Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.03.04, the SlowMist security team received the morph-l2 team's security audit application for morph-l2/go-ethereum, 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" 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 party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

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:

NO.	Audit Items	Result
1	Design Logic Audit	Some Risks
2	Others	Some Risks
3	State Consistency Audit	Passed
4	Failure Rollback Audit	Passed
5	Unit Test Audit	Passed
6	Integer Overflow Audit	Passed
7	Parameter Verification Audit	Passed
8	Error Unhandle Audit	Some Risks

NO.	Audit Items	Result
9	Boundary Check Audit	Passed
10	SAST	Some Risks

3 Project Overview

3.1 Project Introduction

Morph is an innovative force reshaping the consumer blockchain landscape for practical, everyday use. At the core of Morph is a revolutionary approach to Ethereum Layer 2 scalability, harnessing the power of advanced rollup technology.

3.2 Coverage

Target Code and Revision:

https://github.com/morph-l2/go-ethereum/tree/eip4844

Initial audit commit:523eaf60a40ed0f6ea689101aadb9c911bb712dd

Final audit commit: c7c40645201fa0ba87724335ca45a5ec0514d6cf

```
consensus/l2/consensus.go
core/block_validator.go ok
core/blockchain.go ok
core/blockchain_l2.go ok
core/chain_makers.go  ok
core/rawdb/accessors_chain.go
core/rawdb/accessors_l1_msg.go
core/rawdb/accessors_rollup_batch.go
core/rawdb/database.go
core/rawdb/schema.go
core/rawdb/table.go
core/tx_pool.go
core/types/block.go
core/types/gen_batch.go
core/types/gen_batch_sig.go
core/types/gen_header_json.go
core/types/l1_message_tx.go
```

```
core/types/rollup_batch.go
core/types/transaction.go
eth/api.go  ok
eth/api_backend.go ok
eth/backend.go ok
eth/catalyst/api_types.go
eth/catalyst/gen_l2_ed.go
eth/catalyst/gen_l2_sd.go
eth/catalyst/gen_l2blockparams.go
eth/catalyst/l2_api.go
ethclient/authclient/client.go
ethclient/authclient/engine.go
ethclient/ethclient.go
ethclient/gethclient/gethclient.go
miner/miner.go
miner/worker.go
miner/worker_l2.go
```

3.3 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Use of codebases with security risks	SAST	Medium	Fixed
N2	Redundant codes	Others	Suggestion	Fixed
N3	Partial error handling for fillTransactions function Interruptions	Design Logic Audit	Medium	Fixed
N4	Unused formal parameter exists	Others	Information	Acknowledged
N5	Direct comparison of errors	Design Logic Audit	Suggestion	Fixed
N6	Mixed value and pointer receivers	Others	Low	Acknowledged
N7	Reserved word used as name	Others	Information	Fixed
N8	Use of obsolete structures	Others	Low	Acknowledged

NO	Title	Category	Level	Status
N9	Error not handled	Error Unhandle Audit	Suggestion	Acknowledged
N10	Recommendation to evaluate version synchronization for go-ethereum Fork	Others	Suggestion	Acknowledged

4 Findings

4.1 Visibility Description

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

4.2 Vulnerability Summary

[N1] [Medium] Use of codebases with security risks

Category: SAST

Content

- https://github.com/morph-l2/go-ethereum

```
Dependencies go:github.com/btcsuite/btcd:v0.20.1-beta Vulnerable, security version
v0.23.3
https://devhub.checkmarx.com/cve-details/CVE-2022-44797/
https://devhub.checkmarx.com/cve-details/CVE-2022-39389/

Dependency go:golang.org/x/crypto:v0.17.0 Vulnerable, security version v0.21.0
https://devhub.checkmarx.com/cve-details/CVE-2023-42818/
```

Solution

Update to secure version

Status

Fixed; Fixed in https://github.com/morph-l2/go-ethereum/pull/56

[N2] [Suggestion] Redundant codes

Category: Others

Content

- eth/catalyst/l2_api.go

The `extractSkippedQueueIndexes` function is not used.

- core/rawdb/accessors_rollup_batch.go

The `ReadBatchIndexByHash` function is not used.

- core/types/block.go

`EmptyHash` variable not used.

- consensus/l2/consensus.go

`errOut` variable not used.

Solution

If it is certain that these will not be used, then they can be removed.

