RouterProtocol

LP & Staking Smart Contract Audit

Prepared by: **Halborn**

Date of Engagement: **September 22th, 2021 - October 30th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/8/2021	Ataberk Yavuzer
0.2	Document Edits	10/10/2021	Ataberk Yavuzer
0.3	Draft Review	10/11/2021	Gabi Urrutia
1.0	Remediation Plan	10/20/2021	Ataberk Yavuzer
1.1	Remediation Plan Review	10/30/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

RouterProtocol engaged Halborn to conduct a security assessment on their **RouterProtocol** LP & Staking Contracts beginning on September 22th and ending on October 30th, 2021.

The security assessment was scoped to the Github repository of Router-Protocol LP & Staking Contract. An audit of the security risk and implications regarding the changes introduced by the development team at RouterProtocol prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverable set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided six weeks for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the RouterProtocol Team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Ganache, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contracts.

Repository URL: https://github.com/router-protocol/router-bridge-contracts-v2/tree/testcases

- BridgeUpgradeable.sol
- CentrifugeAssetUpgradeable.sol
- VoterUpgradeable.sol
- ERC20SafeUpgradeable.sol
- ERC721SafeUpgradeable.sol
- FeeManagerUpgradeable.sol
- RouterERC20Upgradeable.sol
- RouterERC721Upgradeable.sol
- HandlerReserveUpgradeable.sol
- HandlerHelpersUpgradeable.sol
- ERC20HandlerUpgradeable.sol

Commit ID: 5721d9a51ac19db9fefaeae0125417e29b247673

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	1	5	5

LIKELIHOOD

IMPACT

				(HAL-01)
			(HAL-02) (HAL-03)	
		(HAL-04)		
(HAL-11)	(HAL-06) (HAL-07) (HAL-08) (HAL-09)	(HAL-05)		
(HAL-12) (HAL-13) (HAL-14)	(HAL-10)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF LIQUIDITY LOSS PROTECTION	Critical	SOLVED - 10/20/2021
(HAL-02) VOTER ADMIN CAN VOTE FOR OTHER USERS	High	SOLVED - 10/21/2021
(HAL-03) UUPS VULNERABILITY LEADS TO DESTRUCTION OF THE CONTRACT	High	SOLVED - 10/20/2021
(HAL-04) INVALID ROLE REVOKING PREVENTS TOKEN BURNING	Medium	ACKNOWLEDGED
(HAL-05) MISSING ROLE-BASED ACCESS CONTROL	Low	SOLVED - 10/20/2021
(HAL-06) LACK OF ZERO ADDRESS CHECK	Low	SOLVED - 10/20/2021
(HAL-07) MISSING RE-ENTRANCY GUARD	Low	SOLVED - 10/20/2021
(HAL-08) PRAGMA VERSION	Low	ACKNOWLEDGED
(HAL-09) MISUSE OF PUBLIC FUNCTIONS	Low	NOT APPLICABLE
(HAL-10) FIXED QUORUM ON VOTING MECHANISM	Informational	SOLVED - 10/20/2021
(HAL-11) UNUSED FUNCTIONS AND VARIABLES	Informational	SOLVED - 10/20/2021
(HAL-12) MISNAMED VARIABLE	Informational	SOLVED - 10/20/2021
(HAL-13) UNNECESSARY SAFEMATH IMPLEMENTATION	Informational	SOLVED - 10/20/2021
(HAL-14) MISSING EVENT EMITTING	Informational	SOLVED - 10/20/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF LIQUIDITY LOSS PROTECTION - CRITICAL

Description:

In the BridgeUpgradeable.sol contract, Halborn team noticed a withdrawal function that only can be used by admin of the contract. According to the analysis of the contract, the adminWithdraw function can withdraw all assets through the handler that is allowed to communicate with the Bridge contract. These situations are often enabled because a single executor role through a withdrawal function. While sometimes, the developer or owner does not intend to do this malicious act, the risk still exists if the private key is stolen since there is nothing preventing the key-holder from calling the withdrawal.

Code Location:

Listing 1: BridgeUpgradeable.sol (Lines 429)

```
422 function adminWithdraw(
423         address handlerAddress,
424         address tokenAddress,
425         address recipient,
426         uint256 amountOrTokenID
427     ) public virtual onlyRole(DEFAULT_ADMIN_ROLE) {
428         IERCHandler handler = IERCHandler(handlerAddress);
429         handler.withdraw(tokenAddress, recipient, amountOrTokenID)
                ;
430     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

The adminWithdraw function allow the executors or owners of the system to perform withdraw all amounts from token addresses. The owner should be limited to the minimum operations possible. These functionalities should be split between multiple role-based users with multi-signature wallets for each one. If it is not the intended behavior of the contract, the codes should be deleted from the repository. As another solution, the governance mechanism should be implemented on the critical changes as well as a Timelock.

3.2 (HAL-02) VOTER ADMIN CAN VOTE FOR OTHER USERS - HIGH

Description:

The voting mechanism on the BridgeUpgradeable contract works synchronously with a trusted VoterUpgradeable contract. Contract admin grants Relayer role to specific users. These Relayer users can vote on the proposals. Also, if Voter Admin grants Relayer/Voter role to any user, 1 RRT token minted to that user.

Basically, all Relayer users have 1 RRT to vote and privilege for voting during proposals. On the BridgeUpgradeable side, users create a proposal to be voted by other users. From the Bridge perspective, the voting mechanism looks securely designed. However, VoterUpgradeable contract has flaw on its vote function. It is possible to vote for another user. So, Voter Admin can manipulate all proposal results by voting for other relayers.

Following steps should replicate to reproduce the attack scenario.

BridgeUpgradeable.sol:

1. Call voteProposal function with RELAYER_ROLE privilege from BridgeUpgradeable contract.
2. If proposal is not created yet, `_voter.createProposal(block.number.add(_expiry), uint8(60))` function will be called.
3. This function will also create an id value.
4. If proposal is already created, `_voter.vote(_proposals[proposalHash], 1, msg.sender)` will be called by contract and msg.sender will vote Yes to the proposal.

VoterUpgradeable.sol:

1. Take a note for any ongoing proposal id.
2. The vote(uint256 issueId, uint8 option, address relayer) function is only callable by BRIDGE_ROLE that is Voter Contract Admin privilege.
3. Grant Relayer role to any user with BRIDGE_ROLE privilege. This process will mint 1 RRT to the Relayer. This call is necessary to bypass isValidbalance(relayer) modifier.
4. Call vote() function with new Relayer address.

As a result, it is possible to manipulate all ongoing proposals by Voter Contract Admin.

Code Location:

Listing 2: VoterUpgradeable.sol (Lines 264,269,272,273,274)

