SMART CONTRACT AUDIT REPORT

for

Celer MultiBridge

Prepared By: Xiaomi Huang

PeckShield

February 15, 2023

Document Properties

Client	Celer
Title	Smart Contract Audit Report
Target	Celer MultiBridge
Version	1.0
Author	Xuxian Jiang
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Patrick Lou
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	February 15, 2023	Xuxian Jiang	Final Release
1.0-rc	February 13, 2023	Xuxian Jiang	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction **4**

1.1 About Celer MultiBridge 4

1.2 About PeckShield 5

1.3 Methodology 5

1.4 Disclaimer 7

2 Findings **9**

2.1 Summary 9

2.2 Key Findings 10

3 Detailed Results **11**

3.1 Suggested Adherence Of Checks-Effects-Interactions Pattern 11

3.2 Meaningful Events For Important State Changes 12

3.3 (Limited) Trust Of Admin Keys 13

4 Conclusion **15**

References **16**

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the `Celer MultiBridge` protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Celer MultiBridge

`Celer MultiBridge` is a solution for cross-chain message passing without vendor lock-in and with enhanced security beyond any single bridge. Specifically, a message with multiple copies is sent through different bridges to the destination chains, and will only be executed at the destination chain when the same message has been delivered by a quorum of different bridges. It reduces the trust on each individual bridge and greatly improves the reliability of cross-chain message passing. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of Celer MultiBridge

Item	Description
Name	Celer MultiBridge
Website	https://www.celer.network/
Type	EVM Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	February 15, 2023

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit. Please note that this audit examines only the contracts in the following directory:

contracts/message/apps/multibridge and covers only the Celer-adapter.

- https://github.com/celer-network/sgn-v2-contracts.git (de88762)

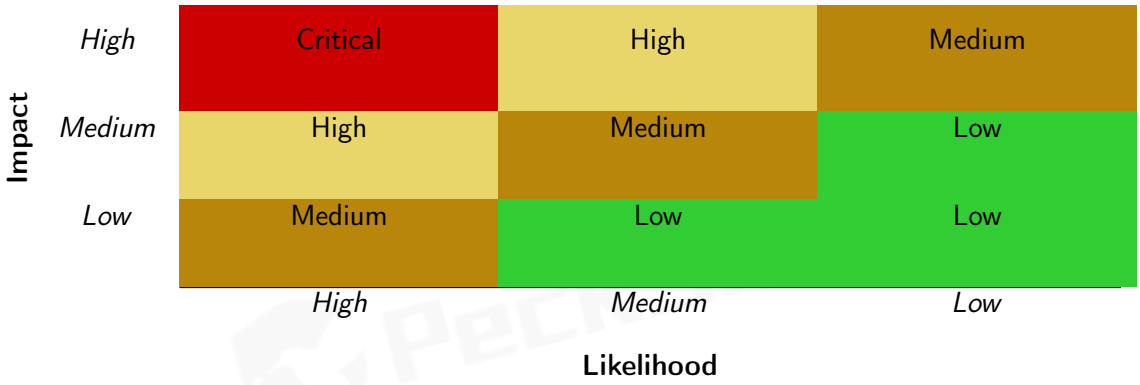
And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/celer-network/sgn-v2-contracts.git (73001ed)

1.2 About PeckShield

PeckShield Inc. [9] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification



1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [8]:

- 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 demonstrates the overall criticality of the risk.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet 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 1.3.

In particular, we perform the audit according to the following procedure:

- 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.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: 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.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [7], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings.

1.4 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 of 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. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the `Celer MultiBridge` implementation. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	2	■■
Informational	1	■
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 low-severity vulnerabilities and 1 informational recommendation.

Table 2.1: Key Celer MultiBridge Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Suggested Adherence Of Checks-Effects-Interactions Pattern	Business Logic	Resolved
PVE-002	Informational	Meaningful Events For Important State Changes	Coding Practices	Resolved
PVE-003	Low	(Limited) Trust Issue Of Admin Keys	Security Features	Resolved

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Suggested Adherence Of Checks-Effects-Interactions Pattern

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `MultiBridgeReceiver`
- Category: Business Logic [6]
- CWE subcategory: CWE-841 [3]

Description

A common coding best practice in Solidity is the adherence of `checks-effects-interactions` principle. This principle is effective in mitigating a serious attack vector known as `re-entrancy`. Via this particular attack vector, a malicious contract can be reentering a vulnerable contract in a nested manner. Specifically, it first calls a function in the vulnerable contract, but before the first instance of the function call is finished, second call can be arranged to re-enter the vulnerable contract by invoking functions that should only be executed once. This attack was part of several most prominent hacks in Ethereum history, including the `DAO` [11] exploit, and the `Uniswap/Lendf.Me` hack [10].

We notice there is an occasion where the `checks-effects-interactions` principle is violated. Specifically, in the `MultiBridgeReceiver` contract, the `_executeMessage()` function (see the code snippet below) is provided to externally call a contract to execute a cross-chained message. However, the invocation of an external contract requires extra care in avoiding the above `re-entrancy`. For example, the interaction with the external contract (line 129) starts before effecting the update on the internal state (line 131), hence violating the principle. In this particular case, if the external contract has certain hidden logic that may be capable of launching `re-entrancy` via the same entry function.

```
119     /**
120      * @notice Execute the message (invoke external call according to the message
             content) if the message
121      * has reached the power threshold (the same message has been delivered by enough
             multiple bridges).
```

```
122      */
123     function _executeMessage(MessageStruct.Message calldata _message, MsgInfo storage
            _msgInfo) private {
124         if (_msgInfo.executed) {
125             return;
126         }
127         uint64 msgPower = _computeMessagePower(_msgInfo);
128         if (msgPower >= (totalPower * quorumThreshold) / THRESHOLD_DECIMAL) {
129             (bool ok, ) = _message.target.call(_message.callData);
130             require(ok, "external message execution failed");
131             _msgInfo.executed = true;
132             emit MessageExecuted(_message.srcChainId, _message.nonce, _message.target,
                    _message.callData);
133         }
134     }
```

Listing 3.1: `MultiBridgeReceiver::_executeMessage()`

Recommendation Apply necessary reentrancy prevention by following the `checks-effects-interactions` principle.