Status

Fixed; Fixed in https://github.com/morph-l2/go-ethereum/pull/59

[N3] [Medium] Partial error handling for fillTransactions function Interruptions

Category: Design Logic Audit

Content

- miner/worker.go

There are three types of errors returned by signalToErr `errBlockInterruptedByNewHead`, `errBlockInterruptedByRecommit`, `errBlockInterruptedByTimeout`

But the switch only determines `errBlockInterruptedByRecommit`, `errBlockInterruptedByNewHead`, This may lead to unknown exceptions.

```
func (w *worker) commitNewWork(interrupt *int32, noempty bool, timestamp int64) {
        ...
        err, _ = w.fillTransactions(w.current, nil, interrupt)
        switch {
        case err == nil:
                // The entire block is filled, decrease resubmit interval in case
                // of current interval is larger than the user-specified one.
                w.adjustResubmitInterval(&intervalAdjust{inc: false})

        case errors.Is(err, errBlockInterruptedByRecommit):
                // Notify resubmit loop to increase resubmitting interval if the
                // interruption is due to frequent commits.
                gaslimit := w.current.header.GasLimit
                ratio := float64(gaslimit-w.current.gasPool.Gas()) /
float64(gaslimit)
                if ratio < 0.1 {
                        ratio = 0.1
                }
                w.adjustResubmitInterval(&intervalAdjust{
                        ratio: ratio,
                        inc:   true,
                })

        case errors.Is(err, errBlockInterruptedByNewHead):
                // If the block building is interrupted by newhead event, discard it
                // totally. Committing the interrupted block introduces unnecessary
                // delay, and possibly causes miner to mine on the previous head,
                // which could result in higher uncle rate.
                w.current.discard()
                return
        } //slowmist//

        w.commit(uncles, w.fullTaskHook, true, tstart)
}
```

Solution

All errors are handled, and you can add a default to add an exception for unknown errors.

Status

Fixed; Fixed in https://github.com/morph-l2/go-ethereum/pull/61

[N4] [Information] Unused formal parameter exists

Category: Others

Content

- consensus/l2/consensus.go

Unused formal parameter seal.

```
func (l2 *Consensus) VerifyHeader(chain consensus.ChainHeaderReader, header
*types.Header, seal bool) error {
        // Short circuit if the parent is not known
        parent := chain.GetHeader(header.ParentHash, header.Number.Uint64()-1)
        if parent == nil {
                return consensus.ErrUnknownAncestor
        }
        // Sanity checks passed, do a proper verification
        return l2.verifyHeader(chain, header, parent)
}
```

Unused formal parameter seal.

```
func (l2 *Consensus) VerifyHeaders(chain consensus.ChainHeaderReader, headers
[]*types.Header, seals []bool) (chan<- struct{}, <-chan error) {
        return l2.verifyHeaders(chain, headers, nil)
}
```

Unused formal parameter chain.

```
func (l2 *Consensus) VerifyUncles(chain consensus.ChainReader, block *types.Block)
error {
        // Verify that there is no uncle block. It's explicitly disabled in the
beacon
        if len(block.Uncles()) > 0 {
                return errTooManyUncles
        }
        return nil
}
```

Unused formal parameter chain.

```
func (l2 *Consensus) Prepare(chain consensus.ChainHeaderReader, header *types.Header)
error {
        header.Difficulty = l2Difficulty
        header.Nonce = l2Nonce
        header.UncleHash = types.EmptyUncleHash
        header.Extra = []byte{} // disable extra field filling with bytes
        // set coinbase to empty address, if feeVault is enabled
```

```
        if l2.config.Scroll.FeeVaultEnabled() {
                header.Coinbase = types.EmptyAddress
        }
        return nil
}
```

Unused formal parameters chain, txs and uncles.

```
func (l2 *Consensus) Finalize(chain consensus.ChainHeaderReader, header *types.Header,
state *state.StateDB, txs []*types.Transaction, uncles []*types.Header) {
        // The block reward is no longer handled here. It's done by the
        // external consensus engine.
        header.Root = state.IntermediateRoot(true)
}
```

Unused formal parameters chain, time and parent.

```
func (l2 *Consensus) CalcDifficulty(chain consensus.ChainHeaderReader, time uint64,
parent *types.Header) *big.Int {
        return l2Difficulty
}
```

Solution

If it is determined that these parameters are unnecessary, they can be deleted.

Status

Acknowledged; Ignored due to interface compatibility issues.

[N5] [Suggestion] Direct comparison of errors

Category: Design Logic Audit

Content

- core/blockchain.go

It is recommended to use errors.Is instead of direct comparison.

```
func (bc *BlockChain) InsertReceiptChain(blockChain types.Blocks, receiptChain
[]types.Receipts, ancientLimit uint64) (int, error) {
        ...
        // Write downloaded chain data and corresponding receipt chain data
        if len(ancientBlocks) > 0 {
```

```
                if n, err := writeAncient(ancientBlocks, ancientReceipts); err != nil
{
                        if err == errInsertionInterrupted { //SLOWMIST//
                                return 0, nil
                        }
                        return n, err
                }
        }
    ...
        if len(liveBlocks) > 0 {
                if n, err := writeLive(liveBlocks, liveReceipts); err != nil {
                        if err == errInsertionInterrupted { //SLOWMIST//
                                return 0, nil
                        }
                        return n, err
                }
        }
    ...
}
```

Solution

Use errors.Is

Status

Fixed; Fixed in https://github.com/morph-l2/go-ethereum/pull/62

[N6] [Low] Mixed value and pointer receivers

Category: Others

Content

In Go, methods can be defined on both value and pointer receivers. When a method is defined on a value receiver, it operates on a copy of the struct, while methods defined on a pointer receiver operate on the original struct.

