Tetragaze

Smart contract audit

Celer

Tetragaze

October 22, 2022 | v 2.0

Pass

Tetragaze Security Team has concluded that there are no issues that can have an impact on contract security. The contract is well written and is production-ready.

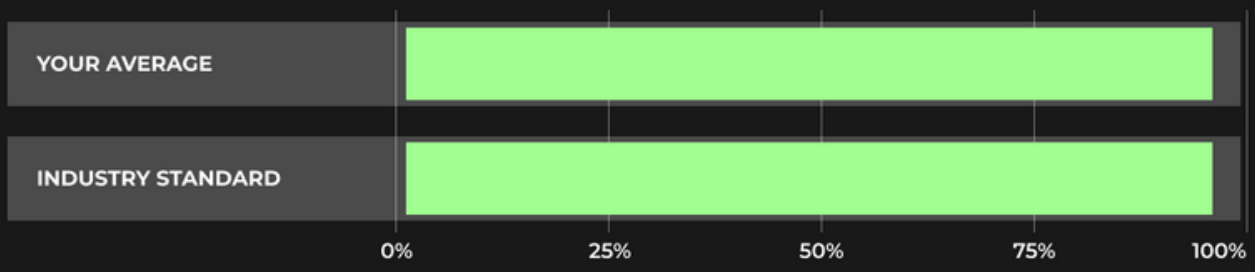
Technical summary

This document outlines the overall security of the cBridge smart contracts, evaluated by Tetragaze Blockchain Security team.
The scope of this audit was to analyze and document the cBridge smart contract codebase for quality, security, and correctness

Contract Status

There were no critical issues found during the audit.

Testable Code



Testable code is 95%, which is close to the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the cBridge team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of content

Auditing Strategy and Techniques Applied

Requirements: FIP8 spec

During the review process, we made sure to carefully check that the token contract:

- Adheres to established Token standards and ensures effective implementation;
- Ensures that documentation and code comments accurately reflect the logic and behavior of the code;
- Ensures that token distribution is in line with calculations;
- Utilizes best practices to efficiently use gas, avoiding unnecessary waste;
- Employs methods that are safe from reentrance attacks and immune to the latest vulnerabilities;
- Follows best practices for code readability

Tetragaze team of expert pentesters and smart contract developers carefully evaluated the repository for security issues, code quality, and compliance with specifications and best practices. They conducted a line-by-line review, documenting any issues that were identified during the process. Special care was taken to ensure that the review was thorough and comprehensive.

1	Due diligence in assessing the overall code quality of the codebase.	2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.	4	Thorough, manual review of the codebase, line-by-line.

Summary

In this audit, we have examined the design and implementation of the cBridge Aptos extension for the Aptos blockchain. The cBridge Aptos is a valuable addition to the Celer Network ecosystem. We found that the current code base is well-organized and structured. Any issues that were identified were promptly addressed. However, we must note that smart contracts are still in the early stages of development. We welcome any feedback or suggestions on our audit findings, methodology, or potential gaps in scope/coverage to improve this report

Structure and Organization of Document

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

Complete Analysis

Delay Enforcement Bypass in execute_delayed_transfer()

High

The cBridge contracts offer a valuable feature for precise risk management, including daily and per-transaction transfer volume restrictions. However, during our review of the delayed_transfer contract, we discovered that the intended delay period for large transfers may not be properly enforced.

To explain further, the code snippet of the execute_delayed_transfer() function does not effectively enforce the intended lock period. As a result, both the peg_bridge::execute_delay_transfer() and vault::execute_delay_transfer() functions can be immediately executed after mint/withdraw operations.

```
80      public ( friend ) fun execute_delayed_transfer (id: vector <u8 >): ( address , string ::
           String , u64) acquires DelayedTransferState {
81          let state = borrow_global_mut < DelayedTransferState >( @celer );
82          assert! ( table :: contains (& state . delay_map , id), DELAYED_TRANSFER_NOT_EXIST );
83          let v = table :: remove (& mut state . delay_map , id);
84          event :: emit_event < DelayedTransferExecutedEvent >(
85          & mut state . delayed_transfer_executed_event ,
86          DelayedTransferExecutedEvent {
87              id ,
88              receiver : v. receiver ,
89              coin_id : v. coin_id ,
90              amt : v.amt ,
91          },
92      );
93      (v. receiver , v. coin_id , v. amt)
```

Recomendation:

The execute_delayed_transfer() function must be declared as a friend in order for a delayed transfer to be executed once the locking period has expired. An example revision is provided below. It is also necessary to strengthen the execute_delayed_transfer() function to ensure proper execution.

