BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT

Parallel Bridge

BridgeableToken

December 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	mimo.capital
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/mimo-capital/titan/blob/d2ebd24dd0b1b507bf311f5e99f89314e0f07910/contracts/0.6/core/VaultsCoreState.sol https://github.com/parallel-protocol/bridging-module/blob/7c3fec46d7750e9dc44e8aa4fdd495ef85b99c2f/contracts/tokens/BridgeableToken.sol
Resolution 1	

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1			
Medium	2			
Low	2			
Informational	2			
Governance	1			
Total	8			

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

VaultsCoreState

The VaultsCoreState contract is used to trigger interest rate calculations of the CDP vault system. The refreshCollateral function of this contract is called from the VaultsCore contract to trigger interest rate calculations during borrowings and repays from the vault. This contract also implements the availableIncome function, which calculates the generated fees of the system.

Appendix: Protocol fee computation update

The protocol fee used to be calculated as the difference between the open debt and the total supply of the stablecoin. After launch, an issue was discovered in this system.

Since the bridging contract directly minted and burnt the principal token, the totalSupply of the principal token was unreliable/manipulatable. Thus if a user bridged their tokens from Ethereum to Polygon, the totalSupply on Ethereum would decrease and the same on Polygon would increase. This led to the system estimating higher than actual fees generated on Ethereum and lower than actual fees generated on Polygon.

To fix this issue, the principal token is now handled with a lock/unlock mechanism instead, which does not change the totalSupply. The bridge contract locks the principal tokens on the source chain instead of burning it and releases them on the destination instead of minting. Only when there are insufficient tokens in the destination chain are principal tokens minted, and this amount is tracked in the principalTokenAmountMinted variable.

During fee calculation, the principalTokenAmountMinted is taken into account as well, and the difference between the totalSupply and principalTokenAmountMinted is the actual amount of stablecoins that has been minted on this chain from the vaults. This modified supply is used to calculate the fees.

a.vaultsData().debt().sub(a.stablex().totalSupply().sub(bridgeableToken.getPrincipalTokenAmountMinted()));

BridgeableToken

The BridgeableToken contract allows an already deployed ERC20 token to be bridgeable by leveraging LayerZero's OFT standard.

The contract works in 2 modes: **Principal token mode** and **OFT mode**.

Principal token mode
This is the bridge's default mode of operation. Tokens are sent cross-chain by locking them in the source chain and releasing them in the destination chain. If enough tokens aren't present in the bridge contract, the principal tokens are burnt/minted.

OFT mode
This mode is entered when the credit limit for a chain has been reached. In this case the user is minted OFT tokens instead of the principal token. The users can later swap these oft tokens for principal tokens once the bridge inflow falls below the credit limit again.

IsolateMode
A toggleIsolateMode function exists to toggle the isIsolateMode variable. This variable is used to prevent the credit debit balance of the bridge from going below 0.

Only the Owner may call toggleIsolateMode.

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 for minting/burning the OFT tokens if the mint/burn limit is reached.

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

dailyCreditLimit : Principal token credit limit per day.
dailyDebitLimit : Principal token debit limit per day.
globalCreditLimit : Maximum amount of principal tokens that can be credited globally.

globalDebitLimit : Maximum amount of principal tokens that can be debited globally.

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

Only the Owner may set the daily and global credit/debit limits.

Privileged Functions

- emergencyWithdraw
- pause
- unpause
- toggleIsolateMode
- setFeesRate
- setDailyCreditLimit
- setGlobalCreditLimit
- setDailyDebitLimit
- setGlobalDebitLimit
- setFeesRecipient

Issue_01	Centralization risk introduced by emergencyWithdraw function
Severity	**Governance**
Description	The bridge is meant to lock most principal tokens. This means the amount of tokens held by the contract will in most cases result in a large amount of tokens being stored on the bridge. The emergency withdrawal function allows the owner to withdraw the total amount without any restrictions. This makes the risk of private key compromise even more severe. Furthermore, when emergencyWithdraw is called, the tokens are sent out and are unable to unlock during credit transactions so the contract is forced to mint out tokens instead and raise the principalTokenAmountMinted. Thus the protocol effectively is indebted to the admin until they return the tokens only after which the principalTokenAmountMinted can return to 0 with debits.
Recommendations	Consider using a time lock for withdrawals by the owner. Allow users in the mean time to exit their position in case they want.
Comments / Resolution	

Issue_02	Erroneous casting causes tokens to be stuck in the bridge
Severity	**High**
Description	Hard casting an int to uint256 leads to an underflow in solidity. In the current code, creditDebitBalance is an integer which is cast into uint256. In certain conditions, this can lead to an underflow, leading to a revert. The issue stems from the internal function _calculatePrincipalTokenAmountToCredit The function will determine the amount of principal token to be minted based on the contracts limits. *principalTokenAmountToCredit = int256(_amount) +* *creditDebitBalance > int256(globalCreditLimit)* *? globalCreditLimit - uint256(creditDebitBalance)* *: _amount;* In the snippet, we will evaluate the principalTokenAmountToCredit based on the conditional operator. The focus of this bug is the true case. Lets assume: creditDebitBalance = -10 globalCreditLimit = 100 _amount = 111 In this case, the condition evaluates to true, and we calculate globalCreditLimit - uint256(creditDebitBalance). However, since the creditDebitBalance is negative, it underflows and the result thus becomes negative as the underflown result is very large. This negative number cannot be stored in principalTokenAmountToCredit since it expects a uint256 and thus reverts.