```
257 function vote(
258         uint256 issueId,
259         uint8 option,
260         address relayer
261     )
262         public
263         virtual
264         onlyRole(BRIDGE_ROLE)
265         isvalidIssue(issueId)
266         isNotVoted(issueId, relayer)
267         isValidOption(option)
268         isNotEnded(issueId)
269         isValidbalance(relayer)
270         returns (bool success)
271     {
272         uint256 balance = balanceOf(relayer);
273         hasVoted[issueId][relayer] = hasVotedStruct(true, option);
274         voteWeight[issueId][option] = voteWeight[issueId][option].
                add(balance);
275         issueMap[issueId].maxVotes = issueMap[issueId].maxVotes.
                add(balance);
276         emit OnVote(issueId, relayer, balance);
277         return true;
```

```
}
```

Listing 3: BridgeUpgradeable.sol (Lines 819,827)

```
811 function voteProposal(
812         uint8 chainID,
813         uint64 depositNonce,
814         bytes32 resourceID,
815         bytes32 dataHash
816     ) public virtual isResourceID(resourceID) onlyRole(
        RELAYER_ROLE) whenNotPaused {
817         bytes32 proposalHash = keccak256(abi.encodePacked(chainID,
             depositNonce, dataHash));
818         if (_proposals[proposalHash] == 0) {
819             uint256 id = _voter.createProposal(block.number.add(
                 _expiry), uint8(60));
820             _proposals[proposalHash] = id;
821             _proposalDetails[id] = proposalStruct(chainID,
                 depositNonce, resourceID, dataHash);
822         } else if (_voter.fetchIsExpired(_proposals[proposalHash])
             ) {
823             _voter.setStatus(_proposals[proposalHash]);
824             return;
825         }
826         if (_voter.getStatus(_proposals[proposalHash]) !=
             VoterUpgradeable.ProposalStatus.Cancelled) {
827             _voter.vote(_proposals[proposalHash], 1, msg.sender);
828         }
829     }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

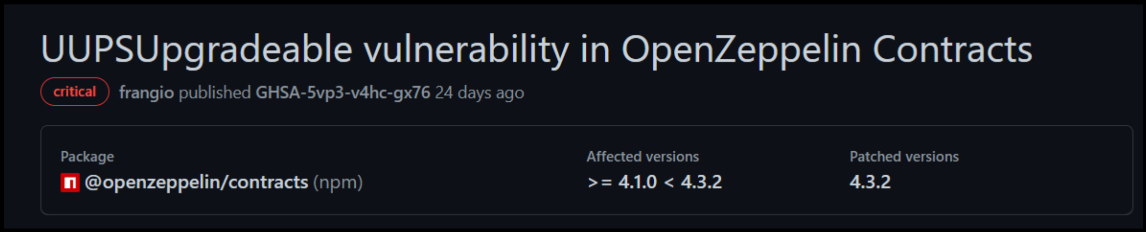
The vote function should be implemented more reliable according to attack scenario above. The Voter contract is used as a Bridge. However, at this stage, it should be considered that Voter Contract Admin can modify relayer variable.

3.3 (HAL-03) UUPS VULNERABILITY LEADS TO DESTRUCTION OF THE CONTRACT - HIGH

Description:

During the audit, it was noted that the proxy model used on contracts is the UUPS Proxy model. It was announced on 9th of September, 2021 that UUPS Proxy is vulnerable to initialization attack. Due to this attack, it is possible to initialize any upgradeable contract which is not initialized yet.

In the details, the vulnerable UUPSUpgradeable contract includes upgradeToAndCall function itself. The upgradeToAndCall function takes two arguments. The first one is newImplementation argument. This argument is necessary to tell which address the contract should upgrade to. The second argument is data. The data argument specifies which function will be called immediately after contract upgrade. At that point, DELEGATECALL function is called by using the data argument. Therefore, it is possible to destruct the contract by upgrading to V2 contract with selfdestruct function.




```
BridgeUpgradeable.sol:9:import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
BridgeUpgradeable.sol:23:contract BridgeUpgradeable is Initializable, PausableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable {
VoterUpgradeable.sol:12:import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
VoterUpgradeable.sol:18:contract VoterUpgradeable is Initializable, AccessControlUpgradeable, ERC20Upgradeable, UUPSUpgradeable {
```

FINDINGS & TECH DETAILS

```
"devDependencies": {
  "@codechecks/client": "^0.1.10",
  "@commitlint/cli": "^12.1.4",
  "@commitlint/config-conventional": "^12.1.4",
  "@ethersproject/abi": "^5.2.0",
  "@ethersproject/abstract-signer": "^5.2.0",
  "@ethersproject/bignumber": "^5.2.0",
  "@ethersproject/bytes": "^5.2.0",
  "@ethersproject/contracts": "^5.2.0",
  "@ethersproject/providers": "^5.2.0",
  "@nomiclabs/hardhat-ethers": "^2.0.2",
  "@nomiclabs/hardhat-etherscan": "^2.1.1",
  "@nomiclabs/hardhat-ganache": "^2.0.0",
  "@nomiclabs/hardhat-waffle": "^2.0.1",
  "@openzeppelin/contracts": "4.2.0",
  "@openzeppelin/contracts-upgradeable": "4.3.1",
  "@openzeppelin/hardhat-upgrades": "^1.10.0",
```

Code Location:

Listing 4: BridgeUpgradeable.sol

```
contract BridgeUpgradeable is Initializable, PausableUpgradeable,
    AccessControlUpgradeable, UUPSUpgradeable {
    using SafeMathUpgradeable for uint256;
...
```

Listing 5: VoterUpgradeable.sol

```
contract VoterUpgradeable is Initializable,
    AccessControlUpgradeable, ERC20Upgradeable, UUPSUpgradeable {
    using SafeMathUpgradeable for uint256;
    using CountersUpgradeable for CountersUpgradeable.Counter;
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

It is recommended to initialize the contracts if they are on deployed state as soon as possible. Also, update the **UUPSUpgradeable.sol** version to the latest version.

References:

UUPSUpgradeable Vulnerability Post-mortem

FINDINGS & TECH DETAILS

3.4 (HAL-04) INVALID ROLE REVOKING PREVENTS TOKEN BURNING - MEDIUM

Description:

The contract admin on the Bridge contract grants Relayer role to specific users. If Contract Admin grants Relayer/Voter role to any user, 1 RRT token minted to that user. So, these Relayer users can vote on the proposals with their tokens. In addition, the totalRelayers variable will be incremented by **1** on every RELAYER_ROLE granting.

During the tests, it has been determined that the revokeRole function on the BridgeUpgradeable contract does not work properly while performing access controls checks.

The contract controls that if a specified address have RELAYER_ROLE and RRT balance of that address is equal to 1, that user's RRT token will be burned and totalRelayers variable will be decremented by **1**.

For example, Alice and Bob are only relayers on the contract. If Relayer Alice send his 1 RRT to Relayer Bob, Bob will have 2 RRTs. So, calling the revokeRole function on Bob will not work properly. The contract will not burn tokens of Bob and it will not decrease count of totalRelayers. As a result, Bob will not have the RELAYER_ROLE anymore. However, he will keep his 2 RRTs and totalRelayers count will be still **2**.

Code Location:

Listing 6: BridgeUpgradeable.sol (Lines 229,230)

```
227 function grantRole(bytes32 role, address account) public virtual
        override onlyRole(getRoleAdmin(role)) {
228         super.grantRole(role, account);
229         if (role == RELAYER_ROLE && _voter.balanceOf(account) == 0
                ether) {
230             _voter.mint(account);
231             totalRelayers = totalRelayers.add(1);
232             emit RelayerAdded(account);
233         }
234     }
```

Listing 7: BridgeUpgradeable.sol (Lines 246,247,248)

```
244 function revokeRole(bytes32 role, address account) public virtual
        override onlyRole(getRoleAdmin(role)) {
245         super.revokeRole(role, account);
246         if (role == RELAYER_ROLE && _voter.balanceOf(account) == 1
                ether) {
247             _voter.burn(account);
248             totalRelayers = totalRelayers.sub(1);
249             emit RelayerRemoved(account);
250         }
251     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to properly implement the control of the token amount in the revokeRole function. The following implementation can be used to fix the vulnerability.

Listing 8: Possible Bug-Fix (Lines 3)

```
1 function revokeRole(bytes32 role, address account) public virtual
    override onlyRole(getRoleAdmin(role)) {
2         super.revokeRole(role, account);
3         if (role == RELAYER_ROLE) {
4             _voter.burn(account);
5             totalRelayers = totalRelayers.sub(1);
6             emit RelayerRemoved(account);
7         }
8     }
```

3.5 (HAL-05) MISSING ROLE-BASED ACCESS CONTROL - LOW

Description:

In smart contracts, implementing a correct Access Control policy is an essential step to maintain security and decentralization of permissions on a token. All the features of the smart contract , such as mint/burn tokens and pause contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Nevertheless, other authorization levels are required to follow the principle of least privilege, also known as least authority. Briefly, any process, user or program only can access to the necessary resources or information. Otherwise, the ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

There are multiple important functionalities on BridgeUpgradeable.sol, VoterUpgradeable.sol and CentrifugeAssetUpgradeable.sol contracts such as adding new relayers, resource setting, voting proposals, pausing the whole contract etc. It is important to divide these functionalities into multiple roles.

Code Location:

Listing 9: BridgeUpgradeable.sol (Lines 287,295)

```
function pause() public virtual onlyRole(DEFAULT_ADMIN_ROLE)
    whenNotPaused {
        _pause();
    }

    /**
        @notice Unpauses deposits, proposal creation and voting,
            and deposit executions.
        @notice Only callable by an address that currently has the
            admin role.
```

```
294     */
295     function unpause() public virtual onlyRole(DEFAULT_ADMIN_ROLE)
            whenPaused {
296         _unpause();
297     }
```

Listing 10: BridgeUpgradeable.sol (Lines 326)

```
326 function adminSetResource(
327         address handlerAddress,
328         bytes32 resourceID,
329         address tokenAddress
330     ) public virtual onlyRole(DEFAULT_ADMIN_ROLE) {
331         _resourceIDToHandlerAddress[resourceID] = handlerAddress;
332         IERCHandler handler = IERCHandler(handlerAddress);
333         handler.setResource(resourceID, tokenAddress);
334     }
```

Listing 11: CentrifugeAssetUpgradeable.sol (Lines 22)

```
22 function store(bytes32 asset) external {
23         require(!_assetsStored[asset], "asset is already stored");
24         _assetsStored[asset] = true;
25         emit AssetStored(asset);
26     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendations:

It is recommended to split roles on important functions such as store(), pause(), unpause(), adminSetResource(), etc. In the other way, any user beside contract/contract admin itself will be able to use these high privileged functions. Also, PAUSER and RESOURCE_SETTER roles should be implemented to the contract.

3.6 (HAL-06) LACK OF ZERO ADDRESS CHECK - LOW

Description:

The ERC20SafeUpgradeable.sol contract have multiple input fields on their both public and private functions. Some of these inputs are required as address variable.

During the test, it has seen all of these inputs are not protected against using the address(0) as the target address. It is not recommended to use zero address as target addresses on the contracts.

Code Location:

Listing 12: ERC20SafeUpgradeable.sol (Lines 153,154)

```
function safeTransferETH(address to, uint256 value) public
    onlyRole(ERC20HANDLER_ROLE) {
        (bool success, ) = to.call{ value: value }(new bytes(0));
        require(success, "safeTransferETH: ETH transfer failed");
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to implement additional address check to detect is current contract getting used as a target address.

Listing 13: ERC20SafeUpgradeable.sol (Lines 153,154)

```
function safeTransferETH(address to, uint256 value) public
    onlyRole(ERC20HANDLER_ROLE) {
        require(to != address(0), "RouterProtocol: zero address is
            not allowed");
        (bool success, ) = to.call{ value: value }(new bytes(0));
        require(success, "safeTransferETH: ETH transfer failed");
    }
```

3.7 (HAL-07) MISSING RE-ENTRANCY GUARD - LOW

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against reentrancy attacks.

Code Location:

Listing 14: Missing Re-Entrancy Guard

```
grantRole(bytes32 role, address account)
adminWithdraw(address handlerAddress, address tokenAddress, address recipient, uint256 amountOrTokenID)
transferFunds(address payable[] calldata addrs, uint256[] calldata amounts)
transferFundsERC20(address[] calldata addrs, address[] calldata tokens, uint256[] calldata amounts)
deposit(uint8 destinationChainID, bytes32 resourceID, bytes calldata data, uint256[] memory distribution, uint256[] memory flags, address[] memory path, address feeTokenAddress)
depositETH(uint8 destinationChainID, bytes32 resourceID, bytes calldata data, uint256[] memory distribution, uint256[] memory flags, address[] memory path, address feeTokenAddress)
stake(bytes32 resourceID, address tokenAddress, uint256 amount)
stakeETH(bytes32 resourceID, address tokenAddress, uint256 amount)
unstake(bytes32 resourceID, address tokenAddress, uint256 amount)
unstakeETH(bytes32 resourceID, address tokenAddress, uint256 amount)
mint(address account)
burn(address account)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

In LP & Staking contract, functions above are missing a nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

3.8 (HAL-08) PRAGMA VERSION - LOW

Description:

The project uses one of the latest pragma version (0.8.2) which was released on 2nd of March, 2021. The latest pragma version (0.8.9) was released in October 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

In the Solitidy Github repository, there is a JSON file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Code Location:

Listing 15

```
ERC721SafeUpgradeable.sol:2:pragma solidity 0.8.2;
BridgeUpgradeable.sol:2:pragma solidity 0.8.2;
interfaces/IOneSplitWrap.sol:2:pragma solidity >=0.8.2;
interfaces/ILiquidityPool.sol:2:pragma solidity >=0.8.2;
interfaces/IWETH.sol:2:pragma solidity >=0.8.2;
interfaces/IBridge.sol:2:pragma solidity 0.8.2;
interfaces/IHandlerReserve.sol:2:pragma solidity >=0.8.2;
interfaces/IGenericHandler.sol:2:pragma solidity 0.8.2;
interfaces/IFeeManager.sol:2:pragma solidity >=0.8.2;
interfaces/IERCHandler.sol:2:pragma solidity 0.8.2;
interfaces/IDepositExecute.sol:2:pragma solidity 0.8.2;
ERC20SafeUpgradeable.sol:2:pragma solidity >=0.8.2;
FeeManagerUpgradeable.sol:3:pragma solidity >=0.8.2;
RouterERC20Upgradable.sol:2:pragma solidity 0.8.2;
test/WETH.sol:20:pragma solidity ^0.8.2;
VoterUpgradeable.sol:2:pragma solidity 0.8.2;
handlers/ERC20HandlerUpgradeable.sol:2:pragma solidity >=0.8.2;
handlers/HandlerReserveUpgradeable.sol:2:pragma solidity >=0.8.2;
handlers/HandlerHelpersUpgradeable.sol:2:pragma solidity >=0.8.2;
CentrifugeAssetUpgradeable.sol:2:pragma solidity 0.8.2;
utils/SafeCastUpgradeable.sol:3:pragma solidity ^0.8.0;
```

```
utils/AddressUpgradeable.sol:3:pragma solidity ^0.8.0;
utils/SafeMathUpgradeable.sol:3:pragma solidity ^0.8.0;
utils/PausableUpgradeable.sol:3:pragma solidity ^0.8.0;
utils/AccessControlUpgradeable.sol:3:pragma solidity ^0.8.0;
RouterERC721Upgradable.sol:2:pragma solidity 0.8.2;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

References:

- Solidity Releases
- Solidity Bugs By Version

3.9 (HAL-09) MISUSE OF PUBLIC FUNCTIONS - LOW

Description:

If functions belonging to external libraries are used without making any changes to them, undesirable results may occur in some cases, such as accessing functions that should not be accessed. During the test, it has seen that the store function of the contract are accessible by all users mistakenly. This function, which are open to use, may cause the smart contract to malfunction in some cases. This situation leads any user to store untrusted assets on the CentrifugeAssetUpgradeable contract.

Code Location:

Listing 16: CentrifugeAssetUpgradeable.sol

```
function store(bytes32 asset) external {
        require(!_assetsStored[asset], "asset is already stored");
        _assetsStored[asset] = true;
        emit AssetStored(asset);
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to implement Role-Based Access Control to the store function. If this function is intended to be used internally, use the internal instead of external keyword.

3.10 (HAL-10) FIXED QUORUM ON VOTING MECHANISM - INFORMATIONAL

Description:

According to the __BridgeUpgradeable_init function on the BridgeUpgradeable.sol contract, that quorum variable is set on contract initialization/upgrade. Main purpose of this variable is defining percentage for yes votes to execute proposals. Even if this variable is defined while initializing, it has no validity in the code. The contract strictly accepts the quorum variable as **60** in the createProposal() function.

Code Location:

Listing 17: BridgeUpgradeable.sol (Lines 819)

```
811 function voteProposal(
812         uint8 chainID,
813         uint64 depositNonce,
814         bytes32 resourceID,
815         bytes32 dataHash
816     ) public virtual isResourceID(resourceID) onlyRole(
        RELAYER_ROLE) whenNotPaused {
817         bytes32 proposalHash = keccak256(abi.encodePacked(chainID,
             depositNonce, dataHash));
818         if (_proposals[proposalHash] == 0) {
819             uint256 id = _voter.createProposal(block.number.add(
                _expiry), uint8(60));
820             _proposals[proposalHash] = id;
821             _proposalDetails[id] = proposalStruct(chainID,
                depositNonce, resourceID, dataHash);
822         } else if (_voter.fetchIsExpired(_proposals[proposalHash])
            ) {
823             _voter.setStatus(_proposals[proposalHash]);
824             return;
825         }
826         if (_voter.getStatus(_proposals[proposalHash]) !=
            VoterUpgradeable.ProposalStatus.Cancelled) {
827             _voter.vote(_proposals[proposalHash], 1, msg.sender);
```

```
        }
    }
```

Listing 18: VoterUpgradeable.sol (Lines 238)

```
229 function createProposal(uint256 endBlock, uint64 quorum)
230         public
231         virtual
232         onlyRole(BRIDGE_ROLE)
233         isValidquorum(quorum)
234         returns (uint256 id)
235     {
236         _IssueCtr.increment();
237         uint256 ctr = _IssueCtr.current();
238         issueMap[ctr] = issueStruct(ProposalStatus.Active, block.
            number, endBlock, quorum, 0, 0);
239         emit OnCreateIssue(ctr);
240         emit OnStatusChange(ctr, issueMap[ctr].status);
241         return ctr;
242     }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendations:

It is recommended to pass the quorum variable as a new argument on the createProposal function.

3.11 (HAL-11) UNUSED FUNCTIONS AND VARIABLES - INFORMATIONAL

Description:

During the test, it was determined that some functions and variables on the contract were not used in any way, although they were defined on the contract. This situation does not pose any risk in terms of security. But it is important for the readability and applicability of the code.

Code Location:

Listing 19: Unused Functions and Variables

```
uint256 private totalRelayers; //this variable will be remain zero
    on Voter contract.
function fetchTotalRelayers() //this function will remain zero on
    Voter contract.
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

It is recommended to review these unused functions and variables, and to delete them from the contract if they will continue to be unused.

3.12 (HAL-12) MISNAMED VARIABLE - INFORMATIONAL

Description:

During the tests, it has seen that the variable name of a function in the contract was misnamed. It is understood that the function was copied from the function above it and the variable name remained as the function above it.

Code Location:

Listing 20: BridgeUpgradeable.sol (Lines 55,59,60)

```
55 function fetch_proposals(bytes32 _id) public view virtual returns
       (uint256) {
56         return _proposals[_id];
57     }
58
59     function fetch_whitelist(address _id) public view virtual
           returns (bool) {
60         return _whitelist[_id];
61     }
```

Listing 21: BridgeUpgradeable.sol (Lines 260)

```
260 function addToWhitelist(address _beneficiary) public virtual
        onlyRole(DEFAULT_ADMIN_ROLE) isWhitelistEnabled {
261         _whitelist[_beneficiary] = true;
262     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

The address _id variable on the fetch_whitelist function should be replaced with address _beneficiary according to the argument name of addToWhitelist function.

3.13 (HAL-13) UNNECESSARY SAFEMATH IMPLEMENTATION - INFORMATIONAL

Description:

During the audit, it was determined that the OpenZeppelin's SafeMath library is actively used for uint256 data type on contracts.

If you're using an unsigned integer in Solidity, the possible values of your variable ranges from 0 to 2 ^ 256. So, it means that if you are around the max value and increment your variable it will go back to 0. The same happens if your variable is at 0 and you subtract one, instead of **overflow** it is called **underflow**.

The SafeMath library also protects contracts for possible integer overflows or underflows. Starting with Solidity version 0.8.0 all arithmetic is checked by default. Therefore, usage of SafeMath library on these contracts are unnecessary.

Code Location:

Listing 22: SafeMath Implementations

```
ERC721SafeUpgradeable.sol:5:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
BridgeUpgradeable.sol:4:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
ERC20SafeUpgradeable.sol:5:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
RouterERC20Upgradable.sol:5:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
VoterUpgradeable.sol:5:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
handlers/ERC20HandlerUpgradeable.sol:4:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
handlers/HandlerHelpersUpgradeable.sol:4:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
RouterERC721Upgradable.sol:5:import "@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol";
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to remove unnecessary SafeMath libraries from contracts if their purpose are preventing integer overflows/underflows.

3.14 (HAL-14) MISSING EVENT EMITTING - INFORMATIONAL

Description:

It has been observed that critical functionality missing emitting event for contract functions. These functions should emit events after completing the transactions. Event emitting on contracts is important for detailed logging of transactions.

Code Location:

Listing 23: Functions with Missing Events

```
addToWhitelist() // BridgeUpgradeable.sol
removeFromWhitelist() // BridgeUpgradeable.sol
setWhitelisting() // BridgeUpgradeable.sol
set_quorum() // BridgeUpgradeable.sol
adminSetLiquidityPool() // BridgeUpgradeable.sol
adminSetGenericResource() // BridgeUpgradeable.sol
adminWithdraw() // BridgeUpgradeable.sol
transferFunds() // BridgeUpgradeable.sol
transferFundsERC20() // BridgeUpgradeable.sol
setBridgeFees() // BridgeUpgradeable.sol
setBridgeFee() // BridgeUpgradeable.sol
setBridgeFee() // BridgeUpgradeable.sol
mint() // VoterUpgradeable.sol
burn() // VoterUpgradeable.sol
_transfer // VoterUpgradeable.sol
safeTransferETH() // ERC20SafeUpgradeable.sol
fundERC721() // ERC721SafeUpgradeable.sol
lockERC721() // ERC721SafeUpgradeable.sol
releaseERC721() // ERC721SafeUpgradeable.sol
mintERC721() // ERC721SafeUpgradeable.sol
burnERC721() // ERC721SafeUpgradeable.sol
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to implement events for the functions in the Code Location section and use these events in related functions.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

AUTOMATED TESTING

Slither Results:

```
BridgeUpgradeable.transferFunds(address[],uint256[]) (contracts/BridgeUpgradeable.sol#439-449) sends eth to arbitrary user
        Dangerous calls:
        - addrs[i].transfer(amounts[i]) (contracts/BridgeUpgradeable.sol#447)
ERC20SafeUpgradeable.safeTransferETH(address,uint256) (contracts/ERC20SafeUpgradeable.sol#153-156) sends eth to arbitrary user
        Dangerous calls:
        - (success) = to.call{value: value}(new bytes(0)) (contracts/ERC20SafeUpgradeable.sol#154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#207-213) uses delegatecall to a input-controlled
 function id
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

IAccessControlUpgradeable is re-used:
        - contracts/utils/AccessControlUpgradeable.sol#13-23
        - node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#8-87
AccessControlUpgradeable is re-used:
        - contracts/utils/AccessControlUpgradeable.sol#63-266
        - node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#49-220
PausableUpgradeable is re-used:
        - contracts/utils/PausableUpgradeable.sol#17-98
        - node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#17-97
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#219) shadows:
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC2771ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/metatx/ERC2771ContextUpgradeable.sol#45) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#64) shadows:
        - ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#214)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#361) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418) shadows:
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
RouterERC721Upgradable.__gap (contracts/RouterERC721Upgradable.sol#107) shadows:
        - ERC721Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#418)
        - PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#96)
        - AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#219)
        - ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#35)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

BridgeUpgradeable.transferFundsERC20(address[],address[],uint256[]) (contracts/BridgeUpgradeable.sol#460-471) ignores return value by IERC20Upgradeable(tokens[i]).transfer(addrs[i],amounts[i]) (contracts/B
ridgeUpgradeable.sol#469)
BridgeUpgradeable.depositETH(uint8,bytes32,bytes,uint256[],uint256[],address[],address) (contracts/BridgeUpgradeable.sol#610-643) ignores return value by IWETH(WETH).transfer(swapDetails.handler,swapDetail
s.srcTokenAmount) (contracts/BridgeUpgradeable.sol#637)
ERC20HandlerUpgradeable.depositETH(bytes32,uint8,uint64,IDepositExecute.SwapInfo) (contracts/handlers/ERC20HandlerUpgradeable.sol#201-252) ignores return value by IWETH(_WETH).transfer(_oneSplitAddress,swa
pDetails.srcTokenAmount) (contracts/handlers/ERC20HandlerUpgradeable.sol#234)
BridgeUpgradeable (contracts/BridgeUpgradeable.sol#23-893) is an upgradeable contract that does not protect its initiliaze functions: BridgeUpgradeable.initialize(uint8,uint256,uint256,address) (contracts/
BridgeUpgradeable.sol#204-211). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#35-38)UUPSUpgradeab
le.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#48-51)VoterUpgradeable (contracts/VoterUpgradeable.sol#18-426) is an upgradeable contrac
t that does not protect its initiliaze functions: VoterUpgradeable.initialize() (contracts/VoterUpgradeable.sol#158-160). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modul
es/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#35-38)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.so
l#48-51)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract

BridgeUpgradeable.grantRole(bytes32,address) (contracts/BridgeUpgradeable.sol#227-234) uses a dangerous strict equality:
        - role == RELAYER_ROLE && _voter.balanceOf(account) == 0 (contracts/BridgeUpgradeable.sol#229)
BridgeUpgradeable.revokeRole(bytes32,address) (contracts/BridgeUpgradeable.sol#244-251) uses a dangerous strict equality:
        - role == RELAYER_ROLE && _voter.balanceOf(account) == 1000000000000000000 (contracts/BridgeUpgradeable.sol#246)
VoterUpgradeable.isNotEnded(uint256) (contracts/VoterUpgradeable.sol#62-68) uses a dangerous strict equality:
        - require(bool,string)((block.number < issueMap[_issue].endBlock) && (issueMap[_issue].status == ProposalStatus.Active),ERC-1202: Voting has ended) (contracts/VoterUpgradeable.sol#63-66)
VoterUpgradeable.isNotvalidIssue(uint256) (contracts/VoterUpgradeable.sol#57-60) uses a dangerous strict equality:
        - require(bool,string)(issueMap[_issue].status == ProposalStatus.Inactive,ERC-1202: A valid proposal) (contracts/VoterUpgradeable.sol#58)
VoterUpgradeable.isPassed(uint256) (contracts/VoterUpgradeable.sol#95-98) uses a dangerous strict equality:
        - require(bool,string)(issueMap[id].status == ProposalStatus.Passed,ERC-1202: Proposal is not passed) (contracts/VoterUpgradeable.sol#96)
VoterUpgradeable.isvalidIssue(uint256) (contracts/VoterUpgradeable.sol#52-55) uses a dangerous strict equality:
        - require(bool,string)(issueMap[_issue].status == ProposalStatus.Active,ERC-1202: Not a valid proposal) (contracts/VoterUpgradeable.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Contract locking ether found:
        Contract ERC20HandlerUpgradeable (contracts/handlers/ERC20HandlerUpgradeable.sol#21-431) has payable functions:
         - ERC20HandlerUpgradeable.receive() (contracts/handlers/ERC20HandlerUpgradeable.sol#114)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

Reentrancy in HandlerReserveUpgradeable._setLiquidityPool(string,string,uint8,address,address) (contracts/handlers/HandlerReserveUpgradeable.sol#138-158):
        External calls:
        - newLPAddr.initialize(name,symbol,decimals) (contracts/handlers/HandlerReserveUpgradeable.sol#151)
        State variables written after the call(s):
        - _contractToLP[contractAddress] = newLPAddress (contracts/handlers/HandlerReserveUpgradeable.sol#157)
Reentrancy in BridgeUpgradeable.voteProposal(uint8,uint64,bytes32,bytes32) (contracts/BridgeUpgradeable.sol#811-829):
        External calls:
        - id = _voter.createProposal(block.number.add(_expiry),uint8(60)) (contracts/BridgeUpgradeable.sol#819)
        State variables written after the call(s):
        - _proposals[proposalHash] = id (contracts/BridgeUpgradeable.sol#820)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#376-397) ignores return value by IERC721Receiver
