Audit Report

Router Bridge

v1.0

November 27, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Kailaasa Infotech Pte Ltd to perform a security audit of several Router bridge components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Repository	Commit hash
https://github.com/router-protocol/router-bridge	a4599217c6f3a25edf82dce27c68aa014f955e63
https://github.com/router-protocol/path-finder-api	cc044d46b4fd347d0cfb18f3e1452a69b2be2baf
https://github.com/router-protocol/router-bridge-contracts-v2	465b3248610f79e653360feaed970ecb08a9a43d
https://github.com/router-protocol/router-crosstalk	090b272d6f94d06681b692aac90d52dec3b9f30e
https://github.com/router-protocol/router-vault	4efdca1f3b2255391e702ac438542ec581f13034

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base’s intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Router Protocol is an extensible multi-directional bridge that connects current and emerging layer 1 and layer 2 blockchains to allow contract-level data flow across them. These can be asset transfers as well as sending arbitrary messages across chains.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase’s code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium-High**	-
Code readability and clarity	**Low-Medium**	-
Level of documentation	**Medium**	-
Test coverage	**Low-Medium**	-

Summary of Findings

No	Description	Severity	Status
1	`RouterCrossTalk`: DoS by frontrunning with approving fees to a value of `0`	Critical	Resolved
2	`RouterERC20Upgradable.sol`: Paused contract does not pause minting/burning	Critical	Resolved
3	Static `feeFactor` allows economic attacks in certain market conditions, and leads to users overpaying in others	Critical	Acknowledged
4	Missing storage gaps for upgradeable contracts might lead to storage slot collisions	Major	Resolved
5	Return value of transfer function not checked	Major	Acknowledged
6	Failure to revoke permission of the previous owner	Major	Resolved
7	Incorrect usage of initializer functions	Major	Resolved
8	Missing external initializer of the RouterERC20Upgradable contract	Major	Acknowledged
9	`universalApprove` does not allow reduction of allowances	Major	Resolved
10	`FetchLiquidity.sol`: Hardcoded invalid exchange contract addresses	Minor	Acknowledged
11	`HandlerReserveUpgradeable.sol`: Deployed liquidity pool contract is not upgradeable	Minor	Resolved
12	`RouterCrossTalk.sol`: Add source chain id to hash	Minor	Resolved
13	`ETHHandler.sol`: Anyone can withdraw ETH funds	Minor	Acknowledged
14	`ERC20HandlerUpgradeable.sol`: Unrecoverable ETH due to unnecessary receive function	Minor	Resolved
15	`TimelockVaults.sol`: Unused pause functionality	Minor	Acknowledged

16	`TimelockVaults.sol`: Owner can grief stakers with a high penalty fee	Minor	Acknowledged
17	`VoterUpgradeable.sol`: `vote` function should check if the proposal has started yet	Minor	Acknowledged
18	Usage of floating pragma	Minor	Resolved
19	Lack of event emission for important state changes impacts off-chain monitoring tools	Minor	Acknowledged
20	NFT logic negatively impacts readability and maintainability	Minor	Acknowledged
21	Unexpected return value of `universalTransfer`	Minor	Acknowledged
22	Usage of ERC-20 functions instead of `SafeERC20` functions	Minor	Resolved
23	Use of constructor in an upgradable contract	Minor	Acknowledged
24	Wrong event is emitted upon expiry changes	Minor	Resolved
25	Centralization risk in emergency role	Minor	Acknowledged
26	Implementation contracts of proxies are not initialized	Minor	Resolved
27	No boundary checks on the expiry of the proposals	Minor	Acknowledged
28	Fixed gas during cross talk linking may be depleted in future Ethereum versions	Minor	Resolved
29	No threshold check on quorum value leads to execution of proposals without voting	Minor	Acknowledged
30	`FetchLiquidity.sol`: DEX return value `_blockTimestampLast` is ignored	Informational	Acknowledged
31	Usage of `transfer` function may cause problems in the future	Informational	Acknowledged
32	Usage of magic numbers may cause problems in the future	Informational	Acknowledged
33	`RouterCrossTalk.sol`: `Unlink` can be called repeatedly which will emit multiple unlink events that may cause issues for off-chain components	Informational	Resolved
34	`RouterCrossTalk.sol`: Misleading naming for	Informational	Resolved

