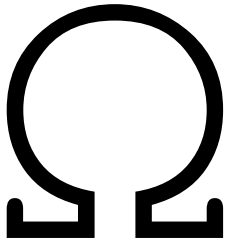
Audit

of Gnosis Token Bridge USDS Upgrade

Final Audit Report

September 11, 2025



Team Omega

Teamomega.eth.limo

Summary **2**
Scope of the Audit **3**
Methods Used **4**
Disclaimer **4**
Severity definitions **4**
Findings **5**
General 5
G1. Tests in CI are failing [info] [resolved] 5
G2. Pin solidity versions [info] [resolved] 5
XDaiForeignBridge.sol 5
XDFB1. An attacker can make user receive DAI instead of USDS and vice versa [medium] [not resolved] 5
XDFB2. Users can avoid paying fees [medium] [not resolved] 6
XDFB3. executeSignaturesGSN sends different tokens before and after the upgrade [low] [not resolved] 6
XDFB4. Use of ignored token parameter may lead to unexpected behavior [low] [partially resolved] 7
XDFB6. Choice of function naming is confusing [info] [resolved] 8
BridgeRouter.sol 8
BR1. setRoute should limit tokens settable [info] [resolved] 8
SavingsDaiConnector.sol 8
SDC1. Outdated error message for interestAmount [info] [resolved] 8
XDaiBridgePeripheralForUsdsPreUsdsUpgrade.sol 9
PUP1. Anyone can cause the contract to permanently fail [medium] [resolved] 9
XDaiBridgePeripheralForDaiPreUsdsUpgrade.sol 9
PDP1. Unused constants defined in the contract [info] [resolved] 9

Summary

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

We found **no high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **3** issues as “medium” - this is an issue we believe you should definitely address. In addition, **3** issues were classified as “low”, and **6** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Severity	Number of issues	Number of resolved issues
High		
Medium	3	1
Low	2	1
Info	6	6

Scope of the Audit

Scope of the Audit

The audit concerns an upgrade to bridge to Gnosis Chain to replace the sDAI with USDS.

The upgrade is developed in the following PR:

> https://github.com/gnosischain/tokenbridge-contracts/

And specifically the following commit:

> 859f4e68ae6f60e99722d646ad0440255f4db95c

The audit concerns the following files:

```
contracts
├── interfaces
│   ├── IBridgeValidators.sol
│   ├── IDaiUsds.sol
│   ├── IERC20.sol
│   ├── IForeignBridge.sol
│   ├── ISUSDS.sol
│   ├── IWETHOmnibridgeRouter.sol
│   └── IXDaiBridgePeripheral.sol
├── upgradeability
│   └── TransparentUpgradeableProxy.sol
└── upgradeable_contracts
    └── erc20_to_native
        ├── BridgeRouter.sol
        ├── SavingsDaiConnector.sol
        ├── XDaiBridgePeripheral.sol
        └── XDaiForeignBridge.sol
```

Resolution

We wrote a preliminary version of this report on April 14, 2025.

After we delivered a preliminary report, the developers addressed the issues mentioned in the report in the following commit:

```
35c57c046040fdb84c1b52e1e31c8a5f145553a0
```

We have checked the changes and updated the report on May 5, 2025.
In September 2025, the developers decided to address issue XDFB1 (which was left unresolved in May). They submitted the following new commit, which we reviewed.

```
f53666fd4f832b1dde479b701d39eddc10b6877c
```

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. Tests in CI are failing [info] [resolved]

The tests that are run as part of the continuous integration in Github Actions are failing.
Recommendation: Make sure your CI tests are passing.
Severity: Info
Resolution: The issue was resolved as recommended.

G2. Pin solidity versions [info] [resolved]

Many files in scope specify a range of suitable Solidity compiler versions:

```
pragma solidity ^0.8.0;
```

The code does not contain any other place where a specific Solidity version is fixed, which means that it depends on the local installation of the developer which version is used. This means that the result of compiling the code is nondeterministic, and this can be a problem - or at least an annoyance - when trying to reproduce the same compiler results - for example, when verifying contracts that are deployed on-chain.
Recommendation: Specify a specific Solidity version that is to be used when compiling the files. A good place is the `foundry.toml` file.
Severity: Info
Resolution: The issue was resolved as recommended.

XDaiForeignBridge.sol

XDFB1. An attacker can make user receive DAI instead of USDS and vice versa [medium] [resolved]

The `executeSignaturesUSDS` and `executeSignatures` functions are essentially identical, with the only difference being the former calls `onExecuteMessageUSDS` and the latter calls `onExecuteMessage`.
Because both functions have identical checks on the signatures, it means both can be triggered by the same bridged message, and since both can be called by anyone, an attacker can frontrun a call to one function by calling the other. The result is that the user will receive the wrong token.

Recommendation: Either add the token address to bridge as part of the message, or add access control on calling these functions, as you already to in `executeSignaturesGSN`:

```
require(isTrustedForwarder(msg.sender), "invalid forwarder");
```

Severity: Medium
Resolution: The issue was resolved by adding the token address as part of the message.

XDFB2. Users can avoid paying fees [medium] [not resolved]

The `executeSignaturesGSN` function allows trusted forwarders to relay messages. A fee will be subtracted from the bridge amount. A user can avoid paying that fee by frontrunning the call to `executeSignaturesGSN`, and replay the message calling `executeSignaturesUSDS` or `executeSignatures`, and receive the full amount.
Recommendation: Avoid ambiguity in messages, and include a flag in the signed message that represents that the transfer includes a fee to be paid. Alternative, add access control to all `executeSignatures**` functions.
Severity: Medium
Resolution: The issue was not resolved. The team has acknowledged the issue, and stated that the GSN functions are no longer supported.

