Cyberscope

Audit Report

Liquid Layer Bridge

March 2024

Network	Sepolia/Liquid Layer Testnets
Audited by	© cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	17 Mar 2024

Source Files

Filename	SHA256
upgradeable_contracts/VersionableBridge.sol	98ce310be97e2b4ed56d300f8dc2915294c900fdecb9dbb664b153a917a34821
upgradeable_contracts/Upgradeable.sol	9ac8b6c1324afa845a4f728dbfec4dd4108ad0522ad1b7263a278132d9a838e8
upgradeable_contracts/Sacrifice.sol	af4f6265d171e2b121a107eb05b4f01d330376f560f28b998e95eb6dc6284fc9
upgradeable_contracts/ReentrancyGuard.sol	5a17bb7d1759774e085266fef8caf0f5dbec4b7d6a72b5c64fc5ceb4b3306c45
upgradeable_contracts/Ownable.sol	3fa8992c2f72fd40f390cb787a4d7c955103f894345e811f1497dc6ef1aa7c9c
upgradeable_contracts/Initializable.sol	d2d60447b3f315c45e93cc27c9b593c79deb4c903dbc98da014dfe042a9c9c42
upgradeable_contracts/HomeOmnibridge.sol	2865f5677cabeee8ddf0813b07c64b5721e0bb5eb35faf487259a98f9f21c3fb
upgradeable_contracts/Claimable.sol	5bb3795d4517542afc4350b467f0c9eb8c0b39fbf897b09a5d3c04fd56063e40
upgradeable_contracts/BasicOmnibridge.sol	0dd7e19335410ffc5b3ea6fb87b832a28846ca3501150f588c1d7287b9b9cd60

upgradeable_contracts/BasicAMBMediator.sol	f1f61f69e5cf7585dbfd61f595ee220e4ee6b209a610e9343cec95f713b92771
upgradeable_contracts/modules/OwnableModule.sol	df9991d6e31b9c9a3b04b10993ef44ff3d2a7bc55889600308b2a9913a3b2cb4
upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitManager.sol	4a7d70666d54039f996fbb60af8a22be4bf3a3b744d1751b60720a887c07abb6
upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitConnector.sol	46fb9c20e21bbd40aa9198dd239bda160675c3cb9b8a8a76e9e336b759fabb16
upgradeable_contracts/modules/forwarding_rules/MultiTokenForwardingRulesManager.sol	681a7c7fc57b002543d3d9a088dafdb78b7bc951d4ae6b47107ef497b2e38dd5
upgradeable_contracts/modules/forwarding_rules/MultiTokenForwardingRulesConnector.sol	57d1a3740c71809b6661c2c56c40ce114212781935778d7f3041795a3c0ceba7
upgradeable_contracts/modules/factory/TokenProxy.sol	c524c04ef08a280c1b45f10026bc98e624f822a10ca58b981aa647a9ae9c6bdb
upgradeable_contracts/modules/factory/TokenFactoryConnector.sol	68084e476e652623d6151cb017f9dadf316d561825f6855b000fd885498a84fc
upgradeable_contracts/modules/factory/TokenFactory.sol	6330f22331a0092add910b0c2c7419eebdcd613b99d9cdee3e3304aa203df058
upgradeable_contracts/components/native/NativeTokensRegistry.sol	2843918507290efaf8a88f1887a20d3459f971d12850e4b4f33b4827b268c0fd
upgradeable_contracts/components/native/MediatorBalanceStorage.sol	c13a3da3f0183095ee2f12c939e32f56a65b1d8194905a86d2c73828e4f6b3c0
upgradeable_contracts/components/common/TransferNativeFeeStorage.sol	97bb426f4525cff0df105a48687ad130088462dc61c7dcae9f97db348ec63eb1
upgradeable_contracts/components/common/TokensRelayer.sol	12a5265d4fbd75f3ff4b754d5f61d3667d48d188ed879ca308165e05479435b5

