Contract Audit: Bridge

Preamble

This audit report was undertaken by @adamdossa for the purpose of providing feedback to the UniTrade team. It has been written without any express or implied warranty.

This audit was done on the code committed to Github as:
https://github.com/UniTradeApp/bridge/tree/ced7db1164cae139424778897cadf7885a254a4f/packages/contracts/contracts
which matched the branch `audit-may-5` at the time of writing.

Disclosure

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. There is no owed duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

Classification

- **Comment** - A note that highlights certain decisions taken and their impact on the contract.
- **Minor** - A defect that does not have a material impact on the contract execution and is likely to be subjective.
- **Moderate** - A defect that could impact the desired outcome of the contract execution in a specific scenario.
- **Major** - A defect that impacts the desired outcome of the contract execution or introduces a weakness that may be exploited.
- **Critical** - A defect that presents a significant security vulnerability or failure of the contract across a range of scenarios.

Audit

This audit covers the contracts:

- Gateway.sol: this contract is intended to be deployed on both chains that the gateway is connecting, and manages moving tokens between chains.
- PriceOracle.sol: this is a wrapper around the UniswapPairsOracle contract, to allow a caller to easily determine the price in DAI of a token.
- UniswapPairsOracle.sol: this contract manages a fixed window oracle based on Uniswap V2 liquidity pools

There are also corresponding interfaces for the oracle contracts, and a mintable ERC20 which were reviewed, but which don't contain on-chain logic directly.

Contract Behaviour

These contracts allow a centralised party (the owner of the Gateway contract) to manage token transfers across EVM chains.

Anyone can call `transferToAnotherChain` on the source chain provided that the `tokenId` is valid, in order to deposit tokens to the Gateway contract.

The centralised party can then call `transferFromAnotherChain` on the destination chain to release tokens held by the Gateway contract back to the user on the destination chain.

The oracles are used to determine the fees levied by the Gateway contract, denominated in the token being transferred.

A brief summary of the expected workflow is:

1. Gateway (and associated oracle contracts) are deployed to both the source and destination chains (e.g. Ethereum and BSC).
2. The Gateway owner calls `addToken` for a given token address, on both the source and destination chains. The `chains` parameter specifies the list of chains on which tokens can be withdrawn from the Gateway contract, as deployed on a given chain.
3. Any user can call `transferToAnotherChain` to transfer / burn a pre-approved token balance to the Gateway contract on the source chain. Fees are levied by the Gateway contract at this point.
4. The Gateway owner calls `transferFromAnotherChain` to transfer / mint tokens on the destination chain to the user. No additional fees are levied by the Gateway contract.

NB - the Gateway contract can be used symmetrically - for example, if ACME tokens can be moved from Ethereum to BSC and from BSC to Ethereum then we would have:

a. Owner calls `addToken` on Ethereum passing `addToken(acmeAddressOnEthereum, ..., [BSCId])`

b. Owner calls `addToken` on BSC passing `addToken(acmeAddressOnBSC, ..., [EthereumId])`

Issues

Minor - `toChainId` is incorrectly typed as uint64

In the function `transferToAnotherChain` the parameter `toChainId` is passed in as a `uint64` rather than the expected type of `uint256`. Solidity will implicitly cast this to a `uint256` but it would be neater to use the correct type throughout rather than relying on implicit casting.

Mitigation

Change the parameter type of `toChainId` from `uint64` to `uint256`.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Minor - front running of fees is possible by Gateway owner

The Gateway contract allows the owner to update the fees levied on a user via `updateNetworkFee` and `setFeeParameters`.

These fee updates are applied immediately, allowing the Gateway owner to front-run a call from a user to `transferToAnotherChain`, meaning that the user could be forced to pay a much larger fee than expected (up to the amount they are depositing to the Gateway contract).

For a user of the Gateway contract this might be off-putting for large transactions, although given that they are trusting the centralised operator to release their funds on the destination chain, this may be a relatively small concern. However in the case of the Gateway owner not releasing tokens on the destination chain, the owner can't directly profit on the source chain, whereas by modifying the fees they can immediately withdraw these fees on the source chain.

