BEOSIN
Blockchain Security

Ronin Bridge

Security Analysis Audit

No. 202408211552

Aug 21st, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Result

The audit was divided into three parts: the first part focused on the contract audit, the second part on the audit of the scripts that led to the attack, and the third part on the audit of the scripts intended to fix the vulnerabilities.

In the contract audit, no major issues were found. During the audit of the 20240716-p3-upgrade-bridge-main-chain.s.sol and 20240716-p2-upgrade-bridge-ronin-chain.s.sol scripts, the previous attack was analyzed, and potential parameter initialization issues were checked to ensure that all parameters of the cross-chain bridge were properly initialized. It was confirmed that only the weight parameters were not initialized.

In the third part, the 20240807-ir-recover script was reviewed, and the entire proposal execution process was analyzed. Based on the code logic, the process to be feasible. Ultimately, the 20240807-ir-recover script passed the tests.

- **Project Description:**

1. Business overview

This audit is a contract security audit of Ronin Bridge and a security audit of deployment scripts and transactions. The audited contracts include: RoninBridgeManager, RoninGatewayV3, MainchainBridgeManager and MainchainGatewayV3.

The RoninBridgeManager contract is the governance management contract on the Ronin chain, responsible for creating, voting on, and executing cross-chain bridge proposals. When a proposal is initiated by a governor, a proposal with an expiration time is created in the current round. Before the expiration time is reached, governors with voting weight can vote on the proposal in the current round. If the proposal's weight reaches the preset threshold before expiration, execution is determined based on the proposal's chain ID. If the chain ID is the Ronin chain itself, the proposal will be executed by the predefined executor within the proposal, which calls the predefined target contract's calldata via the BridgeManager contract. If the chain ID is Ethereum, the proposal will be suspended. Later, the governor on Ethereum will collect the signatures within the proposal and call functions like `relayProposal` in the MainchainBridgeManager contract on Ethereum to generate and create the Ethereum proposal. Since the weight has already been verified on the Ronin chain, this transaction will also pass the current proposal, thereby executing the data within the calldata. The MainchainBridgeManager contract is the governance management contract on Ethereum. Since the contract does not have a voting function, it can only process proposals that have already been signed on the Ronin chain and passed the weight check.

The RoninGatewayV3 and MainchainGatewayV3 contracts are cross-chain bridge contracts on the Ronin chain and Ethereum, respectively, responsible for the generation, receipt, and voting of cross-chain Receipts. If a user wants to perform a cross-chain operation from the Ronin chain, they can first use the `requestWithdrawalFor` function in the RoninGatewayV3 contract to generate a Receipt of the kind Withdrawal, staking the corresponding tokens to the cross-chain bridge to create the cross-chain Receipt. The user can then validate the Receipt's weight using the collected operator signatures. Once the Receipt's weight is sufficient, the cross-chain bridge will send the corresponding amount of tokens to the user recorded in the Receipt, thereby completing the cross-chain operation. A similar process applies when a user wants to perform a cross-chain operation from Ethereum; they can use the `requestDepositFor` function in the MainchainGatewayV3 contract to generate a Receipt of the kind Deposit, and then vote on it on the Ronin chain to complete the cross-chain operation. It is important to note that if the cross-chain operation involves a large amount of funds from Ronin to

Ethereum, it may trigger a fund lock, and the corresponding Receipt will need to be unlocked by an unlock account before the withdrawal can proceed.

1 Overview

1.1 Project Overview

Project Name	Ronin Bridge
Project Language	solidity
Platform	Ronin, Ethereum
Github Link	https://github.com/ronin-chain/bridge-contract/
Contract Scope	/src/ronin/gateway/RoninBridgeManager.sol /src/ronin/gateway/RoninGatewayV3.sol /src/mainchain/MainchainBridgeManager.sol /src/mainchain/MainchainGatewayV3.sol
Script Scope	/script/20240716-upgrade-v3.2.0-mainnet /script/20240807-ir-recover
Audit Commit	132fcc674f46cd900c2dc6bd677379654bd7f639
Audit Transactions	Ronin Transaction: 0xf621da2b000ef3d59d04fe494e6f40b23fe4c2fc29ff32bb9cc9d61eaf28a8a3 Ethereum Transaction: 0xd9e926f03876a286cad87f21015127f7b2b949323d4b4db273c285990a9d336b

1.2 Audit Overview

Audit work duration: Aug 13, 2024 – Aug 21, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

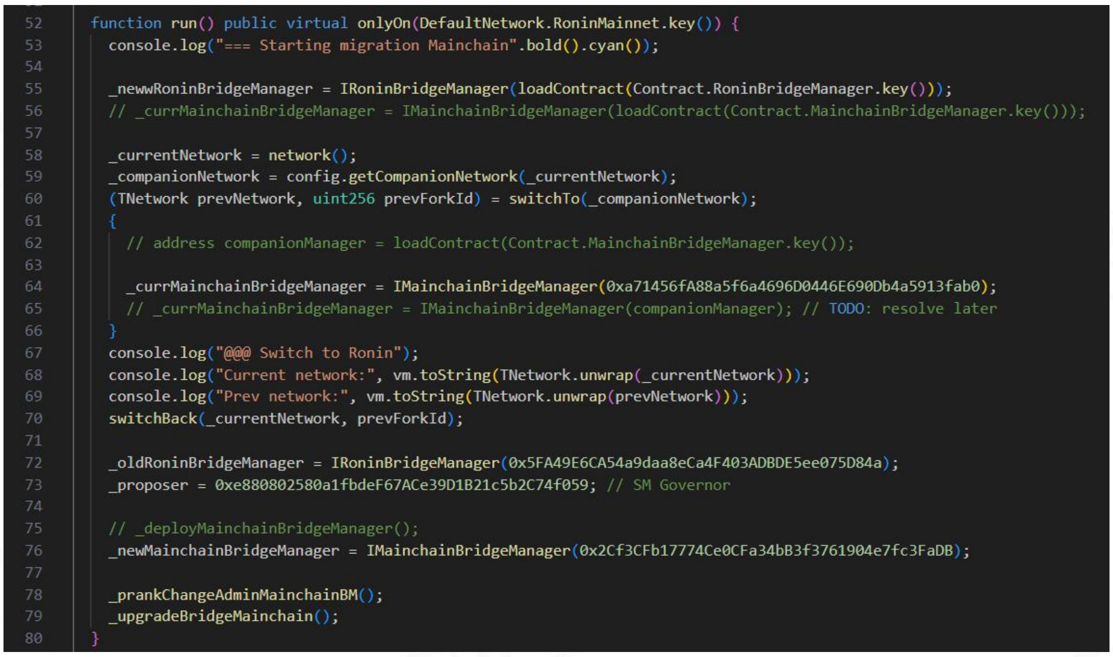
2. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Script Audit

