Connext Security Review

Auditors

Sawmon and Natalie, Lead Security Researcher

Gerard Persoon, Lead Security Researcher

Csanuragjain, Security Researcher

Xiaoming90, Security Researcher

Blockdev, Associate Security Researcher

Report prepared by: Pablo Misirov

December 15, 2022

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Connext is a crosschain liquidity network that enables fast, fully-noncustodial transfers between EVM-compatible chains and L2 systems. It leverages the Ethereum blockchain along with groundbreaking distributed systems tech to enable instant, near-free transfers anywhere in the world.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of connext-nxtp according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 15 days in total, Connext engaged with Spearbit to review the nxtp protocol. In this period of time a total of **214** issues were found.

An architecure diagram can be found in the *Appendix: Architecture* section.

Summary

Project Name	Connext
Repository	nxtp
Commit	32a0370e...74b5c05
Type of Project	Cross Chain Liquidity, Bridge
Audit Timeline	Oct 17th - Nov 4th
Two week fix period	Nov 4th - Nov 18th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	15	15	0
Medium Risk	19	14	5
Low Risk	48	35	13
Gas Optimizations	40	34	6
Informational	92	54	38
Total	**214**	**152**	**62**

5 Findings

5.1 High Risk

5.1.1 `swapInternal()` shouldn't use `msg.sender`

Severity: High Risk

Context:

- BridgeFacet.sol#L337-L369
- BridgeFacet.sol#L659-L750
- AssetLogic.sol#L150-L182
- AssetLogic.sol#L229-L262
- SwapUtils.sol#L798-L826

Description: As reported by the Connext team, the internal stable swap checks if `msg.sender` has sufficient funds on `execute()`. This `msg.sender` is the `relayer` which normally wouldn't have these funds so the swaps would fail. The local funds should come from the Connext diamond itself.

`BridgeFacet.sol`

```
function execute(ExecuteArgs calldata _args) external nonReentrant whenNotPaused returns (bytes32) {
    ...
    (uint256 amountOut, address asset, address local) = _handleExecuteLiquidity(...);
    ...
}
function _handleExecuteLiquidity(...) ... {
    ...
    (uint256 amount, address adopted) = AssetLogic.swapFromLocalAssetIfNeeded(...);
    ...
}
```

`AssetLogic.sol`

```
function swapFromLocalAssetIfNeeded(...) ... {
    ...
    return _swapAsset(...);
}
function _swapAsset(... ) ... {
    ...
    SwapUtils.Swap storage ipool = s.swapStorages[_key];
    if (ipool.exists()) {
        // Swap via the internal pool.
        return  ...   ipool.swapInternal(...) ...
    }
}
```

`SwapUtils.sol`

```
function swapInternal(...) ... {
    IERC20 tokenFrom = self.pooledTokens[tokenIndexFrom];
    require(dx <= tokenFrom.balanceOf(msg.sender), "more than you own");  // msg.sender is the relayer
    ...
}
```

Recommendation: Don't use the balance of `msg.sender`.

Connext: Solved in PR 2120.

Spearbit: Verified.

5.1.2 `MERKLE.insert` does not return the updated tree leaf count

Severity: High Risk

Context: Merkle.sol#L74

Description: The NatSpec comment for `insert` is

```
* @return uint256 Updated count (number of nodes in the tree).
```

But that is not true. If the updated count is $2^k(2n+1)$ where $k, n \in \mathbb{N} \cup 0$ then the return value would be $2n+1$.

Currently, the returned value of `insert` is not being used, otherwise, this could be a bigger issue.

Recommendation: Cache `tree.count + 1` in another variable and return that or do not modify `size` while inserting the new leaf and calculating the new root.

Alternatively, modify the return NatSpec comment to indicate the exact value returned.

Connext: Solved in PR 2211.

Spearbit: Verified. The original function has been removed (with the specific signature). In the new function:

```
function insert(Tree memory tree, bytes32 node) internal pure returns (Tree memory)
```

the returned `Tree` has the correct leaf count.

