Smart Contract Audit Report

IOTAMPC Bridge Smart Contract

20 Mar 2023

Numen Cyber Labs - Security Services

Table of Content

1 EXECUTIVE SUMMARY

Numen Cyber Technology was engaged by IOTAMPC Bridge to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

Any signer can delete a signature and any signer can change the number of signature requests.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

METHODOLOGY

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: High, Medium and Low. Severity is determined by likelihood and impact and can be classified into four categories accordingly, Critical, High, Medium, Low shown in table 1.1.

Table 1.1: Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft fail Attack
	Hard fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure

	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2 FINDINGS OVERVIEW

2.1 PROJECT INFO AND CONTRACT ADDRESS

Project Name: IOTAMPC Bridge

Audit Time: 2023/3/14 - 2022/3/20

Language: solidity

Source Code Link	Commit Hash
https://github.com/TanglePay/bridge	48ad2ac76b5d226502a3c8d7b96c9536c4dd3fa4

2.2 SUMMARY

Severity	Found	
Critical	0	
High	2	
Medium	3	
Low	1	
Informational	4	

2.3 KEY FINDINGS

Two high-risk and medium-risk questions about setting signatories.

ID	Severity	Findings Title	Status	Confirm
NVE-001	High	A signatory can remove multiple signatures	Ignore	Confirmed
NVE-002	High	A signatory can update the number of signatures to zero	Ignore	Confirmed
NVE-003	Medium	Addsigner address cannot be changed after it has been added	Ignore	Confirmed
NVE-004	Medium	Removesigner address cannot be changed after it has been added	Ignore	Confirmed
NVE-005	Medium	The changerequirecount number cannot be changed again after it has been updated	Ignore	Confirmed
NVE-006	Low	Fee extraction security	Ignore	Confirmed
NVE-007	Information	Redundant codes	Ignore	Confirmed
NVE-008	Information	Newowner may be a zero address	Ignore	Confirmed
NVE-009	Information	Signer address and number cannot be too small	Ignore	Confirmed
NVE-010	Information	Any signer can add address(0) as a signer	Ignore	Confirmed

Table 2.1: Key Audit Findings

3 DETAILED DESCRIPTION OF FINDINGS

3.1 A SIGNATORY CAN REMOVE MULTIPLE SIGNATURES

ID: NVE-001

Location: multiSign.sol

Severity: High

Category: Business Issues

Likelihood: Medium

Impact: High

Description:

The removeSigner method is used to remove an existing signer. A malicious signer address can remove signers with zero signers[index] by calling the removeSigner method multiple times. When a signer with zero signers[index] is removed, a new signer with zero signers[index] can also be removed if it is not a malicious signer, that is, a malicious signer may remove multiple signers.

```
72      // To remove a signer
73      function removeSigner(uint8 index) external onlySigner {
74          if (removingIndex == 0) {
75              require(index < signers.length, "wrong index");
76              removingIndex == index;
77              //set isRemoved empty
78              for (uint8 i = 0; i < signers.length; i++) {
79                  delete isRemoved[msg.sender][i];
80              }
81          } else {
82              require(removingIndex == index, "mismatched index");
83          }
84
85          if (!isRemoved[msg.sender][index]) {
86              isRemoved[msg.sender][index] = true;
87              removingCount++;
88          }
89
90          if (removingCount >= requireCount) {
91              address toDelAddr = signers[index];
92              address lastSigner = signers[signers.length - 1];
93              signers[index] = lastSigner;
94              iSigner[lastSigner] = index + 1;
95              delete iSigner[toDelAddr];
96              signers.pop();
97
98              removingIndex = 0;
99              removingCount = 0;
100
101             emit SignerRemoval(toDelAddr);
102         }
103     }
```

Figure 1 removeSigner function

Recommendations:

Suggested removal of signatures to determine if signers[index] is zero when the signature is removed.