Mitigation

Consider adding a timelock approach to fee updates, where they only apply a few blocks after they are updated. The simplest approach is to have a `proposeFeeParameters` and `confirmFeeParameters` that can only be called X blocks after the `proposeFeeParameters` function.

Client Response

Issue not closed with comment:

- For the sake of administrative operations simplicity
- This is a centralized solution where users have to trust us anyway
- It is a minor issue
- I'll keep this issue open to review and possibly implement this in a future version of the code

Minor - tokens can be added twice via `addToken`

It is possible that the same token can be added twice through the `addToken` function, by the owner of the Gateway contract.

This is only possible if the token wasn't previously added to the price oracle via `priceOracle.addToken(tokenAddress);` which could be the case if the `priceOracle` is 0x0 either when the token was first added, or when the token was added for the second time.

Whilst this doesn't seem to cause a direct issue, it could be confusing, for example with different fee settings across the two instances of the token.

Mitigation

See next issue.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/94

Minor - tokens are referenced through `tokenId` rather than through their address

The Gateway contract generally references tokens that have been previously added through `addToken` via their index in the `tokens` storage.

In my opinion it would be cleaner, and less subject to errors to instead reference tokens via their address.

A simple way to do this would be to store a mapping:
`mapping(address => uint256) public addressToTokenId;`

This could also be used to ensure that the same token cannot be added twice by the Gateway owner and make it less susceptible to user error when interacting with the contract.

Mitigation

As above, you could store a mapping from a token address to its index (or its index + 1) in the `tokens` storage. You can then check that this mapping has not been set for a given token, before allowing it to be (re)added in `addTokens`.

Functions such as `transferFromAnotherChain` could then take a token address as their parameter, rather than a `tokenId` and use this mapping to convert to the `tokenId`.

Client Response

Issue not closed with comment:

- Related issue (`addTokens` called twice with the same token) was addressed
- Changing would impact UIs etc.

Minor - `token.transfer(to, amount)` could use safeTransfer

In line:
https://github.com/UniTradeApp/bridge/blob/ced7db1164cae139424778897cadf7885a254a4f/packages/contracts/contracts/Gateway.sol#L118
a call is made to a potentially untrusted token, to transfer tokens to the user on the destination chain.

It would be better to use the `safeTransfer` wrapper which checks the return value of this function, as some token implementations may return `false` rather than reverting on a failure.

Mitigation

Use `safeTransfer` wrapper from `SafeERC20.sol` instead of directly calling `transfer` on the token.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Minor - for MINT release method tokens, you don't need to explicitly track `feeToWithdraw`

For tokens that use the MINT release method, the token balance remaining on the Gateway contract should always be equal to the fees that have been levied on deposits.

On the source chain, the fee is the only portion of deposited tokens which are not burnt, and on the destination chain no token balance is held directly on the Gateway contract (instead balances are minted on demand).

This would simplify calculations in the MINT case and also make the contract more robust for tokens which muck around with token balances (i.e. deflationary / inflationary type ERC20 tokens).

Mitigation

Consider just using the Gateway contracts token balance (i.e. `token.balanceOf(address(this))`) when withdrawing fees for MINT style tokens, rather than tracking through `feeToWithdraw`.

Client Response

Issue not closed with comment:

- Keep code simple (avoiding handling fees differently for each release method);
- This solution doesn't support rebase tokens (that change account balances like AMPL). There are other things to consider for these types of tokens.

Minor - inflationary / deflationary token calculations

The fees levied by the Gateway contract are tracked as absolute values in the `feeToWithdraw` storage. If the amount tracked here cannot be transferred, then no fees for the token can be withdrawn.

As a generalisation of the above issue (although with more complex logic) you could consider tracking fees as a percentage of funds that are deposited on the Gateway contract rather than the absolute amount. This would again potentially make the contract more robust for unusual ERC20 tokens, and avoid the issue that fees might not be withdrawable at all of if the expected fee amount cannot be transferred.