	Given that this calculation is part of the _lzReceive call flow, this can lead to a scenario where a user sends tokens on the source chain but is unable to receive them on the destination chain.
Recommendations	Consider the following changes. *globalCreditLimit - uint256(creditDebitBalance)* should be updated to the following *uint256(int256(globalCreditLimit) - creditDebitBalance)* This will ensure reverts are not possible.
Comments / Resolution	

Issue_03	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_04	Fees are charged on OFT token bridging if limits are reached
Severity	**Medium**
Description	When the credit limits are met, users get minted OFT tokens when bridging which can be swapped for principal tokens at a later stage. However, to allow unsatisfied users to roll back their bridging transaction, the protocol does not charge any fees if OFT tokens are being bridged for principal tokens. This way users can revert their transaction at no fee cost if they are minted OFT tokens. */// @dev Extract from message if the tokens burned from the original chain* */// was the principalToken or the OFT token. If true, fees could be applied.* *(, bool feeApplicable) = abi.decode(_message.composeMsg(), (bytes32, bool));* The code comments confirm that if a user is bridging the principalToken, then fees should be applied. On the other hand, if the

	user is bridging OFT, then no fees should be applied. However, the issue is that this fee-less bridge rollback is not always possible. This happens when the credit limits are met on the source chain as well. Consider the following scenario: 1. Alice wants to bridge principal tokens from chainA to chain B. Alice calls send on chainA and her principal tokens are locked in the bridge. 2. In chainB, the mint limit has already been met. So Alice gets minted OFT tokens instead. 3. Alice is unhappy and wants to revert her transaction. She calls send on chainB with isPrincipalTokenSent set to false. 4. In chainA, the mint limit has already been met. Thus Alice gets minted OFT tokens on chainA as well. In the scenario above, Alice is unable to roll back her bridging transaction for no fees. She can call swapLzTokenToPrincipalToken to swap the OFT tokens for principal tokens on chainA, but this will charge her fees. This issue may happen by chance but can also be triggered by a malicious user. The malicious user can frontrun the transaction and fill the limits in order for the user who will then receive OFT tokens which they have to pay a fee for if they want back the principal tokens.
Recommendations	In the _credit function, if _isFeeApplicable is is false the user should not be paying any fees. This can be done by either minting them principal tokens even if the credit limits have already been met, or by airdropping them the fee amounts later.
Comments / Resolution	

Issue_05	swapLzTokenToPricipalToken can be DOSed in case a whale tries to swap the exact credit limit amount
Severity	**Low**
Description	The swapLzTokenToPricipalToken function is used in case a user wants to exchange his bridgeable tokens into Principal (PAR) tokens. If a user bridges more than the daily Credit Amount, the contract will mint bridge tokens, which then allows the user to swap these later on. The swap function has two user inputs, amount and “to” address. It also verifies that the swapped amount does not exceed the credit limit: *uint256 principalTokenAmountToSend = _calculatePrincipalTokenAmountToCredit(_amount);* *if (principalTokenAmountToSend != _amount) revert ErrorsLib.CreditLimitExceeded();* The _calculatePrincipalTokenAmountToCredit returns the amount of tokens that can be credited to the user. As we can see the if statement makes the transaction revert, in case the user specified input is not equal to the calculated amount. This creates following issue: In case someone tries to swap a large amount of tokens, close to the credit limit, any other transaction that is included before this, will make the large transaction revert. This can happen maliciously or unintentionally. The user has to fix this by submitting a much smaller swap amount then he might have intended. Because this is not a long lasting DOS, the severity should be considered low.

Recommendations	Consider using principalTokenAmountToSend as the real amount instead of just reverting.
Comments / Resolution	

Issue_06	It is impossible to rescue OFT tokens from the contract
Severity	**Low**
Description	The contract includes an emergency function that can withdraw principal token in case it was sent by accident or during an emergency. However, the OFT cannot be withdrawn during an emergency and will remain stuck in the contract.
Recommendations	Consider adding functionality that allows emergency withdrawal of the OFT token
Comments / Resolution	

Issue_07	Bridging will not be possible if all chains are set to isolateMode
Severity	**Informational**
Description	Due to how isolateMode works, if all chains are set to this mode and _config.initialCreditDebitBalance is also 0 on all chains, bridging will be impossible. */// @dev If true, the bridge can't send more tokens than it had received (creditDebitBalance > 0)* *bool private isIsolateMode;* Since every chain will have its creditDebitBalance as 0 and won't be allowed to go negative, bridging will halt.
Recommendations	Ensure that _config.initialCreditDebitBalance is never 0 when isolateMode is desired on all chains.
Comments / Resolution	

Issue_08	Bridge Credit limits can be bypassed if there ever is a vault that accepts bridge tokens as collateral
Severity	**Informational**
Description	The configuration of collateral can be changed by governance. In case there is ever a vault created, which allows the use of bridge tokens as collateral, this will allow users to bypass credit limits of the bridge.
Recommendations	Consider blocking contracts from using bridge tokens as collateral. Or make sure when setting configurations for bridge and vaults, all credit limits add up to the “real” intended credit limit amount.
Comments / Resolution	