upgradeable_contracts/components/common/TokensBridgeLimits.sol	5199de627e448424122e922fd87793fdad6f97f7186ab72851d704c010cb49ad
upgradeable_contracts/components/common/OmnibridgeInfo.sol	a9138f1748c9dc67b5f79ce727809b606dd7b8785b7f8fc2f243a817c90d3fed
upgradeable_contracts/components/common/FailedMessagesProcessor.sol	864234ee913ae4b119da29ded5ff7d9b556c84364361ab98d144e8b1e217438f
upgradeable_contracts/components/common/BridgeOperationsStorage.sol	da83297a98ea833215c911d2479e279582acb0aebd83fb5e202d75ea35c89882
upgradeable_contracts/components/bridged/BridgedTokensRegistry.sol	6958ed165b715fcf010c3ad8d31a9878be235b7844fe90376d23431a0418fdf0
upgradeability/Proxy.sol	5418778e2788f3f14956dd8dce638ef4d7f6f285933b70aa1788c458cb47255b
upgradeability/EternalStorage.sol	746f3281080b5b554576666b79c078ac65954c4c6daa78efd6f89a7582e3e2e6
libraries/TokenReader.sol	b5fdeb0a6000147ef7ecb9feaa84218949ebb64d9d862f64bc67dfde9cc21513
libraries/SafeMint.sol	9ec3d46778eeaccc3b7ad4603714bfd694ab63bcecae25014f959fbea3bcff60
libraries/Bytes.sol	df70aa5700447803365ee47cd328ed09c5cc93a1c9b0c201701450beb086ff64
libraries/AddressHelper.sol	7aa34b60cc8ced2316abfa0f5617fc7996677ed09ed9b7c62919f1a16e8e1c01
interfaces/IUpgradeabilityOwnerStorage.sol	f6c6f26cbdd8b63a23575f4803369a766bdf702c22c83387cfbfa850d3519829
interfaces/IERC677.sol	275122465a235625c8382587acb74168c825886149cfc92895c6b94e2090f26b
interfaces/IERC20Receiver.sol	a1df6fb2f9f5f90614f8435eb637240a9558dd0b0f1b98005288eaea47834c0b

interfaces/IERC20Metadata.sol	6901722e1a10116a7c72f346209e49ea89ef4a0267707e2d1f8125fab58ffb00
interfaces/IBurnableMintableERC677Token.sol	413d9a47d974af1da307cdd977c9062a516359e72629b3c2c1640bfed701cffe
interfaces/IAMB.sol	ba02491f0d9e41f70f369d6a4f94422b7a39db489d78c0a53666be4c0232ede4
@openzeppelin/contracts/utils/Address.sol	405463588905f07e67a5337d4fc462aff87e8d723639dfb587fc24f07e516e34
@openzeppelin/contracts/token/ERC20/SafeERC20.sol	c373cb8cc18d50dd75713df9d906b4e1eb210af74caa4051851976bb30d5cc3e
@openzeppelin/contracts/token/ERC20/IERC20.sol	663bde8def619689c9f219a5774486f1770f3be73411f1fe3ad9071c098c76a4
@openzeppelin/contracts/math/SafeMath.sol	fe67cffc488d9fae5f95e150a3507b7b0881d05388172b965a97c34b417c1a1b
ForeignOmnibridge.sol	adbff6939e3366a01cfe9694f6202084aa49857e17dca505fe4cc8483393ec15

Overview

Cyberscope conducted an audit of eight contracts within the Liquid Layer's Bridge ecosystem. These contracts include the HomeOmnibridge, TokenProxy, two EternalStorageProxy contracts deployed on the Sepolia Testnet, as well as the ForeignOmnibridge, Wrapped LILA, and two EternalStorageProxy contracts deployed on the Liquid Layer Testnet.

The HomeOmnibridge and ForeignOmnibridge contracts serve as the core components responsible for bridging logic on both ends of the bridge. They facilitate the deployment of new tokens to the bridge, with initiation exclusively by the mediator. Bridging a token involves locking funds on one side of the bridge and releasing them on the other side through either minting wrapped tokens or transferring the native token, depending on whether the token is native to the destination address.

EternalStorageProxy contracts are employed to maintain the contract's storage post-upgrade, adhering to a standard pattern.