Mitigation

Consider tracking fees as a percentage of deposited funds, rather than absolute amounts.

Client Response

Issue not closed with comment:

- We don't support rebase tokens like AMPL,
- If we came up in a situation where funds a locked, we can upgrade the contract implementation with a withdraw(tokenAddress,amount) kind of function.

Minor - Duplicated code

In line:
https://github.com/UniTradeApp/bridge/blob/ced7db1164cae139424778897cadf7885a254a4f/packages/contracts/contracts/Gateway.sol#L165
there is duplicated logic. The same check is done 3 lines above.

Mitigation

Remove duplicated code - `require(feeShareToTheTeam <= 1e18, "Gateway: Invalid feeShareToTheTeam");`

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Minor - possible for the service fee to be larger than the amount being deposited

In the case where the USD value of the tokens being transferred, multiplied by the `serviceFeePercent` is less than `minVariableFeeInUsd` it is possible that the fee could be larger than the amount of tokens being transferred.

e.g.

```
usdcAmount == 100 USD
serviceFeePercent == 10%
feeInUsdc == 10 USD
minVariableFeeInUsd == 200 USD
tokenAmount.mul(minVariableFeeInUsd).div(usdcAmount) == tokenAmount.mul(2)
```

Mitigation

You might want to consider having a cap in this case of `serviceFeePercentWhenNoPrice` on the percentage of tokens that are withheld as a fee.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/94

Minor - usage of Uniswap V2 vs. V3

The contract currently uses Uniswap V2 as its price oracle. A new version of Uniswap (V3) has recently been released. V3 updates the approach to oracles, improving the calculation and reducing gas costs - see https://uniswap.org/blog/uniswap-v3/ for details.

Whilst V2 remains on-chain, and will never be removed, it may be that tokens migrate over to V3 liquidity pools, reducing the accuracy of V2 based oracles.

Mitigation

Consider moving to using V3 as a pricing oracle. It should be noted that the pricing oracle address can be updated however, so this could be done after deployment. However, as per the issue above there is no way to re-initialise oracle pairs at the moment if the pricing oracle is updated (unless this was done directly on the new pricing oracle contract).

Client Response

Issue not closed with comment:

- Since the oracle piece is upgradable, and the first tokens supported still using V2, we'll work to support V3 oracle later.

Comment - some confusion in variable names between USDC and DAI

The variables used in fee calculations are e.g. `feeInUsdc` and `usdcAmount`. However, based on the oracle logic you are actually using amounts denominated in DAI rather than USDC.

Mitigation

Consider changing the variable names to more closely match the expected logic.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Comment - fixed window cumulative price oracles can over-weight historical pricing

The UniswapPairsOracle uses a fixed window approach, detailed at:
https://uniswap.org/docs/v2/smart-contract-integration/building-an-oracle/

Depending on the frequency that the Gateway contract is used (specifically the `transferToAnotherChain` function) - which causes the oracle to update its TWAP prices, it is possible that the oracle may place a large weight on historical prices vs. more recent prices.

This "slippage" may be considered a reasonable risk vs. the complexity of an oracle that weights prices based on how long ago they were observed.

Mitigation

If this risk is considered large, you could consider a moving average oracle rather than a fixed window oracle.

Client Response

Issue not closed with comment:

- Not considered risky - will observe the oracle behaviour to see if worth improving it.

Comment - `updateTokenSupportedChain` doesn't support vectorised chainIds

The `addToken` function allows you to specify an array of supported chains, whereas the `updateTokenSupportedChain` only allows a single chainId to be specified on each call.

It might be worth considering making this a vectorised function (similar to `addToken`) as this will be more gas efficient when adding multiple supported chains.

Mitigation

Add a vectorised form of this function `updateTokenSupportedChain`.

Client Response

Issue not closed with comment:

- A new chain addition is rare and I believe they will be done one-by-one.

