SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

CYCLE NETWORK

Overview

Project Summary

- Name: Cycle Network - node
- Platform: Cycle Network
- Language: Go
- Repository:
    - https://github.com/RollNA/cycle-node
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Cycle Network - node
Version	v2
Type	Solidity
Dates	Sep 03 2024
Logs	Aug 23 2024; Sep 03 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	4
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	Removal of force batches functionality may lead to transaction censorship	Low	Business Logic	Mitigated
2	Service downtime during chainID update	Low	Business Logic	Acknowledged
3	Unordered map causes chainID implementation to differ from expected behavior	Low	Business Logic	Resolved
4	Multiple data builds cause unnecessary overhead	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. Removal of force batches functionality may lead to transaction censorship

Severity: Low	Category: Business Logic
Target: - All	

Description

Typically, L2 systems allow users to send transactions to L1, where they are included in a forced batch. The trusted sequencer is then required to incorporate these forced batches in future sequences to maintain its trusted status. Otherwise, users could prove that censorship is occurring, resulting in the revocation of the sequencer's trusted status.

However, in Cycle, this functionality has been removed, which leaves users without the ability to prevent censorship by the sequencer.

Recommendation

Consider adapting the force batches functionality into Cycle.

Status

This issue has been acknowledged by the team. The team has stated that cycle uses a new centralized sequencer that replaces the functionality of forced_batch.

2. Service downtime during chainID update

Severity: Low	Category: Business Logic

Target:
- cycle-node/sequencer/finalizer.go

Description

Cycle supports interactions with multiple L1 and L2 chains for rollup processing and allows the addition of new chains via the PolygonZkEVMChains contract.

cycle-node/sequencer/finalizer.go:L351-L366

```
if f.batch.batchNumber > 0 {
    _, latestVer, err := f.dbManager.GetChainIDsByBatchNumber(ctx,
f.batch.batchNumber-1)
    if err != nil {
        log.Errorf("get chainIds version failed, finalizer exit")
        return
    }
    _, currVer, err := f.dbManager.GetChainIDsByBatchNumber(ctx, f.batch.batchNumber)
    if err != nil {
        log.Errorf("get chainIds version failed, finalizer exit")
        return
    }
    if latestVer != currVer && f.currentChainIdsVer != currVer {
        log.Infof("chainIds version: %v on changing, finalizer exit", currVer)
        return
    }
}
```

However, when a new chain is added, the chainid version increments, causing the sequencer to reject new transactions. This results in a temporary inability to process transactions until the chain is restarted and the configuration is updated

Recommendation

Consider automatically updating the configuration upon detecting a change in the chainID version.

Status

This issue has been acknowledged by the team.

3. Unordered map causes chainID implementation to differ from expected behavior

Severity: Low	Category: Business Logic
Target: - cycle-node/aggregator/aggregator.go	

Description

cycle-node/aggregator/aggregator.go:L809-L815

```
var chainId uint64
for k := range a.defaultChainInfo {
    chainId = k
    break
}
// now that we wait a batch to prove until it is verified on every chain, so any chainId
will be fine, we just take the first one.
lastVerifiedBatch, err := a.State.GetLastVerifiedBatch(ctx, int(chainId), nil)
```

In the getAndLockBatchToProve() function, an attempt is made to iterate over the map and use the first key as the initial chainID. However, since the map is unordered, the actual chainID retrieved is random, which does not align with the expected behavior described in the comments.

Recommendation

Consider using a.l1ChainId to retrieve lastVerifiedBatch instead of iterating over the map:

```
lastVerifiedBatch, err := a.State.GetLastVerifiedBatch(ctx, a.l1ChainId, nil)
```

Status

The team has resolved this issue in commit 6319e47.

4. Multiple data builds cause unnecessary overhead

Severity: Low	Category: Business Logic
Target: - cycle-node/aggregator/utils.go	

Description

cycle-node/aggregator/utils.go:L63-L75

```
func (e Ethermans) RangeBuildTrustedVerifyBatchesTxData(
        lastVerifiedBatch,
        newVerifiedBatch uint64,
        inputs *ethmanTypes.FinalProofInputs,
        fn func(chainID uint64, to *common.Address, data []byte, err error) bool) error {
        for id, em := range e {
                to, data, err := em.BuildTrustedVerifyBatchesTxData(lastVerifiedBatch,
newVerifiedBatch, inputs)
                if !fn(id, to, data, err) {
                        return fmt.Errorf("range exit")
                }
        }
        return nil
}
```

The aggregator repeatedly builds TrustedVerifyBatchesTxData for each extend layer in a loop. However, since the TrustedVerifyBatch sent to each chain uses identical transaction data, this results in unnecessary overhead.