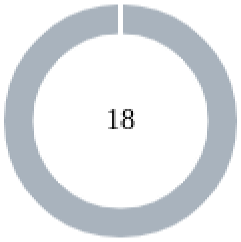
The WLILA contract wraps the LILA native token of the LiquidLayer network, enabling its utilization within the network's dApp ecosystem in a standardized manner.

TokenProxy acts as a proxy to an ERC20 token implementation, serving as an intermediary between the bridge and the respective token.

Test Deployment

Contract	Explorer
HomeOmnibridge	https://sepolia.etherscan.io/address/0x0629C07d768012597cA14580b13Cd33BCef03d76
EternalStorageProxy	https://sepolia.etherscan.io/address/0x88c5844C700F8Feb6Bb32b20eE8EF355224D5c44
EternalStorageProxy	https://sepolia.etherscan.io/address/0xEC0D14f55D626FadAcAaB8529079A4dc79BA2644
TokenProxy	https://sepolia.etherscan.io/address/0x38b1cd4784E09aDc3E8425Ea55086aC750f659E1
ForeignOmnibridge	https://testnet-scan.liquidlayer.network/address/0x985FBA12CAf0A8Ba7EA2ABE2334d69FC68bF6559
EternalStorageProxy	https://testnet-scan.liquidlayer.network/address/0x440D4A77b896520Fae5b457898815184F181b0fe
EternalStorageProxy	https://testnet-scan.liquidlayer.network/address/0xC695e005789D07E846072b8747eD33893AC2f0D7
WLILA	https://testnet-scan.liquidlayer.network/address/0xc2F9874231FEA886F1ecb26E1D4913fD88153BBD

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	16	2	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UPM	Unimplemented Protocol Method	Acknowledged
●	THMC	Token Handling Missing Checks	Acknowledged
●	CART	Callback Always Returns True	Unresolved
●	CACH	Checking Account Code Hash	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	OCV	Outdated Compiler Version	Unresolved
●	PMD	Potential MessageId Duplication	Unresolved
●	ULLCRV	Unused Low Level Call Return Value	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L06	Missing Events Access Control	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

UPM - Unimplemented Protocol Method

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/Upgradeable.sol#L18 contracts/upgradeability/OwnedUpgradeabilityProxy.sol#L29,L40 contracts/upgradeable_contracts/Ownable.sol#L36
Status	Acknowledged

Description

The `contract HomeOmnibridge.sol` uses the `modifier onlyIfUpgradeabilityOwner`. This modifier utilizes the `function upgradeabilityOwner`, by wrapping `address(this)` with the `interface IUpgradeabilityOwnerStorage`. Although, the `contract HomeOmnibridge.sol` doesn't implement this function, and the modifier will revert the transaction to any function utilizing it.

```
function _onlyIfUpgradeabilityOwner() internal view {
    require(msg.sender ==
IUpgradeabilityOwnerStorage(address(this)).upgradeabilityOwner());
}
...
require(msg.sender == upgradeabilityOwner());
...
emit ProxyOwnershipTransferred(upgradeabilityOwner(), newOwner);
...
modifier onlyRelevantSender() {
    (bool isProxy, bytes memory returnData) =

address(this).staticcall(abi.encodeWithSelector(UPGRADEABILITY_OWNER));
    require(
        !isProxy || // covers usage without calling through storage
proxy
            (returnData.length == 32 && msg.sender ==
abi.decode(returnData, (address))) || // covers usage through regular
proxy calls
            msg.sender == address(this) // covers calls through
upgradeAndCall proxy method
    );
    _;
}
...
function fixMediatorBalance(address _token, address _receiver)
    external
    onlyIfUpgradeabilityOwner
    validAddress(_receiver)
...
function claimTokens(address _token, address _to) external
onlyIfUpgradeabilityOwner
...
    function claimTokensFromTokenContract(
        address _bridgedToken,
        address _token,
        address _to
    ) external onlyIfUpgradeabilityOwner
```

Recommendation

The team is advised to define the `upgradeabilityOwner` function within the `HomeOmnibridge` contract. This function should return the address of the upgradeability owner, ensuring consistency with the access control mechanism enforced by the `onlyIfUpgradeabilityOwner` modifier.