Upgradeable(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#383-393)
BridgeUpgradeable.voteProposal(uint8,uint64,bytes32,bytes32) (contracts/BridgeUpgradeable.sol#811-829) ignores return value by _voter.setStatus(_proposals[proposalHash]) (contracts/BridgeUpgradeable.sol#82
3)
BridgeUpgradeable.voteProposal(uint8,uint64,bytes32,bytes32) (contracts/BridgeUpgradeable.sol#811-829) ignores return value by _voter.vote(_proposals[proposalHash],1,msg.sender) (contracts/BridgeUpgradeabl
e.sol#827)
BridgeUpgradeable.cancelProposal(uint8,uint64,bytes32) (contracts/BridgeUpgradeable.sol#840-854) ignores return value by _voter.setStatus(_proposals[proposalHash]) (contracts/BridgeUpgradeable.sol#853)
BridgeUpgradeable.executeProposal(uint8,uint64,bytes,bytes32,uint256[],uint256[],address[]) (contracts/BridgeUpgradeable.sol#867-890) ignores return value by _voter.executeProposal(_proposals[proposalHash]
) (contracts/BridgeUpgradeable.sol#886)
ERC20SafeUpgradeable.mintERC20(address,address,uint256) (contracts/ERC20SafeUpgradeable.sol#85-92) ignores return value by erc20.mint(recipient,amount) (contracts/ERC20SafeUpgradeable.sol#91)
ERC20SafeUpgradeable.burnERC20(address,address,uint256) (contracts/ERC20SafeUpgradeable.sol#100-107) ignores return value by erc20.burn(owner,amount) (contracts/ERC20SafeUpgradeable.sol#106)
ERC721SafeUpgradeable.mintERC721(address,address,uint256,bytes) (contracts/ERC721SafeUpgradeable.sol#84-92) ignores return value by erc721.mint(recipient,tokenID,data) (contracts/ERC721SafeUpgradeable.sol#
91)
ERC721SafeUpgradeable.burnERC721(address,uint256) (contracts/ERC721SafeUpgradeable.sol#99-102) ignores return value by erc721.burn(tokenID) (contracts/ERC721SafeUpgradeable.sol#101)
```

```
ERC20SafeUpgradeable.safeTransferETH(address,uint256).to (contracts/ERC20SafeUpgradeable.sol#153) lacks a zero-check on :
                - (success) = to.call{value: value}(new bytes(0)) (contracts/ERC20SafeUpgradeable.sol#154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

BridgeUpgradeable.transferFunds(address[],uint256[]) (contracts/BridgeUpgradeable.sol#439-449) has external calls inside a loop: addrs[i].transfer(amounts[i]) (contracts/BridgeUpgradeable.sol#447)
BridgeUpgradeable.transferFundsERC20(address[],address[],uint256[]) (contracts/BridgeUpgradeable.sol#460-471) has external calls inside a loop: IERC20Upgradeable(tokens[i]).transfer(addrs[i],amounts[i]) (c
ontracts/BridgeUpgradeable.sol#469)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#383)' in ERC721Upgradeable._che
ckOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#376-397) potentially used before declaration: retval == IERC721Receive
rUpgradeable.onERC721Received.selector (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#384)
Variable 'ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#385)' in ERC721Upgradeable._che
ckOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#376-397) potentially used before declaration: reason.length == 0 (node
_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#386)
Variable 'ERC721Upgradeable._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#385)' in ERC721Upgradeable._che
ckOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#376-397) potentially used before declaration: revert(uint256,uint256)(
32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in HandlerReserveUpgradeable._setLiquidityPool(string,string,uint8,address,address) (contracts/handlers/HandlerReserveUpgradeable.sol#138-158):
        External calls:
        - newLPAddr.initialize(name,symbol,decimals) (contracts/handlers/HandlerReserveUpgradeable.sol#151)
        State variables written after the call(s):
        - _lpToContract[newLPAddress] = contractAddress (contracts/handlers/HandlerReserveUpgradeable.sol#156)
Reentrancy in ERC20HandlerUpgradeable.deposit(bytes32,uint8,uint64,IDepositExecute.SwapInfo) (contracts/handlers/ERC20HandlerUpgradeable.sol#136-192):
        External calls:
        - _reserve.deductFee(swapDetails.feeTokenAddress,swapDetails.depositer,swapDetails.providedFee,transferFee,_ETH,_isFeeEnabled,_bridgeAddress) (contracts/handlers/ERC20HandlerUpgradeable.sol#147-155
)
        - handleDepositForReserveToken(swapDetails) (contracts/handlers/ERC20HandlerUpgradeable.sol#157)
                - _reserve.burnERC20(address(swapDetails.srcTokenAddress),swapDetails.depositer,swapDetails.srcTokenAmount) (contracts/handlers/ERC20HandlerUpgradeable.sol#334)
                - _reserve.lockERC20(address(swapDetails.srcTokenAddress),swapDetails.depositer,address(_reserve),swapDetails.srcTokenAmount) (contracts/handlers/ERC20HandlerUpgradeable.sol#336-341)
        - _reserve.deductFee(swapDetails.feeTokenAddress,swapDetails.depositer,swapDetails.providedFee,exchangeFee,_ETH,_isFeeEnabled,_bridgeAddress) (contracts/handlers/ERC20HandlerUpgradeable.sol#159-167
)
        - _reserve.lockERC20(address(swapDetails.srcTokenAddress),swapDetails.depositer,_oneSplitAddress,swapDetails.srcTokenAmount) (contracts/handlers/ERC20HandlerUpgradeable.sol#169-174)
        - handleDepositForNonReserveToken(swapDetails) (contracts/handlers/ERC20HandlerUpgradeable.sol#175)
                - swapDetails.srcStableTokenAmount = _reserve.swapMulti(_oneSplitAddress,swapDetails.path,swapDetails.srcTokenAmount,0,swapDetails.distribution,swapDetails.flags) (contracts/handlers/ERC20H
andlerUpgradeable.sol#348-355)
                - swapDetails.srcStableTokenAmount = _reserve.swap(_oneSplitAddress,address(swapDetails.srcTokenAddress),swapDetails.srcStableTokenAddress,swapDetails.srcTokenAmount,0,swapDetails.distribut
ion,swapDetails.flags[0]) (contracts/handlers/ERC20HandlerUpgradeable.sol#357-365)
        State variables written after the call(s):
        - _depositRecords[destinationChainID][depositNonce] = DepositRecord(destinationChainID,address(swapDetails.srcTokenAddress),swapDetails.srcStableTokenAddress,swapDetails.srcStableTokenAmount,addres
s(swapDetails.destStableTokenAddress),swapDetails.destStableTokenAmount,address(swapDetails.destTokenAddress),swapDetails.destTokenAmount,resourceID,swapDetails.recipient,swapDetails.depositer,swapDetails.
srcTokenAmount) (contracts/handlers/ERC20HandlerUpgradeable.sol#178-191)
Reentrancy in BridgeUpgradeable.grantRole(bytes32,address) (contracts/BridgeUpgradeable.sol#227-234):
        External calls:
        - _voter.mint(account) (contracts/BridgeUpgradeable.sol#230)