2.1 20240716-p3 script code logic analysis

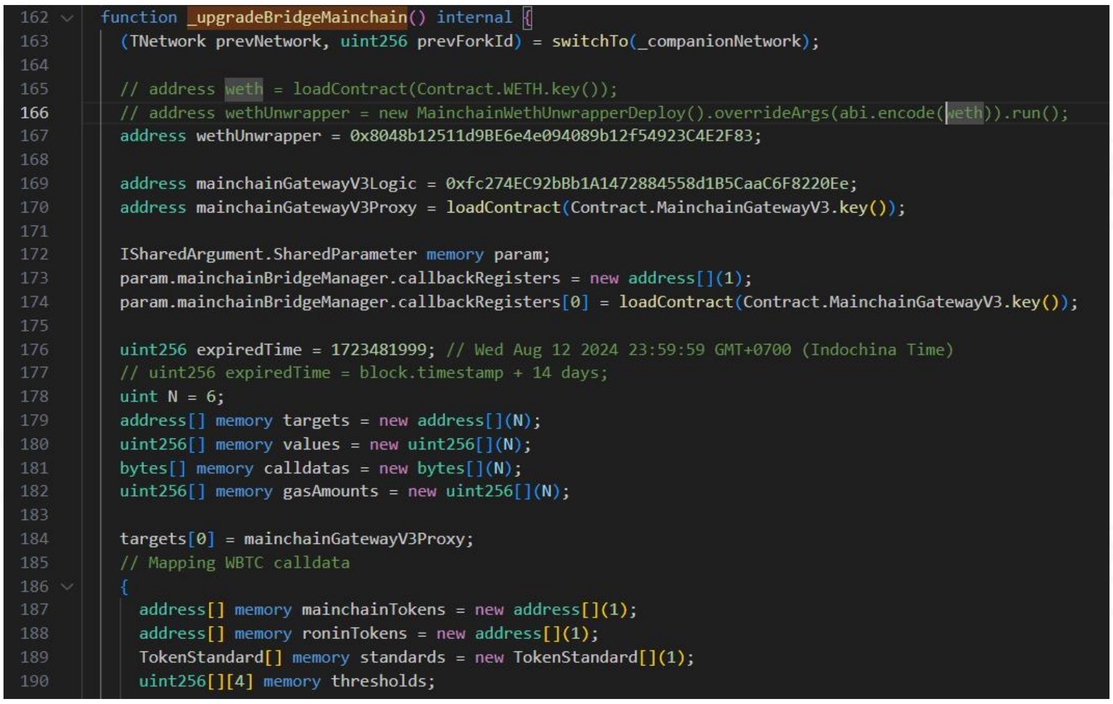
This script is related to the previous Ronin Bridge attack. The focus is on analyzing its logic to identify any other potential issues.



```
function run() public virtual onlyOn(DefaultNetwork.RoninMainnet.key()) {
  console.log("=== Starting migration Mainchain".bold().cyan());

  _newwRoninBridgeManager = IRoninBridgeManager(loadContract(Contract.RoninBridgeManager.key()));
  // _currMainchainBridgeManager = IMainchainBridgeManager(loadContract(Contract.MainchainBridgeManager.key()));

  _currentNetwork = network();
  _companionNetwork = config.getCompanionNetwork(_currentNetwork);
  (TNetwork prevNetwork, uint256 prevForkId) = switchTo(_companionNetwork);
  {
    // address companionManager = loadContract(Contract.MainchainBridgeManager.key());

    _currMainchainBridgeManager = IMainchainBridgeManager(0xa71456fA88a5f6a4696D0446E690Db4a5913fab0);
    // _currMainchainBridgeManager = IMainchainBridgeManager(companionManager); // TODO: resolve later
  }
  console.log("@@@ Switch to Ronin");
  console.log("Current network:", vm.toString(TNetwork.unwrap(_currentNetwork)));
  console.log("Prev network:", vm.toString(TNetwork.unwrap(prevNetwork)));
  switchBack(_currentNetwork, prevForkId);

  _oldRoninBridgeManager = IRoninBridgeManager(0x5FA49E6CA54a9daa8eCa4F403ADBDE5ee075D84a);
  _proposer = 0xe880802580a1fbdeF67ACe39D1B21c5b2C74f059; // SM Governor

  // _deployMainchainBridgeManager();
  _newMainchainBridgeManager = IMainchainBridgeManager(0x2Cf3CFb17774Ce0CFa34bB3f3761904e7fc3FaDB);

  _prankChangeAdminMainchainBM();
  _upgradeBridgeMainchain();
}
```

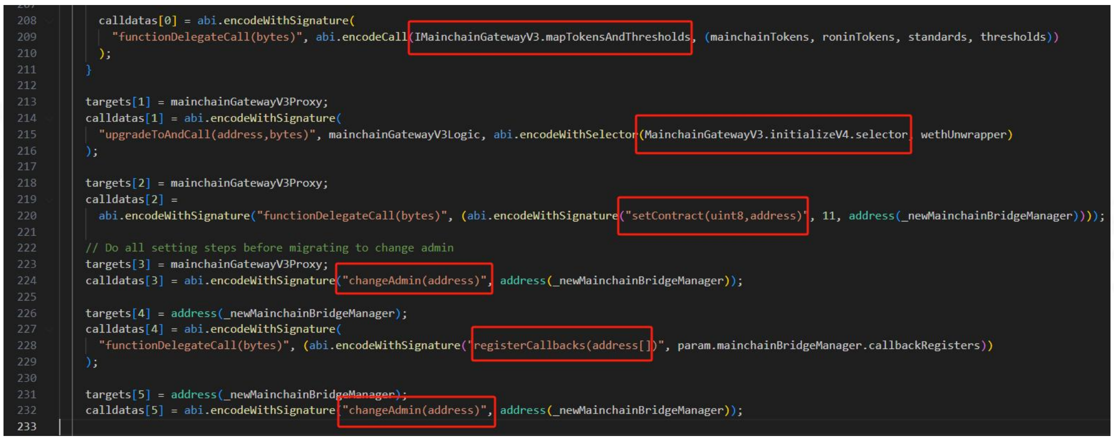
Figure 1 Run function screenshot

The `run` function serves as the entry point for the entire script and primarily handles the migration and upgrade of the bridge manager on the Ronin mainnet. First, it loads the new bridge manager contract instance on the Ronin mainnet. Then, it switches to the companion network (Ethereum mainnet) to load the current main chain bridge manager instance. After performing checks and outputting information, the function switches back to the Ronin network to set the addresses of the old and new bridge managers. Next, it calls `_prankChangeAdminMainchainBM()` to impersonate an admin and change the permissions of the main chain bridge manager. Finally, it invokes `_upgradeBridgeMainchain()` to complete the logic upgrade of the main chain bridge manager.



```
function _upgradeBridgeMainchain() internal {
  (TNetwork prevNetwork, uint256 prevForkId) = switchTo(_companionNetwork);

  // address weth = loadContract(Contract.WETH.key());
  // address wethUnwrapper = new MainchainWethUnwrapperDeploy().overrideArgs(abi.encode(weth)).run();
  address wethUnwrapper = 0x8048b12511d9BE6e4e094089b12f54923C4E2F83;

  address mainchainGatewayV3Logic = 0xfc274EC92bBb1A1472884558d1B5CaaC6F8220Ee;
  address mainchainGatewayV3Proxy = loadContract(Contract.MainchainGatewayV3.key());

  ISharedArgument.SharedParameter memory param;
  param.mainchainBridgeManager.callbackRegisters = new address[](1);
  param.mainchainBridgeManager.callbackRegisters[0] = loadContract(Contract.MainchainGatewayV3.key());

  uint256 expiredTime = 1723481999; // Wed Aug 12 2024 23:59:59 GMT+0700 (Indochina Time)
  // uint256 expiredTime = block.timestamp + 14 days;
  uint N = 6;
  address[] memory targets = new address[](N);
  uint256[] memory values = new uint256[](N);
  bytes[] memory calldatas = new bytes[](N);
  uint256[] memory gasAmounts = new uint256[](N);

  targets[0] = mainchainGatewayV3Proxy;
  // Mapping WBTC calldata
  {
    address[] memory mainchainTokens = new address[](1);
    address[] memory roninTokens = new address[](1);
    TokenStandard[] memory standards = new TokenStandard[](1);
    uint256[][4] memory thresholds;
```

Figure 2 _upgradeBridgeMainchain function screenshot(1/2)



