStarted	External	Can Modify State	-
upgradePreparationStarted	External	Can Modify State	-
upgradeCanceled	External	Can Modify State	-
upgradeFinishes	External	Can Modify State	-
isReadyForUpgrade	External	Can Modify State	-

ZkSync			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
setMaxDepositAmount	External	Can Modify State	-
setWithDrawGasLimit	External	Can Modify State	-
setWithDrawNFTGasLimit	External	Can Modify State	-
setGenesisRootAndAddresses	External	Can Modify State	-
upgrade	External	Can Modify State	-
withdrawERC20Guarded	External	Can Modify State	-
completeWithdrawals	External	Can Modify State	nonReentrant
cancelOutstandingDepositsForExodusMode	External	Can Modify State	nonReentrant
depositETH	External	Payable	nonReentrant
withdrawETH	External	Can Modify State	nonReentrant
withdrawETHWithAddress	External	Can Modify State	nonReentrant
depositERC20	External	Can Modify State	nonReentrant
withdrawERC20	External	Can Modify State	nonReentrant
withdrawERC20WithAddress	External	Can Modify State	nonReentrant
fullExit	External	Can Modify State	nonReentrant
registerDeposit	Internal	Can Modify State	-
registerWithdrawal	Internal	Can Modify State	-
requireActive	Internal	-	-

ZkSync			
addPriorityRequest	Internal	Can Modify State	-
checkWithdrawals	External	Can Modify State	nonReentrant
processOnchainWithdrawals	Internal	Can Modify State	-
<Fallback>	External	Payable	-

ZKBox			
Function Name	Visibility	Mutability	Modifiers
depositNFT	External	Can Modify State	nonReentrant
withdrawNFT	External	Can Modify State	nonReentrant
fullExitNFT	External	Can Modify State	nonReentrant
completeWithdrawalsNFT	External	Can Modify State	nonReentrant
numOfPendingWithdrawalsNFT	External	Can Modify State	-
requireActive	Internal	-	-
addPriorityRequest	Internal	Can Modify State	-
<Fallback>	External	Payable	-
onERC721Received	External	Can Modify State	-

ZkSyncCommitBlock			
Function Name	Visibility	Mutability	Modifiers
commitBlock	External	Can Modify State	nonReentrant
commitMultiBlock	External	Can Modify State	nonReentrant

ZkSyncCommitBlock			
verifyBlock	External	Can Modify State	nonReentrant
createMultiblockCommitment	Internal	Can Modify State	-
verifyBlocks	External	Can Modify State	nonReentrant
revertBlocks	External	Can Modify State	nonReentrant
triggerExodusIfNeeded	External	Can Modify State	-
setAuthPubkeyHash	External	Can Modify State	nonReentrant
createCommittedBlock	Internal	Can Modify State	-
createCommittedMultiBlock	Internal	Can Modify State	-
emitDepositCommitEvent	Internal	Can Modify State	-
emitFullExitCommitEvent	Internal	Can Modify State	-
emitCreatePairCommitEvent	Internal	Can Modify State	-
emitDepositNFTCommitEvent	Internal	Can Modify State	-
emitFullExitNFTCommitEvent	Internal	Can Modify State	-
collectOnchainOps	Internal	Can Modify State	-
collectOnchainMultiOps	Internal	Can Modify State	-
verifyChangePubkeySignature	Internal	-	-
createBlockCommitment	Internal	-	-
commitNextPriorityOperation	Internal	-	-
isBlockCommitmentExpired	Internal	-	-
requireActive	Internal	-	-

ZkSyncCommitBlock			
deleteRequests	Internal	Can Modify State	-
<Fallback>	External	Payable	-

ZkSyncExit			
Function Name	Visibility	Mutability	Modifiers
exit	External	Can Modify State	nonReentrant
exitNFT	External	Can Modify State	nonReentrant
updateBalance	Internal	Can Modify State	-
checkLpL1Balance	Internal	Can Modify State	-
checkPairAccount	Internal	-	-
lpExit	External	Can Modify State	nonReentrant

4.3 Vulnerability Summary

[N1] [Low] Race conditions issue

Category: Race Conditions Vulnerability

Content

The initialize function has no permission control, and there is an issue of being preemptively initialize.