	modifier `isLinkSet` and `isLinkUnSet`		
35	Use of inconsistent code naming conventions	Informational	Acknowledged
36	Incorrect error statement	Informational	Resolved
37	Typographical errors	Informational	Resolved
38	Unconventional naming of functions and events	Informational	Acknowledged
39	Code repetition decreases maintainability	Informational	Acknowledged
40	Unnecessary usage of assembly decreases readability	Informational	Acknowledged
41	Complicated logic for stake function in `TimelockVaults.sol` impacts readability	Informational	Acknowledged
42	Miscellaneous notes	Informational	Acknowledged
43	Gas Optimizations	Informational	Acknowledged

Detailed Findings

1. `RouterCrossTalk`: DoS by frontrunning with approving fees to a value of 0

Severity: Critical

The `RouterCrossTalk` (and `RouterCrossTalkUpgradeable`) contracts implement the function `approveFees` to approve the generic handler to spend fee tokens. This function is intended to only be called by the implementation contract inheriting from `RouterCrossTalk`.

However, the function visibility is external, allowing anyone to approve fees arbitrarily. An attacker could front-run transactions with routerSend calls and call approveFees with a value of 0. Effectively preventing the generic handler from spending fee tokens and leading to a DoS.

Recommendation

We recommend changing the function visibility to internal to prevent anyone external from approving fees.

Status: Resolved

2. `RouterERC20Upgradable.sol`: Paused contract does not pause minting/burning

Severity: Critical

The `router-protocol-router-bridge-contracts-v2/contracts/RouterERC20Upgradable.sol` and `router-protocol-router-bridge-contracts-v2/contracts/RouterERC20UpgradableOld.sol` contracts implement a pause mechanism. Currently, only token transfers are disabled while the contract is paused. However, minting and burning should also be paused to allow mitigation in case of an exploit.

Recommendation

We recommend disabling minting and burning while the contract is paused by using the modifier `whenNotPaused`.

Status: Resolved

3. Static `feeFactor` allows economic attacks in certain market conditions, and leads to users overpaying in others

Severity: Critical

In `router-bridge-contracts-v2/contracts/GenericHandlerUpgradeable.sol:349`, the fee is calculated on the basis of the `feeFactor` that admin sets and the gas value that user provides at the time of deposit. However, the `feeFactor` is calculated in terms of the `feeTokenAddress` so it can match the value of the native token of the destination chain. Since gas prices as well as native token prices are fluctuating over time, but the `feeFactor` is static, a misprising of transactions can easily happen. An attacker can economically attack the Router bridge when the target gas prices native token prices are high and, effectively paying a smaller fee. This would make Router run on a loss. In other market situations, users would effectively overpay for their transactions. In practice, the `feeToken` and native token values would change every second, so it is not possible to appropriately set the `feeFactor` to prevent these issues.

Recommendation

We recommend using an oracle that sets the `feeFactor`, rather then relying on an admin to update the value.

Status: Acknowledged

The team states that they will use a new fee model and an off-chain service to set fees accordingly.

4. Missing storage gaps for upgradeable contracts might lead to storage slot collisions

Severity: Major

For upgradeable contracts, there must be storage gaps to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments" (quote OpenZeppelin). Otherwise, it may be very difficult to write new implementation code.

Without storage gaps, the variables in a child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences to the child contracts, potentially causing loss of user funds or causing the contract to malfunction completely.

Refer to the bottom part of this article: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

This issue exists in both `router-protocol-router-bridge-contracts-v2/contracts/ERC20SafeUpgradeable.sol` and `router-protocol-router-bridge-contracts-v2/contracts/handlers/HandlerHelpersUpgradeable.sol`.

Recommendation

We recommend adding appropriate storage gaps at the end of upgradeable contracts.