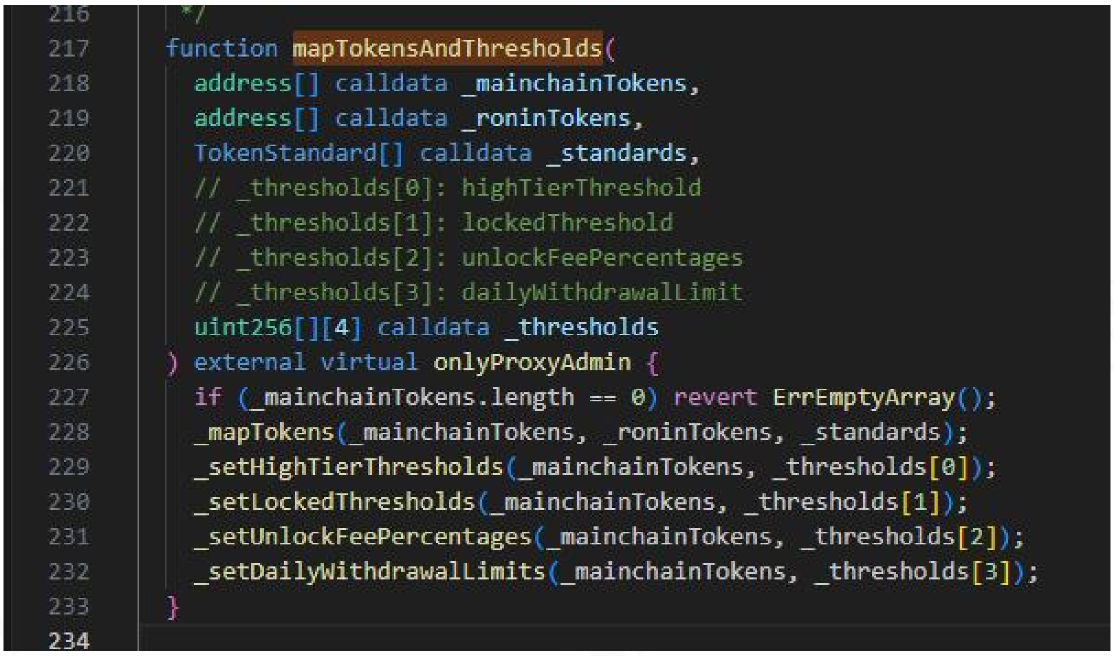
```
    calldatas[0] = abi.encodeWithSignature(
      "functionDelegateCall(bytes)", abi.encodeCall(IMainchainGatewayV3.mapTokensAndThresholds, (mainchainTokens, roninTokens, standards, thresholds))
    );
  }

  targets[1] = mainchainGatewayV3Proxy;
  calldatas[1] = abi.encodeWithSignature(
    "upgradeToAndCall(address,bytes)", mainchainGatewayV3Logic, abi.encodeWithSelector(MainchainGatewayV3.initializeV4.selector, wethUnwrapper)
  );

  targets[2] = mainchainGatewayV3Proxy;
  calldatas[2] =
    abi.encodeWithSignature("functionDelegateCall(bytes)", (abi.encodeWithSignature("setContract(uint8,address)", 11, address(_newMainchainBridgeManager))));

  // Do all setting steps before migrating to change admin
  targets[3] = mainchainGatewayV3Proxy;
  calldatas[3] = abi.encodeWithSignature("changeAdmin(address)", address(_newMainchainBridgeManager));

  targets[4] = address(_newMainchainBridgeManager);
  calldatas[4] = abi.encodeWithSignature(
    "functionDelegateCall(bytes)", (abi.encodeWithSignature("registerCallbacks(address[])", param.mainchainBridgeManager.callbackRegisters))
  );

  targets[5] = address(_newMainchainBridgeManager);
  calldatas[5] = abi.encodeWithSignature("changeAdmin(address)", address(_newMainchainBridgeManager));
```

Figure 3 _upgradeBridgeMainchain function screenshot(2/2)

The `_upgradeBridgeMainchain` function primarily handles the upgrade of the main chain bridge manager and creates a proposal containing several critical steps. Specifically, the function first switches to the companion network and deploys the new contract logic. Next, it sets the mapping rules for the WBTC token using `mapTokensAndThresholds`, upgrades and initializes the Mainchain Gateway V3 logic contract using `initializeV4`, sets the new bridge manager address, and changes the admin permissions. All these actions are bundled into a proposal, which includes steps such as `mapTokensAndThresholds` and `initializeV4`, and is eventually executed by calling the `relayProposal` function on the old bridge manager contract.

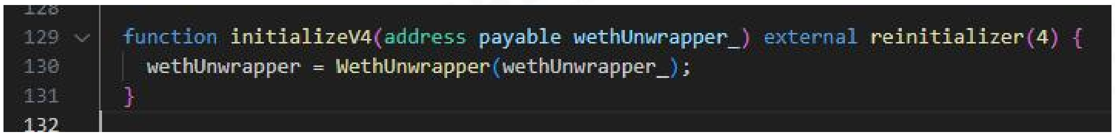
2.2 Check for uninitialized parameters on the chain

Based on the script(20240716-p3) code analysis, the migration script's execution logic can be combined with the on-chain code to check whether, aside from the missing weight updates, there are any other uninitialized parameters in the actual on-chain environment.



```
  */
  function mapTokensAndThresholds(
    address[] calldata _mainchainTokens,
    address[] calldata _roninTokens,
    TokenStandard[] calldata _standards,
    // _thresholds[0]: highTierThreshold
    // _thresholds[1]: lockedThreshold
    // _thresholds[2]: unlockFeePercentages
    // _thresholds[3]: dailyWithdrawalLimit
    uint256[][4] calldata _thresholds
  ) external virtual onlyProxyAdmin {
    if (_mainchainTokens.length == 0) revert ErrEmptyArray();
    _mapTokens(_mainchainTokens, _roninTokens, _standards);
    _setHighTierThresholds(_mainchainTokens, _thresholds[0]);
    _setLockedThresholds(_mainchainTokens, _thresholds[1]);
    _setUnlockFeePercentages(_mainchainTokens, _thresholds[2]);
    _setDailyWithdrawalLimits(_mainchainTokens, _thresholds[3]);
  }
```

Figure 4 mapTokensAndThresholds function screenshot

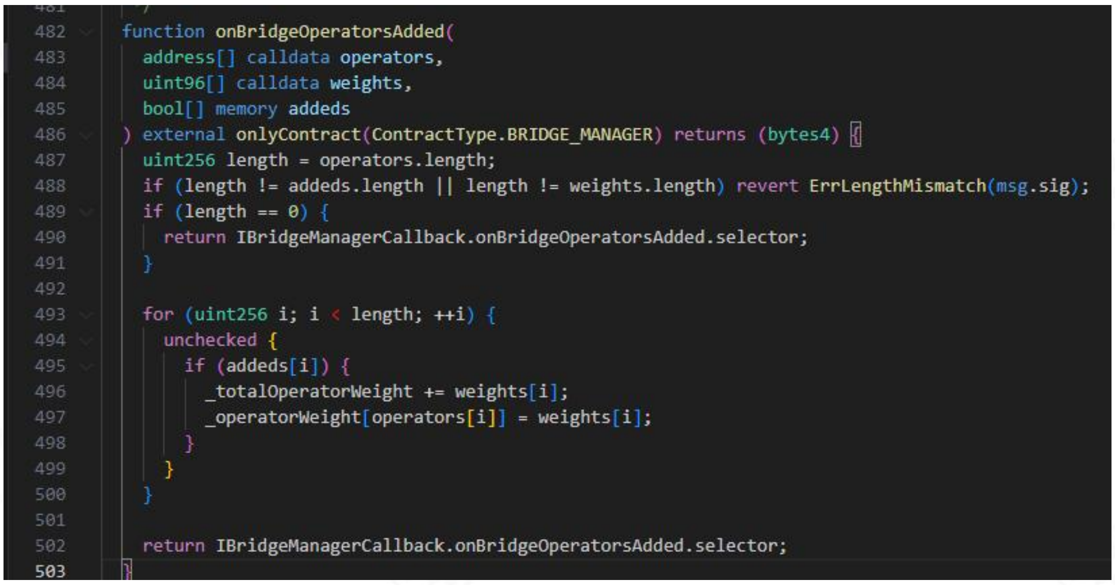


```
function initializeV4(address payable wethUnwrapper_) external reinitializer(4) {
  wethUnwrapper = WethUnwrapper(wethUnwrapper_);
}
```

Figure 5 initializeV4 function screenshot

The first call in the proposal is the mapTokensAndThresholds function, which initializes token mappings and related threshold parameters. It takes the token addresses on the main chain and Ronin network, token standards, and an array containing four types of thresholds. This function maps the tokens to the Ronin network and sets the high-tier thresholds, locked thresholds, unlock fee percentages, and daily withdrawal limits. The second call is the initializeV4 function, used to initialize the WETH address. Apart from these initialization operations, the proposal does not involve updates to other contract parameters.

Combining on-chain contract data and code analysis, it is confirmed that the contract currently lacks updates for _operatorWeight and _totalOperatorWeight. Now there are two ways to solve this problem: the first is to directly utilize the onBridgeOperatorsAdded function that already exists in the current contract to update the operator's weights; the second is to upgrade the contract and implement the initializeV5 function in the upgraded contract to update the weights.

