SMART CONTRACT AUDIT REPORT

for

SGNv2 NFT Bridge

Prepared By: Xiaomi Huang

PeckShield

April 29, 2022

Document Properties

Client	Celer Network
Title	Smart Contract Audit Report
Target	SGNv2 NFT Bridge
Version	1.0
Author	Shulin Bie
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	April 29, 2022	Shulin Bie	Final Release
1.0-rc	April 22, 2022	Shulin Bie	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

Given the opportunity to review the design document and related smart contract source code of the SGNv2 NFT Bridge, 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 SGNv2 NFT Bridge

Celer Network is an Internet-scale, trust-free, and privacy-preserving platform where everyone can quickly build, operate, and use highly scalable decentralized applications. Celer Network provides unprecedented performance and flexibility through innovation in off-chain scaling techniques and incentive-aligned crypto-economics. The audited SGNv2 NFT Bridge is an innovative component of Celer Network to support NFT-related cross-chain transaction between different blockchains, which greatly enriches the Celer Network ecosystem.

Table 1.1: Basic Information of SGNv2 NFT Bridge

Item	Description
Target	SGNv2 NFT Bridge
Website	https://www.celer.network/
Type	EVM Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	April 29, 2022

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 only covers the following contracts: contracts/message/apps/

`NFTBridge.sol`, `contracts/message/apps/NFTMCN.sol`, and `contracts/message/apps/NFTPeg.sol`.

- https://github.com/celer-network/sgn-v2-contracts.git (f765841)

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 (a760605)

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

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

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 `SGNv2 NFT Bridge` 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	2	■ ■
Low	0	
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 medium-severity vulnerabilities and 1 informational recommendation.

Table 2.1: Key SGNv2 NFT Bridge Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Type Inconsistency Of INFT-Bridge::sendMsg()	Business Logic	Fixed
PVE-002	Informational	Meaningful Events For Important State Changes	Coding Practices	Fixed
PVE-003	Medium	Trust Issue Of Admin Keys	Security Features	Confirmed

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 Type Inconsistency Of INFTBridge::sendMsg()

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

Description

In the `SGNv2 NFT Bridge` component, the `MCNNFT` contract is one of the main entries for interaction with users, which supports the `Multi-Chain Native NFT` cross-chain transfers between different blockchains. In particular, one entry routine, i.e., `crossChain()`, is designed to initiate the `Multi-Chain Native NFT` cross-chain transfer on the source chain. While examining its logic, we notice there is a type inconsistency issue in its calling external interfaces.

To elaborate, we show below the related code snippet of the `MCNNFT` contract. In the `crossChain()` routine, the `Multi-Chain Native NFT` token on the source chain is burned firstly (line 62), and then the `NFTBridge::sendMsg()` routine is called (line 63) to send the cross-chain transfer message. Eventually, the corresponding `NFT` token is minted to the recipient on the destination chain. After further analysis, we notice the interface declaration of the `sendMsg()` routine (i.e., `function sendMsg(uint64 _dstChid, address _receiver, uint256 _id, string calldata _uri)external payable`) is inconsistent with its definition (i.e., `function sendMsg(uint64 _dstChid, address _sender, address _receiver, uint256 _id, string calldata _uri)external payable`) in the input parameters field, which will result in unexpected revert.

```
54     // called by user, burn token on this chain and mint same id/uri on dest chain
55     function crossChain(
56         uint64 _dstChid,
57         uint256 _id,
58         address _receiver
59     ) external payable {
```

```
        require(msg.sender == ownerOf(_id), "not token owner");
        string memory _uri = tokenURI(_id);
        _burn(_id);
        INFTBridge(nftBridge).sendMsg{value: msg.value}(_dstChid, _receiver, _id, _uri);
    }
```

Listing 3.1: `MCNNFT::crossChain()`

Recommendation Correct the above-mentioned type inconsistency.

Status The issue has been addressed by the following commit: `68df890`.

3.2 Meaningful Events For Important State Changes

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `NFTBridge`
- 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.

```
    function setDestNFT(address srcNft, uint64 dstChid, address dstNft) external
        onlyOwner {
        destNFTAddr[srcNft][dstChid] = dstNft;
    }

    // set all dest chains
    function setDestNFTs(address srcNft, uint64[] calldata dstChid, address[] calldata
        dstNft) external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destNFTAddr[srcNft][dstChid[i]] = dstNft[i];
        }
    }

    // set destTxFee
```

```
    function setTxFee(uint64 chid, uint256 fee) external onlyOwner {
        destTxFee[chid] = fee;
    }

    // set per chain id, nft bridge address
    function setDestBridge(uint64 dstChid, address dstNftBridge) external onlyOwner {
        destBridge[dstChid] = dstNftBridge;
    }

    // batch set nft bridge addresses for multiple chainids
    function setDestBridges(uint64[] calldata dstChid, address[] calldata dstNftBridge)
        external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destBridge[dstChid[i]] = dstNftBridge[i];
        }
    }

    // send all gas token this contract has to owner
    function claimFee() external onlyOwner {
        payable(msg.sender).transfer(address(this).balance);
    }
```

Listing 3.2: `NFTBridge`

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 The issue has been addressed by the following commit: `450cdaa`.

3.3 Trust Issue Of Admin Keys

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

Description

In the audited part of the `SGNv2 NFT Bridge`, there is a privileged `owner` account that plays a critical role in governing and regulating the protocol-wide operations (e.g., mint `Multi-Chain Native NFT` token arbitrarily). In the following, we show the representative functions potentially affected by the privilege of the `owner` account.

```
    function mint(
        address to,
        uint256 id,
        string memory uri
    ) external onlyOwner {
        _mint(to, id);
        _setTokenURI(id, uri);
    }
```

Listing 3.3: `MCNNFT::mint()`

```
    function setDestNFT(
        address srcNft,
        uint64 dstChid,
        address dstNft
    ) external onlyOwner {
        destNFTAddr[srcNft][dstChid] = dstNft;
    }

    ...

    function setDestBridge(uint64 dstChid, address dstNftBridge) external onlyOwner {
        destBridge[dstChid] = dstNftBridge;
    }
```

Listing 3.4: `NFTBridge::setDestNFT()&&setDestBridge()`

We emphasize that the privilege assignment is indeed necessary and consistent with the protocol design. However, it is worrisome if the privileged account is a plain EOA account. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Note that a compromised privileged account would allow the attacker to modify a number of sensitive system parameters, which directly undermines the assumption of the protocol design.

Recommendation Suggest a multi-sig account plays the privileged `owner` account to mitigate this issue. Additionally, all changes to privileged operations may need to be mediated with necessary timelocks.

Status This issue has been confirmed by the team.

4 Conclusion

In this audit, we have analyzed the SGNv2 NFT Bridge design and implementation. Celer Network is an Internet-scale, trust-free, and privacy-preserving platform where everyone can quickly build, operate, and use highly scalable decentralized applications.The audited SGNv2 NFT Bridge is an innovative component of Celer Network to support NFT-related cross-chain transaction between different blockchains, which greatly enriches the Celer Network ecosystem. 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.