However, it's generally advised to be consistent in your method receiver types to avoid confusion and maintain a clear understanding of how methods interact with the struct. If methods are defined on both value and pointer receivers for the same struct, it might lead to unexpected behavior and confusion for users of the struct.

- core/types/transaction.go

```
func (s TxByPriceAndTime) Len() int { return len(s) }
func (s TxByPriceAndTime) Less(i, j int) bool {
        // If the prices are equal, use the time the transaction was first seen for
```

```
        // deterministic sorting
        cmp := s[i].minerFee.Cmp(s[j].minerFee)
        if cmp == 0 {
                return s[i].tx.time.Before(s[j].tx.time)
        }
        return cmp > 0
}
func (s TxByPriceAndTime) Swap(i, j int) { s[i], s[j] = s[j], s[i] }

func (s *TxByPriceAndTime) Push(x interface{}) {
        *s = append(*s, x.(*TxWithMinerFee))
}

func (s *TxByPriceAndTime) Pop() interface{} {
        old := *s
        n := len(old)
        x := old[n-1]
        *s = old[0 : n-1]
        return x
}
```

- core/types/block.go

```
// Uint64 returns the integer value of a block nonce.
func (n BlockNonce) Uint64() uint64 {
        return binary.BigEndian.Uint64(n[:])
}

// MarshalText encodes n as a hex string with 0x prefix.
func (n BlockNonce) MarshalText() ([]byte, error) {
        return hexutil.Bytes(n[:]).MarshalText()
}

// UnmarshalText implements encoding.TextUnmarshaler.
func (n *BlockNonce) UnmarshalText(input []byte) error {
        return hexutil.UnmarshalFixedText("BlockNonce", input, n[:])
}
```

- core/types/gen_header_json.go

```
func (h Header) MarshalJSON() ([]byte, error) {
    ...
}
```

Solution

To resolve this warning, you should choose one receiver type (either value or pointer) and stick with it consistently throughout your codebase. This helps maintain clarity and predictability in your code. If the methods don't need to modify the struct, using value receivers might be more appropriate. If the methods do need to modify the struct, using pointer receivers would be necessary.

Status

Acknowledged

[N7] [Information] Reserved word used as name

Category: Others

Content

- core/blockchain.go

The variable log conflicts with the name of the imported package.

```
line 1839;
```

- core/blockchain_l2.go

The variable log conflicts with the name of the imported package.

```
line 240;
```

- miner/worker.go

The variable log conflicts with the name of the imported package.

line 756;

- miner/worker.go

The variable state conflicts with the name of the imported package.

```
line 800;
```

- core/rawdb/accessors_chain.go

The variable bytes conflicts with the name of the imported package.

```
line 590;
```

The variable log conflicts with the name of the imported package.

```
line 630;
```

- eth/catalyst/l2_api.go

The variable state conflicts with the name of the imported package.

```
line 86;
```

- eth/catalyst/l2_api.go

Variable json conflicts with imported package name

```
line 215;
line 266;
```

Solution

Don't use reserved words for variables

Status

Fixed; Fixed in https://github.com/morph-l2/go-ethereum/pull/64

[N8] [Low] Use of obsolete structures

Category: Others

Content

- miner/miner.go

Uses the deprecated event.TypeMux

```
line 66:
line 78:
```

- miner/worker.go

Uses the deprecated event.TypeMux

```
line 183;
line 251;
```

Solution

Use event.Feed according to the official recommendation

Status

Acknowledged; The use of this structure will be removed later.

[N9] [Suggestion] Error not handled

Category: Error Unhandle Audit

Content

- miner/worker_l2.go

The error returned by w.commitTransactions is not handled, and if the commitTransactions is an exception, the returned skippedTxs will also be faulty.

```
func (w *worker) simulateL1Messages(genParams *generateParams, transactions
types.Transactions) ([]*types.Transaction, []*types.SkippedTransaction, error) {
    ...
        _, _, skippedTxs := w.commitTransactions(env, txs, env.header.Coinbase, nil)
//SLOWMIST//error no handle

        return env.txs, skippedTxs, nil
}
```

Solution

Handling of returned errors.

Status

Acknowledged; Calling w.commitTransactions inulateL1Message does not return error.

[N10] [Suggestion] Recommendation to evaluate version synchronization for go-ethereum Fork

Category: Others

Content

The latest version of escroll-tech/go-ethereum has been upgraded to v5.1.10. In contrast, the project's forked version is still based on the earlier v3.1.12. Between v3.1.12 and v5.1.10, the official team has corrected several functional issues. To ensure optimal software performance, it is recommended for the project team to pay attention to and evaluate these changes, considering a synchronization to the latest version.

Solution

Recommend keeping an eye out for updates.

Status

Acknowledged; The project has been continuously updated.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002403310002	SlowMist Security Team	2024.03.04 - 2024.03.31	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk, 2 low risk, 4 suggestion vulnerabilities.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist