SECURITY AUDIT REPORT

for

cBridge (Aptos)

Prepared By: Xiaomi Huang

PeckShield

October 18, 2022

Document Properties

Client	Celer Network
Title	Security Audit Report
Target	cBridge Aptos
Version	1.0-rc
Author	Xiaotao Wu
Auditors	Xiaotao Wu, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author(s)	Description
1.0-rc	October 18, 2022	Xiaotao Wu	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 cBridge Aptos 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 Delay Enforcement Bypass in execute_delayed_transfer() 11

3.2 Suggested Consistent Handling Between Aptos and EVM 12

3.3 Strengthened Owner Authentication in peg_bridge Module 14

3.4 Trust Issue of Admin Keys 15

3.5 Addition of Delay Period Update in delayed_transfer Module 16

4 Conclusion **18**

References **19**

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the `Aptos` support in the `cBridge` 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 cBridge Aptos

`Celer cBridge` introduces the cross-chain token bridging experience with deep liquidity for users, highly efficient and easy-to-use liquidity management for both `cBridge` node operators and liquidity providers who do not want to operate `cBridge` nodes, and developer-oriented features such as general message bridging for cases like cross-chain `DEXs` and `NFTs`. The audited `cBridge Aptos` is the extension of `cBridge` in the `Aptos` blockchain, and currently supports a canonical token bridge model. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of cBridge Aptos

Item	Description
Name	Celer Network
Website	https://www.celer.network/
Type	Aptos
Language	Move
Audit Method	Whitebox
Latest Audit Report	October 18, 2022

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

- https://github.com/celer-network/cbridge-aptos.git (d18e182)

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

- https://github.com/celer-network/cbridge-aptos.git (0969711)

1.2 About PeckShield

PeckShield Inc. [10] 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

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [9]:

- 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.

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

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) [8], 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 `cBridge Aptos` implementations. 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	1	■
Medium	1	■
Low	3	■ ■ ■
Informational	0	
Total	5	

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 high-severity vulnerability, 1 medium-severity vulnerability, and 3 low-severity vulnerabilities.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	High	Delay Enforcement Bypass in execute_-delayed_transfer()	Business Logic	Fixed
PVE-002	Low	Suggested Consistent Handling Between Aptos and EVM	Coding Practices	Fixed
PVE-003	Low	Strengthened Owner Authentication in peg_bridge Module	Business Logic	Fixed
PVE-004	Medium	Trust Issue of Admin Keys	Security Features	Confirmed
PVE-005	Low	Addition of Delay Period Update in delayed_transfer Module	Business Logic	Fixed

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 Delay Enforcement Bypass in execute_delayed_transfer()

- ID: PVE-001
- Severity: High
- Likelihood: High
- Impact: Medium
- Target: delayed_transfer.move
- Category: Business Logic [7]
- CWE subcategory: CWE-837 [3]

Description

The cBridge contracts add a much-needed feature for fine-grained risk controls, i.e., the daily transfer volume restriction and the per-transaction transfer volume restriction. While examining the implementation of the delayed_transfer contract, we notice the intended enforcement of delay period for large transfers may be bypassed.

To elaborate, we show below the code snippet of the related execute_delayed_transfer() function. It comes to our attention that the current implementation does not enforce the intended locking period. In other words, both peg_bridge::execute_delay_transfer() and vault::execute_delay_transfer() functions can be executed immediately after the mint/withdraw operations.

```
    public(friend) fun execute_delayed_transfer(id: vector<u8>): (address, string::
        String, u64) acquires DelayedTransferState {
        let state = borrow_global_mut<DelayedTransferState>(@celer);
        assert!(table::contains(&state.delay_map, id), DELAYED_TRANSFER_NOT_EXIST);
        let v = table::remove(&mut state.delay_map, id);
        event::emit_event<DelayedTransferExecutedEvent>(
            &mut state.delayed_transfer_executed_event,
            DelayedTransferExecutedEvent {
                id,
                receiver: v.receiver,
                coin_id: v.coin_id,
                amt: v.amt,
            },
        );
        (v.receiver, v.coin_id, v.amt)
```

```
    }
```

Listing 3.1: `delayed_transfer::execute_delayed_transfer()`

Recommendation Only allows a delayed transfer to be executed if the locking period expires. An example revision is shown as follows. In addition, the affected `execute_delayed_transfer()` function needs to be strengthened to have the `friend` declaration.

```
    public(friend) fun execute_delayed_transfer(id: vector<u8>): (address, string::String, u64) acquires DelayedTransferState {
        let state = borrow_global_mut<DelayedTransferState>(@celer);
        assert!(table::contains(&state.delay_map, id), DELAYED_TRANSFER_NOT_EXIST);
        let v = table::remove(&mut state.delay_map, id);
        assert!(v.blk_ts > timestamp::now_seconds() + state.delay_period,
            DELAYED_TRANSFER_STILL_LOCKED);
        event::emit_event<DelayedTransferExecutedEvent>(
            &mut state.delayed_transfer_executed_event,
            DelayedTransferExecutedEvent {
                id,
                receiver: v.receiver,
                coin_id: v.coin_id,
                amt: v.amt,
            },
        );
        (v.receiver, v.coin_id, v.amt)
    }
```

Listing 3.2: `delayed_transfer::execute_delayed_transfer()`

Status The issue has been fixed by this commit: `1a1320d`.

3.2 Suggested Consistent Handling Between Aptos and EVM

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Multiple contracts`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1099 [1]

Description

The `bridge` contract allows to update the `signers` if the transaction has been signed by the current `signers` with at least $\frac{2}{3}$ of the total decision-making powers. While examining the `update_signers()` routine, we notice the current implementation is inconsistent with the `EVM` counterpart(s).

To elaborate, we show below its implementation. Specifically, the variable `state.trigger_time` is assigned with the `cur_blk_time` value of the current `Aptos` block, instead of the `blk_time` value in `EVM`. (line 154).

```
142     public entry fun update_signers(pbmsg: vector<u8>, sigs: vector<vector<u8>>)
            acquires BridgeState {
143         let (blk_time, new_signers, total_power) = decode_update_signers_pb(pbmsg);
144         assert!(total_power > 0u128, INVALID_TOTAL_POWR);
145         let cur_blk_time = timestamp::now_seconds();
146         let state = borrow_global_mut<BridgeState>(@celer);
147         assert!(blk_time <= cur_blk_time + 3600 == true, LESS_THEN_CUR_BLK_TIME);
148         assert!(blk_time > state.trigger_time == true, TRIGGER_TIME_TOO_FAR);
149         let sign_data = state.domain_prefix;
150         vector::append(&mut sign_data, b".UpdateSigners");
151         vector::append(&mut sign_data, pbmsg);
152         assert!(verify_sig_by_signers(sign_data, sigs, state) == true, VERIFY_SIG_FAIL);
153         state.signers = new_signers;
154         state.trigger_time = cur_blk_time;
155         state.total_power = total_power;
156         event::emit_event<SignersUpdatedEvent>(
157             &mut state.signers_updated_events,
158             SignersUpdatedEvent {
159                 signers: new_signers,
160             },
161         );
162     }
```

Listing 3.3: `bridge::update_signers()`

Note this inconsistent handling issue also exists in the `peg_bridge::mint()/burn()`, `vault::deposit()/withdraw()` functions.

Recommendation Use the same handling logic with the `EVM` blockchain for above mentioned functions.

Status The issue has been fixed by this commit: `0969711`.

3.3 Strengthened Owner Authentication in peg_bridge Module

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: peg_bridge.move
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The peg_bridge contract provides a public add_token_capabilities() function to add burn_cap, burn_cap , and burn_cap capabilities for an account associated with a specified CoinType. Our analysis with this routine shows its current implementation can be improved.

To elaborate, we show below the full implementation of the add_token_capabilities() function. Specifically, it should only allow the type_info::account_address(&type_info::type_of<CoinType>()) address to set various capabilities for a specified CoinType, instead of currently allowing anyone to call this function. The same issue is also applicable to other routines with signer as its arguments, including rm_token_capabilities() and add_token().

```
    public entry fun add_token_capabilities<CoinType>(
        owner: &signer,
        burn_cap: coin::BurnCapability<CoinType>,
        freeze_cap: coin::FreezeCapability<CoinType>,
        mint_cap: coin::MintCapability<CoinType>
    ) {
        let addr = signer::address_of(owner);
        assert!(exists<Capabilities<CoinType>>(addr) == false,
            CAPABILITIES_ALREADY_EXIST);
        let coin_cap = Capabilities<CoinType> {
            burn_cap,
            freeze_cap,
            mint_cap,
        };
        move_to(owner, coin_cap);
    }
```

Listing 3.4: peg_bridge::add_token_capabilities()

Recommendation Add the necessary owner validation in the above-mentioned add_token_capabilities function.

Status The issue has been fixed by this commit: 7eab5c5.

3.4 Trust Issue of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `Multiple contracts`
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In `cBridge Aptos`, there is a privileged account, i.e., `@celer`. This account plays a critical role in governing and regulating the system-wide operations (e.g., add/remove `governor`, add/remove `pauser`, reset `signers` for `cBridge`, update `delay_period`, set `epoch_length`, etc.). Our analysis shows that this privileged account needs to be scrutinized. In the following, we use the `admin_manager` contract as an example and show the representative functions potentially affected by the privileges of the `@celer` account.

```
    public entry fun add_governor(owner: &signer, governor: address) acquires AdminState {
        let addr = signer::address_of(owner);
        assert!(addr == @celer, NOT_OWNER);
        assert!(exists<AdminState>(addr), ADMIN_STATE_NOT_EXIST);
        let state = borrow_global_mut<AdminState>(addr);
        assert!(table::contains(&state.governors, governor) == false, GOVERNOR_EXIST);
        table::add(&mut state.governors, governor, true);
    }

    public entry fun rm_governor(owner: &signer, governor: address) acquires AdminState {
        let addr = signer::address_of(owner);
        assert!(addr == @celer, NOT_OWNER);
        assert!(exists<AdminState>(addr), ADMIN_STATE_NOT_EXIST);
        let state = borrow_global_mut<AdminState>(addr);
        assert!(table::contains(&state.governors, governor), GOVERNOR_NOT_EXIST);
        table::remove(&mut state.governors, governor);
    }

    public entry fun add_pauser(owner: &signer, pauser: address) acquires AdminState {
        let addr = signer::address_of(owner);
        assert!(addr == @celer, NOT_OWNER);
        assert!(exists<AdminState>(addr), ADMIN_STATE_NOT_EXIST);
        let state = borrow_global_mut<AdminState>(addr);
        assert!(table::contains(&state.pausers, pauser) == false, PAUSER_EXIST);
        table::add(&mut state.pausers, pauser, true);
    }

    public entry fun rm_pauser(owner: &signer, pauser: address) acquires AdminState {
        let addr = signer::address_of(owner);
```

```
        assert!(addr == @celer, NOT_OWNER);
        assert!(exists<AdminState>(addr), ADMIN_STATE_NOT_EXIST);
        let state = borrow_global_mut<AdminState>(addr);
        assert!(table::contains(&state.pausers, pauser), PAUSER_NOT_EXIST);
        table::remove(&mut state.pausers, pauser);
    }
```

Listing 3.5: admin_manager.move

We understand the need of the privileged functions for proper cBridge Aptos operations, but at the same time the extra power to the @celer may also be a counter-party risk to the cBridge Aptos users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Make the list of extra privileges granted to cBridge Aptos explicit to cBridge Aptos users.

Status This issue has been confirmed.

3.5 Addition of Delay Period Update in delayed_transfer Module

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: delayed_transfer.move
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The Celer cBridge protocol is no exception. Specifically, if we examine the delayed transfer feature, it has defined a number of protocol-wide risk parameters, such as delay_period and delay_threshold. In the following, we show the corresponding routines that allow for their changes.

To elaborate, we show below the full implementation of the add_token() function. Specifically, it allows for the dynamic addition of a new token with customized delay_threshold and vol_cap. However, it comes to our attention the setter for the important delay_period is currently missing.

```
    public entry fun add_token<CoinType>(
        governor: &signer,
        min_burn: u64,
        max_burn: u64,
        delay_threshold: u64,
        vol_cap: u64) acquires PegBridgeState {
```

```
        assert!(admin_manager::is_governor(signer::address_of(governor)), NOT_GOVERNOR);
        assert!(exists<PegBridgeState>(@celer), STATE_NOT_EXIST);
        let state = borrow_global_mut<PegBridgeState>(@celer);
        let coin_id = type_info::type_name<CoinType>();
        if (table::contains(&state.coin_map, coin_id)) {
            let cur_state = table::borrow_mut(&mut state.coin_map, coin_id);
            cur_state.min_burn = min_burn;
            cur_state.max_burn = max_burn;
            cur_state.delay_threshold = delay_threshold;
            cur_state.vol_cap = vol_cap;
        } else {
            table::add(&mut state.coin_map, coin_id, CoinConfig {
                min_burn,
                max_burn,
                delay_threshold,
                vol_cap,
            });
        }
    }
```

Listing 3.6: `peg_bridge::add_token()`

These parameters define various aspects of the protocol operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the update logic on the `delay_period` parameter needs to be supported.

Recommendation Add necessary setters to support dynamic reconfiguration of important protocol-wide parameters.

Status The issue has been fixed by this commit: `1a1320d`.

4 Conclusion

In this audit, we have analyzed the `cBridge Aptos` design and implementation. The audited `cBridge Aptos` is the extension of `cBridge` in `Aptos` blockchain, 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-1099: Inconsistent Naming Conventions for Identifiers. https://cwe.mitre.org/data/definitions/1099.html.

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

[3] MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

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

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

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

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

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

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

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