Status This issue has been fixed in the following commit: `73001ed`.

3.2 Meaningful Events For Important State Changes

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Adapter Contracts`
- Category: Coding Practices [5]
- CWE subcategory: CWE-563 [2]

Description

In `Ethereum`, the `event` is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an `event` is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. `Events` can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

While examining the events that reflect the protocol dynamics, we notice there are several privileged routines that lack meaningful events to reflect their changes. In the following, we show several representative routines.

```
60     function updateSenderAdapter(uint64[] calldata _srcChainIds, address[] calldata
           _senderAdapters)
61         external
```

```
62          override
63          onlyOwner
64      {
65          require(_srcChainIds.length == _senderAdapters.length, "mismatch length");
66          for (uint256 i = 0; i < _srcChainIds.length; i++) {
67              senderAdapters[_srcChainIds[i]] = _senderAdapters[i];
68          }
69      }
70
71      function setMultiBridgeReceiver(address _multiBridgeReceiver) external override
            onlyOwner {
72          multiBridgeReceiver = _multiBridgeReceiver;
73      }
```

Listing 3.2: Multiple Setters in `CelerReceiverAdapter`

With that, we suggest to emit meaningful events in these privileged routines. Also, the key event information is better `indexed`. Note each emitted event is represented as a topic that usually consists of the signature (from a `keccak256` hash) of the event name and the types (`uint256`, `string`, etc.) of its parameters. Each indexed type will be treated like an additional topic. If an argument is not indexed, it will be attached as data (instead of a separate topic). Considering that the key information is typically queried, it is better treated as a topic, hence the need of being `indexed`.

Recommendation Properly emit the above-mentioned events with accurate information to timely reflect state changes. This is very helpful for external analytics and reporting tools.

Status This issue has been fixed in the following commit: `73001ed`.

3.3 (Limited) Trust Of Admin Keys

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Multiple Contracts`
- Category: Security Features [4]
- CWE subcategory: CWE-287 [1]

Description

In the `Celer MultiBridge` protocol, there is a privileged adapter-specific `owner` account that plays a critical role in governing and regulating each individual adapter (e.g., add/remove sender/receiver adapters and update multibridge sender/receiver). In the following, we show the representative functions potentially affected by the privilege of the `owner` account.

```
60      function updateSenderAdapter(uint64[] calldata _srcChainIds, address[] calldata
            _senderAdapters)
```

```
61          external
62          override
63          onlyOwner
64      {
65          require(_srcChainIds.length == _senderAdapters.length, "mismatch length");
66          for (uint256 i = 0; i < _srcChainIds.length; i++) {
67              senderAdapters[_srcChainIds[i]] = _senderAdapters[i];
68          }
69      }
70
71      function setMultiBridgeReceiver(address _multiBridgeReceiver) external override
            onlyOwner {
72          multiBridgeReceiver = _multiBridgeReceiver;
73      }
```

Listing 3.3: `CelerReceiverAdapter::updateSenderAdapter()/setMultiBridgeReceiver()`

```
41      function updateReceiverAdapter(uint64[] calldata _dstChainIds, address[] calldata
            _receiverAdapters)
42          external
43          override
44          onlyOwner
45      {
46          require(_dstChainIds.length == _receiverAdapters.length, "mismatch length");
47          for (uint256 i = 0; i < _dstChainIds.length; i++) {
48              receiverAdapters[_dstChainIds[i]] = _receiverAdapters[i];
49          }
50      }
51
52      function setMultiBridgeSender(address _multiBridgeSender) external override
            onlyOwner {
53          multiBridgeSender = _multiBridgeSender;
54      }
```

Listing 3.4: `CelerSenderAdapter::updateReceiverAdapter()/setMultiBridgeSender()`

We emphasize that the privilege assignment is indeed necessary and consistent with the protocol design. In addition, we notice that each owner is restricted to each individual adapter, which is accommodated by the current trust model behind the proposed multi-bridge solution. In other words, even if an adapter is compromised, it only affects the assigned weight for the adapter, not other functional adapters.

Recommendation While the owner' privilege is well contained, it is still suggested to have a multi-sig account to exercise the privileged `owner` account. If necessary, apply the best practices to have the privileged operation mediated with a timelock.

Status This issue has been confirmed by the team. Specifically, each adapter could be implemented with/without owner and is depended on each bridge provider. If an owner is specified, then it should be renounced before integration.

4 | Conclusion

In this audit, we have analyzed the `Celer MultiBridge` design and implementation. It is a solution for cross-chain message passing without vendor lock-in and with enhanced security beyond any single bridge. Specifically, a message with multiple copies is sent through different bridges to the destination chains, and will only be executed at the destination chain when the same message has been delivered by a quorum of different bridges. It reduces the trust on each individual bridge and greatly improves the reliability of cross-chain message passing. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[3] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

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

[5] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[6] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[7] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[8] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.

[10] PeckShield. Uniswap/Lendf.Me Hacks: Root Cause and Loss Analysis. https://medium.com/@peckshield/uniswap-lendf-me-hacks-root-cause-and-loss-analysis-50f3263dcc09.

[11] David Siegel. Understanding The DAO Attack. https://www.coindesk.com/understanding-dao-hack-journalists.