Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.03.19, the SlowMist security team received the Brevis Network team's security audit application for uniswap-rebate, 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 lead, black, grey box assists" 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 team 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.

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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit

Serial Number	Audit Class	Audit Subclass
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-
17	Circuit Trusted Setup Risks	-
18	Overflow of Circuit Operations	-
19	Input Signal Cracking	-
20	Input Signal Leakage	-

3 Project Overview

3.1 Project Introduction

This is the circuit code of a Uniswap rebate system, mainly used for verifying transactions and calculating fuel rebates.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of exception handling	Design Logic Audit	Suggestion	Fixed
N2	Error in calculating the maximum block number	Design Logic Audit	High	Fixed
N3	Integer overflow risk	Integer Overflow and Underflow Vulnerability	Low	Acknowledged
N4	Incomplete circuit constraints lead to numerical extensibility risks.	Lack of constrains	Low	Acknowledged
N5	The length of the unchecked input Receipts	Design Logic Audit	Low	Acknowledged
N6	Pool ID logical flaw in verification	Lack of constrains	High	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/brevis-network/uniswap-rebate/blob/v4/circuit/circuit.go

Initial audit commit: 331209aef6e7d477156f609637081d771a369b83

Final audit commit: 9334e4757661d7749e15163cbb4ed11d2a6cfaf7

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

4.3 Vulnerability Summary

[N1] [Suggestion] Lack of exception handling

Category: Design Logic Audit

Content

In the `Hex2Bytes` function, the error returned by `hex.DecodeString` is not handled.:

- circuit/circuit.go

```
func Hex2Bytes(s string) (b []byte) {
   if len(s) >= 2 && s[0] == '0' && (s[1] == 'x' || s[1] == 'X') {
       s = s[2:]
   }
   // hex.DecodeString expects an even-length string
   if len(s)%2 == 1 {
       s = "0" + s
   }
   b, _ = hex.DecodeString(s)
   return b
}
```

Solution

Check if any errors have occurred and handle them.

Status

Fixed

[N2] [High] Error in calculating the maximum block number

Category: Design Logic Audit

Content

When the condition is not met, `maxBlk` should be returned instead of `minBlk`. The current implementation may cause the maximum block number to be calculated incorrectly, affecting the valid range of the rebate amount.

- circuit/circuit.go

```
  maxBlk := sdk.Reduce(swaps, sdk.ConstUint32(0), func(maxBlk sdk.Uint32, r
sdk.Receipt) sdk.Uint32 {
        return api.Uint32.Select(
            api.Uint32.IsGreaterThan(r.BlockNum, maxBlk),
            r.BlockNum,
            minBlk,
        )
  })
```

Solution

```
return api.Uint32.Select(
            api.Uint32.IsGreaterThan(r.BlockNum, maxBlk),
            r.BlockNum,
            maxBlk,
        )
```

Status

Fixed

[N3] [Low] Integer overflow risk

Category: Integer Overflow and Underflow Vulnerability

Content

The `Add` method of the API does not perform overflow checking. When `a + b > 2^32` or `a + b > 2^248`, the calculation result will be incorrect due to numerical overflow.

- circuit/circuit.go

```
curTxGas = api.Uint32.Add(curTxGas, c.GasPerSwap)
//…
api.Uint32.Add(curTxGas, c.GasPerTx)
//…
totalRebate = api.Uint248.Add(totalRebate, api.Uint248.Mul(api.ToUint248(toAdd),
r.BlockBaseFee))
```

Solution

Check if the calculation result has overflowed.

Status

Acknowledged

[N4] [Low] Incomplete circuit constraints lead to numerical extensibility risks.

Category: Lack of constrains

Content

The input to the circuit is a multi-level complex data structure. These data are defined outside the Define function, and all fields must be constrained within the circuit to ensure that the input data is fully verified and proven. If some of the input fields are not constrained within the circuit, these fields will be at risk of malleability attacks. This means that after the proof is generated, an attacker can modify the unconstrained input data, bringing unknown risks.

Pay attention to the following defined data input structure, where all unconstrained fields are marked as Unconstrained.

- circuit/circuit.go

```
func (c *GasCircuit) Define(api *sdk.CircuitAPI, in sdk.DataInput) error {
//...
}


type DataInput struct {
   Receipts     DataPoints[Receipt]
   StorageSlots DataPoints[StorageSlot] //SlowMist// Unconstrained
   Transactions DataPoints[Transaction] //SlowMist// Unconstrained
}


const NumMaxLogFields = 4


type Receipt struct {
   BlockNum       Uint32
   BlockBaseFee   Uint248
   MptKeyPath     Uint32
   Fields         [NumMaxLogFields]LogField  //SlowMist// 2 Fields unconstrained
   BlockTimestamp Uint248 //SlowMist// Unconstrained
}


type LogField struct {
```

```
    Contract Uint248
    LogPos Uint32  //SlowMist// Unconstrained
    EventID Uint248
    IsTopic Uint248 //SlowMist// Unconstrained
    Index Uint248  //SlowMist// Unconstrained
    Value Bytes32
}
```

Solution

Use the API to constrain all input data fields.

Status

Acknowledged; The correctness of the data itself is guaranteed by the brevis prover.

[N5] [Low] The length of the unchecked input Receipts

Category: Design Logic Audit

Content

The length of `in.Receipts.Raw` is not verified to be at least `MaxSwapNum + 1`. If the length is insufficient, it may lead to out-of-bounds array access.

- circuit/circuit.go

```
    for i := 1; i < MaxSwapNum; i++ {
        cur := in.Receipts.Raw[i]
        next := in.Receipts.Raw[i+1]
        api.Uint32.AssertIsEqual(api.Uint32.Select(
            api.Uint32.IsZero(c.TxGasCap[i-1]), // TxGasCap index is 1 less than
receipt
            api.Uint32.And(
                api.Uint32.IsEqual(cur.BlockNum, next.BlockNum),
                api.Uint32.IsEqual(cur.MptKeyPath, next.MptKeyPath)),
            sdk.ConstUint32(1),
        ), sdk.ConstUint32(1))
    }
```

Solution

Use API to restrict input length.

Status

Acknowledged; Allocate returns the maximum length of Receipts supported by the circuit, and the Brevis SDK ensures that it will not exceed `MaxReceipts`.

[N6] [High] Pool ID logical flaw in verification

Category: Lack of constrains

Content

During each loop iteration, the value of `eligible` is overwritten. If a previously matched pool sets `eligible` to 1, but a subsequent non-matching pool resets it back to the current value (still 1).

Assuming the first pool matches, `eligible` will be set to 1, but if the last pool does not match and `eligible` is still 1 at that time, the final value remains 1. This means the outcome depends on the order of pool checks, not just whether there is a match.

The intended implementation seems to be to check if at least one pool matches, but the current implementation does not correctly express this logic.

- circuit/circuit.go

```
eligible := sdk.ConstUint32(0) // check event poolid with all poolids
for j := range MaxPoolNum {
    // if event poolid matches poolid, set eligiblePoolId to 1, otherwise keep as is
    eligible = api.Uint32.Select(
        sdk.Uint32{Val: api.Bytes32.IsEqual(poolIDs[j], r.Fields[0].Value).Val},
        sdk.ConstUint32(1),
        eligible,
    )
}
```

Solution

Clarify the eligible conditions.

Status

Fixed; Change to using `Or` to make it look clearer.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002503210002	SlowMist Security Team	2025.03.19 - 2025.03.21	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 2 high risk, 3 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