Result: **Not passed**

Fix Result:

Ignore

3.2 A SIGNATORY CAN UPDATE THE NUMBER OF SIGNATURES TO ZERO

ID: NVE-002 Location: multiSign.sol

Severity: High Category: Business Issues

Likelihood: High Impact: High

Description:

The changeRequireCount method is used to update the number of signatories. Any signatory can change the number of signatures to zero by calling it multiple times, and if there is a malicious signatory, any signatory can call the signature implementation method call individually when the number of signatures is changed to zero.

```
109     // to change the requireCount
110     function changeRequireCount(uint8 newCount) external onlySigner {
111         if (newRequireCount == 0) {
112             require(newCount <= signers.length, "wrong count");
113             newRequireCount == newCount;
114             //set isChangeCount empty
115             for (uint8 i = 0; i < signers.length; i++) {
116                 delete isChangeCount[msg.sender];
117             }
118         } else {
119             require(newCount == newRequireCount, "mismatched count");
120         }
121 
122         if (!isChangeCount[msg.sender]) {
123             isChangeCount[msg.sender] = true;
124             changingCount++;
125         }
126 
127         if (changingCount >= requireCount) {
128             requireCount = newCount;
129 
130             newRequireCount = 0;
131             changingCount = 0;
132 
133             emit ChangeRequireCount(newCount);
134         }
135     }
```

Figure 2 changeRequireCount function

Recommendations:

It is recommended that a judgement be added and that the number of strong judgement signatories not be too small.

Result: Not passed

Fix Result:

Ignore

3.3 ADDSIGNER ADDRESS CANNOT BE CHANGED AFTER IT HAS BEEN ADDED

ID: NVE-003 Location: multiSign.sol

Severity: Medium Category: Business Issues

Likelihood: Medium Impact: Medium

Description:

addSigner method is used to add a new signer, any signer submitted to addSigner address is not address(0), the remaining signature address also can not set other addresses for the signer, if there is a malicious signer to submit a signature, the rest of the benign signer can only agree to sign, otherwise the addSigner add signature method will not work properly, once the signature provided by a malicious signer is added, it will cause the risk of the existence of multiple malicious signatures.

```
39      // To add a signer
40      function addSigner(address signer) external onlySigner {
41          if (addingSigner == address(0)) {
42              require(iSigner[signer] == 0, "already exist");
43              addingSigner = signer;
44              //set isAdded empty
45              for (uint8 i = 0; i < signers.length; i++) {
46                  delete isAdded[msg.sender][signer];
47              }
48          } else {
49              require(addingSigner == signer, "wrong signer");
50          }
51
52          if (!isAdded[msg.sender][signer]) {
53              isAdded[msg.sender][signer] = true;
54              addingCount++;
55          }
56
57          if (addingCount >= requireCount) {
58              signers.push(signer);
59              iSigner[signer] = uint8(signers.length);
60
61              addingSigner = address(0);
62              addingCount = 0;
63
64              emit SignerAddition(signer);
65          }
66      }
```

Figure 3 addSigner function

Recommendations:

It is recommended that time judgements be added so that when an address is older than the time a signature is added, a new signatory can be resubmitted.

Result: Not passed

Fix Result:

Ignore

3.4 REMOVESIGNER ADDRESS CANNOT BE CHANGED AFTER IT HAS BEEN ADDED

ID: NVE-004

Location: multiSign.sol

Severity: Medium

Category: Business Issues

Likelihood: Medium

Impact: Medium

Description:

removeSigner method is used to remove existing signers, any signer submitted to add removingIndex address is not address(0), the remaining signature address can not remove other signers, if there is a malicious signer to remove the signature, the remaining benign signers can only agree to sign, otherwise the removeSigner method to remove the signature will not work properly, once the removal of the signature provided by the malicious signer, it will cause the risk of multiple benign signers removed.

```
72      // To remove a signer
73      function removeSigner(uint8 index) external onlySigner {
74          if (removingIndex == 0) {
75              require(index < signers.length, "wrong index");
76              removingIndex == index;
77              //set isRemoved empty
78              for (uint8 i = 0; i < signers.length; i++) {
79                  delete isRemoved[msg.sender][i];
80              }
81          } else {
82              require(removingIndex == index, "mismatched index");
83          }
84
85          if (!isRemoved[msg.sender][index]) {
86              isRemoved[msg.sender][index] = true;
87              removingCount++;
88          }
89
90          if (removingCount >= requireCount) {
91              address toDelAddr = signers[index];
92              address lastSigner = signers[signers.length - 1];
93              signers[index] = lastSigner;
94              iSigner[lastSigner] = index + 1;
95              delete iSigner[toDelAddr];
96              signers.pop();
97
98              removingIndex = 0;
99              removingCount = 0;
100
101             emit SignerRemoval(toDelAddr);
102         }
103     }
```

Figure 4 removeSigner function

Recommendations:

Suggest adding a time determination so that when an address is older than the signature removal time, then a new removal signer can be resubmitted.