        State variables written after the call(s):
        - totalRelayers = totalRelayers.add(1) (contracts/BridgeUpgradeable.sol#231)
Reentrancy in BridgeUpgradeable.revokeRole(bytes32,address) (contracts/BridgeUpgradeable.sol#244-251):
        External calls:
        - _voter.burn(account) (contracts/BridgeUpgradeable.sol#247)
        State variables written after the call(s):
        - totalRelayers = totalRelayers.sub(1) (contracts/BridgeUpgradeable.sol#248)
Reentrancy in HandlerReserveUpgradeable.stake(address,address,uint256) (contracts/handlers/HandlerReserveUpgradeable.sol#62-71):
        External calls:
        - lockERC20(tokenAddress,depositor,address(this),amount) (contracts/handlers/HandlerReserveUpgradeable.sol#68)
                - (success,returndata) = address(token).call(data) (contracts/ERC20SafeUpgradeable.sol#145)
        - mintERC20(_contractToLP[tokenAddress],depositor,amount) (contracts/handlers/HandlerReserveUpgradeable.sol#69)
                - erc20.mint(recipient,amount) (contracts/ERC20SafeUpgradeable.sol#91)
        State variables written after the call(s):
        - _stakedRecords[depositor][tokenAddress] = _stakedRecords[depositor][tokenAddress] + (amount) (contracts/handlers/HandlerReserveUpgradeable.sol#70)
Reentrancy in HandlerReserveUpgradeable.stakeETH(address,address,uint256) (contracts/handlers/HandlerReserveUpgradeable.sol#73-81):
        External calls:
        - mintERC20(_contractToLP[tokenAddress],depositor,amount) (contracts/handlers/HandlerReserveUpgradeable.sol#79)
                - erc20.mint(recipient,amount) (contracts/ERC20SafeUpgradeable.sol#91)
        State variables written after the call(s):
        - _stakedRecords[depositor][tokenAddress] = _stakedRecords[depositor][tokenAddress] + (amount) (contracts/handlers/HandlerReserveUpgradeable.sol#80)
Reentrancy in HandlerReserveUpgradeable.unstake(address,address,uint256) (contracts/handlers/HandlerReserveUpgradeable.sol#90-99):
        External calls:
        - burnERC20(_contractToLP[tokenAddress],unstaker,amount) (contracts/handlers/HandlerReserveUpgradeable.sol#96)
                - erc20.burn(owner,amount) (contracts/ERC20SafeUpgradeable.sol#106)
        - releaseERC20(tokenAddress,unstaker,amount) (contracts/handlers/HandlerReserveUpgradeable.sol#97)
                - (success,returndata) = address(token).call(data) (contracts/ERC20SafeUpgradeable.sol#145)
        State variables written after the call(s):
        - _stakedRecords[unstaker][tokenAddress] = _stakedRecords[unstaker][tokenAddress] - (amount) (contracts/handlers/HandlerReserveUpgradeable.sol#98)
Reentrancy in HandlerReserveUpgradeable.unstakeETH(address,address,uint256,address) (contracts/handlers/HandlerReserveUpgradeable.sol#101-113):
        External calls:
        - burnERC20(_contractToLP[tokenAddress],unstaker,amount) (contracts/handlers/HandlerReserveUpgradeable.sol#108)
                - erc20.burn(owner,amount) (contracts/ERC20SafeUpgradeable.sol#106)
        State variables written after the call(s):
        - _stakedRecords[unstaker][tokenAddress] = _stakedRecords[unstaker][tokenAddress] - (amount) (contracts/handlers/HandlerReserveUpgradeable.sol#109)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#169-173)
name() should be declared external:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#67-69)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#75-77)
decimals() should be declared external:
        - ERC20Upgradeable.decimals() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#92-94)
        - RouterERC20Upgradable.decimals() (contracts/RouterERC20Upgradable.sol#83-85)
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101)
transfer(address,uint256) should be declared external:
        - ERC20Upgradeable.transfer(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#118-121)
allowance(address,address) should be declared external:
        - ERC20Upgradeable.allowance(address,address) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#126-128)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#137-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#155-169)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#183-186)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#202-210)
balanceOf(address) should be declared external:
        - ERC721Upgradeable.balanceOf(address) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#68-71)
name() should be declared external:
        - ERC721Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#85-87)
symbol() should be declared external:
        - ERC721Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#92-94)
tokenURI(uint256) should be declared external:
        - ERC721Upgradeable.tokenURI(uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#99-104)
approve(address,uint256) should be declared external:
        - ERC721Upgradeable.approve(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#118-128)
setApprovalForAll(address,bool) should be declared external:
        - ERC721Upgradeable.setApprovalForAll(address,bool) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#142-147)
transferFrom(address,address,uint256) should be declared external:
        - ERC721Upgradeable.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#159-168)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721Upgradeable.safeTransferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#173-179)
fetchMAX_RELAYERS() should be declared external:
        - BridgeUpgradeable.fetchMAX_RELAYERS() (contracts/BridgeUpgradeable.sol#27-29)
fetchMAX_FEE_SETTERS() should be declared external:
        - BridgeUpgradeable.fetchMAX_FEE_SETTERS() (contracts/BridgeUpgradeable.sol#31-33)
fetch_chainID() should be declared external:
        - BridgeUpgradeable.fetch_chainID() (contracts/BridgeUpgradeable.sol#35-37)
fetch_expiry() should be declared external:
        - BridgeUpgradeable.fetch_expiry() (contracts/BridgeUpgradeable.sol#39-41)
fetch_whitelistEnabled() should be declared external:
        - BridgeUpgradeable.fetch_whitelistEnabled() (contracts/BridgeUpgradeable.sol#43-45)
```

As a result of the tests completed with the Slither tool, some results were obtained and these results were reviewed by Halborn. In line with the reviewed results, it was decided that some vulnerabilities were false-positive and these results were not included in the report. The actual vulnerabilities are already included in the findings on the report.

THANK YOU FOR CHOOSING

HALBORN