



SMART CONTRACT AUDIT REPORT

for

SVM-EVM MultiToken Bridge



Prepared By: Xiaomi Huang

PeckShield
August 24, 2024

Document Properties

| | |
|----------------|-----------------------------|
| Client | BOOL |
| Title | Smart Contract Audit Report |
| Target | MultiToken Bridge |
| Version | 1.0 |
| Author | Xuxian Jiang |
| Auditors | Daisy Cao, Xuxian Jiang |
| Reviewed by | Xiaomi Huang |
| Approved by | Xuxian Jiang |
| Classification | Public |

Version Info

| Version | Date | Author(s) | Description |
|---------|-----------------|--------------|----------------------|
| 1.0 | August 24, 2024 | Xuxian Jiang | Final Release |
| 1.0-rc | August 23, 2024 | Xuxian Jiang | Release Candidate #1 |

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 MultiToken Bridge | 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 | Redundant ReentrancyGuard Inheritance in MultiTokenBridge | 11 |
| 3.2 | Trust Issue of Admin Keys | 12 |
| 4 | Conclusion | 14 |
| | References | 15 |

1 | Introduction

Given the opportunity to review the design document and related source code of the `MultiToken Bridge` 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 MultiToken Bridge

`MultiToken Bridge` proposes the much-needed bridge solutions by building upon the `Bool Network AMT` module to enable cross-chain transfers between `SatoshiVM` and EVM-compliant chains. The `Bool Network AMT` module allows for arbitrary message transmission (AMT) across heterogeneous networks. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of MultiToken Bridge

| Item | Description |
|---------------------|-----------------|
| Name | BOOL |
| Type | Smart Contract |
| Language | Solidity |
| Audit Method | Whitebox |
| Latest Audit Report | August 24, 2024 |

In the following, we show the Git repository of reviewed files and the commit hash values used in this audit.

- <https://github.com/SatoshiVM/svm-evmBridge-audits.git> (af4d033)

1.2 About PeckShield

PeckShield Inc. [7] 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 | | | | |
|--------|----------|------------|--------|-----|
| | High | Medium | Low | |
| High | Critical | High | Medium | |
| Medium | High | Medium | Low | |
| Low | Medium | Low | Low | |
| | | High | Medium | Low |
| | | Likelihood | | |

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [6]:

- 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.

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 accordingly classified into four categories, 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

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 |

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) [5], 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. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

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 design and implementation of the `MultiToken Bridge` protocol. 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 logics, 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 | 1 |  |
| Low | 0 | |
| Informational | 1 |  |
| Total | 2 | |

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 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 1 medium-severity vulnerability and 1 informational issue.

Table 2.1: Key Audit Findings

| ID | Severity | Title | Category | Status |
|---------|---------------|---|-------------------|-----------|
| PVE-001 | Informational | Redundant ReentrancyGuard Inheritance in MultiTokenBridge | Coding Practices | Confirmed |
| PVE-002 | Medium | Trust Issue Of Admin Keys | Security Features | Mitigated |

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 Redundant ReentrancyGuard Inheritance in MultiTokenBridge

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: MultiTokenBridge
- Category: Coding Practices [\[4\]](#)
- CWE subcategory: CWE-1126 [\[1\]](#)

Description

The MultiToken Bridge protocol makes good use of a number of reference contracts, such as ERC20, SafeERC20, and ReentrancyGuard, to facilitate its code implementation and organization. For example, the MultiTokenBridge smart contract has so far imported at least five reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

For example, if we examine closely the MultiTokenBridge::bridgeOut() implementation, there is a nonReentrant modifier (line 125) with the purpose of safeguarding against possible reentrancy. Our analysis indicates that this routine strictly follows the widely-adopted checks-effects-interactions practice, which makes this modifier not actually needed. With its removal, we also notice the ReentrancyGuard inheritance of MultiTokenBridge becomes unnecessary.

```
37     function bridgeOut(  
38         uint32 dstChainId,  
39         uint256 tokenId,  
40         uint256 amount,  
41         address dstRecipient,  
42         address payable refundAddress,  
43         bytes calldata customData  
44     )  
45     public  
46     payable
```

```

47     override
48     whenNotPaused(PausedType.BridgeOut)
49     nonReentrant
50     returns (bytes32 txUniqueIdentification)
51 {
52     TokenInfo memory tokenInfo = tokenIdToInfo[tokenId];
53
54     // Inputs validation
55     require(amount > 0, "ZERO_AMOUNT");
56     require(dstRecipient != address(0), "NULL_RECIPIENT");
57     _checkTokenConfig(tokenInfo);
58     ...
59 }

```

Listing 3.1: MultiTokenBridge::nonReentrant()

Recommendation Consider the removal of the redundant code with a simplified, consistent implementation.

Status The issue has been confirmed.

3.2 Trust Issue of Admin Keys

- ID: PVE-002
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: MultiTokenBridge
- Category: Security Features [3]
- CWE subcategory: CWE-287 [2]

Description

In MultiToken Bridges, there is a privileged owner account that plays a critical role in governing and regulating the system-wide operations (e.g., configure various parameters and pause/unpause bridges). It also has the privilege to control or govern the flow of assets managed by this protocol. Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```

279     function fixTokenConfig(uint256 tokenId) external onlyOwner {
280         // Can only fix configs for existing tokenId, i.e. token exists and the config
           is not fixed
281         require(
282             tokenIdToInfo[tokenId].token != address(0) &&
283             !tokenIdToInfo[tokenId].isConfigFixed,
284             "INVALID_TOKEN_ID"
285         );
286

```

```

287     tokenIdToInfo[tokenId].isConfigFixed = true;
288 }
289
290 /**
291  * @notice In case of an incorrect token configuration, the owner can set a new
292  *         token Id with the correct configuration
293  * @param tokenId The globally unique token Id
294  * @param isNativeAsset Indicates whether the token is a native asset (lock & unlock
295  *         or mint & burn)
296  * @param isGasToken Indicates whether the token is a gas token
297  * @param token The address of the token
298  * @param commonDecimal The globally shared decimal, used for the cross-chain
299  *         conversion
300  */
301 function setNewTokenId(
302     uint256 tokenId,
303     bool isNativeAsset,
304     bool isGasToken,
305     address token,
306     uint8 commonDecimal
307 ) external onlyOwner {
308     ...
309 }

```

Listing 3.2: Example Privileged Functions in MultiTokenBridge

Note that if the privileged owner account is a plain EOA account, this may be worrisome and pose counter-party risk to the exchange users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated as the team makes use of a multisig to act as the privileged owner.

4 | Conclusion

In this audit, we have analyzed the design and implementation of the MultiToken Bridge protocol, which proposes the much-needed bridge solutions by building upon the Bool Network AMT module to enable cross-chain transfers between SatoshiVM and EVM-compliant chains. The Bool Network AMT module allows for arbitrary message transmission (AMT) across heterogeneous networks. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that Solidity-based 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-1126: Declaration of Variable with Unnecessarily Wide Scope. <https://cwe.mitre.org/data/definitions/1126.html>.
- [2] MITRE. CWE-287: Improper Authentication. <https://cwe.mitre.org/data/definitions/287.html>.
- [3] MITRE. CWE CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [4] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [5] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [7] PeckShield. PeckShield Inc. <https://www.peckshield.com>.