Recommendation

There's no need to repeatedly build tx data within the for loop.

Status

The team has resolved this issue in commit f907979.

2.3 Informational Findings

5. Redundant Code

Severity: Informational	Category: Redundancy

Target:
- cycle-node/jsonrpc/endpoints_zkevm.go
- cycle-node/state/pgstatestorage.go

Description

Unused code should be removed before deploying the contract to mainnet.

1. In the GetLatestGerByChain() function, the fullTx parameter is declared but not utilized.
cycle-node/jsonrpc/endpoints_zkevm.go:L188

```
GetLatestGerByChain(chainId int, fullTx bool) (interface{}, types.Error)
```

2. There are unresolved debug outputs present in the GetTxsOlderThanNL1Blocks() function.
cycle-node/state/pgstatestorage.go:L121-L136

```
err = e.QueryRow(ctx, getBatchNumByBlockNumFromVirtualBatch, blockNum,
chainId).Scan(&batchNum)
    if errors.Is(err, pgx.ErrNoRows) {
        fmt.Println("0000000000002")
        return nil, ErrNotFound
    } else if err != nil {
        fmt.Println("222222222222")
        return nil, err
    }
    rows, err := e.Query(ctx, getTxsHashesBeforeBatchNum, batchNum)
    if errors.Is(err, pgx.ErrNoRows) {
        fmt.Println("00000000000003")
        return nil, ErrNotFound
    } else if err != nil {
        fmt.Println("3333333333333333")
        return nil, err
    }
```

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit f907979.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit ebe7caf:

File	SHA-1 hash
aggregator/aggregator.go	03c46a6475cd01036e9ea52c88175045b1c18164
aggregator/utils.go	8e124919d6de69b91e914e907fbe89f140ccabbc
cmd/approve.go	3340241dce46ee0ea6833f20781b873d0f381a35
cmd/dumpstate.go	a54dd458657892645b7f4372da18f20d641ecd52
cmd/run.go	513935b72aaed5ec030554ded143f38e51637329
config/config.go	b5b5ca63312b2144188820e429ce6ce72e575693
config/default.go	b63c0b05b065aa8e6162bf9ad742c9041359ef71
config/network.go	4d234ab520a40e9ea4e1d28940c8751d93e0056e
etherman/etherman.go	88485d3d9e63012f220050c8c5be231199da68c2
ethtxmanager/config.go	fe14e817b965275702c44b0e05e40737c6280805
ethtxmanager/ethtxmanager.go	8c029449ae9cbd790211257055c66f4697f4c4b4
ethtxmanager/pgstorage.go	d2902e6c06f47f6a366e6dcbd3b63a96486dbc90
jsonrpc/client/zkevm.go	f4fc89491a834c06b7934d2a044330704f83e1e1
jsonrpc/endpoints_zkevm.go	17d15c66f12c097ebd77ac79bd92369aa74a08e1
jsonrpc/types/codec.go	8f43df72b966083fbdc1501170a7c22b4b32bfa2
sequencer/closingsignalsmanager.go	174f8bb670e1159810c75ad212db40e10886d5ee
sequencer/config.go	f540345df43239df2c6a70e46d6d5dfb94af917e
sequencer/dbmanager.go	8dac4ba84ea978cb14a3ef6d38798fcb2398e1b9
sequencer/finalizer.go	18043f10e9d8438195d397ce4bff18871184468d
sequencer/sequencer.go	69c367192bbe333d0ab89cec3a4ee4d171163870
sequencesender/config.go	3d0dd929963ca8e06fe27f4bb7c3e7ac7bf97757
sequencesender/sequencesender.go	00a89bfcccdd399734739078ee8e5269f41f03dd
state/chainids.go	1f98091fc3d8763962c909ad23830b5747d8e258

state/genesis.go	1cf53393a9ee1a020d75598ce92b340ae4d1c866
state/pgstatestorage.go	c991ccf2ffcd2c76b8fdcde1a92140704ded427e
state/proof.go	6effdc251b43de22759a6f72ebf6f04b916ed0b1
synchronizer/config.go	63aca6ddd40f8d516241090ea33e0a56bdfda92e
synchronizer/l1_rollup_info_consumer.go	d6fff58140ddfe796f7fb2adb90646ab367ecd92
synchronizer/synchronizer.go	620aaa0ceb7ce5701cbbf7d8434ef70a8849a294