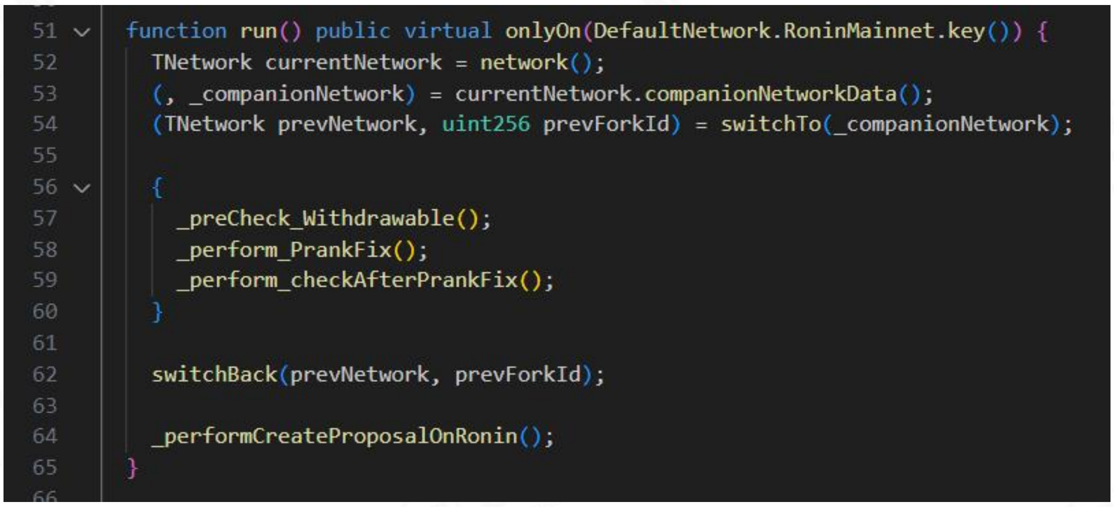


```
function onBridgeOperatorsAdded(
  address[] calldata operators,
  uint96[] calldata weights,
  bool[] memory addeds
) external onlyContract(ContractType.BRIDGE_MANAGER) returns (bytes4) {
  uint256 length = operators.length;
  if (length != addeds.length || length != weights.length) revert ErrLengthMismatch(msg.sig);
  if (length == 0) {
    return IBridgeManagerCallback.onBridgeOperatorsAdded.selector;
  }

  for (uint256 i; i < length; ++i) {
    unchecked {
      if (addeds[i]) {
        _totalOperatorWeight += weights[i];
        _operatorWeight[operators[i]] = weights[i];
      }
    }
  }

  return IBridgeManagerCallback.onBridgeOperatorsAdded.selector;
}
```

Figure 6 onBridgeOperatorsAdded function screenshot

2.3 20240807-ir-recover script code logic analysis

This section provides an analysis of the script logic used to fix the vulnerability, and based on this analysis, confirms the feasibility of the fix.



```
function run() public virtual onlyOn(DefaultNetwork.RoninMainnet.key()) {
  TNetwork currentNetwork = network();
  (, _companionNetwork) = currentNetwork.companionNetworkData();
  (TNetwork prevNetwork, uint256 prevForkId) = switchTo(_companionNetwork);

  {
    _preCheck_Withdrawable();
    _perform_PrankFix();
    _perform_checkAfterPrankFix();
  }

  switchBack(prevNetwork, prevForkId);

  _performCreateProposalOnRonin();
}
```

Figure 7 run function screenshot

The script logic is generally divided into four steps:

1. `_preCheck_Withdrawable`: Perform a preliminary check by simulating operations to cancel the pause and ensure that funds can be withdrawn normally.

2. `_perform_PrankFix`: Execute the fix by creating a proposal and calling the `onBridgeOperatorsAdded` function within the proposal to set the weights.

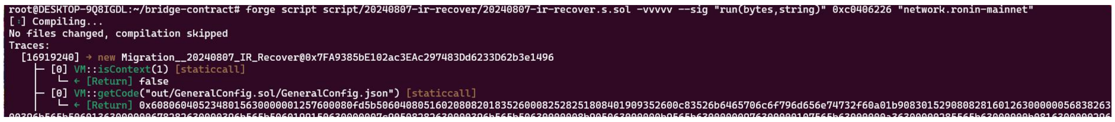
3. `_perform_checkAfterPrankFix`: Check after the fix to ensure that withdrawal operations can proceed normally.

4. `_performCreateProposalOnRonin`: Initiate a proposal on the Ronin chain.

2.4. Test the 20240807-ir-recover script

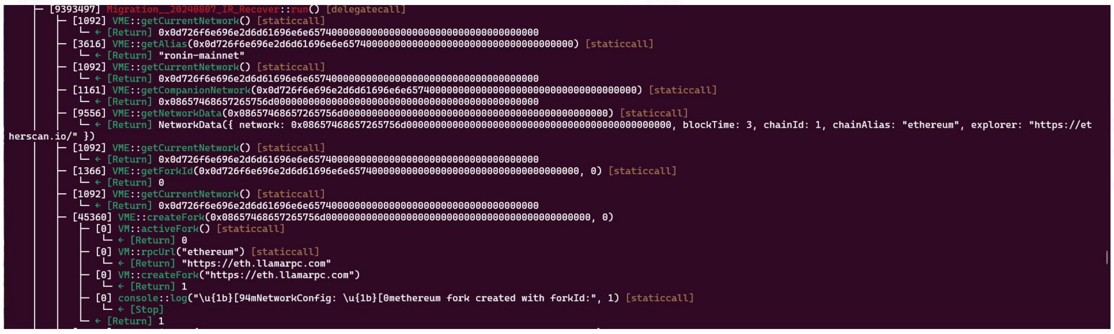
This section describes the execution of the fix script and assesses whether it effectively resolves the vulnerability based on the test results.

Script execution test screenshot:



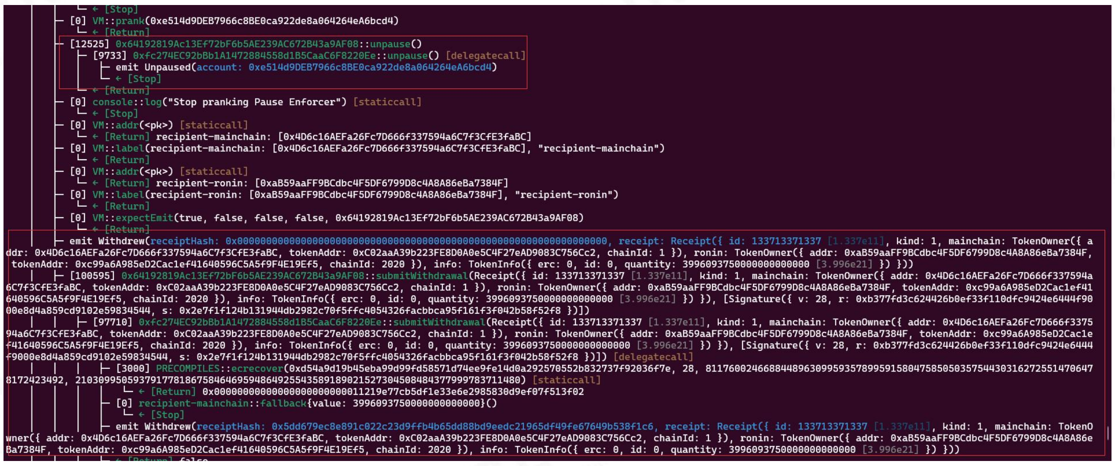
```
root@DESKTOP-9Q8IGDL:~/bridge-contract# forge script script/20240807-ir-recover/20240807-ir-recover.s.sol -vvvvv --sig "run(bytes,string)" 0xc0406226 "network.ronin-mainnet"
[:] Compiling...
No files changed, compilation skipped
Traces:
  [16919240] → new Migration__20240807_IR_Recover@0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
    ├─ [0] VM::isContext(1) [staticcall]
    │   └─ ← [Return] false
    ├─ [0] VM::getCode("out/GeneralConfig.sol/GeneralConfig.json") [staticcall]
    │   └─ ← [Return] 0x6080604052348015630000001257600080fd5b506040805160208082018352600082528251808401909352600c83526b6465706c6f796d656e74732f60a01b908301529080828160126300000056838263
```

Screenshot of the run function call in the script:



