BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT:

Parallel Bridge

BridgeableToken

July 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Parallel Bridge - BridgeableToken
Website	gov.mimo.capital
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/parallel-protocol/bridging-module/tree/8f5d7cbfb71c58208179a2e1501bf31a9cd42476
Resolution 1	

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High				
Medium	2			
Low	1			
Informational	1			
Governance				
Total	4			

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

BridgeableToken

The BridgeableToken contract allows an already deployed ERC-20 token to be bridgeable by leveraging LayerZero's OFT standard.

Key features of the BridgeableToken are:

- Ability to burn principal tokens and send a message to LayerZero to mint the mirrored token on the other chain.

- Ability to mint principal tokens by receiving a message from LayerZero from the mirrored BridgeableToken contract on the other chain.

- Ability to cap the principal token amount to mint and burn (per day and globally). Since minting tokens from a received message must not revert, when the mint limit is reached, the user will receive OFT tokens instead of principal tokens.

- Ability to swap OFT tokens to principal tokens if the limits are not reached.

Technical Details

LayerZero OFT standard
The OFT standard from LayerZero allows fungible tokens to be transferred across multiple blockchains without asset wrapping or middlechains. The BridgeableToken contract is designed to comply with this standard to mint/burn the OFT tokens if the mint/burn limit is reached. More details about the OFT standard can be found under in the official L0 docs.

Receiving Credit Messages from LayerZero
When the contract receives a message from LayerZero, it will mint tokens to the specified receiver address. Depending on the mint limits, the receiver will be credited with principal tokens and/or OFT tokens.

Sending Messages to LayerZero

Users can send principal tokens or OFT tokens to a receiver address on another chain by calling the send function. A check regarding burn limits and isIsolateMode is performed before sending the message to LayerZero. If any check fails, the function will revert. The user's tokens will be burned before sending the message to LayerZero.

Swap OFT tokens to principal tokens

As users can receive OFT tokens instead of principal tokens when one of the mint limit is reached, the contract provides a swapLzTokenToPrincipalToken function to swap these OFT tokens for principal tokens according to the limits.

Pause

A pause function exists to prevent new send() calls from being executed. This is useful in the event of a bug or security vulnerability.

Only the Owner can pause the contract.

Unpause

An unpause function exists to unpase the send() function.

Only the Owner can unpause the contract

IsolateMode

A toggleIsolateMode function exists to toggle the isIsolateMode variable. This variable is used to prevent the contract from burning more principal tokens than what it has minted. This is useful when the contract is in a state where it has minted more principal tokens than it has burned.

Only the Owner may call toggleIsolateMode.

Principal token mint/burn limits

Daily and global mint/burn limits on the principal token:

dailyMintLimit : Maximum amount of principal tokens that can be minted in a day.
dailyBurnLimit : Maximum amount of principal tokens that can be burned in a day.
globalMintLimit : Maximum amount of principal tokens that can be minted globally.
globalBurnLimit : Maximum amount of principal tokens that can be burned globally.

When a burn limit is reached, the contract will simply revert future send requests for principal tokens.
When a mint limit is reached, the contract will mint OFT tokens instead of principal tokens.

Only the Owner may set the daily and global mint/burn limits.

Privileged Functions

- pause
- unpause
- toggleIsolateMode
- setFeesRate
- setMintDailyLimit
- setGlobalMintLimit
- setBurnDailyLimit
- setGlobalBurnLimit
- setFeesRecipient

Issue_01	lzReceive reverts if netMintedAmount surpasses globalMintLimit
Severity	**Medium**
Description	globalMintValue is set during contract deployment and can be re-set at any time by the owner. netMintedAmount tracks the amount of principalToken minted and burned. The purpose of globalMintValue is to restrict the maximum number of principal tokens minted on a chain. In case when netMintedAmount becomes equal to globalMintValue, users can't mint any more principal tokens and the logic defaults to minting OFTs instead. The issue is that _creditPrincipalToken internal function that gets called during receiving logic doesn't check if netMintedAmount > globalMintValue, and instead always assumes this holds true. There is one instance where this can break, it's because the setGlobalMintLimit function can be used to set the globalMintValue to any value. It can happen by accident or during an emergency that globalMintValue was set to a value lower than the netMintedAmount resulting in all the lzReceive calls reverting.
Recommendations	Consider adding the condition inside the _calculatePrincipalTokenAmountToMint function: if(netMintedAmount > int256(globalMintLimit) return 0
Comments / Resolution	