Status: Resolved

5. Return value of transfer function not checked

Severity: Major

The return value of the solidity `transfer` function is not checked throughout the contracts in several places.

For instance in the `universalTransfer` and `universalTransferFrom` in `router-protocol-path-finder-api/Contracts/UniversalERC20.sol`.

Recommendation

Always check the return value of any contract call inside the contracts.

Status: Acknowledged

6. Failure to revoke permission of the previous owner

Severity: Major

The `_setLiquidityPoolOwner` function of `router-protocol-router-bridge-contracts-v2/contracts/handlers/HandlerReserveUpgradeable.sol` does not revoke the permissions of the previous owner upon setting a new pool owner.

Recommendation

Consider using `revokeRole` for the previous owner.

Status: Resolved

The team has added revoking access logic.

7. Incorrect usage of initializer functions

Severity: Major

To avoid the double initialization problem with upgradeable contracts inheriting from multiple contracts, `__{ContractName}_init_unchained` functions are used to initialize the contract without calls to parent initializers.

The following contracts are upgradeable and are possibly inherited by other contracts, hence, the `__{ContractName}_init_unchained` initialize functions have to work properly:

- `router-protocol-router-bridge-contracts-v2/contracts/FeeManagerGenericUpgradeable.sol` - missing `address handlerAddress` initialization and admin role setup
- `router-protocol-router-bridge-contracts-v2/contracts/FeeManagerUpgradeable.sol` - missing `address handlerAddress` initialization and admin role setup
- `router-protocol-router-crosstalk/contracts/RouterCrossTalkUpgradeable.sol` - Missing `address _handler` initialization in unchained init function

Recommendation

We recommend adding the initialization logic to the `__{ContractName}_init_unchained` functions and call this function from within the `__{ContractName}_init` function.

Status: Resolved

8. Missing external initializer of the `RouterERC20Upgradable` contract

Severity: Major

`router-bridge-contracts-v2/contracts/RouterERC20Upgradable.sol` doesn't have the `initialize` function. Because of this, the proxy would not initialize with the right storage values.

Currently, constructor is performing the initialization of the code which only initializes the implementation contract storage instead of the proxy storage.

Recommendation

We recommend adding the `initialize` function to `RouterERC20Upgradable` if the contract is supposed to support upgrades. Otherwise we recommend removing support for upgrades and removing the `Upgradable` suffix from the contract name.

Status: Acknowledged

The team has explained that this smart contract is no longer in use.

9. `universalApprove` does not allow reduction of allowances

Severity: Major

The `universalApprove` function in `router-protocol-path-finder-api/UniversalERC20.sol` does not consider the case in which `allowance > amount`. Therefore, the function doesn't allow decreasing the allowance.

Recommendation

Update the function logic to allow the decrease of allowance.

Status: Resolved

The team added `safeApprove(0)` to reset the current allowance before setting the actual allowance amount.

10. `FetchLiquidity.sol`: Hardcoded invalid exchange contract addresses

Severity: Minor

The `router-protocol-path-finder-api/contracts/FetchLiquidity.sol` contract has hardcoded exchange contract addresses that are currently either invalid or set to the testnet contract addresses.

For instance, `SushiV2Factory` on Ethereum Mainnet is `0xC0AEe478e3658e2610c5F7A4A2E1777cE9e4f2Ac` and `0xc35DADB65012eC5796536bD9864eD8773aBc74C4` on testnets.

Recommendation

We recommend setting the exchange contract addresses on deployment via the `FetchLiquidity` constructor.

Status: Acknowledged

The team will use the appropriate exchange contract addresses for each deployed chain.

11. `HandlerReserveUpgradeable.sol`: Deployed liquidity pool contract is not upgradeable

Severity: Minor

Liquidity pools for reserve tokens (for instance `RBNB`, `RMATIC`, or `RUSDC`) are manually deployed as transparent proxies. Additionally, it is possible to deploy new liquidity pools via `router-protocol-router-bridge-contracts-v2/contracts/handlers/HandlerReserveUpgradeable._setLiquidityPool`. Contrary to the manually deployed, upgradeable contracts, those newly deployed liquidity pool contracts are not upgradeable.