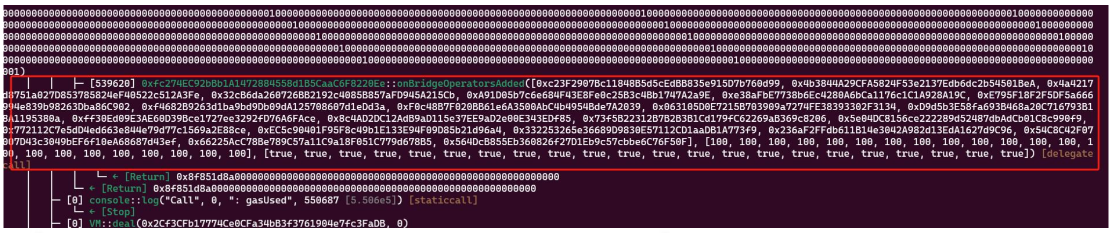
```
    ├─ [9393497] Migration__20240807_IR_Recover::run() [delegatecall]
    │   ├─ [1092] VME::getCurrentNetwork() [staticcall]
    │   │   └─ ← [Return] 0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000
    │   ├─ [3616] VME::getAlias(0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← [Return] "ronin-mainnet"
    │   ├─ [1092] VME::getCurrentNetwork() [staticcall]
    │   │   └─ ← [Return] 0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000
    │   ├─ [1161] VME::getCompanionNetwork(0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← [Return] 0x08657468657265756d000000000000000000000000000000000000000000000000
    │   ├─ [9556] VME::getNetworkData(0x08657468657265756d000000000000000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← [Return] NetworkData({ network: 0x08657468657265756d000000000000000000000000000000000000000000000000, blockTime: 3, chainId: 1, chainAlias: "ethereum", explorer: "https://et
herscan.io/" })
    │   ├─ [1092] VME::getCurrentNetwork() [staticcall]
    │   │   └─ ← [Return] 0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000
    │   ├─ [1366] VME::getForkId(0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000, 0) [staticcall]
    │   │   └─ ← [Return] 0
    │   ├─ [1092] VME::getCurrentNetwork() [staticcall]
    │   │   └─ ← [Return] 0x0d726f6e696e2d6d61696e6e6574000000000000000000000000000000000000
    │   ├─ [45360] VME::createFork(0x08657468657265756d000000000000000000000000000000000000000000000000, 0)
    │   │   ├─ [0] VM::activeFork() [staticcall]
    │   │   │   └─ ← [Return] 0
    │   │   ├─ [0] VM::rpcUrl("ethereum") [staticcall]
    │   │   │   └─ ← [Return] "https://eth.llamarpc.com"
    │   │   ├─ [0] VM::createFork("https://eth.llamarpc.com")
    │   │   │   └─ ← [Return] 1
    │   │   ├─ [0] console::log("\u{1b}[94mNetworkConfig: \u{1b}[0methereum fork created with forkId:", 1) [staticcall]
    │   │   │   └─ ← [Stop]
    │   │   └─ ← [Return] 1
```

Successful withdrawals were made using fake credentials, confirming that the vulnerability persisted after the suspension was lifted.



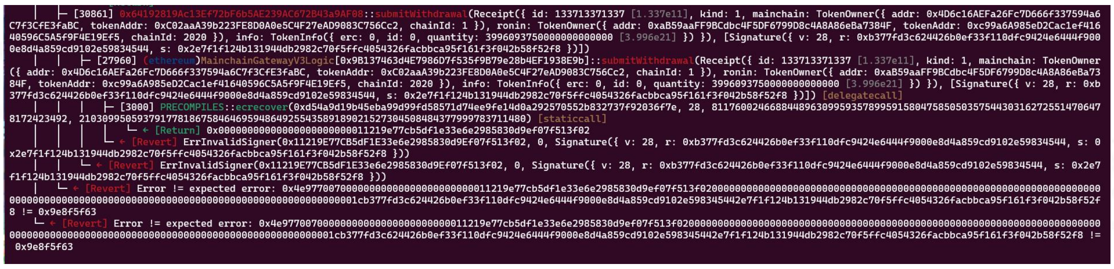
```
    │   │   └─ ← [Stop]
    │   ├─ [0] VM::prank(0xe514d9DEB7966c8BE0ca922de8a064264eA6bcd4)
    │   │   └─ ← [Return]
    │   ├─ [12525] 0x64192819Ac13Ef72bF6b5AE239AC672B43a9AF08::unpause()
    │   │   ├─ [9733] 0xfc274EC92bBb1A1472884558d1B5CaaC6F8220Ee::unpause() [delegatecall]
    │   │   │   ├─ emit Unpaused(account: 0xe514d9DEB7966c8BE0ca922de8a064264eA6bcd4)
    │   │   │   └─ ← [Stop]
    │   │   └─ ← [Return]
    │   ├─ [0] console::log("Stop pranking Pause Enforcer") [staticcall]
    │   │   └─ ← [Stop]
    │   ├─ [0] VM::addr(<pk>) [staticcall]
    │   │   └─ ← [Return] recipient-mainchain: [0x4D6c16AEFa26Fc7D666f337594a6C7f3CfE3faBC]
    │   ├─ [0] VM::label(recipient-mainchain: [0x4D6c16AEFa26Fc7D666f337594a6C7f3CfE3faBC], "recipient-mainchain")
    │   │   └─ ← [Return]
    │   ├─ [0] VM::addr(<pk>) [staticcall]
    │   │   └─ ← [Return] recipient-ronin: [0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F]
    │   ├─ [0] VM::label(recipient-ronin: [0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F], "recipient-ronin")
    │   │   └─ ← [Return]
    │   ├─ [0] VM::expectEmit(true, false, false, false, 0x64192819Ac13Ef72bF6b5AE239AC672B43a9AF08)
    │   │   └─ ← [Return]
    │   ├─ emit Withdrew(receiptHash: 0x0000000000000000000000000000000000000000000000000000000000000000, receipt: Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenOwner({ a
ddr: 0x4D6c16AEFa26Fc7D666f337594a6C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F,
 tokenAddr: 0xc99a6A985eD2Cac1ef41640596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }))
    │   ├─ [100595] 0x64192819Ac13Ef72bF6b5AE239AC672B43a9AF08::submitWithdrawal(Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenOwner({ addr: 0x4D6c16AEFa26Fc7D666f337594a
6C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F, tokenAddr: 0xc99a6A985eD2Cac1ef41
640596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }), [Signature({ v: 28, r: 0xb377fd3c624426b0ef33f110dfc9424e6444f90
00e8d4a859cd9102e59834544, s: 0x2e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 })])
    │   │   ├─ [97710] 0xfc274EC92bBb1A1472884558d1B5CaaC6F8220Ee::submitWithdrawal(Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenOwner({ addr: 0x4D6c16AEFa26Fc7D666f3375
94a6C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F, tokenAddr: 0xc99a6A985eD2Cac1e
f41640596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }), [Signature({ v: 28, r: 0xb377fd3c624426b0ef33f110dfc9424e6444
f9000e8d4a859cd9102e59834544, s: 0x2e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 })]) [delegatecall]
    │   │   │   ├─ [3000] PRECOMPILES::ecrecover(0xd54a9d19b45eba99d99fd58571d74ee9fe14d0a292570552b832737f92036f7e, 28, 81176002466884489630995935789959158047585050357544303162725514706478172423492, 21030995059379177818675846469594864925543589189021527304508484377999783711480) [staticcall]
    │   │   │   │   └─ ← [Return] 0x00000000000000000000000011219e77cb5df1e33e6e2985830d9ef07f513f02
    │   │   │   ├─ [0] recipient-mainchain::fallback{value: 3996093750000000000000}()
    │   │   │   │   └─ ← [Stop]
    │   │   │   ├─ emit Withdrew(receiptHash: 0x5dd679ec8e891c022c23d9ffb4b65dd88bd9eedc21965df49fe67649b538f1c6, receipt: Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenO
wner({ addr: 0x4D6c16AEFa26Fc7D666f337594a6C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86e
Ba7384F, tokenAddr: 0xc99a6A985eD2Cac1ef41640596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }))
```

The onBridgeOperatorsAdded function was successfully called through the proposal, and the weights were successfully updated.



```
0000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000000000000
0000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000
0000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000
0000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010
0000000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
001)
    │   │   ├─ [539620] 0xfc274EC92bBb1A1472884558d1B5CaaC6F8220Ee::onBridgeOperatorsAdded([0xc23F2907Bc11848B5d5cEdBB835e915D7b760d99, 0x4b3844A29CFA5824F53e2137Edb6dc2b54501BeA, 0x4a4217
d8751a027D853785824eF40522c512A3Fe, 0x32cB6da260726BB2192c4085B857aFD945A215Cb, 0xA91D05b7c6e684F43E8Fe0c25B3c4Bb1747A2a9E, 0xe38aFbE7738b6Ec4280A6bCa1176c1C1A928A19C, 0xE795F18F2F5DF5a666
994e839b98263Dba86C902, 0xf4682B9263d1ba9bd9Db09dA125708607d1eDd3a, 0xF0c48B7F020BB61e6A3500AbC4b4954Bde7A2039, 0x063105D0E7215B703909a7274FE38393302F3134, 0xD9d5b3E58fa693B468a20C716793B1
8A1195380a, 0xff30Ed09E3AE60D39Bce1727ee3292fD76A6FAce, 0x8c4AD2DC12AdB9aD115e37EE9aD2e00E343EDf85, 0x73f5B22312B7B2B3B1Cd179fC62269aB369c8206, 0x5e04DC8156ce222289d52487dbAdCb01C8c990f9,
0x772112C7e5dD4ed663e844e79d77c1569a2E88ce, 0xEC5c90401F95F8c49b1E133E94F09D85b21d96a4, 0x332253265e36689D9830E57112CD1aaDB1A773f9, 0x236aF2FFdb611B14e3042A982d13EdA1627d9C96, 0x54C8C42F07
007D43c3049bEF6f10eA68687d43ef, 0x66225AcC78Be789C57a11C9a18F051C779d678B5, 0x564DcB855Eb360826f27D1Eb9c57cbbe6C76F50F], [100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 1
00, 100, 100, 100, 100, 100, 100, 100, 100], [true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true]) [delegate
call]
    │   │   │   └─ ← [Return] 0x8f851d8a00000000000000000000000000000000000000000000000000000000
    │   │   └─ ← [Return] 0x8f851d8a00000000000000000000000000000000000000000000000000000000
    │   ├─ [0] console::log("Call", 0, ": gasUsed", 550687 [5.506e5]) [staticcall]
    │   │   └─ ← [Stop]
    │   ├─ [0] VM::deal(0x2Cf3CFb17774Ce0CFa34bB3f3761904e7fc3FaDB, 0)
```