- contracts/nft/ZKBoxNFT.sol#L54-L60

```
function initialize(bytes calldata initializationParameters) external {
        require(externAccountSeqId == 0, "duplicate init");
        _name = "ZKBox";
        _symbol = "ZKBox";
        _setOwner(abi.decode(initializationParameters, (address)));
```

```
       externAccountSeqId = 1;
   }
```

The setZkSyncAddress function has no permission control, and there is an issue of being preemptively setZkSyncAddress.

- contracts/contracts/nft/ZKBoxNFT.sol#L62-L65

```
function setZkSyncAddress(address _zksyncAddress) external {
       require(zksCore == address(0), "ZKBoxNFT: already initialized");
       zksCore = _zksyncAddress;
   }
```

The setZkSyncAddress function has no permission control, and there is an issue of being preemptively setZkSyncAddress.

- contracts/contracts/uniswap/UniswapV2Factory.sol#L20-L23

```
function setZkSyncAddress(address _zksyncAddress) external {
       require(zkSyncAddress == address(0), "szsa1");
       zkSyncAddress = _zksyncAddress;
   }
```

The initialize function has no permission control, and there is an issue of being preemptively initialize.

- contracts/contracts/ZkSync.sol#L151-L173

```
function initialize(bytes calldata initializationParameters) external {
       require(address(governance) == address(0), "init0");
       initializeReentrancyGuard();

       (
           address _governanceAddress,
           address _verifierAddress,
           address _verifierExitAddress,
           address _pairManagerAddress,
           address _zkBoxNFT
       ) = abi.decode(initializationParameters, (address, address, address, address, 
address));

       verifier = Verifier(_verifierAddress);
       verifierExit = VerifierExit(_verifierExitAddress);
       governance = Governance(_governanceAddress);
       pairmanager = UniswapV2Factory(_pairManagerAddress);
       zkBoxNFT = IZKBoxNFT(_zkBoxNFT);

       maxDepositAmount = DEFAULT_MAX_DEPOSIT_AMOUNT;
       withdrawGasLimit = ERC20_WITHDRAWAL_GAS_LIMIT;
       withdrawNFTGasLimit = ERC721_WITHDRAWAL_GAS_LIMIT;

   }
```

Solution

It is recommended to perform authentication processing for initialization related operations.

Status

Confirmed; After communication and feedback, the zkswap project team will deploy the contract in a unified script.

After the contract is deployed, zkswap project team will control these functions to be called by the DeployFactory immediately.

[N2] [Suggestion] Missing event record

Category: Others

Content

There is no event record for modifying _contractURI, which is not conducive to the review of community users.

- contracts/contracts/nft/ZKBoxNFT.sol#L177-L179

```
    function setContractURI(string memory contractURI_) public onlyOwner {
        _contractURI = contractURI_;
    }
```

Solution

It is recommended to add event records to facilitate community users to obtain new _contractURI in time.

Status

Ignored

[N3] [Low] Missing overflow/underflow check

Category: Integer Overflow and Underflow Vulnerability

Content

The code involved in arithmetic operations in the contract did not use SafeMath for overflow/underflow checking, which affected the code involved in arithmetic operations in the project.

For example, the following function：

- contracts/contracts/nft/ZKBoxNFT.sol#L239-L259

```
function genWithdrawItems(uint32 n) external onlyZksCore returns (WithdrawItem[]
memory) {
       uint32 toProcess = Utils.minU32(n, numOfPendingWithdrawals);
       uint32 startIndex = firstPendingWithdrawal;
       firstPendingWithdrawal += toProcess;
       numOfPendingWithdrawals -= toProcess;
       WithdrawItem[] memory items = new WithdrawItem[](toProcess);
       PendingWithdrawal memory pw;
       L1Info memory info;
       for (uint32 i = startIndex; i < startIndex + toProcess; ++i) {
           pw = pendingWithdrawals[i];
           delete pendingWithdrawals[i];
           info = infoMapL1[pw.globalId];
           items[i - startIndex] = WithdrawItem({
               tokenContract: info.tokenContract,
               tokenId: info.tokenId,
               to: pw.to,
               globalId: pw.globalId
           });
       }
       return items;
   }
```

Solution

It is recommended that the code involving arithmetic operations use SafeMath for overflow checking or use solidity 0.8.0 and above version.

