Polygon Bridge Mapping Audit

Report

DATE 27/12/2021

Conducted By: Barney Chambers (Dev Barnyard)

Disclaimer

not represent the opinions of the authors of the document. and is provided for information and evaluation purposes only and is not currently a formal offer capable of acceptance.

Objectives

The audit is being undertaken to ensure that NFT World's ERC20 token contracts conform to the Polygon bridge mapping specification. The contracts described in this document are intended to be integrated into the *Polygon bridge* and *Polygon mapper*, and must conform to the Polygon Mintable Assets Specification

Objective 1 - Ensure the NFT World token to be deployed on the Ethereum mainnet, conforms to the specifications outlined on the Polygon Mintable Asset Specification

Objective 2 - Ensure the NFT World token to be deployed on the Polygon mainnet, conforms to the specifications outlined on the Polygon Mintable Asset Specification

Report completed	By: Barney Chambers	Date: 27/12/2021
Report distributed	By: Barney Chambers	Date: 27/12/2021

The type of bridge has to be selected from the "Choose map type" dropdown.

NFT World are bridging using the Polygon PoS bridge.

The type of your token can be selected by switching among the three tabs marked as "ERC20", "ERC721" and "ERC1155". For mapping any other token standard, you can reach out to the Polygon team on Discord or create a ticket here and keep "Token Mapping" in the ticket title.

WRLD token inherits the ERC20 standard.

"Choose network" will let you select the network on which you need the mapping to be done. For mainnet mappings you can choose Ethereum - Polygon Mainnet and for testnet mappings you can choose Goerli Testnet - Mumbai.

Token mapping will be performed on the Ethereum Mainnet, chain ID 0x1

Enter your Ethereum/Goerli token address in the "Ethereum token address" field. Ensure that your token contract code is verified on the Ethereum/Goerli blockchain explorers.

To be addressed, once the tokens have been deployed.

In case you need a standard ERC20/ERC721/ERC1155 child token, you may leave the "Polygon token address" field empty. But, if you need a custom child token (standard ERC functions + custom functions), you can follow this guide to create a custom child token. Once you deploy your custom child token, you can mention the contract address in the "Polygon token address" field. Please ensure that you verify your child token contract code too on Polygon/Mumbai explorer.

The WRLD ERC20 token does not directly inherit ChildMintableERC20.sol however, all necessary functionality from this contract has been included in the WRLD contract.

Note: You will have to verify the contract code of the WRLD contract on Polygonscan, this can be done during deployment using Hardhat

https://moralis.io/how-to-verify-a-smart-contract-with-hardhat/

If your root token is verified, the name, symbol and decimals fields will be automatically filled for you and these fields cannot be edited.

Not applicable.

You may choose either "Polygon Mintable" or a "Non Polygon Mintable" token from the drop down. More details on the Polygon Mintable tokens can be found here.

“Polygon Mintable” is the required selection.

Objective 1 - Contract To Be Deployed On Ethereum

Polygon's requirements: If you want to deploy the contract by yourself, ensure that the deposit, withdraw and mint functions are present.

The smart contract *WLRD_Token_Ethereum.sol* inherits the interface IMintableERC20.sol and overrides the mint() function by calling the inherited function like so:

```
  function _mint(address to, uint256 amount) internal override(ERC20, ERC20Capped) {
    super._mint(to, amount);
  }
}
```

the implementation of the *super._mint()* function in this context is the standard ERC20 mint function defined by Open Zeppelin, which is called upon by the NPM installation script:

```
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
    }

```

```
  },
  "devDependencies": {
    "@nomiclabs/hardhat-ethers": "^2.0.3",
    "@nomiclabs/hardhat-etherscan": "^2.1.8",
    "@nomiclabs/hardhat-waffle": "^2.0.1",
    "@openzeppelin/contracts": "^4.4.0",
    "chai": "^4.3.4",
    "dotenv": "^10.0.0",
    "eslint": "^7.32.0",
    "eslint-config-prettier": "^8.3.0",
```

The deposit() and withdraw() functions are correctly implemented as per Polygon's requirements, but the implementation style does not follow

Polygon's recommendation to inherit from ChildMintableERC20.sol. However, the result is the same. PREDICATE_ROLE is given minting privileges.

```
function deposit(address user, bytes calldata depositData) external override only(DEPOSITOR_ROLE) {
  uint256 amount = abi.decode(depositData, (uint256));
  _mint(user, amount);
}
```

```
function withdraw(uint256 amount) external {
  _burn(_msgSender(), amount);
}
```

```
function mint(address user, uint256 amount) external override only(PREDICATE_ROLE) {
  _mint(user, amount);
}
```

