SALUS SECURITY

AUG 2024

CODE SECURITY ASSESSMENT

CYCLE NETWORK

Overview

Project Summary

- Name: Cycle Network
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/RollNA/cycle-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Cycle Network
Version	v2
Type	Solidity
Dates	Aug 7 2024
Logs	Jul 29 2024, Aug 7 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	2
Total informational issues	1
Total	5

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Non-compatible with btc ecosystem	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Implementation contract could be initialized by anyone	Low	Business Logic	Acknowledged
4	Lack of input validation in delChain()	Low	Business Logic	Resolved
5	Redundant Code	Informational	Redundancy	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Non-compatible with btc ecosystem

Severity: Medium	Category: Business Logic
Target: - contracts/PolygonZkEVMBridge.sol	

Description

In PolygonZkEVMBridge contract, there are two ways to transfer native tokens, `bridgeAsset()` and `bridgeMessage()`.

These functions can work well based on one assumption: All related chains' native token is Ether. The vulnerability is that the assumption is incorrect, and the Cycle chain is expected to connect with btc ecosystems, such as the BTC chain, or Layer2 chain in the BTC ecosystem. And in BTC or related Layer2, the native token is not Ether. This will break current ether transfer function.

contracts/PolygonZkEVMBridge.sol:L266-L302

```
function bridgeMessage(
    uint32 destinationNetwork,
    address destinationAddress,
    bool forceUpdateGlobalExitRoot,
    bytes calldata metadata
) external payable ifNotEmergencyState {
   ...
   _deposit(
       getLeafValue(
           _LEAF_TYPE_MESSAGE,
           networkID,
           msg.sender,
           destinationNetwork,
           destinationAddress,
           msg.value,
           keccak256(metadata),
           networkID
       )
   );
   ...
}
```

contracts/PolygonZkEVMBridge.sol:L440-L477

```
function claimMessage(
    bytes32[_DEPOSIT_CONTRACT_TREE_DEPTH] calldata smtProof,
    uint32 fromNetworkID,
    uint32 index,
    bytes32 mainnetExitRoot,
    bytes32 rollupExitRoot,
    uint32 originNetwork,
    address originAddress,
    uint32 destinationNetwork,
    address destinationAddress,
    uint256 amount,
    bytes calldata metadata
) external ifNotEmergencyState {
   …
   // Execute message
   // Transfer ether
   /* solhint-disable avoid-low-level-calls */
   (bool success, ) = destinationAddress.call{value: amount}(
       abi.encodeCall(
           IBridgeMessageReceiver.onMessageReceived,
           (originAddress, originNetwork, metadata)
       )
   );
   …
}
```

Recommendation

Add some input limitations to avoid these scenarios.

Status

This issue has been resolved by the team with commit c0af850.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contracts/PolygonZkEVM.sol - contracts/PolygonZkEVMBridge.sol	

Description

In PolygonZkEVM and other contracts, there exists a privileged owner role. This owner has authority over key operations such as updating critical parameters and choose the trusted sequencer and aggregator.

If the owner's private key were compromised, an attacker could exploit this access to withdraw all tokens.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team. The team has stated that after deployment the admin role will be migrated to a multi-sig wallet.

3. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - contracts/PolygonZkEVMBridge.sol - contracts/PolygonZkEVMChains.sol - contracts/periphery/ERC20Bridge.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the sequenceBatches() and other functions in `` `PolygonZkEVMBridge/PolygonZkEVMChains/ERC20Bridge` ``'s implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the `` `PolygonZkEVMBridge/PolygonZkEVMChains/ERC20Bridge` `` contract to automatically lock it when it is deployed.

Status

This issue has been acknowledged by the team.

4. Lack of input validation in delChain()	
Severity: Low	Category: Business Logic
Target: - contracts/PolygonZkEVMChains.sol	

Description

The PolygonZkEVMChains contract aims to manage the chains which will be connected with the Cycle chain.

The admin can add or delete one chain via addChain()/delChain(). The vulnerability is that we don't check the input parameter `_chainID` is one existing supported chain. If we delete one unsupported chain, one existing supported chain will be deleted.

contracts/PolygonZkEVMChains.sol:L45-L57

```
function delChain(uint64 _batchNum, uint256 _chainID) external onlyAdmin() {
    uint i = 0;
    for(;i < allChainIDs.length; i++){
        if (allChainIDs[i] == _chainID){
            break;
        }
    }
    for(;i < allChainIDs.length-1; i++){
        allChainIDs[i] = allChainIDs[i+1];
    }
    allChainIDs.pop();
    emit UpdateChainIDs(_batchNum, allChainIDs);
}
```

Recommendation

Add some input parameter check to make sure the `_chainID` is one existing supported chainID.

Status

This issue has been resolved by the team with commit a43bfff.

2.3 Informational Findings

5. Redundant Code	
Severity: Informational	Category: Redundancy
Target: - contracts/PolygonZkEVMChains.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following contracts are not being utilized:

contracts/PolygonZkEVMChains.sol:L9-L13

```
contract PolygonZkEVMChains is
   OwnableUpgradeable,
   EmergencyManager,
   IPolygonZkEVMErrors
{
    ...
}
```

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team with commit a43bfff.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 166db5d:

File	SHA-1 hash
PolygonZkEVMChains.sol	be557f5f7fd51ff77c1a47bf75cf2f92d4ed08b6
PolygonZkEVMBridge.sol	eaed5ae47202adcf945662ecfab08cd6179f23f6
PolygonZkEVM.sol	34201f5b21296f6b3a517ebb7152ef3ee19fa520
PolygonZkEVMGlobalExitRoot.sol	e3e792dfb642f625e0542c6cedfd22ecdd28fe5a
PolygonZkEVMGlobalExitRootL2.sol	907a4a0b6c6e7436f23e9eb1450d64e6843e6055
PolygonZkEVMTimelock.sol	e2220000add0c3a82324b2fffef1e615a6458a59
PolygonZkEVMDeployer.sol	87878d7b9a1dfc1c8844587dd45f796e36181267
DepositContract.sol	56b982040d0bfa312d681c6369eee431ba9c2a37
EmergencyManager.sol	e5535f95b992de4b8d974bee0b1d194718d33be5
GlobalExitRootLib.sol	cbe88865963252964569aeb61e78342265624d38
TokenWrapped.sol	39aca0b51e7ce3c35f169a7ebc1012c0559ba31b
PolygonZkEVMUpgraded.sol	f76634d0608f55ee47411df1713312e5a652e7ae
ERC20Bridge.sol	f23a56a229dbb7821c0e56ac4e91af2212f639c3
FflonkVerifier.sol	0a51c78661c1bcf5333753012e77d46d83568aec