PUBLIC

Code Assessment

of the Omnibridge Version 6.0 Smart Contracts

September 07, 2021

Produced for

by

Contents

1 Executive Summary

Dear POA Network Team,

First and foremost we would like to thank you for giving us the opportunity to assess the Omnibridge Version 6.0. This document outlines the findings, limitations, and methodology of our assessment.

Only two partially corrected findings remain. These findings have been largely mitigated and hence their original severities no longer reflect the current state. In the current code version miner cooperation is required to exploit the first finding. Regarding the second finding the attacker no longer has any financial gain, they can just prevent a regular GSN execution at their own expense.

We hope that this assessment provides valuable findings. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Partially Corrected	1
Medium-Severity Findings	3
• Code Corrected	2
• Code Partially Corrected	1
Low-Severity Findings	13
• Code Corrected	11
• Specification Changed	2

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Omnibridge Version 6.0 repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Omnibridge

V	Date	Commit Hash	Note
1	August 02 2021	68ad223a39bf0c23d4e891f7e25c66aa3f3f99a7	Initial Version
2	August 20 2021	76867baad2884b39e739cebd0ac99cbafe38a395	Second Version
3	September 03 2021	b4935ff2c3b5ac897517af19b6fb626a800eee98	Third Version

Tokenbridge

V	Date	Commit Hash	Note
1	August 06 2021	3519ddb8c2c75a84f1452224103f509a909a0578	Initial Version
2	August 20 2021	e44f4d8bf4fe665c784c7071a77ce635f41b9558	Second Version

For the Omnibridge solidity smart contracts, the compiler version `0.7.5` was chosen. For the Tokenbridge solidity smart contracts, the compiler version `0.4.24` was chosen.

This report is only concerned with the upgrade from Omnibridge v1.0.0 to Omnibridge v1.1.0-rc1 and Tokenbridge v5.5.0-rc0 to Tokenbridge v6.0.0-rc0. Further it is restricted to the changes within the following files:

Omnibridge

- contracts/upgradeable_contracts/BasicOmnibridge.sol
- contracts/upgradeable_contracts/components/common/InterestConnector.sol
- contracts/upgradeable_contracts/components/common/OmnibridgeInfo.sol
- contracts/upgradeable_contracts/ForeignOmnibridge.sol
- contracts/upgradeable_contracts/modules/interest/AAVEInterestERC20.sol
- contracts/upgradeable_contracts/modules/interest/CompoundInterestERC20.sol
- contracts/upgradeable_contracts/Upgradeable.sol

Tokenbridge

- contracts/gsn/BasePaymaster.sol
- contracts/ERC677BridgeToken.sol
- contracts/gsn/BaseRelayRecipient.sol
- contracts/gsn/token_paymaster/TokenPaymaster.sol
- contracts/gsn/utils/GsnEip712Library.sol
- contracts/helpers/AMBBridgeHelper.sol
- contracts/helpers/Erc20ToNativeBridgeHelper.sol

- contracts/libraries/ArbitraryMessage.sol
- contracts/PermittableToken.sol
- contracts/upgradeable_contracts/arbitrary_message/AsyncInformationProcessor.sol
- contracts/upgradeable_contracts/arbitrary_message/HomeAMB.sol
- contracts/upgradeable_contracts/arbitrary_message/MessageDelivery.sol
- contracts/upgradeable_contracts/erc20_to_native/CompoundConnector.sol
- contracts/upgradeable_contracts/erc20_to_native/ForeignBridgeErcToNative.sol
- contracts/upgradeable_contracts/GSNForeignERC20Bridge.sol
- contracts/upgradeable_contracts/InterestReceiverBase.sol
- contracts/upgradeable_contracts/InterestReceiverStakeBuyback.sol
- contracts/upgradeable_contracts/InterestReceiverSwapToETH.sol

The `erc20token` used in `GSNForeignERC20Bridge` contract is assumed to be Maker DAI token with 18 decimals.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

The Omnibridge upgrade allows tokens that are locked on the foreign side of the bridge to accrue interest. The `InterestConnector` contract can be used to select an `InterestImplementation` which is then used to invest tokens. There are two implementations currently available, one for Compound and one for AAVE. Both implementations specify an `InterestReceiver` which receives the accrued interest, the Compound connector additionally specifies a `CompReceiver` to receive the governance tokens. In order to account for tokens that are not directly stored in the bridge contract, the `_unaccountedBalance` function was added, which calculates the balance not stored in the bridge, e.g., invested tokens. The `_releaseTokens` function was modified to account for the fact that tokens may have to be withdrawn from the `InterestImplementation` in order to release them.