Objective 2 - Contract To Be Deployed On Polygon

Polygon's requirements: **A token contract has to be deployed on the Ethereum chain and it should look like this: MintableERC20 - https://github.com/maticnetwork/pos-portal/blob/master/flat/DummyMintableERC20.sol**

The smart contract *WLRD_Token_Polygon.sol* must inherit or include the following functionalities. In this document, each required inheritance is described and referenced to the actual implementation chosen in *WLRD_Token_Polygon.sol*

```
contract DummyMintableERC20 is
    ERC20,
    AccessControlMixin,
    NativeMetaTransaction,
    ContextMixin,
    IMintableERC20
{
```

Required:

The IERC20 interface found at IERC20.sol and the standard implementation of ERC20.sol

Implementation:

IERC20.sol is inherited by ERC20.sol. ERC20.sol is inherited by *WLRD_Token_Polygon.sol*

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/metatx/ERC2771Context.sol";
import "./common/IChildToken.sol";
import "./common/AccessControlMixin.sol";
import "./common/NativeMetaTransaction.sol";
import "./common/ContextMixin.sol";

```

```
contract WRLD_Token_Polygon is
  ERC20, ERC20Capped, ReentrancyGuard,
  ERC2771Context, IChildToken, AccessControlMixin,
  NativeMetaTransaction, ContextMixin
{
```

Required:

The AccessControlMixin contract, to be inherited by the ERC20 token to be deployed on the Polygon blockchain.

```
// File: contracts/common/AccessControlMixin.sol

pragma solidity 0.6.6;


contract AccessControlMixin is AccessControl {
    string private _revertMsg;
    function _setupContractId(string memory contractId) internal {
        _revertMsg = string(abi.encodePacked(contractId, ": INSUFFICIENT_PERMISSIONS"));
    }

    modifier only(bytes32 role) {
        require(
            hasRole(role, _msgSender()),
            _revertMsg
        );
        _;
    }
}

```

Implementation:

The context contract is imported as follows.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/metatx/ERC2771Context.sol";
import "./common/IChildToken.sol";
import "./common/AccessControlMixin.sol";
import "./common/NativeMetaTransaction.sol";
import "./common/ContextMixin.sol";

```

The image above references the following code.

contracts > common > AccessControlMixin.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/access/AccessControl.sol";

contract AccessControlMixin is AccessControl {
  string private _revertMsg;

  function _setupContractId(string memory contractId) internal {
    _revertMsg = string(abi.encodePacked(contractId, ": INSUFFICIENT_PERMISSIONS"));
  }

  modifier only(bytes32 role) {
    require(
      hasRole(role, _msgSender()),
      _revertMsg
    );

    _;
  }
}
```

Required:

The NativeMetaTransaction functionality. Implemented here

Implemented:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/metatx/ERC2771Context.sol";
import "./common/IChildToken.sol";
import "./common/AccessControlMixin.sol";
import "./common/NativeMetaTransaction.sol";
import "./common/ContextMixin.sol";

contract WRLD_Token_Polygon is
  ERC20, ERC20Capped, ReentrancyGuard,
  ERC2771Context, IChildToken, AccessControlMixin,
  NativeMetaTransaction, ContextMixin
{
  uint feeBps;
  uint feeFixed;
  uint feeCap;
  address private feeRecipient;

  bytes32 public constant DEPOSITOR_ROLE = keccak256("DEPOSITOR_ROLE");
```

Required:

The ContextMixin functionality. Implemented here

Implemented:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/metatx/ERC2771Context.sol";
import "./common/IChildToken.sol";
import "./common/AccessControlMixin.sol";
import "./common/NativeMetaTransaction.sol";
import "./common/ContextMixin.sol";

contract WRLD_Token_Polygon is
  ERC20, ERC20Capped, ReentrancyGuard,
  ERC2771Context, IChildToken, AccessControlMixin,
  NativeMetaTransaction, ContextMixin
{
```

Required:

The IMintableERC20 interface, with a standardised *mint()* implementation. Implemented here. DEFAULT_ADMIN_ROLE has minting privileges.

Implemented:

```
function mint(address to, uint256 amount) public only(DEFAULT_ADMIN_ROLE)  {
  _mint(to, amount);
}
```

```
    */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
    }
```

Conclusion

The contracts *WLRD_Token_Ethereum.sol* and *WLRD_Token_Polygon.sol* satisfy the requirements stated in the Polygon Mintable Assets documentation for bridging between the Ethereum mainnet and Polygon mainnet through the Polygon PoS bridge.

Document ratification and history			
Approved by:	Barney Chambers		
Date approved:	December 27 2021	Date placed on electronic library:	N/A
Review period:	1 Day		
Authors:	Barney Chambers	Document Owner:	ArkDev
Version number as approved and published:	1	Unique identifier no.:	NFTWRLD261221