Status

Confirmed; After communication and feedback, the zkswap project team reported that the transaction volume in the initial stage of the project will not be very large, and there will be no possibility of overflow/underflow. With the subsequent development of the business, solidity 0.8.0 and above will be gradually used

[N4] [Critical] Missing permission check

Category: Authority Control Vulnerability

Content

When the creatorId is not 0, it is not checked whether msg.sender is the owner or authorizer of this NFT.

- contracts/contracts/ZKBox.sol

```
   function depositNFT(IERC721 _token, uint256 _tokenId, address _franklinAddr)
external nonReentrant {
       requireActive();
       uint64 nextPriorityRequestId = firstPriorityRequestId +
totalOpenPriorityRequests;
       Operations.DepositNFT memory op = zkBoxNFT.onDeposit(_token, _tokenId,
_franklinAddr);
       if (op.creatorId == 0) {
           _token.safeTransferFrom(msg.sender, address(this), _tokenId);
           require(_token.ownerOf(_tokenId) == address(this), "ZKBoxNFT: depositNFT
failed");
       }
       bytes memory pubData = Operations.writeDepositNFTPubdata(op);
       addPriorityRequest(nextPriorityRequestId, Operations.OpType.DepositNFT,
pubData, "");
       emit OnchainDepositNFT(
           msg.sender,
           address(_token),
           _tokenId,
           _franklinAddr
```

```
        );
    }
```

Solution

It is recommended to add permission check,and make sure msg.sender is the owner or authorizer of the NFT.

Status

Fixed

[N5] [Suggestion] Enhancement suggestions for permission checking

Category: Authority Control Vulnerability

Content

In the code, the request of fullExitNFT is recorded in priorityRequests, and there is no logic to check the relationship between owner and globalId accountId in the code.

- contracts/contracts/ZKBox.sol#L51-L67

```
    function fullExitNFT(uint32 _accountId, uint64 _globalId) external nonReentrant {
        requireActive();
        require(_accountId <= MAX_ACCOUNT_ID, "fee11");
        require(_globalId > 0, "ZKBoxNFT: invalid exit id");
        bytes memory pubData =
Operations.writeFullExitNFTPubdata(Operations.FullExitNFT({
            accountId: _accountId,
            globalId: _globalId,
            creatorId: 0,
            seqId: 0,
            uri: 0,
            owner: msg.sender,
            success: 0
        }));
        uint64 nextPriorityRequestId = firstPriorityRequestId +
totalOpenPriorityRequests;
        addPriorityRequest(nextPriorityRequestId, Operations.OpType.FullExitNFT,
pubData, "");
        emit OnchainFullExitNFT(_accountId, msg.sender, _globalId);
    }
```

Solution

It is recommended to add authentication operations in the contract to ensure the correctness of the data.

Status

Confirmed; After communication with zkswap project, the authentication is performed on L2.

[N6] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

The visibility of the numOfPendingWithdrawalsNFT function should be view, and no data writing operation is found in the code.

- contracts/contracts/ZKBox.sol#L90-L92

```
function numOfPendingWithdrawalsNFT() external returns(uint32) {
    return zkBoxNFT.numOfPendingWithdrawals();
}
```

Solution

It is recommended to add view visibility.

Status

Fixed

[N7] [Low] DoS issue

Category: Denial of Service Vulnerability

Content

There is no maximum length limit in the for loop, so when the number of loops is large, it will cause out of gas and then revert, which will cause the previous operation to consume the gas in vain.

- contracts/contracts/ZkSyncCommitBlock.sol#L142-L154

```
    function createMultiblockCommitment(uint32 _blockNumberFrom, uint32
_blockNumberTo)
    internal returns (uint32[] memory blockSizes, uint256[] memory inputs) {
        uint32 numberOfBlocks = _blockNumberTo - _blockNumberFrom + 1;
        blockSizes = new uint32[](numberOfBlocks);
        inputs = new uint256[](numberOfBlocks);
        for (uint32 i = 0; i < numberOfBlocks; i++) {
            blockSizes[i] = blocks[_blockNumberFrom + i].chunks;

            bytes32 blockCommitment = blocks[_blockNumberFrom + i].commitment;
            uint256 mask = (~uint256(0)) >> 3;
            inputs[i] = uint256(blockCommitment) & mask;
        }
    }
```

Solution

It is recommended to have a maximum number of times limit or to have it in off-chain There is a limitation to avoid the waste of gas after DoS.