Team Update

The team will use the `HomeOmnibridge.sol` through a proxy that implements the `upgradeabilityOwner()` method.

THMC - Token Handling Missing Checks

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/BasicOmnibridge.sol#L158
Status	Acknowledged

Description

During the assessment of the `BasicOmnibridge` contract, it was observed that the function `handleNativeTokens` lacks a security check before processing native tokens. The function `handleNativeTokens` is responsible for handling native tokens bridged from the other chain. However, it does not include a verification step to ensure that the token being handled is registered as a native token.

```
function handleNativeTokens(
    address _token,
    address _recipient,
    uint256 _value
) external onlyMediator {
    _ackBridgedTokenDeploy(_token);

    _handleTokens(_token, true, _recipient, _value);
}
```

Recommendation

In order to ensure the contract's robustness and security, it is advisable to implement a verification step within the `handleNativeTokens` function to validate whether the token being processed is registered as a native token using the `isRegisteredAsNativeToken` function. This check should be performed before any token handling logic is executed

Team Update

The team does the appropriate checking at `_prepareMessage` before passing the data to the AMB Bridge.

CART - Callback Always Returns True

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/components/common/TokensRelayer.sol#L24
Status	Unresolved

Description

The contract `TokensRelayer` contains a `callback` function named `onTokenTransfer` that is designed to handle token transfers. However, a notable concern arises from the unconditional return value of true at the end of this function, irrespective of the execution path taken within the function. This finding warrants attention as it may introduce unintended behavior.

```
function onTokenTransfer(
    address _from,
    uint256 _value,
    bytes memory _data
) external returns (bool) {
    if (!lock()) {
        bytes memory data = new bytes(0);
        address receiver = _from;
        if (_data.length >= 20) {
            receiver = Bytes.bytesToAddress(_data);
            if (_data.length > 20) {
                assembly {
                    let size := sub(mload(_data), 20)
                    data := add(_data, 20)
                    mstore(data, size)
                }
            }
        }
        bridgeSpecificActionsOnTokenTransfer(msg.sender, _from,
receiver, _value, data);
    }
    return true;
}
```

Recommendation

To address this finding and ensure the contract's robustness and security, it is advisable to implement a conditional return statement within the `onTokenTransfer` function. This return statement should reflect the outcome of the `bridgeSpecificActionsOnTokenTransfer` function, ensuring that true is returned only when the bridge-specific actions are successfully executed.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/factory/TokenProxy.sol#L60,61,62,63,64
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
decimals
owner
bridgeContractAddr
bridgeFeeAddr
bridgeFeeAmount
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	upgradeable_contracts/HomeOmnibridge.sol#L47,79,123,125,151,159 upgradeable_contracts/BasicOmnibridge.sol#L48,122,144,220,221,222,249,252,254,273,352,441,455 upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitManager.sol#L23,51,61,64,171,172,173,174,175,197,215,216,217,218 upgradeable_contracts/modules/factory/TokenFactory.sol#L46 upgradeable_contracts/components/common/TokensRelayer.sol#L107,113,115,117,124 upgradeable_contracts/components/common/TokensBridgeLimits.sol#L137,138,150,151,163,164,175,176,186,187,244,263 upgradeability/Proxy.sol#L21 WLILA.sol#L26,50,53
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(msg.sender ==
IUpgradeabilityOwnerStorage(address(this)).upgradeabilityOwner())
require(msg.sender == owner())