Issue_02	Fees can be bypassed if mint limits are met
Severity	**Medium**
Description	The bridge charges fees only when principal tokens (PTs) are consumed on the source chain and PTs are dispersed on the destination chain. If the mint limits are met on a chain, OFT tokens are instead minted out, which are meant to be placeholder tokens which can later be swapped out to PTs. If the bridge mints or burns OFTs, no fees are charged. This creates a scenario where users can bypass the fee mechanism of the bridge by cleverly timing their bridging activities. This is done in 3 steps: 1. Bridge PTs from chain A to OFTs on chain B. This can be achieved by backrunning a large bridge transfer which maxes out the mint limit and the user will be minted OFT tokens instead. 2. Bridge OFTs from chain B for OFTs on chain A. Similar to 1, this is also achieved by running the transaction after the mint limits on chain A are met. 3. Bridge OFTs on chain A for PTs on chain B. This allows users to bridge over PTS from A to B without paying any fees. An easier way to carry this out is just by using DEXs. It can be assumed that users will be providing liquidity in PT-OFT pairs to collect swap fees in times when the mint limit is hit on the bridge contract. PT-OFT rates will always be less than 1, i.e. OFTs will always be valued less than PTs since converting OFTs to PTs in the designed way incurs fees. Users looking to bridge can then just buy these cheaper OFTs from the market and use those OFTs to bridge over to other chains and skip paying bridging fees. This is actually the most efficient method for the user and can even net them a profit.

	Say the bridge fee is 0.5%. So users can swap 1000 OFTs for 995 PTs via the bridge contract. So lets say the DEX curve also has the same price of 0.995, and a swap fee of 0.01% ike in curve stablepools. Users can either bridge 1000 PTs from chain A into 995 PTs on chainB, paying the 5 PT bridge fees, or they can use the 1000 PTs to buy up 1005 OFTs, then bridge them for 1005 PTs on the destination bridge. Bots will of course balance the pools to remove this arbitrage scenario but this way will always be more efficient than using the bridge and paying bridge fees. Stable swap pools can have fees as low as 0.01%, so in most cases swapping to OFTs and bridging will be the most efficient way to go about for users, bypassing protocol fees.
Recommendations	Consider always charging fees if principal tokens are paid out on the destination chain. This can negatively affect only a certain category of users: those who bridged and got OFT tokens and want to roll that back. These users can be tracked and be offered rebates for the spent fees via airdrops or protocol token compensations. the protocol can also incentivize OFT-PT liquidity pools on the destination bridge offering another way out of the OFT tokens. The mint capability status of the destination chain at the time of bridging will also be available to the user, so they can be made aware of the consequences if very little mint capability is present on the other side.
Comments / Resolution	

Issue_03	getMaxBurnableAmount function returns the wrong value
Severity	**Low**
Description	The getMaxBurnableAmount computes the max burnable amount according to: uint256 max = netMintedAmount < 0 ? MathLib.abs(globalBurnLimit + netMintedAmount) : MathLib.abs(globalBurnLimit - netMintedAmount); If we take: globalBurnLimit = -100 ether netMintedAmount = -50 ether; The computed maximum amount would be 150 ether while it should be 50 ether. Since this is a view-only function, this issue is only rated as low severity.
Recommendations	Compute the maximum amount regardless of the netMintedAmount being a positive or negative number with the following formula: uint256 max = MathLib.abs(globalBurnLimit - netMintedAmount);
Comments / Resolution	

Issue_04	composeMsg length is unrestricted
Severity	**Informational**
Description	The composeMsg function should only contain a single boolean value encoded to signal if on the source chain, the principal token was burned. Due to how the message is decoded, it is possible to pass a very long sequence of bytes after the initial boolean is encoded. There is no immediate security threat due to this possibility however it is still advised to restrict the length of the composeMsg.
Recommendations	Consider checking the length of composeMsg to be equal to 32 bytes: require(_sendParam.composeMsg.length == 32);
Comments / Resolution	