XDFB3. executeSignaturesGSN sends different tokens before and after the upgrade [low] [not resolved]

The developers have taken care to make sure that before and after the upgrade from DAI to USDS, the relayer function `executeSignatures` will not change semantics: it will continue to release DAI tokens to the users. To release USDS, relayers call the new function `executeSignaturesUSDS`. This provides a smooth upgrade path for integrators, which can continue to expect the same behavior before or after the upgrade. However, `executeSignaturesGSN` does not follow this pattern - it will claim DAI before the upgrade, and USDS after the upgrade. This may lead to unexpected behavior for users and integrators.
Severity: Low
Recommendation: Use the same upgrade pattern in `executeSignaturesGSN` that you used in `executeSignatures`
Resolution: The issue was not resolved. The team has acknowledged the issue, and stated that the GSN functions are no longer supported.

XDFB4. Use of ignored token parameter may lead to unexpected behavior [low] [partially resolved]

There are many functions in the reviewed contracts that take a token parameter. This parameter is often, but not always, ignored - instead, these functions will often operate on the value returned by `erc20Token()` or `daiToken()`. (In `XDAIForeignBridge`, both these functions return the hardcoded value of `0xdC035D45d973E3EC169d2276DDab16f1e407384F`, which is the USDS token)

There are some places in the code where this behavior can be particularly confusing or unexpected.

- For example, the `invest` function takes a `token` parameter. It is protected by a decorator i`nterestEnabled(_token)` - which will makes sure that the `_token` parameter is either the DAI address (assuming that the current version has not enabled any other tokens) until `swapSDAIToUSDS` is called) or `USDS` (after the upgrade, and after `initializeInterest` has been called by the owner). This is a `public` function, and although the risk is very small, there may be a moment during the upgrade process where the "investedAmount" of DAI is updated but USDS is actually invested, which would basically designate the entire investment amount as interest.
- The function `disableInterest` accepts any token, and will mark the token address passed as `disabled`, but will always try to withdraw USDS.
- The `previewWithdraw(_token, _amount)` function will return the amount of SUSDS shares corresponding to the `amount` of USDS, but ignore the `_token` argument.
- The `ensureEnoughTokens` function has a `token` parameter, but it is only ever passed the `daiToken()` address. In its internal logic, it depends on `_withdrawTokens`, which will withdraw the `daiToken()` - so should only ever be used with it.

Recommendation: We recommend removing unused parameters from the function signatures where possible - that means in all internal functions. If that is cumbersome for external functions, or if there are reasons (e.g. backwards compatibility or adherence to a certain interface) to keep the signatures as they are, we recommend that all functions that accept a token parameter should verify the token passed in the USDS, and revert otherwise.
Severity: Low
Resolution: The issue was partially resolved. Functions now verify that the token passed is the correct token, but internal functions still pass a token argument even though that is always the same token. Also the verification for the correct token is done against the literal address, though it is clearer and less error prone to use a constant instead.

XDFB6. Choice of function naming is confusing [info] [resolved]

The functions `erc20Token()` and `daiToken()` both return the hardcoded value of `0xdC035D45d973E3EC169d2276DDab16f1e407384F`, which is the USDS token. The function `sdaiToken()` will return the address of SUSDS.

This can be quite confusing, and the code is replete with comments clarifying that:

```
/// daiToken() returns USDS address
```

Recommendation: Use the `USDS` constant (or define a new function `usdsToken()`) and use that internally in all places where the above comment is necessary. You can keep the public-facing `daiToken()` and `sdaiToken()` functions (and have them return the USDS addresses) for backward compatibility if that is deemed necessary, but there is no need to use these internally.
Severity: Info
Resolution: The issue was resolved as recommended.

BridgeRouter.sol

BR1. setRoute should limit tokens settable [info] [resolved]

The `setRoute` allows setting routes for any token. However, any route besides those of USDS and DAI are ignored, and so it is better to explicitly disable setting them to avoid possible confusion in the future.
Recommendation: Revert if a user tries to set a route for tokens other than USDS or DAI.
Severity: Info
Resolution: The issue was resolved as recommended.

SavingsDaiConnector.sol

SDC1. Outdated error message for interestAmount [info] [resolved]

The error message in line 35 says Not DAI, but with the new change, this should be updated like other messages were to say: “Not USDS”.
Recommendation: Update the error message to say: “Not USDS”.
Severity: Info
Resolution: The issue was resolved as recommended.

XDaiBridgePeripheralForUsdsPreUsdsUpgrade.sol

PUP1. Anyone can cause the contract to permanently fail [medium] [resolved]

The `relayTokens` function has a check to verify that the amount of DAI received from the DAIUSDS contract is equal to the amount sent. The check is done by comparing the contract’s DAI balance to the `amount` parameter of the function.

However, if anyone sends DAI directly into that contract, the check will fail, as the contract will have more DAI than expected. And since there is no way to withdraw the excess DAI, all future calls to `relayTokens` will revert.
Recommendation: The check for the swap result is not necessary, as the DAIUSDS contract always returns a 1:1 ratio, so we would recommend removing the check. However, if you choose to leave it, we recommend to check the balance is greater than or equal to, instead of checking strict equality, and also add it to `XDaiBridgePeripheral.sol relayTokens`, so the check is done consistently for swaps.
Severity: Medium
Resolution: The issue was resolved as recommended.

XDaiBridgePeripheralForDaiPreUsdsUpgrade.sol

PDP1. Unused constants defined in the contract [info] [resolved]

The contract defines `DAIUSDS` and `USDS` constants, but these are never used in the contract and can be removed.
Recommendation: Remove the unused constants from the contract
Severity: Info
Resolution: The issue was resolved as recommended.