Gnosis Bridge

Final Audit Report

August 29, 2023

Team Omega

teamomega.eth.limo

Summary 2
Scope of the Audit 3
Resolution 3
Methods Used 3
Disclaimer 4
Severity definitions 4
Findings 4
General 4
G1. No working tests or continuous integration [info] [not resolved] 4
ERC20Bridge.sol 5
E1. _relayInterest function will revert if current balance is less than the interest [medium] [resolved] 5
InterestConnector.sol 5
I1. Cannot pay out interest if the amount of interest accumulated exceeds the daily limits [medium] [resolved] 5
I2. The owner of the contract can mint infinite tokens on the destination chain [low] [resolved] 6
I3. payInterest has unnecessary side effects [low] [resolved] 6
I4. Potential reentrancy from calls to the token and withdrawal [low] [resolved] 7
I5. payInterest is vulnerable to overflow errors [low] [resolved] 7
I6. Duplicated check on receiver address [info] [resolved] 8
SavingsDAIConnector.sol 8
S1. interestAmount gives unexpected results [low] [resolved] 8
S2. SavingsDAIConnector does not have complete functionality [info] [not resolved] 9

Summary

Gnosis has asked Team Omega to audit the update of their bridge contracts.

We found no **high severity issue** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **2** issues as “medium” - this is an issue we believe you should definitely address. In addition, **5** issues were classified as “low”, and **3** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

After receiving a preliminary report, Gnosis has updated the repository and has resolved all serious issues we reported.

Severity	**Number of issues**	**Number of resolved issues**
High	0	0
Medium	2	2
Low	5	5
Info	3	1

Scope of the Audit

Scope of the Audit
The audit concerns the Solidity files in the following repository and branch:

```
https://github.com/Luigy-Lemon/tokenbridge-contracts/tree/DSR
```

And specifically commit 9eb8f1d00741271b44b3c83f042fb9f6882705f1 at commit

This repository is a fork of `https://github.com/gnosischain/tokenbridge-contracts` at commit `57dd76e1bc091888783ddd21b367d9c2934f1c83`

The scope of the audit is limited to the changes between these two commits.

Resolution

The issues were addressed in commit `676e65da7a933adb2ee752d536e925ee22728537`.
We subsequently checked the changes and updated the report.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. The tools have detected a number of low severity issues, concerning mostly the variables naming and external calls, were found. We have included any relevant issues below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. No working tests or continuous integration [info] [not resolved]

The branch that we audited did not have updated instructions on installation and testing, and does not have continuous integration configured.

Working tests help confirm the behavior of the code and catch errors, and continuous integration helps to keep the code consistent as it changes over time.

Recommendation: Update the instructions on how to install and run the tests in the repository. Configure continuous integration on GitHub to run the tests on each commit, and update the instructions for running the tests. Make sure your tests have 100% coverage.

Severity: Info

Resolution: The issue was not resolved.

ERC20Bridge.sol

E1. _relayInterest function will revert if current balance is less than the interest [medium] [resolved]

On line 35, the current balance of the bridge contract is compared to the amount to be relayed:

```
require(erc20token().balanceOf(address(this)) > _amount , "Not enough Balance");
```

There is no reason to implement this check. The collateral for the tokens that are being relayed is not only held in token balance of the bridge contract, but is also deposited in the sDAI contract, and in any case the collateral is guaranteed to be present by the code in the `payInterest` function.

The check however will make the `InterestConnector.payInterest` function revert if the amount of interest that is to be relayed is higher than the current balance, which could cause the transfer to fail.

Recommendation: Remove this line.

Severity: Medium

Resolution: The issue was resolved as recommended.

InterestConnector.sol

I1. Cannot pay out interest if the amount of interest accumulated exceeds the daily limits [medium] [resolved]

The `payoutInterest()` function will revert if the amount of interest accumulated is above the `dailyLimit()` or exceeds `maxPerTx()`.
Recommendation: Even if in realistic scenarios these limits are not expected to be reached, we recommend to adopt changes so that basic security parameters do not need to be adjusted to execute a function such as `payoutInterest`. A straightforward way to implement this is to add an `_amount` parameter to the `payoutInterest` signature, and use its value as a cap on the amount of interest paid out in this transaction.
Severity: Medium
Resolution: The issue was resolved as recommended.

I2. The owner of the contract can mint infinite tokens on the destination chain [low] [resolved]

The `payInterest` function takes a `_token` parameter. When called in the context of `SavingsDAIConnector`, the `_token` argument is passed to the functions `interestAmount` and `_safeWithdrawTokens`, which ignore the value and instead operate on the hardcoded DAI contract address. However, functions such as `investedAmount` and `_setInvestedAmount` do not ignore the `_token` parameter, and will read and write their data at a key generated from the value of `_token`.
In practice, this means that if `payInterest` is called with a different address than the DAI contract, tokens will be withdrawn from the sDAI investment pool, but the `investedAmount` counter of the DAI token will not be updated correspondingly. This makes it possible to relay an unlimited amount of tokens to the `interestReceiver` on the destination chain by repeatedly calling `payInterest`.
A condition for this attack to work though is that the `interestEnabled` flag for `_token` is set to true, and the `interestReceiver` value is set as well. These flags are controlled by the owner. To set this flag, also the implementation of `_isInterestSupported` must return true for the given token address.
If these parameters are configured for a token different from DAI, whether intentionally or by mistake, this allows anyone to send an unlimited amount of DAI on the destination chain to the `interestReceiver` of that token.
The contracts in the repository that use `InterestConnector`, namely `CompoundConnector` and `SavingsConnector`, hardcode `_isInterestSupported` to return true only when the DAI contract address is set, and so the vulnerability is not exploitable with the deployable contracts in the current version of the repository.
See also S1, which describes a similar bug.

