Security Audit

Date : 2024-04-28

Zus Contracts

Executive Summary

Type	Smart Contract Audit
Audit Timeline	9 days
Runtime Environment	EVM
Languages	Solidity

Scope

Github Repository Link : https://github.com/0chain/token_bridge_SC

/contracts/Bridge.sol

/contracts/Token.sol

/contracts/Authorizable.sol

/contracts/Authorizers.sol

/contracts/interfaces/.*

Summary of Findings

ID	Name	Description	Severity
M-01	Inconsistent ERC20 Token Behavior Handling	Unsafe use of transfer/transferFrom/approve	Medium
M-02	Block Gas Limit for Bridge::resetUserNonceMinted() might exceed on function call	Function call might revert because of unbounded iteration	Medium
G-01	It costs more gas to initialize variables to zero than to let the default of zero be applied	Variables initialized to 0 is redundant	Gas
G-02	Change function visibility from public to external	Changing Function visibility can help save some gas on function calls	Gas
G-03	For Loop can be optimized in Bridge::resetUserNonceMinted()	For loop can be optimized	Gas
G-04	Using `calldata` instead of `memory`	Use calldata instead of memory	Gas

	for read-only arguments saves gas		
G-05	Functions guaranteed to revert when called by normal users can be marked `payable`	Make the functions Bridge::withdraw() and Bridge::resetUserNonceMinted() as payable	Gas
I-01	Incorrect or Misleading Documentation	Incorrect comments have been added in the code	Informational
I-02	Redundant Event Emission	Two events emitting the same values	Informational

Findings

[M-01] Inconsistent ERC20 Token Behavior Handling

Severity: Medium

Location : Bridge.sol

Description: The contract directly uses token.transfer and token.transferFrom methods, which could lead to issues with tokens that do not follow the **ERC20** standard return value convention. While the contract does check for failures by requiring that these methods return **true**, some tokens might **revert on failure** instead of returning **false.** Also its using the **approve**() function that is prone to **frontrunning** . Use **safeApprove**() instead.

Impact

This behavior can cause incompatibilities with certain **ERC20** tokens that only **revert** on errors without returning a **boolean value.** Also the **approve** function can be frontrunned.

Recommendation
To ensure compatibility across all types of ERC20 tokens, including those that revert on failure, integrate **OpenZeppelin's SafeERC20** library. This library provides methods like **safeTransfer** and **safeTransferFrom**, which handle both returning false and reverting, making token interactions safer and more predictable.

Proof of Concept

```
require(
            token.transfer(args_.to, args_.amount),
            "Bridge: transfer out of pool failed"
        );
```

```
        token.approve(address(this), tokenBalance);
        return token.transferFrom(address(this), owner(), tokenBalance);
```

‘

Github Permalink :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L163-#L164
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L151

Similar Findings :
https://solodit.xyz/issues/m-04-use-safetransfer-instead-of-transfer-code4rena-yield-yield-micro-contest-1-git

[M-02] Block Gas Limit for Bridge::resetUserNonceMinted() might exceed

Severity: Medium

Location : Bridge.sol

Description: The function **resetUserNonceMinted**() contains a for loop over the mapping userNonceMinted . If **userNonceMinted** would be too big then the loop would run out of gas and **resetUserNonceMinted**() would revert. This would mean that **resetUserNonceMinted**() cannot be executed anymore.

Impact
The reset nonce function cant be executed if the function is reverting everytime.

Recommendation
Consider resetting the nonce for a single index at a time or it can also be done for a range of indexes at a time. This would allow the nonce to be reset without exceeding the block gas limit.