The Tokenbridge upgrade enables GSN functionality, which essentially allows users to execute transactions despite not being able to pay for gas costs. Instead, they give a relayer a signature, who executes the transaction on their behalf, under the assumption that the user will repay the gas costs using tokens in their account.

In order to allow this functionality, there is a `TokenPayMaster` and a `RelayRecipient` contract. They both receive calls from an external `RelayHub` contract, which does most of the bookkeeping. The `TokenPaymaster` first receives a call to the `preRelayedCall` function, where it can decide to reject the call. Then the `RelayRecipient`, in this case the `GSNForeignERC20Bridge`, is called through a trusted forwarder contract. Here, the tokens are bridged and partially send to the `TokenPaymaster`.

Next, the `RelayHub` calls the `postRelayedCall` function of the `TokenPaymaster`. Here, the exact gas costs are calculated, then the necessary amount of tokens are traded to ETH via uniswap and sent to the `RelayHub`. Any remaining bridged tokens are sent to the user.

It is important that the `TokenPaymaster` correctly estimates how much gas is used after the gas cost calculation (to send the remaining tokens to the `RelayHub` and user), otherwise it may consistently send too little or too much. If it sends too much, then value is lost. If it doesn't pay enough, the paymaster's balance in the `RelayHub` will eventually drop below the threshold where it can cover the maximum cost of a call, meaning the `RelayHub` will no longer forward the calls.

The Tokenbridge upgrade also unlocks asynchronous message passing, whereby the home chain can query low-level information about the foreign chain that normally would not be available on smart-contract level through the JSON-RPC interface.

2.2.1 Trust Assumptions

The interest providers, in particular Aave and Compound, are trusted as they could otherwise attack the contracts in various ways, e.g., through reentrancy or by simply stealing funds. Their reward tokens are also trusted not to contain any callbacks that could lead to reentrancies.

The bridged tokens are also trusted as they could otherwise be used to perform reentrancy attacks.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Interest Payments Prone to Price Manipulation Attacks Code Partially Corrected

Medium-Severity Findings	1

- Anyone Can Steal GSN Fees Code Partially Corrected

Low-Severity Findings	0

5.1 Interest Payments Prone to Price Manipulation Attacks

Security High Version 1 Code Partially Corrected

The `InterestReceiverStakeBuyback` and `InterestReceiverSwapToETH` contracts both utilize Uniswap in order to swap tokens. There is some slippage protection, but no protection against price manipulation is built in.

Because the `payInterest` function in the `InterestConnector` is external and permissionless, anyone can call it at any time. Therefore, once enough interest accrues, one could take out a flash loan to manipulate the relevant Uniswap pools and call `payInterest` to make a profit.

Additionally, even if `payInterest` weren't permissionless, it is possible for the Uniswap call in `onInterestReceived` to fail, leaving the tokens sitting in the `InterestReceiver` contract. Here, the `onInterestReceived` function is external and permissionless, meaning it's once again prone to being attacked via price manipulation.

Furthermore, assuming the interest payment was not callable by untrusted parties at all, it would still be possible to manipulate the price by sandwiching that transaction between two others, either by chance or by collusion with miners.

Code partially corrected:

Only EOA modifier was added to `payInterest` function, thus no flash loans can be used to manipulate the pools. The modifier allows calls only when `msg.sender == tx.origin`.

The calls to uniswap in `onInterestReceived` in InterestReceiverSwapToETH and InterestReceiverStakeBuyback contracts now will revert, if uniswap call does so.

POA Network accepted the risk of transaction front-running and risk of placing a `payInterest` transaction between the attacker's transactions. POA Network also accepted the risk of storing other ERC20 tokens directly on the interest receiver contract.

5.2 Anyone Can Steal GSN Fees

Security | Medium | Version 1 | Code Partially Corrected

In GSN contracts the `Forwarder` is an intermediate contract which checks the user's signature, handles user nonces, and then forwards the request.

However, as anyone can observe the user signature in the transaction pool and use it to call the `Forwarder` contract directly, some of the protections enforced by the `RelayHub` are lost. These protections include that GSN transactions must be triggered by an externally owned account. Instead, the following attack becomes possible:

- Observe user signature in GSN transaction and front-run with own transaction
- Own transaction:
  - Call `Forwarder` with intercepted signatures to send `maxTokensFee` to the `TokenPaymaster`.
  - Call `TokenPaymaster` with fake inputs to send `maxTokensFee` to the attacker. This issue is described in Missing Access Control for postRelayedCall.

Note that users might choose a high value for `maxTokensFee` as they expect to receive the remaining value that wasn't used for gas payments. Also note that the attacker's transaction is cheaper than a regular GSN transaction as no checks within the `RelayHub` and no execution of `preRelayedCall` need to be performed. Hence, the attacker gets more funds than a faithful GSN relayer would have received.

Code partially corrected:

The fix of the Missing Access Control for postRelayedCall issue makes it possible for attacker only to lock the `maxTokensFee` inside the `TokenPaymaster`. This risk was accepted by the client, since the locked ERC20 can be withdrawn only by the owner of the contract. Later, lost fees can later manually be returned to users, according to POA Network.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- Missing Access Control for postRelayedCall Code Corrected
- Outdated GSN Interface Code Corrected

Low-Severity Findings	13

- Decimals in GSN Token Handling Code Corrected
- Delete Statements Can Be Combined Code Corrected
- Events on Interest Params Change Code Corrected
- Gas Saving if No Interest Implementation Exists Code Corrected
- Incorrect Documentation Comments Specification Changed
- Interest Implementation With Non-Revoked Allowance Code Corrected
- Interest Implementations Cannot Handle Non-Standard Tokens Code Corrected
- Interface IInterestImplementation Events Code Corrected
- Message Offset Incorrect Specification Changed
- Minimum Gas Computed Incorrectly Code Corrected
- No Claiming of Aave Rewards Code Corrected
- Potentially Conflicting Getters Code Corrected
- Unnecessary Balance Query Code Corrected

6.1 Missing Access Control for postRelayedCall

Security Medium Version 1 Code Corrected

The `postRelayedCall` function of the `TokenPaymaster` contract is public and allows anyone to call it. In this function the uniswap router is called and leftover funds are sent back to the user. Thus, this function modifies the state. Without an access control check that only allows the `RelayHub` to call it, an attacker can craft call data that will drain the ERC20 balance of the contract. Note that normally the contract is not supposed to have an ERC20 balance, however, this can change in situations as described in Anyone can steal GSN fees.

The `preRelayedCall` and `postRelayedCall` functions both have following comment in the official GSN `IPaymaster` class:

```
* MUST be protected with relayHubOnly() in case it modifies state.
```

The `preRelayedCall` function does not modify state but it could still make sense to add an access control check.

Code corrected:

The modifier `relayHubOnly` was added to `preRelayedCall` and `postRelayedCall`.

6.2 Outdated GSN Interface

Design | Medium | Version 1 | Code Corrected

The contracts use an outdated GSN interface in the definition of `IPaymaster`. In particular the following differences exist:

- The function `getGasLimits` should now be called `getGasAndDataLimits()` and should return four instead of three values.
- A `trustedForwarder` function should be implemented.

Furthermore, the `RelayData` struct has changed. As less context information is available the security guarantees are weaker.

Code corrected:

The GSN Interface was updated to version `2.2.0`.

6.3 Decimals in GSN Token Handling

Design | Low | Version 1 | Code Corrected

When the `ForeignBridgeErcToNative` is used in Gas Station Network (GSN)-mode, token decimals are treated unevenly. Generally, as part of this report we assume that as only the DAI tokens is used in GSN mode, the token decimals will always be 18. And hence token decimals on foreign and home side will be the same.

However, the `ForeignBridgeErcToNative` will shift token amounts to account for different decimals when transferring tokens:

```
uint256 unshiftMaxFee = _unshiftValue(fee);
uint256 unshiftLeft = _unshiftValue(amount - fee);
```

Furthermore, sometimes the code does assume that token decimals are the same. As part of the `preRelayedCall` the `TokenPayMaster` estimates the value of the provided tokens by calling:

```
uint256 potentialWeiIncome = router.getAmountsOut(maxTokensFee, tokenWethPair)[1];
```

Hence, overall either the shifts can be avoided or they should be consistently performed everywhere.

Code corrected:

The shifts were removed. The `XDaiForeignBridge` was introduced as a specialized contract to deal with DAI only.

6.4 Delete Statements Can Be Combined

Design Low Version 1 Code Corrected

In the function `forceDisable` in AAVEInterestERC20.sol and CompoundInterestERC20.sol, an instance of `InterestParams` is deleted.

```
delete params.aToken;
delete params.dust;
delete params.investedAmount;
delete params.minInterestPaid;
delete params.interestReceiver;
```

Because all fields of the struct are deleted, these delete statements can be combined into:

```
delete interestParams[_token];
```

Code corrected:

The statements were combined.