Recommendation: In `SavingsDAIConnector`, overwrite `payInterest` and check if the `_token` parameter is equal to the DAI token. In general, consider following our recommendation in S1.
Severity: Low
Resolution: The issue was resolved by adding a check that the `_token` argument passed to `payInterest` is the same as `erc20token()`.

I3. payInterest has unnecessary side effects [low] [resolved]

The current implementation of `payoutInterest` does two logically unrelated things:

1. It calculates the amount of interest accumulated and relays that amount to the destination chain (by emitting a `UserRequestForAffirmation` event)
2. It checks if the current token balance of the bridge is below the `minCashThreshold`, and if so withdraws up to `interest` tokens from the sDAI contract to cover the difference

These two actions (relaying interest, vs keeping the balance of the bridge above `minCashThreshold`) are not in any way related. Doing these two actions at the same time is confusing, and will make callers spend gas on operations that they otherwise might not have requested.
Recommendation: We recommend removing the logic that handles withdrawals from the sDAI contract from the `payInterest` function. No withdrawal needs to be made for paying interest. As there already are two other functions that can be used to “refill” the bridge balance, namely `XDAIForeignBridge.ensureEnoughTokens()` and `XDAIForeignBridge.refillBridge()` this logic can safely be removed.
Severity: Low
Resolution: The issue was resolved as recommended.

I4. Potential reentrancy from calls to the token and withdrawal [low] [resolved]

The `payInterest` function makes multiple external calls in the middle of its execution, such as to the `_token.balanceOf` and the `_withdrawTokens` function (which is not part of the `InterestConnector’s` implementation).
Especially because this is a generic call to any `_token`, this opens a possibility for a reentrancy attack, where the `_token`, or any address they may call, could re-enter the `payInterest` in the middle of its execution and cause the interest to be paid multiple times.
We also take the opportunity to note that this vulnerability exists in other parts of the code as well, parts that were not part of the scope of this audit.
Recommendation: Make external calls at the end of the function where possible, after you make state changes. For example, it is prudent to call the `_withdrawTokens` after the call to

`_setInvestedAmount`. Also consider using the OpenZeppelin `ReentrancyGuard` for further protection against re-entrancy, especially by the token itself.
Severity: Low
Resolution: The issue was resolved. These parts of the code were removed.

I5. payInterest is vulnerable to overflow errors [low] [resolved]

As the contracts are written in solidity 4, the functions will not revert if there are arithmetical over- and underflow errors. In the payInterest function, there are checks to avoid underflows, but there is still the possibility of overflows. This may only occur in extreme cases where `balance + interest` or `minCash + interest` is greater than the maximum of `uint256`, but it is best practice to use SafeMath to avoid even such edge cases.
Recommendation: As a general precaution, we recommend to use the SafeMath library, as is done in many other parts of the code. Also, if you choose to follow our recommendation on I3, most of these calculations can be removed.
Severity: low
Resolution: The issue was resolved. These parts of the code were removed.

I6. Duplicated check on receiver address [info] [resolved]

On line 140 there is a check that the receiver address is not the 0 address, yet this check also happens in the `_relayInterest` function and so can be removed.
Recommendation: Remove the check that is in line 140 or the one inside the `_relayInterest` function.
Severity: Info
Resolution: The issue was resolved as recommended.

SavingsDAIConnector.sol

S1. interestAmount gives unexpected results [low] [resolved]

The `interestAmount` function takes a `_token` argument. It will compare the withdrawable balance from the sDAI contract with the internally tracked `investedAmount(_token)`. If the `_token` parameter is not equal to the sDAI address, this will compare two values that are not related. When called upstream, for example in `payInterest`, this may lead to unexpected results and even loss of funds - see for example I1.

Recommendation: Either remove the `_token` argument altogether, or, if you must keep the function signature, just ignore it in the function body, and calculate the interest for the sDAI contract only, or revert if the address is different than the DAI address.
Severity: Low
Resolution: The issue was resolved as recommended. The function will now revert if the token passed is not DAI.

S2. SavingsDAIConnector does not have complete functionality [info] [not resolved]

The `SavingsDAIConnector` contains functions to invest its DAI balance in the sDAI contracts, but contains no functionality to withdraw these invested tokens. The `XDAIForeignBridge` contract, which inherits from `SavingsDAIConnector`, implements this functionality. This means that if `SavingsDAIConnector` is deployed and used as-is, funds will be invested in the sDAI contract that can never be withdrawn.
Recommendation: To make it clear that `SavingsDAIConnector` is a mix-in contract, and should not be deployed as-is, consider making this very clear in the documentation, and mark it as abstract by adding function signatures for functions such as `onExecuteMessageGSN`.
Severity: Info
Resolution: The issue was not resolved.