require(
            !isProxy || // covers usage without calling through storage
proxy
                (returnData.length == 32 && msg.sender ==
abi.decode(returnData, (address))) || // covers usage through regular
proxy calls
                msg.sender == address(this) // covers calls through
upgradeAndCall proxy method
        )
require(newOwner != address(0))
require(!isInitialized())
require(msg.sender == address(this))
require(!lock())
require(withinExecutionLimit(_token, _value))
require(_receiver != address(0) && _receiver !=
mediatorContractOnOtherSide())
require(withinLimit(_token, _value))
require(_implementation != implementation)
require(Address.isContract(implementation))
require(version > _version)
require(_impl != address(0))
require(msg.sender == upgradeabilityOwner())
require(status)
require(balanceOf[msg.sender] >= wad)
require(balanceOf[src] >= wad)
require(allowance[src][msg.sender] >= wad)
...
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/BasicAMBMediator.sol#L68 contracts/upgradeable_contracts/BasicAMBMediator.sol#L77 contracts/upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitConnector.sol#L35 contracts/upgradeable_contracts/modules/factory/TokenFactoryConnector.sol#L35 contracts/upgradeable_contracts/modules/forwarding_rules/MultiTokenForwardingRulesConnector.sol#L35 contracts/upgradeable_contracts/components/common/TransferNativeFeeStorage.sol#L32
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function _setBridgeContract(address _bridgeContract) internal
...
function _setMediatorContractOnOtherSide(address _mediatorContract)
internal
...
function _setGasLimitManager(address _manager) internal
...
function _setTokenFactory(address _tokenFactory) internal
...
function _setForwardingRulesManager(address _manager) internal
...
function _setTransferFeeAddress(address _feeAddress) internal
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

OCV - Outdated Compiler Version

Criticality	Minor / Informative
Location	contracts/upgradeability/EternalStorage.sol#L1
Status	Unresolved

Description

During our assessment of the smart contract codebase, we identified that the Solidity compiler version being utilized is outdated. The current version employed is 0.4.24, which has been surpassed by numerous updates and improvements in subsequent versions.

- Security Vulnerabilities: Using an outdated compiler version poses potential security risks as newer versions often contain patches for vulnerabilities discovered in earlier releases. By sticking with an outdated version, the smart contract becomes susceptible to known vulnerabilities that have been addressed in newer releases.
- Compatibility Issues: As the Ethereum ecosystem evolves, libraries and tools are updated to align with newer Solidity compiler versions. By staying with an outdated version, compatibility issues may arise with other contracts or external tools, hindering interoperability and potentially leading to unforeseen errors.
- Lack of Language Features: Newer versions of Solidity often introduce enhancements, optimizations, and additional language features that can improve code efficiency, readability, and maintainability. Utilizing an outdated compiler version limits access to these advancements, potentially hindering the development process and reducing the overall quality of the smart contract codebase.

```
pragma solidity 0.4.24;
```