Result: **Not passed**

Fix Result:

Ignore

3.5 THE CHANGEREQUIRECOUNT NUMBER CANNOT BE CHANGED AGAIN AFTER IT HAS BEEN UPDATED

ID: NVE-005

Location: multiSign.sol

Severity: Medium

Category: Business Issues

Likelihood: Medium

Impact: Medium

Description:

The changeRequireCount method is used to update the number of signers. After any signer submits a non-zero newRequireCount value, the remaining signature addresses cannot be changed to other values either. If there is a malicious signer changing the number of signatures, the remaining benign signers can only agree to change their signatures, otherwise the changeRequireCount method to update the number of signatures will not work properly, and once the same malicious number of signatures is submitted, it will cause a security incident.

```
109     // to change the requireCount
110     function changeRequireCount(uint8 newCount) external onlySigner {
111         if (newRequireCount == 0) {
112             require(newCount <= signers.length, "wrong count");
113             newRequireCount == newCount;
114             //set isChangeCount empty
115             for (uint8 i = 0; i < signers.length; i++) {
116                 delete isChangeCount[msg.sender];
117             }
118         } else {
119             require(newCount == newRequireCount, "mismatched count");
120         }
121 
122         if (!isChangeCount[msg.sender]) {
123             isChangeCount[msg.sender] = true;
124             changingCount++;
125         }
126 
127         if (changingCount >= requireCount) {
128             requireCount = newCount;
129 
130             newRequireCount = 0;
131             changingCount = 0;
132 
133             emit ChangeRequireCount(newCount);
134         }
135     }
```

Figure 5 changeRequireCount function

Recommendations:

Suggest adding a time determination so that when an address exceeds a signature count update event, a new signature count update can be resubmitted.

Result: Not passed

Fix Result:

Ignore

3.6 FEE EXTRACTION SECURITY

ID: NVE-006 Location: wrap.sol

Severity: Low Category: Business Issues

Likelihood: Low Impact: Medium

Description:

The withdrawFee method is used to withdraw the team fee, but may steal the team fee when the owner privileged role is malicious.

The unWrapFee method should be used to transfer the team fee, but when the owner privileged role is malicious, the fee to address can be set to the address of the person who stole the money.

```
83      //withdraw the fee to dev team
84      function withdrawFee(address to, uint256 fee) external {
85          require(msg.sender == owner, "forbidden");
86          feeSum -= fee;
87          _mint(to, fee);
88      }
89
90      //unwrap the fee to dev team
91      function unWrapFee(
92          bytes32 to,
93          bytes32 symbol,
94          uint256 fee
95      ) external {
96          require(msg.sender == owner, "forbidden");
97          feeSum -= fee;
98          emit UnWrap(owner, to, symbol, fee);
99      }
```

Figure 6 withdrawFee and unWrapFee function

Recommendations:

It is recommended that the owner's privileged role be managed using multiple signatures.

Result: Pass

Fix Result:

Ignore

3.7 REDUNDANT CODES

ID: NVE-007

Location: multiSign.sol

Severity: Information

Category: Business Issues

Likelihood: Low

Impact: Low

Description:

The addSigner method is used to add a new signer and clear the original data on the first call, but the contract uses a for loop statement to make a judgement that has no real meaning and it is recommended to remove the for loop statement here to avoid wasting Gas.

```
39      // To add a signer
40      function addSigner(address signer) external onlySigner {
41          if (addingSigner == address(0)) {
42              require(iSigner[signer] == 0, "already exist");
43              addingSigner = signer;
44              //set isAdded empty
45              for (uint8 i = 0; i < signers.length; i++) {
46                  delete isAdded[msg.sender][signer];
47              }
48          } else {
49              require(addingSigner == signer, "wrong signer");
50          }
51
52          if (!isAdded[msg.sender][signer]) {
53              isAdded[msg.sender][signer] = true;
54              addingCount++;
55          }
56
57          if (addingCount >= requireCount) {
58              signers.push(signer);
59              iSigner[signer] = uint8(signers.length);
60
61              addingSigner = address(0);
62              addingCount = 0;
63
64              emit SignerAddition(signer);
65          }
66      }
```

Figure 7 addSigner function

Recommendations:

To avoid wasting Gas, it is recommended to remove the for loop statement here.