Proof of Concept :

```
    function resetUserNonceMinted() public onlyOwner { // @audit make it
external go1
        // slither-disable-start costly-loop
        uint256 counter = userAddressesCounter;
        for (uint i ; i <= counter; ) { // @audit GO the loop can be
changed to save gas cost
            delete userNonceMinted[userAddresses[i]];
            unchecked {
                ++i;
            }
        }
        // slither-disable-end costly-loop
    }
```

Github Permalink :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L63-L69

Similar Findings :
https://solodit.xyz/issues/m-11-block-usage-of-addcuratedpool-code4rena-spartan-protocol-spartan-protocol-contest-git

[G-01] It costs more gas to initialize variables to zero than to let the default of zero be applied

Severity: Gas

Location : Bridge.sol

Description
Solidity does not recognize null as a value, so uint variables are initialized to zero. Setting a uint variable to zero is redundant and can waste gas.

Impact
The contract size will be a little more and the gas cost required will be more .

Recommendation
No need to initialize the variables to 0

POC:

```
    /// @notice A global burn nonce. Its first value should be 1, not 0
    uint256 private burnNonce = 0; // @audit GO no need to initialize this
again

    /// @notice Amount of user addresses, which at least once executed
'mint' function
    uint256 private userAddressesCounter = 0; // @audit dont initilize
this as 0
```

Github Permalink :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L22
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L25

Similar Findings :
https://solodit.xyz/issues/g-07-it-costs-more-gas-to-initialize-variables-to-zero-than-to-let-the-default-of-zero-be-applied-code4rena-joyn-joyn-contest-git

[G-02] Change function visibility from public to external

Severity: Gas

Location : Bridge.sol

Description
For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge

Recommendation
Change the visibility of the given functions to external to save some gas

POC
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L56
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L63
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L74

[G-03] For Loop can be optimized in Bridge::resetUserNonceMinted()

Severity: Gas

Location : Bridge.sol

Description
The function **resetUserNonceMinted** uses a for-loop to iterate over all user addresses and reset their associated nonces. This loop accesses the storage for each iteration, which is costly in terms of gas.

Recommendation
Cache **'userAddressesCounter'** as reading variable from memory is much less gas consuming than reading from storage. Also don't initialize 'i' as its by default initialized to 0. Also increment 'i' in an unchecked block as that also saves gas and overflowing i is virtually impossible. Consider changing the loop to something like this :-

```
 function resetUserNonceMinted() public onlyOwner { // @audit make it
external go1
        // slither-disable-start costly-loop
        uint256 counter = userAddressesCounter;
        for (uint i ; i <= counter; ) { // @audit GO the loop can be
changed to save gas cost
            delete userNonceMinted[userAddresses[i]];
            unchecked {
                ++i;
            }
        }
        // slither-disable-end costly-loop
    }
```

POC
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L63-L69

[G-04] Using `calldata` instead of `memory` for read-only arguments saves gas

Severity: Gas

Location : Bridge.sol , Token.sol

Description
Use calldata for arguments that are not changed while the function calls.

Recommendation
Change memory to calldata in the function parameters.

POC :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L87
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L133
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Token.sol#L13

[G-05] Functions guaranteed to revert when called by normal users can be marked `payable`

Severity: Gas

Location : Bridge.sol

Description
If a function modifier such as **onlyOwner** is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Recommendation
Make the functions Bridge::withdraw() and Bridge::resetUserNonceMinted() as payable to save some gas

POC :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L159
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L63

[I-01] Incorrect or Misleading Documentation

Severity: Informational

Location: Bridge.sol

Description
The documentation for **'userNonceMinted'** mapping is not correct. The documentation states that the mapping stores the burn nonce whereas the implementation of the mapping is done in the mint function.

Recommendation
Correct the documentation that states that the mapping contains a burn nonce attached

POC :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L31

[I-02] Redundant Event Emission

Severity: Low Risk

Location: Bridge.sol

Description
The contract emits two very similar events, **Burned** and **BurnedFullIndex**, after burning tokens, which could be consolidated into a single event to streamline event handling and reduce gas costs associated with logging multiple events.

Recommendation
Consider merging these events into one that captures all necessary data, optimizing event emission and simplifying frontend interaction.

POC :
https://github.com/0chain/token_bridge_SC/blob/master/contracts/Bridge.sol#L95-L96