Recommendation

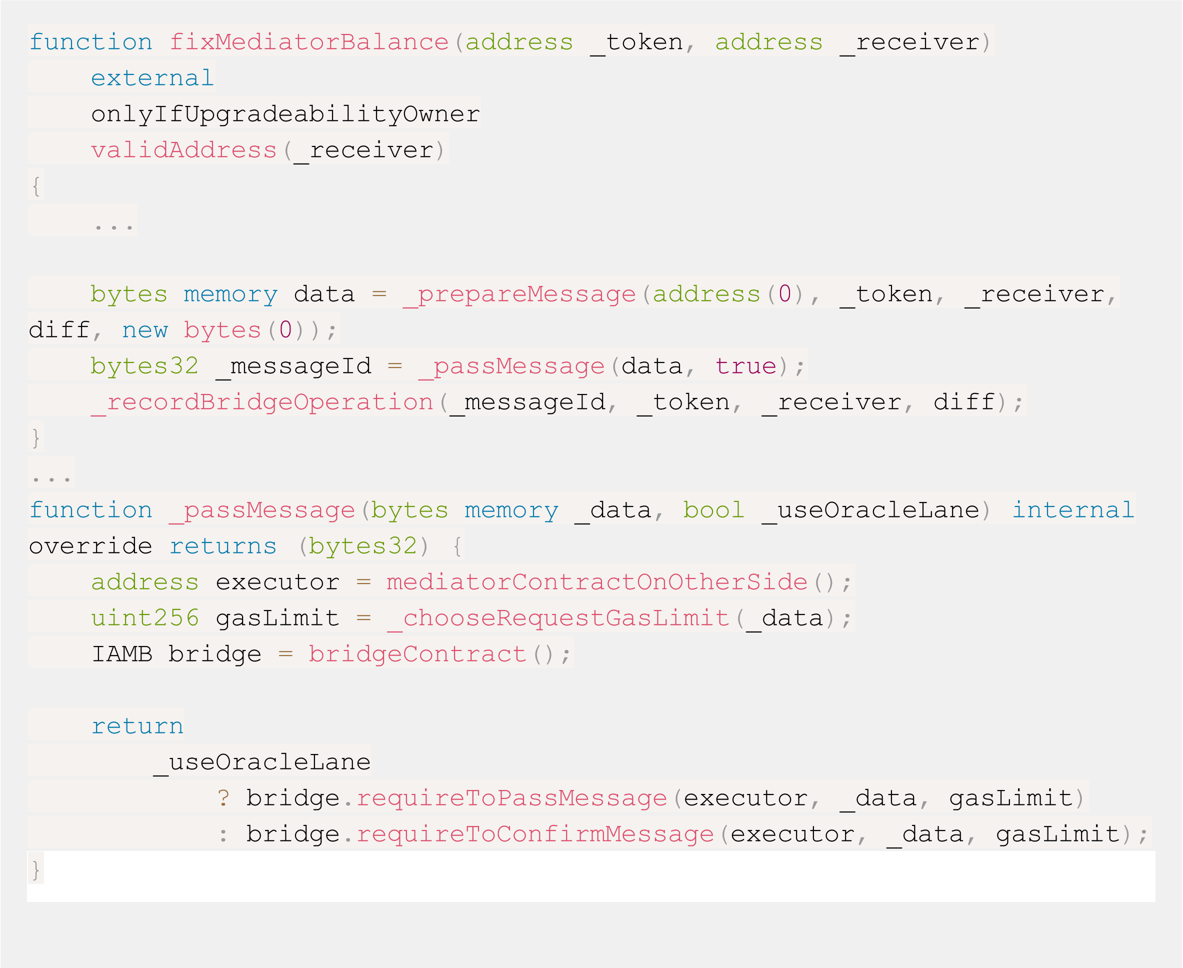
It is strongly advised to upgrade the Solidity compiler version to the latest stable release (at the time of assessment, Solidity 0.8.x). By doing so, the smart contract will benefit from the latest security patches, improved language features, and better compatibility with the broader Ethereum ecosystem. Additionally, ensuring regular updates to the compiler version should be incorporated into the development workflow to mitigate future risks associated with using outdated technology.

PMD - Potential MessageId Duplication

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/BasicOmnibridge.sol#L244 contracts/upgradeable_contracts/HomeOmnibridge.sol#L178
Status	Unresolved

Description

The `_messageId` serves as a unique identifier for messages passed through the bridge. The `_messageId` is retrieved from external functions `requireToPassMessage` and `requireToConfirmMessage`, both defined in the interface IAMB. Depending on the implementation of these external functions, there's a risk of generating duplicate `_messageId`. A duplicate `_messageId` can lead to unintended consequences.



```
function fixMediatorBalance(address _token, address _receiver)
    external
    onlyIfUpgradeabilityOwner
    validAddress(_receiver)
{
    ...

    bytes memory data = _prepareMessage(address(0), _token, _receiver,
diff, new bytes(0));
    bytes32 _messageId = _passMessage(data, true);
    _recordBridgeOperation(_messageId, _token, _receiver, diff);
}
...
function _passMessage(bytes memory _data, bool _useOracleLane) internal
override returns (bytes32) {
    address executor = mediatorContractOnOtherSide();
    uint256 gasLimit = _chooseRequestGasLimit(_data);
    IAMB bridge = bridgeContract();

    return
        _useOracleLane
            ? bridge.requireToPassMessage(executor, _data, gasLimit)
            : bridge.requireToConfirmMessage(executor, _data, gasLimit);
}
```

Recommendation

The team could mitigate this issue by introducing an additional check within the method that returns this unique identifier, in order to ensure the uniqueness of the generated `_messageId` .