Complete Analysis

Trust Issue of Admin Keys

Medium

The @celer account in cBridge Aptos holds a privileged position, and is responsible for overseeing and controlling various system-wide functions such as adding or removing governors, pausers, and resetting signers for cBridge. It can also update the delay period and set the epoch length. Our analysis has determined that this privileged account requires careful examination. As an example, we will use the admin_manager contract to demonstrate the functions that may be impacted by the privileges of the @celer account.

```
37    public entry fun add_governor ( owner : &signer , governor : address ) acquires AdminState
          {
38        let addr = signer :: address_of ( owner );
39        assert! ( addr == @celer , NOT_OWNER );
40        assert! ( exists < AdminState >( addr ), ADMIN_STATE_NOT_EXIST );
41        let state = borrow_global_mut < AdminState >( addr );
42        assert! ( table :: contains (& state . governors , governor ) == false , GOVERNOR_EXIST
);
43        table :: add (& mut state . governors , governor , true );
44    }
45
46    public entry fun rm_governor ( owner : & signer , governor : address ) acquires AdminState
          {
47        let addr = signer :: address_of ( owner );
48        assert! ( addr == @celer , NOT_OWNER );
49        assert! ( exists < AdminState >( addr ), ADMIN_STATE_NOT_EXIST );
50        let state = borrow_global_mut < AdminState >( addr );
51        assert! ( table :: contains (& state . governors , governor ), GOVERNOR_NOT_EXIST );
52        table :: remove (& mut state . governors , governor );
53    }
54
55    public entry fun add_pauser ( owner : & signer , pauser : address ) acquires AdminState {
56        let addr = signer :: address_of ( owner );
57        assert! ( addr == @celer , NOT_OWNER );
58        assert! ( exists < AdminState >( addr ), ADMIN_STATE_NOT_EXIST );
59        let state = borrow_global_mut < AdminState >( addr );
60        assert! ( table :: contains (& state . pausers , pauser ) == false , PAUSER_EXIST );
61        table :: add (& mut state . pausers , pauser , true );
62    }
63
64    public entry fun rm_pauser ( owner : & signer , pauser : address ) acquires AdminState {
65        let addr = signer :: address_of ( owner );
66        assert! ( addr == @celer , NOT_OWNER );
67        assert! ( exists < AdminState >( addr ), ADMIN_STATE_NOT_EXIST );
68        let state = borrow_global_mut < AdminState >( addr );
69        assert! ( table :: contains (& state . pausers , pauser ), PAUSER_NOT_EXIST );
70        table :: remove (& mut state . pausers , pauser );
71    }
```

Complete Analysis

Suggested Consistent Handling Between Aptos and EVM

Low

The bridge contract allows for the updating of signers if the transaction has been signed by the current signers with at least 23 decision-making powers. During our examination of the update_signers() routine, we observed that the current implementation does not align with its EVM counterpart(s). To explain further, the variable state.trigger_time is assigned the value of cur_blk_time for the current Aptos block instead of using the blk_time value from 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    }
```

Recomendation:

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

Complete Analysis

Update to include delay period in the delayed_transfer module

Low

The Celer cBridge protocol, like many other DeFi protocols, allows for the dynamic configuration of various system-wide parameters. One such feature is the delayed transfer feature, which has defined protocol-wide risk parameters such as delay_period and delay_threshold. The add_token() function, which allows for the dynamic addition of a new token with customized delay_threshold and vol_cap, has been implemented. However, we have noticed that the setter for the crucial delay_period is currently absent.

```
165    public entry fun add_token < CoinType >(
166        governor : & signer ,
167        min_burn : u64 ,
168        max_burn : u64 ,
169        delay_threshold : u64 ,
170        vol_cap : u64 ) acquires PegBridgeState {
171        assert! ( admin_manager :: is_governor ( signer :: address_of ( governor )),
NOT_GOVERNOR );
172        assert! ( exists < PegBridgeState >( @celer ), STATE_NOT_EXIST );
173.       let state = borrow_global_mut < PegBridgeState >( @celer );
174        let coin_id = type_info :: type_name < CoinType >() ;
175        if ( table :: contains (& state . coin_map , coin_id )) {
176             let cur_state = table :: borrow_mut (& mut state . coin_map , coin_id );
177             cur_state . min_burn = min_burn ;
178             cur_state . max_burn = max_burn ;
179             cur_state . delay_threshold = delay_threshold ;
180             cur_state . vol_cap = vol_cap ;
181        } else {
182            table :: add (& mut state . coin_map , coin_id , CoinConfig {
183                min_burn ,
184                max_burn ,
185                delay_threshold ,
186                vol_cap ,
187            });
188        }
189    }
```

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.

Complete Analysis

Recomendation:

Implement setter functions to allow for the dynamic adjustment of significant protocol-level parameters during operation.

We are thankful for the opportunity to collaborate with the cBridge team.

This document's statements should not be taken as investment or legal advice, and the authors cannot be held responsible for any decisions made based on them.

It is suggested by the Tetragaze Security Team that the cBridge team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.