Comment - check that a network fee is set when adding supported chains

The Gateway contract uses `networksFee` to track the cost (in ETH / BSC) that must be paid by a user depositing to the source chain, to cover the costs of the `transferFromAnotherChain` on the destination chain.

However, there is no check that this value is set (i.e. non-zero) when adding support for a particular `chainId` in the `addToken` function.

Mitigation

Consider checking whether `networkFee` is set when adding a supported chain for a particular token.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/94

Comment - no event emitted from `setPriceOracle`

Generally events are emitted when configuration of the Gateway is modified, but in this case no event is emitted. This may be useful for UIs to display these updates.

Mitigation

Consider adding an event when the price oracle is updated.

Client Response

Issue not closed with comment:

- It isn't an event that we want to track from our UI.

Comment - network fees are denominated in the source chain token, but paid in the destination chain token

In order to pay for the transaction on the destination chain, a fee in the source chain network token is levied on deposits.

The Gateway owner should manage any price volatility between these tokens appropriately to ensure that there is sufficient buffer to account for varying fees on the destination chain, and changes in the destination vs. source network token prices (i.e. the ETH / BSC price).

Mitigation

None really possible - just flagging it up. It might be possible to use an oracle to help manage the ETH / BSC price volatility so that you can denominate the cost in the destination chain token, rather than the source chain token.

Client Response

Issue not closed with comment:
- Yes, there isn't a definitive solution for that. We've implemented logic for the fees monitoring/updating some times per day. We have the same logic running for our beta version of the bridge and it's working well.
- Will monitor the fee setter component during the first days to see if it's working as expected and improve it if needed.

Comment - small typo in "Gateway: token does't exist"

There is a small typo in this `require` error message - "does't" should be "doesn't".

Mitigation

Fix typo.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Comment - small ordering optimisation

In line:
https://github.com/UniTradeApp/bridge/blob/ced7db1164cae139424778897cadf7885a254a4f/packages/contracts/contracts/Gateway.sol#L108
the initialisation of `token` could be moved below the check that `requestKey` has not been previously processed.

This would be a minor gas optimisation in the case that this `require` fails.

Mitigation

Move initialisation of `token` below the `require` check on `requestKey` not having been seen previously.

Client Response

Fixed via:
https://github.com/UniTradeApp/bridge/pull/92

Testing

The repo contains detailed test cases that cover the majority of the contract functionality, with 52 unit tests included which all pass.

There is good coverage across all of the in-scope contracts:

```
 -------------------------|----------|----------|----------|----------|----------------|
File                     |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------------|----------|----------|----------|----------|----------------|
 contracts/              |    99.24 |    83.33 |       96 |    98.48 |                |
  Gateway.sol            |     98.8 |    86.36 |    93.75 |    97.62 |        217,245 |
  PriceOracle.sol        |      100 |    58.33 |      100 |      100 |                |
  UniswapPairsOracle.sol |      100 |    93.75 |      100 |      100 |                |
 contracts/interfaces/   |      100 |      100 |      100 |      100 |                |
  IERC20Mintable.sol     |      100 |      100 |      100 |      100 |                |
  IPriceOracle.sol       |      100 |      100 |      100 |      100 |                |
  IUniswapPairsOracle.sol |      100 |      100 |      100 |      100 |                |
 contracts/tokens/       |        0 |        0 |        0 |        0 |                |
  FROGE_goerli.sol       |        0 |        0 |        0 |        0 |... 687,689,690 |
  FROGE_rinkeby.sol      |        0 |        0 |        0 |        0 |... 705,707,708 |
-------------------------|----------|----------|----------|----------|----------------|
All files                |    26.37 |    26.55 |    17.27 |    26.86 |                |
-------------------------|----------|----------|----------|----------|----------------|
```

Of the uncovered lines in Gateway.sol, one related to chainId which is not used on-chain, and the other to the case where there is no fee to be withdrawn for the supplied token. The latter case may be worth adding as a test case.