ULLCRV - Unused Low Level Call Return Value

Criticality	Minor / Informative
Location	contracts/upgradeable_contracts/BasicOmnibridge.sol#L468
Status	Unresolved

Description

The smart contract `BasicOmnibridge` includes a `callback` function named `_receiverCallback`, which is intended to call the `_recipient`'s `onTokenBridged` callback function. Upon closer inspection it has been observed that it fails to handle or make use of the return value from this call.

```
function _receiverCallback(
    address _recipient,
    address _token,
    uint256 _value,
    bytes memory _data
) internal {
    if (Address.isContract(_recipient)) {
_recipient.call(abi.encodeWithSelector(IERC20Receiver.onTokenBridged.selector, _token, _value, _data));
    }
}
```

Recommendation

To address this finding and ensure the robustness of the `BasicOmnibridge` contract, it is recommended to enhance the `_receiverCallback` function to appropriately handle the return value of the low-level call. This may involve implementing mechanisms for error detection, processing success or failure outcomes, and adjusting the contract's logic accordingly.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/factory/TokenProxy.sol#L19,22 WLILA.sol#L4,5,6
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 internal totalSupply
bool internal mintingFinished
string public name      = "Wrapped LILA"
string public symbol    = "WLILA"
uint8  public decimals = 18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitManager.sol#L75,86,99,100,101,119,129,138,170,192,210 upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitConnector.sol#L19 upgradeable_contracts/modules/factory/TokenProxy.sol#L27 upgradeable_contracts/modules/factory/TokenFactory.sol#L45,59,60,61,62,63,64 upgradeable_contracts/HomeOmnibridge.sol#L36,37,38,39,40,41,42,43,44,45,74,75,76,77,94,103,145,146,147,148,149 upgradeable_contracts/components/native/MediatorBalanceStorage.sol#L16 upgradeable_contracts/components/common/BridgeOperationsStorage.sol#L15,23,32,40,49,57 upgradeable_contracts/components/bridged/BridgedTokensRegistry.sol#L17,26 upgradeable_contracts/BasicOmnibridge.sol#L43,44,68,69,70,71,72,73,93,94,95,96,97,98,99,116,117,118,137,138,139,140,159,160,161,178,179,180,181,195,206,207,208,219,244,271,285,286,287 upgradeable_contracts/BasicAMBMediator.sol#L36,44 libraries/TokenReader.sol#L6,8,10,26,44,61 libraries/AddressHelper.sol#L15
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.

6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
bool _lock
address _newOwner
uint256 _gasLimit
bytes4 _selector
address _token
bytes memory _data
uint256[] calldata _gasLimits
address _manager
address _sender
address _receiver
bool _enable
bytes32 internal DOMAIN_SEPARATOR
address _tokenFactory
address _tokenImage
uint256 _value
address payable _receiver
address _to
IBurnableMintableERC677Token _token
function NAME() external view;
function SYMBOL() external view;
function DECIMALS() external view;
address _nativeToken
address _bridgedToken
bytes memory _bytes
address _bridgeContract
address _mediatorContract

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	upgradeability/EternalStorage.sol#L9,11,13
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping(bytes32 => string) internal stringStorage
mapping(bytes32 => bytes) internal bytesStorage
mapping(bytes32 => int256) internal intStorage
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/OwnableModule.sol#L33
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = _newOwner
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitManager.sol#L76 upgradeable_contracts/modules/components/common/TransferNativeFeeStorage.sol#L19
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
defaultGasLimit = _gasLimit
...
function _setTransferFeeAmount(uint256 _feeAmount) internal
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	upgradeable_contracts/BasicAMBMediator.sol#L93 @openzeppelin/contracts/utils/Address.sol#L53,79,104,114 @openzeppelin/contracts/token/ERC20/SafeERC20.sol#L37,48,53
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function maxGasPerTx() internal view returns (uint256) {
        return bridgeContract().maxGasPerTx();
    }