Recommendation

We recommend implementing a factory contract that deploys a liquidity pool contract `RouterERC20Upgradable` via a transparent proxy.

Status: Resolved

The team removed the functionality to deploy new liquidity pools via the `_setLiquidityPool` function.

12. `RouterCrossTalk.sol`: Add source chain id to hash

Severity: Minor

A hash of all data sent or received by a `router-protocol-router-crosstalk/contracts/RouterCrossTalk.sol` contract is generated via the `_hash` function. However, if the same data is sent from multiple different source chains, the hash will be exactly the same. A contract inheriting from `RouterCrossTalk` which stores all hashes in a mapping would therefore encounter issues due to colliding hashes.

Recommendation

We recommend adapting the `_hash` function to incorporate the source chain id.

Status: Resolved

13. `ETHHandler.sol`: Anyone can withdraw ETH funds

Severity: Minor

The `router-protocol-router-bridge-contracts-v2/contracts/handlers/ETHHandler.sol` contract is used as a helper contract to unwrap wrapped native tokens (`WETH`, `WFTM`, `WAVAX`, …). After unwrapping, native token funds are usually immediately withdrawn via the `withdraw` function by the caller contract.

However, as the `ETHHandler` contract implements a `receive` function, native tokens can be transferred to the contract aside from unwrapping. Anyone can watch the native token balance of the contract and immediately withdraw accidentally sent funds from someone else.

Recommendation

We recommend adding access control checks to only allow the `router-protocol-router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol` and `router-protocol-router-bridge-contracts-v2/contracts/handlers/HandlerReserveUpgradeable.sol` contracts to withdraw ETH funds.

Status: Acknowledged

The team decided to not implemented our recommendation, as deposits and withdrawals take place in the same transaction.

14. `ERC20HandlerUpgradeable.sol`: Unrecoverable ETH due to unnecessary receive function

Severity: Minor

The `router-protocol-router-bridge-contracts-v2/contracts/handlers/ERC20HandlerUpgradeable.sol` contract does not expect direct ETH transfers. However, there is a `receive` function to receive ETH. Anyone accidentally sending ETH to this contract will lose their funds.

Recommendation

We recommend removing the `receive` function.

Status: Resolved

15. `TimelockVaults.sol`: Unused pause functionality

Severity: Minor

The `router-protocol-router-vault/contracts/TimelockVaults.sol` contract inherits from `PausableUpgradeable` to add pause/unpause functionality. However, the contract does not currently take advantage of pausing/unpausing.

Recommendation

We recommend adding the modifier `whenNotPaused` to the `stake` function as well as adding `pause` and `unpause` owner-callable functions to the contract.

Status: Acknowledged

The team states that the `TimelockVaults.sol` contract is not used.

16. `TimelockVaults.sol`: Owner can grief stakers with a high penalty fee

Severity: Minor

Withdrawing staked tokens from the `router-protocol-router-vault/contracts/TimelockVaults.sol` contract before the tenure ends is subject to penalty fees. The penalty fee is calculated based on the storage variable `penaltyFactor` and can be set by the owner via the function `setPenaltyFactor`. However, an owner can prevent users from using the emergency withdrawal by setting the penalty factor to a maximum value of `99`, effectively introducing a penalty of 99%.

Recommendation

We recommend adding a reasonable max penalty fee boundary for `penaltyFactor`.

Status: Acknowledged

The team states that the `TimelockVaults.sol` contract is not used.

17. `VoterUpgradeable.sol`: `vote` function should check if the proposal has started yet

Severity: Minor

Currently the voting start for a proposal is hardcoded to the block number at the time of proposal creation in `router-protocol-router-bridge-contracts-v2/contracts/VoterUpgradeable.sol`. But as this contract is upgradable, a future implementation could allow setting the `startBlock` block number value for a new proposal.

Recommendation

We recommend implementing a modifier `isStarted` for the `vote` function to check whether voting for a given proposal has already started.