Status

Confirmed

[N8] [Suggestion] Business logic is not clear

Category: Design Logic Audit

Content

The ZkSync contract has already assigned zkSyncCommitBlockAddress to address(this) in the constructor, but the setGenesisRootAndAddresses function in the contract can modify zkSyncCommitBlockAddress. Before the modification, there is a condition that requires (zkSyncCommitBlockAddress == address(0), "sraa1"), but The value is already available at the time of construction, so setGenesisRootAndAddresses cannot be called to modify it. This part of the business logic is not very clear and needs to be symmetrical with the developer.

- contracts/contracts/ZkSync.sol#L196-207

```
function setGenesisRootAndAddresses(bytes32 _genesisRoot, address
_zkSyncCommitBlockAddress,
       address _zkSyncExitAddress, address _zkBoxAddress) external {
       // This function cannot be called twice as long as
       // _zkSyncCommitBlockAddress and _zkSyncExitAddress have been set to
       // non-zero.
       require(zkSyncCommitBlockAddress == address(0), "sraa1");
       require(zkSyncExitAddress == address(0), "sraa2");
       blocks[0].stateRoot = _genesisRoot;
       zkSyncCommitBlockAddress = _zkSyncCommitBlockAddress;
       zkSyncExitAddress = _zkSyncExitAddress;
       zkBoxAddress = _zkBoxAddress;
   }
```

- contracts/contracts/ZkSync.sol#L140

```
   constructor() public {
       governance = Governance(msg.sender);
       zkSyncCommitBlockAddress = address(this);
       zkSyncExitAddress = address(this);
       zkBoxAddress = address(this);
   }
```

There is no logic to assign values to the zkSyncCommitBlockAddress variable in the ZKBox contract. It is necessary to communicate with the developer to see if the code here is wrong.

- contracts/contracts/ZKBox.sol#L132-L144

```
   function() external payable {
       address nextAddress = zkSyncCommitBlockAddress;
       require(nextAddress != address(0), "zkSyncCommitBlockAddress should be set");
       // Execute external function from facet using delegatecall and return any
value.
       assembly {
           calldatacopy(0, 0, calldatasize())
           let result := delegatecall(gas(), nextAddress, 0, calldatasize(), 0, 0)
           returndatacopy(0, 0, returndatasize())
           switch result
           case 0 {revert(0, returndatasize())}
           default {return (0, returndatasize())}
```

```
    }
}
```

The function here can be called arbitrarily without authentication, and the global variable has not been found to be used.

contracts/contracts/ZkSync.sol#L110-L130

```
function upgradeNoticePeriodStarted() external {

}

/// @notice Notification that upgrade preparation status is activated
function upgradePreparationStarted() external {
    upgradePreparationActive = true;
    upgradePreparationActivationTime = now;
}

/// @notice Notification that upgrade canceled
function upgradeCanceled() external {
    upgradePreparationActive = false;
    upgradePreparationActivationTime = 0;
}

/// @notice Notification that upgrade finishes
function upgradeFinishes() external {
    upgradePreparationActive = false;
    upgradePreparationActivationTime = 0;
}
```

Solution

It is recommended to communicate the specific business design and calling process with the development team, and clarify the implementation of the code.

Status

Ignored; Through communication and feedback, the zkswap project team explained that the contract calls setGenesisRootAndAddresses to implement the assignment operation of zkSyncCommitBlockAddress by a proxy contract;

these "upgrade" functions is mainly used for upgradeGateKeeper, and the Zksync contract is deployed in proxy mode and can be upgraded, so it can be called arbitrarily here.

[N9] [Suggestion] Logic contract upgrade security reminder

Category: Others

Content

Need to confirm with the development whether zkSyncCommitBlockAddress can be upgraded, because there is an issue with unclear business logic in the code, N8.

