---

title: Boss Bridge Audit Report author: Mauro Júnior date: February 14, 2025 header-includes:

- \usepackage
- \usepackage

\begin \centering \begin[h] \centering \includegraphics[width=0.5\textwidth] \end \vspace*{2cm} {\Huge\bfseries Boss Bridge Initial Audit Report\par} \vspace{1cm} {\Large Version 0.1\par} \vspace{2cm} {\Large\itshape Mauro\par} \vfill {\large \today\par} \end

\maketitle

Boss Bridge Audit Report

Prepared by: Mauro Júnior Lead Auditors:

- [Mauro Júnior]

Assisting Auditors:

- None

Table of contents

▶ See table

About YOUR_NAME_HERE

Disclaimer

The YOUR_NAME_HERE 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

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

Audit Details

The findings described in this document correspond the following commit hash:

```
07af21653ab3e8a8362bf5f63eb058047f562375
```

Scope

```
#-- src
|   #-- L1BossBridge.sol
|   #-- L1Token.sol
|   #-- L1Vault.sol
|   #-- TokenFactory.sol
```

Protocol Summary

The Boss Bridge is a bridging mechanism to move an ERC20 token (the "Boss Bridge Token" or "BBT") from L1 to an L2 the development team claims to be building. Because the L2 part of the bridge is under construction, it was not included in the reviewed codebase.

The bridge is intended to allow users to deposit tokens, which are to be held in a vault contract on L1. Successful deposits should trigger an event that an off-chain mechanism is in charge of detecting to mint the corresponding tokens on the L2 side of the bridge.

Withdrawals must be approved operators (or "signers"). Essentially they are expected to be one or more off-chain services where users request withdrawals, and that should verify requests before signing the data users must use to withdraw their tokens. It's worth highlighting that there's little-to-no on-chain mechanism to verify withdrawals, other than the operator's signature. So the Boss Bridge heavily relies on having robust, reliable and always available operators to approve withdrawals. Any rogue operator or compromised signing key may put at risk the entire protocol.

Roles

- Bridge owner: can pause and unpause withdrawals in the `L1BossBridge` contract. Also, can add and remove operators. Rogue owners or compromised keys may put at risk all bridge funds.

- User: Accounts that hold BBT tokens and use the `L1BossBridge` contract to deposit and withdraw them.
- Operator: Accounts approved by the bridge owner that can sign withdrawal operations. Rogue operators or compromised keys may put at risk all bridge funds.

Executive Summary

Issues found

Severity	**Number of issues found**
High	4
Medium	1
Low	0
Info	2
Gas	0
Total	7

Findings

Highs

[H-1] Arbitrary `from` issue in `L1BossBridge:depositTokensToL2` function, causing a attacker to call this functions with params `from`: any address that has approved tokens to bridge and steal their tokens by doing a deposit `to`: the attacker address

Description There is an arbitrary `from` in the `depositTokensToL2` function, that allows any user to call it with a `from` address of any account that has approved tokens, With this feature, attackers could just call this function to move tokens from a victim account which has a balance greater than zero, and since this will move the tokens to the vault, they can assign their `L2recipient` address of themselves to steal the funds.

Proof of Concept

1. User calls deposits tokens to L2
2. User passes as from parameter an address of some user that has approved tokens to the bridge.
3. User passes as recipient parameter his own address
4. User steals other tokens.

▶ PoC

Recommended Mitigations Consider adding a replay protection mechanism in the withdraw functionality, like nonces or deadlines.

[H-4] The `L1BossBridge::sendToL1` function allows arbitrary calls and thisenables users to call `L1Vault::approveTo` and give themselves infinite allowance of vault funds

Description The `L1BossBridge` contract includes the `sendToL1` function that, if called with a valid signature by an operator, can execute arbitrary low-level calls to any given target. Because there's no restrictions neither on the target nor the calldata, this call could be used by an attacker to execute sensitive contracts of the bridge. For example, the `L1Vault` contract.

The `L1BossBridge` contract owns the `L1Vault` contract. Therefore, an attacker could submit a call that targets the vault and executes is `approveTo` function, passing an attacker-controlled address to increase its allowance. This would then allow the attacker to completely drain the vault.

It's worth noting that this attack's likelihood depends on the level of sophistication of the off-chain validations implemented by the operators that approve and sign withdrawals. However, we're rating it as a High severity issue because, according to the available documentation, the only validation made by off-chain services is that "the account submitting the withdrawal has first originated a successful deposit in the L1 part of the bridge". As the next PoC shows, such validation is not enough to prevent the attack.

Proof of Concept

To reproduce, include the following test in the `L1BossBridge.t.sol` file:

▶ PoC

Recommended Mitigations

Consider disallowing attacker-controlled external calls to sensitive components of the bridge, such as the `L1Vault` contract.

Mediums

[M-1] Withdrawals are prone to unbounded gas consumption due to return bombs

Description During withdrawals, the L1 part of the bridge executes a low-level call to an arbitrary target passing all available gas. While this would work fine for regular targets, it may not for adversarial ones.

In particular, a malicious target may drop a return bomb (https://github.com/nomad-xyz/ExcessivelySafeCall) to the caller. This would be done by returning an large amount of returndata in the call, which Solidity would copy to memory, thus increasing gas costs due to the expensive memory operations. Callers unaware of this risk may not set the transaction's gas limit sensibly, and therefore be tricked to spent more ETH than necessary to execute the call.

Recommended Mitigations If the external call's returndata is not to be used, then consider modifying the call to avoid copying any of the data. This can be done in a custom implementation, or reusing external libraries such as this one (https://github.com/nomad-xyz/ExcessivelySafeCall).

Informational

[I-1] In `L1BossBridge::DEPOSIT_LIMIT` this is a constant variable should be constant since it doesn't change

Description The `DEPOSIT_LIMIT` is a variable that doesn't change everytime it's used and it isn't updated, and also it's set to a single value.

Recommended Mitigations It should be added the `constant` keyword to save gas and not be keeping reading from storage everytime it is being called.

[I-2] Insufficient test covarage.

Running tests... | File | % Lines | % Statements | % Branches | % Funcs | | -------------------- | -------------- | -------------- | ------------- | ------------- | | src/L1BossBridge.sol | 86.67% (13/15) | 90.00% (18/20) | 83.33% (5/6) | 83.33% (5/6) | | src/L1Vault.sol | 0.00% (0/1) | 0.00% (0/1) | 100.00% (0/0) | 0.00% (0/1) | | src/TokenFactory.sol | 100.00% (4/4) | 100.00% (4/4) | 100.00% (0/0) | 100.00% (2/2) | | Total | 85.00% (17/20) | 88.00% (22/25) | 83.33% (5/6) | 77.78% (7/9) |

Recomended Mitigations Try to get these up to at least 90% all of them.