Status: Acknowledged

18. Usage of floating pragma

Severity: Minor

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. See https://swcregistry.io/docs/SWC-103 for reference.

Affected contracts:

- `router-protocol-router-vault/contracts/TimelockVaults.sol`
- `router-protocol-router-bridge-contracts-v2/contracts/ERC20SafeUpgradeable.sol`

Recommendation

We recommend locking the pragma version in all contracts and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements may be allowed to float when a contract is intended for consumption by other developers. Otherwise, the developer would need to manually update the pragma in order to compile it locally.

Status: Resolved

The team has updated to contracts to use the appropriate compiler version.

19. Lack of event emission for important state changes impacts off-chain monitoring tools

Severity: Minor

When changing state variables events are not emitted. Emitting events allows monitoring activities with off-chain monitoring tools. The following functions do not currently emit events:

In `router-protocol-router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol`:

- `adminChangeTrustedForwarder(address newTF)`

In `router-protocol-router-bridge-contracts-v2/contracts/FeeManagerUpgradeable.sol`:

- `setHandler(address _handler)`

In `router-protocol-router-bridge-contracts-v2/contracts/handlers/HandlerHelpersUpgradeable.sol`:

- `setLiquidityPoolOwner(address newOwner,address tokenAddress,address lpAddress)`
- `_setResource(bytes32 resourceID, address contractAddress)`
- `_setBurnable(address contractAddress, bool status)`
- `_setOneSplitAddress(address contractAddress)`

In `router-protocol-router-vault/contracts/TimelockVaults.sol`:

- `setPenaltyFactor(uint256 factor)`
- `setTimeToWeight(uint256 lockperiod, uint256 weight)`

- `setMultiTimeToWeight(uint256[] memory lockperiod, uint256[] memory weight)`
- `setMaxUserStakeLimit(uint256 _maxUserStakeLimit)`
- `setMaxTotalStakedLimit(uint256 _maxTotalStakedLimit)`
- `withdrawPenalty()`

Recommendation

We recommend emitting events for important state variable changes.

Status: Acknowledged

20. NFT logic negatively impacts readability and maintainability

Severity: Minor

Non-fungible (ERC-721) token transfers are currently not supported by the protocol, but the codebase contains unfinished code relating to NFTs. Most of the code related to the transfer of NFTs is already commented out. However, in `router-protocol-router-bridge/chains/**/writer.go`, there is still logic to create proposals for NFT transfers.

Unfinished functionality as well as commented-out code has a negative impact on the readability and maintainability of the codebase.

Recommendation

We recommend removing all commented code related to NFTs as well as the logic to handle ERC-721 transfers.

Status: Acknowledged

21. Unexpected return value of `universalTransfer`

Severity: Minor

The `universalTransfer` function in `router-protocol-path-finder-api/contracts/UniversalERC20.sol` returns a `bool` value, but the function does not check for the return value of the `transfer` function. Instead, it uses `safeTransfer` for transferring ERC-20 tokens.

Recommendation

In order to comply with `SafeERC20` standards, we recommend removing the return value of the function and `revert` in the case of a `false` return value of the `transfer` function.

Status: Acknowledged

22. Usage of ERC-20 functions instead of `SafeERC20` functions

Severity: Minor

The `universalTransferFrom` and `universalApprove` functions in `router-protocol-path-finder-api/contracts/UniversalERC20.sol` use ERC-20 `transferFrom` and `approve` functions instead of their `SafeERC20` `safeTransferFrom` and `safeApprove` counterparts.

Recommendation

Since this contract takes advantage of `SafeERC20`, we recommend using `SafeERC20` functions instead of ERC-20 functions.

Status: Resolved

The team has updated the contract to use the recommended functions.

23. Use of constructor in an upgradable contract

Severity: Minor

The `router-protocol-router-bridge-contracts-v2/contracts/RouterERC20Upgradable.sol` contract has a constructor in which the initializer has been called. However, an upgradable contract should not have a constructor.

Recommendation

Remove the constructor.

Status: Acknowledged

The team states that this smart contract is no longer in use.