5.1.3 `PolygonSpokeConnector` or `PolygonHubConnector` can get compromised and DoSed if an address(0) is passed to their constructor for `_mirrorConnector`

Severity: High Risk

Context:

- FxBaseChildTunnel.sol#L38-L41
- FxBaseRootTunnel.sol#L58-L61
- Connector.sol#L119-L121
- PolygonSpokeConnector.sol#L78-L82
- PolygonHubConnector.sol#L51-L55

Description: `PolygonSpokeConnector` (`PolygonHubConnector`) inherits from `SpokeConnector` (`HubConnector`) and `FxBaseChildTunnel` (`FxBaseRootTunnel`). When `PolygonSpokeConnector` (`PolygonHubConnector`) gets deployed and its constructor is called, if `_mirrorConnector == address(0)` then setting the `mirrorConnector` storage variable is skipped:

```
// File: Connector.sol#L118-L121
if (_mirrorConnector != address(0)) {
  _setMirrorConnector(_mirrorConnector);
}
```

Now since the `setFxRootTunnel` (`setFxChildTunnel`) is an unprotected endpoint that is not overridden by `PolygonSpokeConnector` (`PolygonHubConnector`) anyone can call it and assign their own `fxRootTunnel` (`fxChildTunnel`) address (note, `fxRootTunnel` (`fxChildTunnel`) is supposed to correspond to `mirrorConnector` on the destination domain).

Note that the `require` statement in `setFxRootTunnel` (`setFxChildTunnel`) only allows `fxRootTunnel` (`fxChildTunnel`) to be set once (non-zero address value) so afterward even the owner cannot update this value.

If at some later time the owner tries to call `setMirrorConnector` to assign the `mirrorConnector`, since `_setMirrorConnector` is overridden by `PolygonSpokeConnector` (`PolygonHubConnector`) the following will try to execute:

```
// File: PolygonSpokeConnector.sol#L78-L82
function _setMirrorConnector(address _mirrorConnector) internal override {
  super._setMirrorConnector(_mirrorConnector);
  setFxRootTunnel(_mirrorConnector);
}
```

Or for `PolygonHubConnector`:

```
// File: PolygonHubConnector.sol#L51-L55
function _setMirrorConnector(address _mirrorConnector) internal override {
  super._setMirrorConnector(_mirrorConnector);
  setFxChildTunnel(_mirrorConnector);
}
```

But this will revert since `fxRootTunnel` (`fxChildTunnel`) is already set. Thus if the owner of `PolygonSpokeConnector` (`PolygonHubConnector`) does not provide a non-zero address value for `mirrorConnector` upon deployment, a malicious actor can set `fxRootTunnel` which will cause:

1. Rerouting of messages from Polygon to Ethereum to an address decided by the malicious actor (or vice versa for `PolygonHubConnector`).
2. DoSing the `setMirrorConnector` and `setFxRootTunnel` (`fxChildTunnel`) endpoints for the owner.

Recommendation: Make sure either a non-zero address value of `_mirrorConnector` is submitted/enforced in `PolygonSpokeConnector`'s (`PolygonHubConnector`) constructor or override the `setFxRootTunnel` (`setFxChildTunnel`) endpoint to disallow a call from a random user, enabling a select few privileged users to be able to call.

Connext: Only owner can update now. Solved in PR 2387.

Spearbit: Verified.

5.1.4 A malicious `owner` or user with a `Role.Router` role can drain a `router`'s liquidity

Severity: High Risk

Context:

- RoutersFacet.sol#L263-L267
- RoutersFacet.sol#L297
- RoutersFacet.sol#L498
- BridgeFacet.sol#L622

Description: A malicious `owner` or user with `Role.Router` `Role` denominated as *A* in this example, can drain a `router`'s liquidity for a current `router` (a `router` that has already been added to the system and might potentially have added big liquidities to some assets).

