Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.10.10, the SlowMist security team received the Bitget Wallet team's security audit application for Eip-7702 implement of morph-l2, 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	Passed

3 Project Overview

3.1 Project Introduction

Official Golang implementation of the morph-l2 protocol.

3.2 Coverage

Target Code and Revision:

https://github.com/morph-l2/go-ethereum/pull/196/files

```
//Audit the following files in PR:
core/rawdb/accessors_indexes.go
core/state/state_object.go
core/state/statedb.go
core/state/statedb_hooked.go
core/state_processor.go
core/state_transition.go
core/tx_pool.go
core/types/access_list_tx.go
core/types/dynamic_fee_tx.go
core/types/legacy_tx.go
core/types/tx_blob.go
core/types/transaction_signing.go
core/vm/eips.go
core/vm/evm.go
core/vm/instructions.go
core/vm/interpreter.go
core/vm/jump_table.go
internal/ethapi/api.go
```

Fix PRs:

https://github.com/morph-l2/go-ethereum/pull/207/files

https://github.com/morph-l2/go-ethereum/pull/208/files

https://github.com/morph-l2/go-ethereum/pull/209/files

3.3 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	EVM context error	Design Logic Audit	Medium	Fixed
N2	Transaction fees unnecessary increase	Design Logic Audit	Suggestion	Acknowledged
N3	L1DataFee deduction/booking asymmetry at FeeVault closure	Design Logic Audit	High	Fixed
N4	Return error not handled	Error Unhandle Audit	Low	Fixed
N5	Missing default branch in switch `tx.Type()`	Design Logic Audit	Low	Acknowledged
N6	RPC wallet module remote access risk	Others	Information	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] EVM context error

Category: Design Logic Audit

Content

In the transaction execution entry point `ApplyTransactionWithEVM`, the `OnTxStart` callback of the Tracer is called before setting `evm.TxContext`, causing the callback to obtain the VM context from the previous transaction (or default empty context).

- core/state_processor.go

```
if hooks := evm.Config.Tracer; hooks != nil {
    if hooks.OnTxStart != nil {
        hooks.OnTxStart(evm.GetVMContext(), tx, msg.From()) // Trigger here
    }
    if hooks.OnTxEnd != nil {
        defer func() { hooks.OnTxEnd(receipt, err) }()
    }
}

// VM context for this transaction is set here
txContext := NewEVMTxContext(msg)
evm.TxContext = txContext
```

Solution

Set `evm.TxContext` before calling `OnTxStart`, to ensure the callback gets the VM context of the current transaction.

Status

Fixed

[N2] [Suggestion] Transaction fees unnecessary increase

Category: Design Logic Audit

Content

The current implementation charges each EIP-7702 authorization tuple with the "new account" cost CallNewAccountGas(25_000) in intrinsic gas. If later it is found that the authorized account already exists, the difference (25_000 - 12_500) will be refunded. Although this "pay first, refund later" strategy may result in net costs close to the expected value, it directly increases the "minimum gas requirement for a transaction (intrinsic gas)" to 25k per authorization.

- core/state_transition.go

```
// Charge each item in the authorization list with the "new account" cost in internal
gas calculation:
if authList != nil {
   gas += uint64(len(authList)) * params.CallNewAccountGas
}
// ...

// When applying for authorization, if the account already exists, the difference is
refunded
if st.state.Exist(authority) {
   st.state.AddRefund(params.CallNewAccountGas - params.TxAuthTupleGas)
}
```

The minimum gas requirement was raised to "25k per authorization", causing transactions with gasLimit estimated at "12.5k per authorization" to be rejected at the step "intrinsic gas less than required".

Solution

Align the "minimum gas requirement" with the "expected net cost" to avoid relying on refund paths to correct prices.

Status

Acknowledged

[N3] [High] L1DataFee deduction/booking asymmetry at FeeVault closure

Category: Design Logic Audit

Content

In `buyGas()`, the `st.l1DataFee` is deducted from the sender's balance only when `FeeVaultEnabled()` is true and the transaction is not an `L1MessageTx`; but in the settlement stage `TransitionDb()`, regardless of whether `FeeVault` is enabled, the `l1DataFee + l2Fee` is sent to the fee recipient (non-L1MessageTx process). This results in the sender not being charged the `L1DataFee` when `FeeVault` is disabled, yet the fee is still "sent out", equivalent to creating money out of thin air.