24. Wrong event is emitted upon expiry changes

Severity: Minor

The `adminChangeExpiry` function of `BridgeUpgradeable.sol` in `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:379` emits the `expiryChanged` event with the `_quorum` instead of the new expiry value that gets set in the function call. Because of this, off-chain components may process incorrect data.

Recommendation

Use `expiryChanged(_expiry)` instead.

Status: Resolved

25. Centralization risk in emergency role

Severity: Minor

In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:495`, the `adminWithdraw` function provides the power to the emergency role holder to move all the funds from the reserve at once. This is a centralization risk. A compromised key could lead to user funds being lost.

Recommendation

We recommend removing the function or breaking it up into the following two-step process: First lock the bridge contracts and then second, after a cooldown period perform the second step of fund transferral.

Status: Acknowledged

The team states they accept the risk associated with this issue.

26. Implementation contracts of proxies are not initialized

Severity: Minor

Throughout the codebase, logic or implementation contracts are not initialized properly which means anyone else can do the initialization of the contracts and impose different states and even assume ownership of these implementation contracts as well. Currently there are no functions that directly get affected because of this but in future if the contracts get upgraded and introduce gateways this lack of initialization may be exploited.

We consider this a minor issue since it cannot be exploited in the current contracts. The severity of this issue would be critical if there was any function present in the system that can get affected by this lack of initializtion.

Recommendation

We recommend adding the `_disableInitializers` function provided by function provided by OpenZeppelin in the constructor of every implementation contract.

Status: Resolved

27. No boundary checks on the expiry of the proposals

Severity: Minor

In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:378`, a new expiry can be set by the admin, but there is no minimum/maximum value validation on that new expiry. This makes it possible to set the expiry to 0 or an arbitrarily high number, which may lead to unusability.

We consider this issue to be minor since it can only be caused by the admin.

Recommendation

We recommend adding minimum and maximum bounds on the new expiry value.

Status: Acknowledged

28. Fixed gas during cross talk linking may be depleted in future Ethereum versions

Severity: Minor

In `router-bridge-contracts-v2/contracts/GenericHandlerUpgradeable.sol:112`, `crossTalk` linking consumes a fixed gas amount that is hardcoded within the contract. If future Ethereum upgrade introduces changes to the gas price of opcodes, then this call may run out of gas.

Recommendation

We recommend making the gas limit configurable.

Status: Resolved

29. No threshold check on quorum value leads to execution of proposals without voting

Severity: Minor

It `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:367`, the quorum can be changed by the admin, without any minimum quorum value check. This poses a centralization issue, since an admin can set the quorum value to zero, allowing a single validator to control the bridge.

Recommendation

We recommend hard-coding and enforcing a minimum quorum value, e.g. ⅔ of the number of validators whitelisted in the bridge.

Status: Acknowledged

30. `FetchLiquidity.sol`: DEX return value `_blockTimestampLast` is ignored

Severity: Informational

The `router-protocol-path-finder-api/contracts/FetchLiquidity.sol` contract used within the Pathfinder API fetches the current token pair reserve info from multiple exchanges. The DEX response includes the variable `_blockTimestampLast`, which is the last block during which an interaction with the pair on the DEX occurred. That value is currently ignored and not returned within the Pathfinder API response.

Recommendation

We recommend including the DEX return value `_blockTimestampLast` in the `dexResponse` struct.

Status: Acknowledged

31. Usage of `transfer` function may cause problems in the future

Severity: Informational

The `transfer` function has been used in the code repeatedly. However, considering that this function uses a fixed gas amount, its usage is not recommended, since the gas needs might change in the future, which might lead to errors.

Instances of functions that use the `transfer` function:

a) The `universalTransfer` function in the `UniversalERC20` library in `router-protocol-path-finder-api`
b) The universalTransferFrom function in the `UniversalERC20` library in `router-protocol-path-finder-api`
c) The universalTransferFromSenderToThis function in the `UniversalERC20` library at `router-protocol-path-finder-api`

Recommendation

We recommend using the `call` function instead.