After verifying the updated weights, another attempt to withdraw token failed, proving the vulnerability had been fixed.



```
    │   ├─ [30861] 0x64192819Ac13Ef72bF6b5AE239AC672B43a9AF08::submitWithdrawal(Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenOwner({ addr: 0x4D6c16AEFa26Fc7D666f337594a6
C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa7384F, tokenAddr: 0xc99a6A985eD2Cac1ef416
40596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }), [Signature({ v: 28, r: 0xb377fd3c624426b0ef33f110dfc9424e6444f900
0e8d4a859cd9102e59834544, s: 0x2e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 })])
    │   │   ├─ [27960] (ethereum)MainchainGatewayV3Logic[0x9B137463d4E7986D7f535f9B79e28b4EF1938E9b]::submitWithdrawal(Receipt({ id: 133713371337 [1.337e11], kind: 1, mainchain: TokenOwner
({ addr: 0x4D6c16AEFa26Fc7D666f337594a6C7f3CfE3faBC, tokenAddr: 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, chainId: 1 }), ronin: TokenOwner({ addr: 0xaB59aaFF9BCdbc4F5DF6799D8c4A8A86eBa73
84F, tokenAddr: 0xc99a6A985eD2Cac1ef41640596C5A5f9F4E19Ef5, chainId: 2020 }), info: TokenInfo({ erc: 0, id: 0, quantity: 3996093750000000000000 [3.996e21] }) }), [Signature({ v: 28, r: 0xb
377fd3c624426b0ef33f110dfc9424e6444f9000e8d4a859cd9102e59834544, s: 0x2e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 })]) [delegatecall]
    │   │   │   ├─ [3000] PRECOMPILES::ecrecover(0xd54a9d19b45eba99d99fd58571d74ee9fe14d0a292570552b832737f92036f7e, 28, 8117600246688448963099593578995915804758505035754430316272551470647
8172423492, 21030995059379177818675846469594864925543589189021527304508484377999783711480) [staticcall]
    │   │   │   │   └─ ← [Return] 0x00000000000000000000000011219e77cb5df1e33e6e2985830d9ef07f513f02
    │   │   │   └─ ← [Revert] ErrInvalidSigner(0x11219E77CB5dF1E33e6e2985830d9Ef07f513f02, 0, Signature({ v: 28, r: 0xb377fd3c624426b0ef33f110dfc9424e6444f9000e8d4a859cd9102e59834544, s: 0
x2e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 }))
    │   │   └─ ← [Revert] ErrInvalidSigner(0x11219E77CB5dF1E33e6e2985830d9Ef07f513f02, 0, Signature({ v: 28, r: 0xb377fd3c624426b0ef33f110dfc9424e6444f9000e8d4a859cd9102e59834544, s: 0x2e7
f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 }))
    │   └─ ← [Revert] Error != expected error: 0x4e97700700000000000000000000000011219e77cb5df1e33e6e2985830d9ef07f513f020000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000001cb377fd3c624426b0ef33f110dfc9424e6444f9000e8d4a859cd9102e598345442e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f
8 != 0x9e8f5f63
    └─ ← [Revert] Error != expected error: 0x4e97700700000000000000000000000011219e77cb5df1e33e6e2985830d9ef07f513f0200000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000001cb377fd3c624426b0ef33f110dfc9424e6444f9000e8d4a859cd9102e598345442e7f1f124b131944db2982c70f5ffc4054326facbbca95f161f3f042b58f52f8 !=
 0x9e8f5f63
```

3 Contract Audit

2.1 Proposal Security

[Ronin-01] Propose Function

Lines	RoninBridgeManager.sol#L35-57
Description	The `_proposeProposalStruct` function allows a governor to create a corresponding callback proposal. The proposal includes information such as the target contract to be called, callback data, and expiration time. Subsequent voting can be done using functions like `castProposalBySignatures`. The `propose` function can only be called by the Governor, and the expiration of the old proposal will be checked when creating a proposal. When the old proposal expires, the current proposal will replace the round of the old proposal, thus ensuring the safety of the round. Creating a proposal requires classifying the proposal's chainid, which affects whether the subsequent voting type is a signed vote or an unsigned vote.

```
  function _proposeProposalStruct(Proposal.ProposalDetail memory
proposal, address creator) internal virtual returns (uint256 round_) {
    uint256 chainId = proposal.chainId;
    if (chainId == 0) revert ErrInvalidChainId(msg.sig, 0,
block.chainid);
    proposal.validate(_proposalExpiryDuration);
    bytes32 proposalHash = proposal.hash();
    round_ = _createVotingRound(chainId);
    _saveVotingRound(vote[chainId][round_], proposalHash,
proposal.expiryTimestamp);
    if (round_ != proposal.nonce) revert
ErrInvalidProposalNonce(msg.sig);
    emit ProposalCreated(chainId, round_, proposalHash, proposal,
creator);
  }
```

[Ronin-02] CastVote Function

Lines	RoninBridgeManager.sol#L67-73
Description	The _castVote function allows voting on the proposal for the current round, but the voting process is divided into two types: signed voting and unsigned voting. Signed voting is typically used for cross-chain proposals, where the chainid is not 2020. In this type of voting, a signature is recorded during the voting process. Once the signature weight of the entire proposal accumulates sufficiently, the governor will handle the signature list on the corresponding chain of the proposal in the form of a RelayPropose. Unsigned voting is generally used for proposals on the current chain. When the weight accumulates sufficiently, the current chain's calldata will be invoked. After each governor votes, the contract records the signature hash for the corresponding proposal to prevent the reuse of signatures for the same proposal, ensuring the security of the signatures.

```
  function _castVote(
    Proposal.ProposalDetail memory proposal,
    Ballot.VoteType support,
    uint256 minimumForVoteWeight,
    uint256 minimumAgainstVoteWeight,
    address voter,
    Signature memory signature,
    uint256 voterWeight
  ) internal virtual returns (bool done) {
    uint256 chainId = proposal.chainId;
    uint256 round_ = proposal.nonce;
    ProposalVote storage _vote = vote[chainId][round_];
    if (_tryDeleteExpiredVotingRound(_vote)) {
      return true;
    }
    if (round[proposal.chainId] != round_) revert
ErrInvalidProposalNonce(msg.sig);
    if (_vote.status != VoteStatus.Pending) revert
ErrVoteIsFinalized();
    if (_voted(_vote, voter)) revert ErrAlreadyVoted(voter);
    _vote.voted[voter] = true;
    // Stores the signature if it is not empty
    if (signature.r > 0 || signature.s > 0 || signature.v > 0) {
```

```
    _vote.sig[voter] = signature;
  }
  emit ProposalVoted(_vote.hash, voter, support, voterWeight);
  uint256 _forVoteWeight;
  uint256 _againstVoteWeight;
  if (support == Ballot.VoteType.For) {
    _vote.forVoteds.push(voter);
    _forVoteWeight = _vote.forVoteWeight += voterWeight;
  } else if (support == Ballot.VoteType.Against) {
    _vote.againstVoteds.push(voter);
    _againstVoteWeight = _vote.againstVoteWeight += voterWeight;
  } else {
    revert ErrUnsupportedVoteType(msg.sig);
  }
  if (_forVoteWeight >= minimumForVoteWeight) {
    done = true;
    _vote.status = VoteStatus.Approved;
    emit ProposalApproved(_vote.hash);
    if (proposal.isAutoExecute()) {
      _tryExecute(_vote, proposal);
    }
  } else if (_againstVoteWeight >= minimumAgainstVoteWeight) {
    done = true;
    _vote.status = VoteStatus.Rejected;
    emit ProposalRejected(_vote.hash);
  }
}
```