- core/state_transition.go

```
// Pre-withdrawal only occurs when FeeVault is open (and not L1MessageTx):
func (st *StateTransition) buyGas() error {
   // ...
   if st.evm.ChainConfig().Morph.FeeVaultEnabled() {
       if !st.msg.IsL1MessageTx() {
```

```
            log.Debug("Adding L1DataFee", "l1DataFee", st.l1DataFee)
            mgval = mgval.Add(mgval, st.l1DataFee)
        }
    }
    // balanceCheck branch also only adds l1DataFee when FeeVault is open
    // ...
}

// Add L1DataFee unconditionally to the fee recipient at settlement time
l2Fee := new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), effectiveTip)
fee := new(big.Int).Add(st.l1DataFee, l2Fee) // Add l1DataFee regardless of the
FeeVault switch status
st.state.AddBalance(st.evm.FeeRecipient(), fee,
tracing.BalanceIncreaseRewardTransactionFee)
```

Solution

Ensure that "pre-deduction" and "posting" are strictly symmetrical.

Status

Fixed

[N4] [Low] Return error not handled

Category: Error Unhandle Audit

Content

(1). After a chain reorganization (reorg), txpool calculates the difference between the old chain's discarded transactions and the new chain's included transactions and attempts to reinject. The current implementation calls addTxsLocked to process the re-injected transactions, but completely ignores the errors and dirty account set returned.

- core/tx_pool.go

```
func (pool *TxPool) reset(oldHead, newHead *types.Header) {
//...
    pool.addTxsLocked(reinject, false)
```

(2). In the RPC method PublicBlockChainAPI.GetBlockReceipts, the current implementation returns (nil, nil) directly when the underlying call returns an error. According to the comment, the method should return JSON null when "the

block does not exist"; however, the cases of "the block does not exist" and "query error" are now merged into the same return value null.

- internal/ethapi/api.go

```
func (s *PublicBlockChainAPI) GetBlockReceipts(ctx context.Context, blockNrOrHash
rpc.BlockNumberOrHash) ([]map[string]interface{}, error) {
   block, err := s.b.BlockByNumberOrHash(ctx, blockNrOrHash)
   if block == nil || err != nil {
       // When the block doesn't exist, the RPC method should return JSON null
       // as per specification.
       return nil, nil
   }
```

The client and indexer cannot distinguish between "the block really does not exist" and "read error / timeout / internal error", leading to incorrect monitoring and data consistency judgments.

(3). In the RPC method PendingTransactions, the error return value of types.Sender is ignored

- internal/ethapi/api.go

```
func (s *PublicTransactionPoolAPI) PendingTransactions() ([]*RPCTransaction, error) {
   //...
   for _, tx := range pending {
       from, _ := types.Sender(s.signer, tx)
```

Solution

Handle errors returned from function calls.

Status

Fixed

[N5] [Low] Missing default branch in switch `tx.Type()`

Category: Design Logic Audit

Content

There is no default branch in the switch `tx.Type()` of `NewRPCTransaction`. Known types (AccessList, DynamicFee, L1Message, SetCode) are all handled, but unknown or erroneous types will not cause an error.

- internal/ethapi/api.go

```
func NewRPCTransaction(tx *types.Transaction, blockHash common.Hash, blockNumber
uint64, blockTime uint64, index uint64, baseFee *big.Int, config *params.ChainConfig)
*RPCTransaction {
   //...
   switch tx.Type() {
   case types.AccessListTxType:
       //...
   case types.DynamicFeeTxType:
       //...
   case types.L1MessageTxType:
       //...
   case types.SetCodeTxType:
       //...
   }
   return result
}
```

Solution

Add default branch.

Status

Acknowledged

[N6] [Information] RPC wallet module remote access risk

Category: Others

Content

If a node exposes "private" RPC interfaces related to account management (HTTP/WS), attackers can remotely initiate sensitive operations: creating accounts, importing private keys, unlocking accounts and initiating signatures. Additionally, new accounts do not validate password strength, and weak passwords may be exploited through brute force attacks.

Although there is an RPC configuration switch, once exposed to the public network or misconfigured, it may lead to persistent account unlocking, leakage of key, or passive signing, resulting in risks to funds and identity security.

- internal/ethapi/api.go

Solution

N/A

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002510160002	SlowMist Security Team	2025.10.10 - 2025.10.16	Low Risk

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 1 high risk, 1 medium risk, 2 low risk, 1 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