Status: Acknowledged

32. Usage of magic numbers may cause problems in the future

Severity: Informational

Magic numbers, instances of integer values, are used across the codebase. Such numbers without context can be hard to track, which may lead to errors if the values are changed to inconsistent states in future upgrades.

Recommendation

The contract `router-protocol-router-bridge-contracts-v2/contracts/VoterUpgradeable.sol` defines valid voting options as `1` for `YES` and `2` for `NO`. Instead of using the integer values directly, we recommend adding an enum and replacing all occurrences with the equivalent enum value.

Additionally, in `router-protocol-router-bridge-contracts-v2/contracts/VoterUpgradeable.sol` and `router-protocol-router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol`, a valid relayer voter balance is defined as `1 ETH`. We recommend using a shared `constant` variable instead.

Status: Acknowledged

33. `RouterCrossTalk.sol`: `Unlink` can be called repeatedly which will emit multiple unlink events that may cause issues for off-chain components

Severity: Informational

Contracts are linked to a given `chainId` via the `Link` and `Unlink` functions in `router-protocol-router-crosstalk/contracts/RouterCrossTalk.sol`. Currently, the `Unlink` function can be repeatedly called even though the address is already unlinked. This will emit an event for each call, which may lead to issues with off-chain event monitoring tools.

Recommendation

We recommend adding the modifier `isLinkUnSet` to the `Unlink` function.

Status: Resolved

34. `RouterCrossTalk.sol`: Misleading naming for modifier `isLinkSet` and `isLinkUnSet`

Severity: Informational

Contrary to the modifier name `isLinkSet`, the modifier actually checks if a given `_chainID` has no contract address defined. Additionally, the modifier `isLinkUnSet` checks if a given `_chainID` has a contract address defined.

The naming of both `modifiers` can lead to potential confusion and miss-use leading to issues in the future.

Recommendation

We recommend swapping the names for both mentioned modifiers: Rename the modifier `isLinkSet` to `isLinkUnSet` and `isLinkUnSet` to `isLinkSet` in the `router-protocol-router-crosstalk/contracts/RouterCrossTalk.sol` contract.

Status: Resolved

35. Use of inconsistent code naming conventions

Severity: Informational

Across all contracts, different naming conventions are used for variables and functions. For instance, some functions use Pascal case (e.g. `GetProposalHash` in `router-protocol-router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol`) instead of camel case (e.g. `getProposalHash`).

Recommendation

We recommend maintaining a consistent code style throughout the codebase by following the Solidity naming convention, which can be found at https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions.

Status: Acknowledged

36. Incorrect error statement

Severity: Informational

In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:210`, a check is performed whether a proposal with the given `proposalHash` already exists. If not, an error will be thrown. The current error message reads "`BirdgeUpgradeable: Proposals Already Exists`", which is incorrect.

Recommendation

We recommend replacing the current error statement "`BridgeUpgradeable: Proposal does not exists`".

Status: Resolved

37. Typographical errors

Severity: Informational

In `router-crosstalk/contracts/RouterCrossTalk.sol:76` and `RouterCrossTalkUpgradeable.sol:82`, the function name `fetchFeetToken` contains a typographical error, it should be named `fetchFeeToken`.

Recommendation

We recommend renaming the function to `fetchFeeToken`.

Status: Resolved

38. Unconventional naming of functions and events

Severity: Informational

Throughout the codebase, unconventional naming is used for functions and events.

Below are some examples:

- In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:31-59`, function are named using `snake_case` while it is convention to use `camelCase`.
- In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:63`, function are named using `PascalCase` while it is convention to use `camelCase`.
- In `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:113`, Struct are name as `mixedCase` while recommended to use `CapWords`.
- `router-bridge-contracts-v2/contracts/BridgeUpgradeable.sol:126` Event are name as `mixedCase` while recommended to use `CapWords`.

Recommendation

We recommend following the consistent naming styling recommended here in the whole codebase so it is easier to read and easy to differentiate as well.