6.5 Events on Interest Params Change

Design Low Version 1 Code Corrected

The `CompoundInterestERC20` and `AAVEInterestERC20` contracts do not emit any events when values in `interestParams` are changed or initialized first time. List of such functions:

- `setMinInterestPaid`
- `setInterestReceiver`
- `setDust`
- `enableInterestToken`
- `setMinCompPaid` in `CompoundInterestERC20`
- `setCompReceiver` in `CompoundInterestERC20`

To make monitoring easier it can be helpful to add such events.

Code corrected:

Events were added.

6.6 Gas Saving if No Interest Implementation Exists

Design Low Version 1 Code Corrected

For each token an `IInterestImplementation` can be chosen inside the `ForeignOmnibridge`. When a token is handled this is checked as follows:

```
IInterestImplementation impl = interestImplementation(_token);
if (Address.isContract(address(impl))) {
```

In case no `IInterestImplementation` has been registered the `impl` value will be the zero address. Querying the existence of a contract at this address requires caching the account (due to EIP-2929) and hence incurs non-trivial gas costs. This could be avoided if the check would be modified to first handle the zero address.

Code corrected:

The more effective `address(impl) != address(0)` check was introduced.

6.7 Incorrect Documentation Comments

Correctness | Low | Version 1 | Specification Changed

The documentation on the `withdraw` function in CompoundInterestERC20.sol and AAVEInterestERC20.sol reads:

```
/**
 * @dev Withdraws at least the given amount of tokens from the Compound protocol.
 * ...
 * @param _amount minimal amount of tokens to withdraw.
 */
function withdraw(address _token, uint256 _amount) external override onlyMediator {
    InterestParams storage params = interestParams[_token];
    uint256 invested = params.investedAmount;
    uint256 redeemed = _safeWithdraw(_token, _amount > invested ? invested : _amount);
    params.investedAmount = redeemed > invested ? 0 : invested - redeemed;
    IERC20(_token).transfer(mediator, redeemed);
}
```

However, if the `_amount` specified is greater than the amount invested, only the amount invested will actually be redeemed. Therefore, an amount of tokens less than the specified `_amount` can be withdrawn, contradicting the documentation.

Additionally, the comment on the `_enableInterestToken` function in CompoundInterestERC20.sol reads:

```
/**
 * ...
 * @param _interestReceiver address of the interest receiver for underlying token and associated COMP tokens.
 * ...
 */
function enableInterestToken(ICToken _cToken, uint96 _dust, address _interestReceiver, uint256 _minInterestPaid) external onlyOwner {
    // ...
}
```

However, the account that receives COMP tokens is `compReceiver`, which is set in the constructor.

Lastly, the comment on the `_enableInterestToken` function in AAVEInterestERC20.sol also mentions COMP tokens even though no COMP tokens will be handled.

Specification changed:

The comments were fixed and don't have mentioned contradictions anymore.

6.8 Interest Implementation With Non-Revoked Allowance

Design Low Version 1 Code Corrected

When `enableInterestToken` is called within the `CompoundInterestERC20` and `AAVEInterestERC20` contracts, the respective interest implementation approves the lending pool for the maximum amount of tokens. When `forceDisable` is called, this allowance is not revoked.

Code corrected:

The allowance is set to 0 in `forceDisable` function now.

6.9 Interest Implementations Cannot Handle Non-Standard Tokens

Design Low Version 1 Code Corrected

Both the Compound and AAVE platforms can work with non-standard ERC20 tokens. However, the `CompoundInterestERC20` and `AAVEInterestERC20` do not use `SafeERC20` functions for transfers and allowances of the tokens. Hence, they will fail to handle such tokens.

Code corrected:

The SafeERC20 is now used for token transfers. Allowances are handled based on the token type and the situation.

6.10 Interface `IInterestImplementation` Events

Design Low Version 1 Code Corrected

Both `CompoundInterestERC20` and `AAVEInterestERC20` define the following events:

```
event PaidInterest(address indexed token, address to, uint256 value);
event ForceDisable(address token, uint256 tokensAmount, uint256 cTokensAmount, uint256 investedAmount);
```

Since they both also implement `IInterestImplementation` interface, such events can be defined in there.

Code corrected:

Events are now defined in `IInterestImplementation`.