[Ronin-03] Execute Function

Lines	RoninBridgeManager.sol#L187-189
Description	The `execute` function is called by the proposal's executor to attempt to execute the proposal, invoking the calldata on the target contract with the contract's authority. When executing a proposal call, the contract checks whether the hash of the currently passed proposal is consistent with the hash of the ledger vote, and checks the voting status and whether the caller is the executor of the proposal. These checks ensure the completeness of the proposal execution.

```
  function _executeWithCaller(Proposal.ProposalDetail memory proposal,
address caller) internal {
    bytes32 proposalHash = proposal.hash();
    ProposalVote storage _vote =
vote[proposal.chainId][proposal.nonce];
    if (_vote.hash != proposalHash) {
      revert ErrInvalidProposal(proposalHash, _vote.hash);
    }
    if (_vote.status != VoteStatus.Approved) revert
ErrProposalNotApproved();
    if (caller != proposal.executor) revert ErrInvalidExecutor();
    _tryExecute(_vote, proposal);
  }
```

[Ronin-04] DeleteExpired Function

Lines	RoninBridgeManager.sol#L208-213
Description	The deleteExpired function allows for manually deleting expired proposals. It is important to note that even if deleteExpired is not called, expired proposals can still be automatically deleted when creating new proposals. When creating a proposal, the contract will call the _tryDeleteExpiredVotingRound function to delete the expired proposal according to the current proposal status. This ensures the security of the contract round.

```
  function _tryDeleteExpiredVotingRound(ProposalVote storage
proposalVote) internal returns (bool isExpired) {
    isExpired = _getChainType() == ChainType.RoninChain &&
proposalVote.status == VoteStatus.Pending &&
proposalVote.expiryTimestamp <= block.timestamp;
    if (isExpired) {
      emit ProposalExpired(proposalVote.hash);
      for (uint256 _i; _i < proposalVote.forVoteds.length;) {
        delete proposalVote.voted[proposalVote.forVoteds[_i]];
        delete proposalVote.sig[proposalVote.forVoteds[_i]];
        unchecked {
          ++_i;
        }}
      for (uint256 _i; _i < proposalVote.againstVoteds.length;) {
        delete proposalVote.voted[proposalVote.againstVoteds[_i]];
        delete proposalVote.sig[proposalVote.againstVoteds[_i]];
        unchecked {
          ++_i;
        }
      }
      delete proposalVote.status;
      delete proposalVote.hash;
      delete proposalVote.againstVoteWeight;
      delete proposalVote.forVoteWeight;
      delete proposalVote.forVoteds;
      delete proposalVote.againstVoteds;
      delete proposalVote.expiryTimestamp;
    }
  }
```

[Ronin-05] RelayProposal Function

Lines MainchainBridgeManager.sol#L39-46

Description The relayProposal function can relay proposals that have already been passed on the Ronin chain. When the Ethereum-type proposal on the Ronin chain has sufficient weight to pass, the governor can collect the corresponding signatures in the proposal and use the relayProposal function to create and execute the corresponding proposal. relayProposal will only be executed when the proposal of the ronin chain is passed, and cannot be executed by voting alone in Ethereum, this design pattern ensures that voting on proposals is conducted on the ronin chain, making it easier to carry out and manage proposals.

```
  function _relayVotesBySignatures(
    Proposal.ProposalDetail memory _proposal,
    Ballot.VoteType[] calldata _supports,
    Signature[] calldata _signatures,
    bytes32 proposalHash
  ) internal {
    if (!(_supports.length > 0 && _supports.length ==
_signatures.length)) revert ErrLengthMismatch(msg.sig);
    bytes32 _forDigest =
ECDSA.toTypedDataHash(_proposalDomainSeparator(),
Ballot.hash(proposalHash, Ballot.VoteType.For));
    bytes32 _againstDigest =
ECDSA.toTypedDataHash(_proposalDomainSeparator(),
Ballot.hash(proposalHash, Ballot.VoteType.Against));
    address[] memory _forVoteSigners = new
address[](_signatures.length);
    address[] memory _againstVoteSigners = new
address[](_signatures.length);
    {
      uint256 _forVoteCount;
      uint256 _againstVoteCount;
      {
        address _signer;
        address _lastSigner;
        Ballot.VoteType _support;
        Signature calldata _sig;
        for (uint256 _i; _i < _signatures.length;) {
          _sig = _signatures[_i];
          _support = _supports[_i];
          if (_support == Ballot.VoteType.For) {
            _signer = ECDSA.recover(_forDigest, _sig.v, _sig.r, _sig.s);
            _forVoteSigners[_forVoteCount++] = _signer;
          } else if (_support == Ballot.VoteType.Against) {
            _signer = ECDSA.recover(_againstDigest, _sig.v, _sig.r,
_sig.s);
            _againstVoteSigners[_againstVoteCount++] = _signer;
          } else {
            revert ErrUnsupportedVoteType(msg.sig);
          }
          if (_lastSigner >= _signer) revert ErrInvalidOrder(msg.sig);
          _lastSigner = _signer;
          unchecked {
            ++_i;
          }
        }
      }
      assembly {
        mstore(_forVoteSigners, _forVoteCount)
        mstore(_againstVoteSigners, _againstVoteCount)
      }
    }
    ProposalVote storage _vote =
vote[_proposal.chainId][_proposal.nonce];
    uint256 _minimumForVoteWeight = _getMinimumVoteWeight();
    uint256 _totalForVoteWeight = _sumWeight(_forVoteSigners);
    if (_totalForVoteWeight >= _minimumForVoteWeight) {
      if (_totalForVoteWeight == 0) revert
ErrInvalidVoteWeight(msg.sig);
      _vote.status = VoteStatus.Approved;
      emit ProposalApproved(_vote.hash);
      _tryExecute(_vote, _proposal);
      return;
    }
    uint256 _minimumAgainstVoteWeight = _getTotalWeight() -
_minimumForVoteWeight + 1;
```

```
    uint256 _totalAgainstVoteWeight = _sumWeight(_againstVoteSigners);
    if (_totalAgainstVoteWeight >= _minimumAgainstVoteWeight) {
      if (_totalAgainstVoteWeight == 0) revert
ErrInvalidVoteWeight(msg.sig);
      _vote.status = VoteStatus.Rejected;
      emit ProposalRejected(_vote.hash);
      return;
    }
    revert ErrRelayFailed(msg.sig);
  }
```

2.2 Cross-chain Bridge Security

[Ronin-06] RequestWithdrawal Function

Lines	RoninGatewayV3.sol#L208-210
Description	The `requestWithdrawalFor` function can generate a cross-chain Receipt on the Ronin chain. If the user later obtains enough weighted signatures, they can use the corresponding cross-chain Receipt and signatures on Ethereum to withdraw funds. It is important to note that when withdrawing corresponding cross-chain funds from Ethereum, there is a fund threshold. If the withdrawal amount exceeds this threshold, a specific unlock account will need to call the `unlockWithdrawal` function to unlock the corresponding Receipt for withdrawal.The generation of the receipt includes the kind of Withdrawal type. This type of receipt can only be redeemed on Ethereum, which ensures the legitimacy of the receipt.

```
  function _requestWithdrawalFor(Transfer.Request calldata _request,
address _requester, uint256 _chainId) internal {
    _request.info.validate();
    _checkWithdrawal(_request);
    MappedToken memory _token = getMainchainToken(_request.tokenAddr,
_chainId);
    if (_request.info.erc != _token.erc) revert
ErrInvalidTokenStandard();
    _request.info.handleAssetIn(_requester, _request.tokenAddr);
    _storeAsReceipt(_request, _chainId, _requester, _token.tokenAddr);
  }
```