Status: Acknowledged

39. Code repetition decreases maintainability

Severity: Informational

Code repetition exists in multiple places in the codebase, which decreases the readability of the code and is prone to errors when functionality is changed/extended inconsistently in future upgrades. Instances of code repetition:

- The logic for checking the private key pairs' passwords in `router-protocol-router-bridge/cmd/router-bridge/main.go`
- The logic for calculating the `proposalHash` has been used in the `BridgeUpgradeable.sol` contract repeatedly despite the fact that the `GetProposalHash` function exists and could be used

Recommendation

We recommend extracting the repeated code into functions.

Status: Acknowledged

40. Unnecessary usage of assembly decreases readability

Severity: Informational

The `isContract` function of `router-protocol-router-bridge-contracts/contracts/utils/AddressUpgradeable.sol` uses assembly code which is unnecessary and negatively impacts the readability of the contract.

Recommendation

Use return `account.code.length > 0;` instead of assembly code.

Status: Acknowledged

41. Complicated logic for stake function in `TimelockVaults.sol` impacts readability

Severity: Informational

The `_updateReward` function is performing two different functions, which goes against best practices and reduces code readability and maintainability. Specifically:

a) Using `userVaults[msg.sender].length` as a function parameter for calling `_updateReward` in the `stake` function of `router-protocol-router-vault/contracts/TimelockVaults.sol` can cause complications. Although out-of-bound array index access has been avoided in lines `114` and `115` by returning `0`, this pattern goes against best practices.

b) The `_updateReward` function in `router-protocol-router-vault/contracts/TimelockVaults.sol` has two purposes: updating the `_userRewardPerTokenPaid` value and calculating the `_earned` value. In this case, the value of `_earned` will be `0` for the new block. However, at first, it looks like the contract adds the earn value for all the other previous blocks in the new one as well.

Recommendation

We recommend breaking the `_updateReward` function down into two separate functions `_updateReward` and `_updateRewardPerToken`.

Status: Acknowledged

The team states that this smart contract is no longer in use.

42. Miscellaneous notes

Severity: Informational

Consider fixing the following items:

a) `BridgeUpgradeable.adminSetTokenDecimals`: Incorrect NatSpec function comment - should be `"Sets decimals for token on target chain"`

b) Lack of revert messages:

  i) `BridgeUpgradeable.sol:596`: `require(IWETH(weth).transfer(msg.sender, msg.value));`

  ii) `BridgeUpgradeable.sol:606`: `require(_genericWhitelist[_resourceIDToHandlerAddress[_resourceID]] == true);`

  iii) `BridgeUpgradeable.sol:607`: `require(_resourceIDToHandlerAddress[_resourceID] == msg.sender);`

  iv) `BridgeUpgradeable.sol:722`: `assert(IWETH(WETH).transfer(handler, amount));`

  v) `VoterUpgradeable.sol:48`: `require(msg.sender == bridge);`

c) `VoterUpgradeable.sol:23`: Unused enum value `Inactive`

d) Use of `assert` instead of `require`:

  i) `BridgeUpgradeable.sol:722`: `assert(IWETH(WETH).transfer(handler, amount));`

  ii) `ERC20HandlerUpgradeable.sol:356`: `assert(IWETH(_WETH).transfer(address(_reserve), amount));`

e) Remove the following unused contracts:

  i) `ERC721SafeUpgradeable.sol`

  ii) `CentrifugeAssetUpgradeable.sol`

  iii) `RouterERC721Upgradable.sol`

f) Rename the `isProposalExists` modifier of `BridgeUpgradeable.sol` to `doesProposalExist`

Status: Acknowledged

43. Gas Optimizations

Severity: Informational

a) The storage variable `__gap` is not needed for non-upgradeable contracts: `router-protocol-router-crosstalk/contracts/RouterCrossTalk.sol`

b) The implementation of `_updateReward`, `earned`, and `rewardPerToken` functions of `router-protocol-router-vault/contracts/TimelockVaults.sol` are not efficient enough, in these functions times and time values equaling 0 cause one side of the formula to evaluate to 0, unnecessarily consuming gas. These 3 functions can be broken down to several new functions which makes them more gas efficient.

Status: Acknowledged