Reference：https://dydx.foundation/blog/en/outage-1

```
contracts/contracts/ZKBox.sol#L132-L144
   function() external payable {
       address nextAddress = zkSyncCommitBlockAddress;
       require(nextAddress != address(0), "zkSyncCommitBlockAddress should be set");
       // Execute external function from facet using delegatecall and return any 
value.
       assembly {
           calldatacopy(0, 0, calldatasize())
           let result := delegatecall(gas(), nextAddress, 0, calldatasize(), 0, 0)
           returndatacopy(0, 0, returndatasize())
           switch result
           case 0 {revert(0, returndatasize())}
           default {return (0, returndatasize())}
       }
   }
```

Solution

If it can be upgraded, it is recommended to pay attention to the data structure of the logical contract to be consistent with the previous one when upgrading.

Status

Confirmed; The zkswap project team will pay attention to the compatibility of the data structure of the upgraded new contract with the old code

[N10] [Suggestion] Token compatibility security reminder

Category: Others

Content

When creating a pair, pay attention to reviewing the compatibility of the token and the project. Currently, the project is not compatible with deflation and inflation type tokens. If (lpTokenId> 0), the account is not calculated by calculating the actual balance received by the contract. When encountering deflationary or inflationary tokens, accounting errors will occur.

- contracts/contracts/ZkSync.sol#L345-L387

```
function depositERC20(IERC20 _token, uint104 _amount, address _franklinAddr) external
nonReentrant {
       requireActive();

       // Get token id by its address
       uint16 lpTokenId = tokenIds[address(_token)];
       uint16 tokenId = 0;
       if (lpTokenId == 0) {
           // This means it is not a pair address
           tokenId = governance.validateTokenAddress(address(_token));
       } else {
           lpTokenId = validatePairTokenAddress(address(_token));
       }

       uint256 balance_before = 0;
       uint256 balance_after = 0;
       uint128 deposit_amount = 0;
       if (lpTokenId > 0) {
           // Note: For lp token, main contract always has no money
           balance_before = _token.balanceOf(msg.sender);
           pairmanager.burn(address(_token), msg.sender,
SafeCast.toUint128(_amount));
           balance_after = _token.balanceOf(msg.sender);
           deposit_amount = SafeCast.toUint128(balance_before.sub(balance_after));
           require(deposit_amount <= maxDepositAmount, "fd011");
           registerDeposit(lpTokenId, deposit_amount, _franklinAddr);
       } else {
           balance_before = _token.balanceOf(address(this));
           require(Utils.transferFromERC20(_token, msg.sender, address(this),
SafeCast.toUint128(_amount)), "fd012");
           // token transfer failed deposit
           balance_after = _token.balanceOf(address(this));
           deposit_amount = SafeCast.toUint128(balance_after.sub(balance_before));
           require(deposit_amount <= maxDepositAmount, "fd013");
           registerDeposit(tokenId, deposit_amount, _franklinAddr);
       }
```

Solution

It is recommended to consider compatibility issues when adding lptoken.

Status

Confirmed; After communication and feedback, the project team explained, that for the LPToken generated by the project team, there will be no such issue. Only some ERC20 Tokens have such deflation, which has been dealt with in else.

[N11] [Low] Gas token attack issue

Category: Gas Optimization Audit

Content

The business design allows others to perform the withdrawETHWithAddress operation. Since there is no limit to the gaslimit of the call, if to is a malicious contract, the gas of the caller can be attacked through gastoken.

Reference: https://floriantramer.com/docs/slides/CESC18gastoken.pdf

- contracts/contracts/ZkSync.sol#L342-348

```
   function withdrawETHWithAddress(uint128 _amount, address payable _to) external
nonReentrant {
       require(_to != address(0), "ipa11");
       registerWithdrawal(0, _amount, _to);
       (bool success,) = _to.call.value(_amount)("");
       require(success, "fwe12");
       // ETH withdraw failed
   }
```

Solution

It is recommended to evaluate whether to add a gas limit according to the business design of the project.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002110220002	SlowMist Security Team	2021.10.15 - 2021.10.22	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 critical risk, 4 low risk, 6 suggestion vulnerabilities. And 3 low risk, 3 suggestion vulnerabilities were confirmed; 2 suggestion vulnerabilities were ignored; All other 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