6.11 Message Offset Incorrect

Correctness Low Version 1 Specification Changed

The function `parseMessage` is preceded by a comment about the offsets of the different parts within the message. It contains:

```
// offset 104: 20 bytes :: address - contract address to prevent double spending
```

However the correct offset is 116. The code implements it correctly.

Specification changed:

The comment now has the correct offset of 116.

6.12 Minimum Gas Computed Incorrectly

Correctness Low Version 1 Code Corrected

As part of the AMB a gas limit for the call on the other side can be provided. The provided gas limit is checked against a minimum gas usage computed based on the number of bytes within a call:

```
function getMinimumGasUsage(bytes _data) public pure returns (uint256 gas) {
    // From Ethereum Yellow Paper
    // 68 gas is paid for every non-zero byte of data or code for a transaction
    // Starting from Istanbul hardfork, 16 gas is paid (EIP-2028)
    return _data.length.mul(16);
}
```

However, as the call is made from within a smart contract, these costs do not apply. They only apply to initial calldata of a transaction. The dominating cost, namely the cost for the CALL itself (increased by EIP-2929) is not considered.

Code corrected:

The new code version uses a fixed value according to the new gas prices as minimum gas requirement.

6.13 No Claiming of Aave Rewards

Design Low Version 1 Code Corrected

Aave issues rewards for participants of the system which can be claimed using the `claimRewards` function. However, the `AAVEInterestERC20` contract currently does not collect any of these rewards.

Code corrected:

The functionality to claim the rewards from AAVE's IncentivesController contract was added to the AAVEInterestERC20 contract.

6.14 Potentially Conflicting Getters

Design | Low | Version 1 | Code Corrected

The `ForeignBridgeErcToNative` contract has getter functions that return potentially conflicting information:

- `erc20token()`: This function describes the ERC20 token that is bridged on the foreign side. It reads the return value from storage.
- `daiToken()`: This function also describes the ERC20 token that is bridged on the foreign side. It is set to a constant address.

Code corrected:

The `XDaiForeignBridge` was introduced as a specialized contract to deal with DAI only.

6.15 Unnecessary Balance Query

Design | Low | Version 1 | Code Corrected

In the function `forceDisable` in AAVEInterestERC20.sol, `aToken.balanceOf` is queried in order to withdraw all invested funds:

```
uint256 aTokenBalance = aToken.balanceOf(address(this));
// try to redeem all aTokens
try lendingPool().withdraw(_token, aTokenBalance, mediator) {
    aTokenBalance = 0;
} catch {
    aToken.transfer(mediator, aTokenBalance);
}
```

This balance query is not necessary, because according to the documentation of `LendingPool.withdraw()`, one can set the balance to `type(uint256).max` (or `uint256(-1)`) to withdraw the entire balance. In case the withdraw fails, the balance could be queried inside the catch block in order to transfer the aTokens to the mediator.

```
/**
 * ...
 * @param amount The underlying amount to be withdrawn
 *   - Send the value type(uint256).max in order to withdraw the whole aToken balance
 * ...
**/
function withdraw(address asset, uint256 amount, address to) external returns (uint256);
```

Code corrected:

The updated withdraw code does not have unnecessary balance query.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Changing the Number of Required Signatures

Note Version 1 Acknowledged

AMB messages require a certain number of signatures. However, this threshold can be changed. If changed, a number of issues might occur. For example, it is impossible to submit additional signatures if a message has already been marked as processed due to sufficient signatures. However, additional signatures would be required to successfully handle the message on the other side of the bridge.

Acknowledged:

POA Network acknowledged that such issues could arise and presented a work-around where one side of the bridge temporarily rejects new requests.

7.2 Hardcoded GSN Gas Costs

Note Version 1 Acknowledged

The GSN-related contracts contains a number of constants describing the gas costs of certain functions or operations. Past hard-forks have changed the gas costs of EVM operations and future hard-forks are expected to introduce further changes. Hence, the constants might no longer work in the future. However, they have been chosen with some margin so that fairly dramatic gas cost changes would have to occur for these constants to stop working.

Acknowledged:

The client acknowledged the issue, but it is out of their control as it is a general issue with GSN.

7.3 Validation of Used dataType

Note Version 1 Acknowledged

AMB messages can carry a `dataType`. Together with other components, this value of part of the message header created within the `_packHeader` function. Here, this `uint256` value is being packed into a single byte. No length check are performed on this value, while length checks are performed on some of the other values. Hence, a length check could be added. However, with currently used hard-coded values no issue will arise.

Acknowledged:

POA Network plans such improvements for future version with more complicated data types.