[Ronin-07] DepositFor Function

Lines RoninGatewayV3.sol#L146-148

Description The depositFor function allows the Operator to vote on the generated Receipt on Ethereum. Once the Receipt accumulates sufficient weight, the contract will send the corresponding cross-chain funds to the user recorded in the Receipt. When withdrawing funds, the chainid and kind types are checked to avoid multiple chains being reused. This cross-chain withdrawal requires enough operators to call and vote, which ensures the overall security of the receipt.

```
  function _depositFor(Transfer.Receipt memory receipt, address
operator, uint256 minVoteWeight) internal {
    uint256 id = receipt.id;
    receipt.info.validate();
    if (receipt.kind != Transfer.Kind.Deposit) revert
ErrInvalidReceiptKind();
    if (receipt.ronin.chainId != block.chainid) revert
ErrInvalidChainId(msg.sig, receipt.ronin.chainId, block.chainid);
    MappedToken memory token =
getMainchainToken(receipt.ronin.tokenAddr,
receipt.mainchain.chainId);
    if (!(token.erc == receipt.info.erc && token.tokenAddr ==
receipt.mainchain.tokenAddr)) {
      revert ErrInvalidReceipt();
    }
    IsolatedGovernance.Vote storage _proposal =
depositVote[receipt.mainchain.chainId][id];
    bytes32 _receiptHash = receipt.hash();
    VoteStatus status = _castIsolatedVote(_proposal, operator,
minVoteWeight, _receiptHash);
    emit DepositVoted(operator, id, receipt.mainchain.chainId,
_receiptHash);
    IBridgeTracking bridgeTrackingContract =
IBridgeTracking(getContract(ContractType.BRIDGE_TRACKING));
    if (status == VoteStatus.Approved) {
      _proposal.status = VoteStatus.Executed;
      receipt.info.handleAssetOut(payable(receipt.ronin.addr),
receipt.ronin.tokenAddr, IWETH(address(0)));
      bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteK
ind.Deposit, receipt.id);
      emit Deposited(_receiptHash, receipt);
    }
    bridgeTrackingContract.recordVote(IBridgeTracking.VoteKind.Depos
it, receipt.id, operator);
  }
```

[Ronin-08] SubmitWithdrawal Function

Lines MainchainGatewayV3.sol#L173-175

Description If a user has obtained enough weighted signatures, they can call the `submitWithdrawal` function and pass in the signature list to use the operator's signatures to verify the weight of the Receipt, thereby claiming the cross-chain funds. It is important to note that when withdrawing corresponding cross-chain funds from Ethereum, there is a fund threshold. If the withdrawal amount exceeds this threshold, a specific unlock account will need to call the `unlockWithdrawal` function to unlock the corresponding Receipt for withdrawal.

```
  function _submitWithdrawal(Transfer.Receipt calldata receipt,
Signature[] memory signatures) internal virtual returns (bool locked)
{
    uint256 id = receipt.id;
    uint256 quantity = receipt.info.quantity;
    address tokenAddr = receipt.mainchain.tokenAddr;
    receipt.info.validate();
    if (receipt.kind != Transfer.Kind.Withdrawal) revert
ErrInvalidReceiptKind();
    if (receipt.mainchain.chainId != block.chainid) {
      revert ErrInvalidChainId(msg.sig, receipt.mainchain.chainId,
block.chainid);
    }
    MappedToken memory token =
getRoninToken(receipt.mainchain.tokenAddr);
    if (!(token.erc == receipt.info.erc && token.tokenAddr ==
receipt.ronin.tokenAddr && receipt.ronin.chainId == roninChainId)) {
      revert ErrInvalidReceipt();
    }
    if (withdrawalHash[id] != 0) revert
ErrQueryForProcessedWithdrawal();
    if (!(receipt.info.erc == TokenStandard.ERC721
|| !_reachedWithdrawalLimit(tokenAddr, quantity))) {
      revert ErrReachedDailyWithdrawalLimit();
    }
    bytes32 receiptHash = receipt.hash();
    bytes32 receiptDigest = Transfer.receiptDigest(_domainSeparator,
receiptHash);
```

```
    uint256 minimumWeight;
    (minimumWeight, locked) = _computeMinVoteWeight(receipt.info.erc,
tokenAddr, quantity);
    {
      bool passed;
      address signer;
      address lastSigner;
      Signature memory sig;
      uint256 accumWeight;
      for (uint256 i; i < signatures.length; i++) {
        sig = signatures[i];
        signer = ECDSA.recover({ hash: receiptDigest, v: sig.v, r: sig.r,
s: sig.s });
        if (lastSigner >= signer) revert ErrInvalidOrder(msg.sig);
        lastSigner = signer;
        uint256 w = _getWeight(signer);
        if (w == 0) revert ErrInvalidSigner(signer, w, sig);
        accumWeight += w;
        if (accumWeight >= minimumWeight) {
          passed = true;
          break;
        }
      }
      if (!passed) revert ErrQueryForInsufficientVoteWeight();
      withdrawalHash[id] = receiptHash;
    }
    if (locked) {
      withdrawalLocked[id] = true;
      emit WithdrawalLocked(receiptHash, receipt);
      return locked;
    }
    _recordWithdrawal(tokenAddr, quantity);
    receipt.info.handleAssetOut(payable(receipt.mainchain.addr),
tokenAddr, wrappedNativeToken);
    emit Withdrew(receiptHash, receipt);
  }
```

[Ronin-09] UnlockWithdrawal Function

Lines	MainchainGatewayV3.sol#L180-205
Description	If the amount of cross-chain funds to be claimed by the user is large enough, the Receipt will be locked. At this time, the unlock account will need to call the `unlockWithdrawal` function to unlock the Receipt so that the user can claim the corresponding cross-chain funds. When performing the unlocking operation, a portion of the funds will be used as commission for the unlock account, which ensures the enthusiasm for the unlock account.

```
  function unlockWithdrawal(Transfer.Receipt calldata receipt) external
onlyRole(WITHDRAWAL_UNLOCKER_ROLE) {
    bytes32 _receiptHash = receipt.hash();
    if (withdrawalHash[receipt.id] != receipt.hash()) {
      revert ErrInvalidReceipt();
    }
    if (!withdrawalLocked[receipt.id]) {
      revert ErrQueryForApprovedWithdrawal();
    }
    delete withdrawalLocked[receipt.id];
    emit WithdrawalUnlocked(_receiptHash, receipt);
    address token = receipt.mainchain.tokenAddr;
    if (receipt.info.erc == TokenStandard.ERC20) {
      TokenInfo memory feeInfo = receipt.info;
      feeInfo.quantity = _computeFeePercentage(receipt.info.quantity,
unlockFeePercentages[token]);
      TokenInfo memory withdrawInfo = receipt.info;
      withdrawInfo.quantity = receipt.info.quantity - feeInfo.quantity;
      feeInfo.handleAssetOut(payable(msg.sender), token,
wrappedNativeToken);
      withdrawInfo.handleAssetOut(payable(receipt.mainchain.addr),
token, wrappedNativeToken);
    } else {
      receipt.info.handleAssetOut(payable(receipt.mainchain.addr),
token, wrappedNativeToken);
    }
    emit Withdrew(_receiptHash, receipt);
  }
```

[Ronin-10] RequestDepositFor Function

Lines MainchainGatewayV3.sol#L156-158

Description The requestDepositFor function can generate a cross-chain Receipt on Ethereum. Subsequently, the operator on the Ronin chain can vote on the Receipt. Once the accumulated weight of the votes is sufficient, the contract will send the corresponding cross-chain funds to the user recorded in the Receipt. This withdrawal method is convenient for users to make manual proposals, but it requires sufficient security protection for the operator's signature.

```
 function _requestDepositFor(Transfer.Request memory _request, address
_requester) internal virtual {
    MappedToken memory _token;
    address _roninWeth = address(wrappedNativeToken);
    _request.info.validate();
    if (_request.tokenAddr == address(0)) {
      if (_request.info.quantity != msg.value) revert
ErrInvalidRequest();
      _token = getRoninToken(_roninWeth);
      if (_token.erc != _request.info.erc) revert
ErrInvalidTokenStandard();
      _request.tokenAddr = _roninWeth;
    } else {
      if (msg.value != 0) revert ErrInvalidRequest();
      _token = getRoninToken(_request.tokenAddr);
      if (_token.erc != _request.info.erc) revert
ErrInvalidTokenStandard();
      _request.info.handleAssetIn(_requester, _request.tokenAddr);
      if (_roninWeth == _request.tokenAddr) {
        wrappedNativeToken.approve(address(wethUnwrapper),
_request.info.quantity);
        wethUnwrapper.unwrap(_request.info.quantity);
      }
    }
    uint256 _depositId = depositCount++;
    Transfer.Receipt memory _receipt =
_request.into_deposit_receipt(_requester, _depositId,
_token.tokenAddr, roninChainId);
```

```
    emit DepositRequested(_receipt.hash(), _receipt);
  }
```

4 Appendix

4.1 Vulnerability Assessment Metrics and Status in Smart Contracts

4.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

4.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

4.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

4.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

4.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	SPL Token Standards
		Visibility Specifiers
		Lamport Check
		Account Check
		Signer Check
		Program Id Check
		Deprecated Items
		Redundant Code
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

4.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

4.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com