function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");
...
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call
failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/OwnableModule.sol#L17,33 upgradeable_contracts/modules/factory/TokenProxy.sol#L61,63 upgradeable_contracts/modules/factory/TokenFactory.sol#L19
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
owner = _owner
owner = _newOwner
bridgeFeeAddr = _feeAddr
tokenImage = _tokenImage
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	upgradeable_contracts/modules/gas_limit/SelectorTokenGasLimitManager.sol#L141 upgradeable_contracts/modules/factory/TokenProxy.sol#L53,82 upgradeable_contracts/components/common/TokensRelayer.sol#L35 upgradeable_contracts/BasicOmnibridge.sol#L50,487 upgradeability/Proxy.sol#L22 ForeignOmnibridge.sol#L281,380,750,879,890,899,1188,1843,1968,1997,2151,2588
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            // Even though this is not the same as
boolStorage[keccak256(abi.encodePacked("lock"))],
            // since solidity mapping introduces another level of
addressing, such slot change is safe
            // for temporary variables which are cleared at the end of
the call execution.
            res :=
sload(0x6168652c307c1e813ca11cfb3a601f1cf3b22452021a5052d8b05f1f1f8a3e92
) // keccak256(abi.encodePacked("lock"))
        }

assembly {
            // Even though this is not the same as
boolStorage[keccak256(abi.encodePacked("lock"))],
            // since solidity mapping introduces another level of
addressing, such slot change is safe
            // for temporary variables which are cleared at the end of
the call execution.

sstore(0x6168652c307c1e813ca11cfb3a601f1cf3b22452021a5052d8b05f1f1f8a3e9
2, _lock) // keccak256(abi.encodePacked("lock"))
        }
assembly { codehash := extcodehash(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
assembly {
            // EIP 1967
            //
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1)

sstore(0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bb
c, _tokenImage)
        }

assembly {
            impl :=
sload(0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc
)
        }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	@openzeppelin/contracts/utils/Address.sol#L3 @openzeppelin/contracts/token/ERC20/SafeERC20.sol#L3 @openzeppelin/contracts/token/ERC20/IERC20.sol#L3 @openzeppelin/contracts/math/SafeMath.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.7.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
EternalStorage	Implementation			
TokenProxy	Implementation	Proxy		
		Public	✓	-
	implementation	Public		-
	getTokenProxyInterfacesVersion	External		-
BasicOmnibridge	Implementation	Initializable, Upgradeable, Claimable, OmnibridgeInfo, TokensRelayer, FailedMessagesProcessor, BridgedTokensRegistry, NativeTokensRegistry, MediatorBalanceStorage, TokenFactoryConnector, TokensBridgeLimits		
		Public	✓	-
	deployAndHandleBridgedTokens	External	✓	onlyMediator
	deployAndHandleBridgedTokensAndCall	External	✓	onlyMediator
	handleBridgedTokens	External	✓	onlyMediator

	handleBridgedTokensAndCall	External	✓	onlyMediator
	handleNativeTokens	External	✓	onlyMediator
	handleNativeTokensAndCall	External	✓	onlyMediator
	isRegisteredAsNativeToken	Public		-
	executeActionOnFixedTokens	Internal	✓	
	setCustomTokenAddressPair	External	✓	onlyOwner
	fixMediatorBalance	External	✓	onlyIfUpgradeabilityOwner validAddress
	claimTokens	External	✓	onlyIfUpgradeabilityOwner
	claimTokensFromTokenContract	External	✓	onlyIfUpgradeabilityOwner
	_recordBridgeOperation	Internal	✓	
	_prepareMessage	Internal	✓	
	_getMinterFor	Internal		
	_releaseTokens	Internal	✓	
	_getBridgedTokenOrDeploy	Internal	✓	
	_receiverCallback	Internal	✓	
	_transformName	Internal		
	_unaccountedBalance	Internal		
	_handleTokens	Internal	✓	
ForeignOmnibridge	Implementation	BasicOmnibridge, GasLimitManager, InterestConnector		
		Public	✓	BasicOmnibridge

	initialize	External	✓	onlyRelevantSender
	upgradeToReverseMode	External	✓	-
	_handleTokens	Internal	✓	
	bridgeSpecificActionsOnTokenTransfer	Internal	✓	
	_releaseTokens	Internal	✓	
	_passMessage	Internal	✓	
	_unaccountedBalance	Internal		

Inheritance Graph

See the detailed images in the github repository.

Flow Graph

See the detailed images in the github repository.

Summary

This audit reviews the smart contracts of Liquid Layer’s bridge. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io