Result: **Pass**

Fix Result:

Ignore

3.8 NEWOWNER MAY BE A ZERO ADDRESS

ID: NVE-009

Location: Ownable.sol

Severity: Information

Category: Business Issues

Likelihood: Low

Impact: Low

Description:

The transferOwner method is used to modify the new owner privileged address, but the privileged address may be called accidentally when the transferOwner method is called resulting in the newOwner becoming address(0), but it can also be called again to modify the newOwner address normal address.

```
9     //transfer the owner
10    function transferOwner(address _owner) external {
11        require(msg.sender == owner, "forbidden");
12        newOwner = _owner;
13    }
14
15    //accept the owner
16    function acceptSetter() external {
17        require(msg.sender == newOwner, "forbidden");
18        owner = newOwner;
19        newOwner = address(0);
20    }
```

Figure 8 transferOwner function

Recommendations:

It is recommended that the owner privileged role avoid passing in the address(0) parameter when calling the transferOwner method.

Result: Pass

Fix Result:

Ignore

3.9 SIGNER ADDRESS AND NUMBER CANNOT BE TOO SMALL

ID: NVE-009 Location: multiSign.sol

Severity: Information Category: Business Issues

Likelihood: Low Impact: Low

Description:

As a multiSign contract, if the number of signatories or signatures is small, it may lead to one address controlling the signature and there is a security risk. It is recommended to add N signatories and require a threshold of more than N/2 signatures.

```
23      constructor(address[] memory _signers, uint8 _requireCount) {
24          for (uint8 i = 0; i < _signers.length; i++) {
25              require(
26                  iSigner[_signers[i]] == 0 && _signers[i] != address(0),
27                  "invalid or duplicated signer"
28              );
29              iSigner[_signers[i]] = i + 1;
30          }
31          signers = _signers;
32          requireCount = _requireCount;
33      }
```

Figure 9 multiSign contract constructor

Recommendations:

It is recommended that the threshold for the number of signatures required to add N signatories is greater than N/2.

Result: Pass

Fix Result:

Ignore

3.10 ANY SIGNER CAN ADD ADDRESS(0) AS A SIGNER

ID: NVE-010

Location: multiSign.sol

Severity: Information

Category: Business Issues

Likelihood: Low

Impact: Low

Description:

The addSigner method is used to add a new signer, but an arbitrary signer can be added as a signer by calling the addSigner method multiple times with the ADDRESS(0) address, and adding ADDRESS(0) as a signer has no significant effect on the overall signing mechanism.

```
39      // To add a signer
40      function addSigner(address signer) external onlySigner {
41          if (addingSigner == address(0)) {
42              require(iSigner[signer] == 0, "already exist");
43              addingSigner = signer;
44              //set isAdded empty
45              for (uint8 i = 0; i < signers.length; i++) {
46                  delete isAdded[msg.sender][signer];
47              }
48          } else {
49              require(addingSigner == signer, "wrong signer");
50          }
51
52          if (!isAdded[msg.sender][signer]) {
53              isAdded[msg.sender][signer] = true;
54              addingCount++;
55          }
56
57          if (addingCount >= requireCount) {
58              signers.push(signer);
59              iSigner[signer] = uint8(signers.length);
60
61              addingSigner = address(0);
62              addingCount = 0;
63
64              emit SignerAddition(signer);
65          }
66      }
```

Figure 10 addSigner function

Recommendations:

It is recommended to determine if the signer is ADDRESS(0) when adding the signature address.

Result: **Pass**

Fix Result:

Ignore

4 CONCLUSION

In this audit, we thoroughly analyzed **IOTAMPC Bridge** smart contract implementation. The problems found are described and explained in detail in Section 3. The issues identified in the audit have been raised with the project and the two high risk vulnerabilities that need to be amended as soon as possible are: a signer can remove multiple signatures; and a signer can update the number of signatures to zero. We therefore consider the audit result to be **Not passed**.To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5 APPENDIX

5.1 BASIC CODING ASSESSMENT

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 soft fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 hard fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behaviour

- Description: Examine whether sensitive behaviour present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6 DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without Numen's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Numen to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Numen's position is that each company and individual are responsible for their own due diligence and continuous security. Numen's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound). https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation. https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure. https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors. https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_ Rating_Methodology

Numen Cyber Technology Pte. Ltd.

11 North Buona Vista Drive, #04-09,

The Metropolis, Singapore 138589

Tel: 65-63555555

Fax: 65-63666666

Email: sales@numencyber.com

Web: https://numencyber.com