Here is how *A* can do it (can also be done atomically):

1. Remove the `router` by calling `removeRouter`.
2. Add the `router` back by calling `setupRouter` and set the `owner` and `recipient` parameters to accounts *A* has access to / control over.
3. Loop over all tokens that the `router` has liquidity and call `removeRouterLiquidityFor` to drain/redirect the funds into accounts *A* has control over.

That means all `routers` would need to put their trust in the `owner` (of this `connext` instance) and any `user` who has a `Role.Router` `Role` with their liquidity. So the setup is not trustless currently.

Recommendation: To remove this trust assumption a redesign is required for how `routers` get integrated into this system. And it starts from here, it would be best to have the function in a form like `function addRouter(IRouter`

router) (renamed setupRouter to addRouter). Where IRouter is an interface that tries to shape some requirements that the router would need to have. A router:

1. Needs to be able to set its own owner or recipient if required. It might not always be required.
2. Needs to be able to sign transfers and bid for those transfers to a sequencer.
3. If approved for using Aave Portal, it might need to be able to call repayAavePortal. But it is not necessary since anyone can call repayAavePortalFor to repay the fees/debts for this router.
4. Can implement calling to addRouterLiquidity to add liquidity. But it is not necessary since anyone can call addRouterLiquidityFor for this router.
5. If the router does not register an account as its owner (also needs to be implemented in this contract for this new redesign) it needs to implement calling removeRouterLiquidity to remove its liquidity. If it does register an owner, implementing calls to removeRouterLiquidity is not necessary since the router's owner can call removeRouterLiquidityFor.

Connext: Solved in PR 2413.

Spearbit: Verified.

5.1.5 Users are forced to accept any slippage on the destination chain

Severity: High Risk

Context: BridgeFacet.sol#L28

Description: The documentation mentioned that there is cancel function on the destination domain that allows users to send the funds back to the origin domain, accepting the loss incurred by slippage from the origin pool. However, this feature is not found in the current codebase.

If the high slippage rate persists continuously on the destination domain, the users will be forced to accept the high slippage rate. Otherwise, their funds will be stuck in Connext.

Recommendation: Implement the cancel function on the destination domain to allow users send funds back to the origin domain if they choose not to accept the high slippage rate on the destination domain.

Connext: Solved in PR 2456.

Spearbit: Verified.

5.1.6 Preservation of msg.sender in ZkSync could break certain trust assumption

Severity: High Risk

Context: Any contract deployed on ZkSync that relies on msg.sender

Description: For ZkSync chain, the msg.sender is preserved for L1 -> L2 calls. One of the rules when pursuing a cross-chain strategy is to never assume that address control between L1 and L2 is always guaranteed. For EOAs (i.e., non-contract accounts), this is generally true that any account that can be accessed on Ethereum will also be accessible on other EVM-based chains. However, this is not always true for contract-based accounts as the same account/wallet address might be owned by different persons on different chains. This might happen if there is a poorly implemented smart contract wallet factory on multiple EVM-based chains that deterministically deploys a wallet based on some user-defined inputs.

For instance, if a smart contract wallet factory deployed on both EVM-based chains uses deterministic CREATE2 which allows users to define its salt when deploying the wallet, Bob might use ABC as salt in Ethereum and Alice might use ABC as salt in Zksync. Both of them will end up getting the same wallet address on two different chains. A similar issue occurred in the Optimism-Wintermute Hack, but the actual incident is more complicated.

Assume that 0xABC is a smart contract wallet owned and deployed by Alice on ZkSync chain. Alice performs a xcall from Ethereum to ZkSync with delegate set to 0xABC address. Thus, on the destination chain (ZkSync), only Alice's smart contract wallet 0xABC is authorized to call functions protected by the onlyDelegate modifier.

Bob (attacker) saw that the `0xABC` address is not owned by anyone on Ethereum. Therefore, he proceeds to take ownership of the `0xABC` by interacting with the wallet factory to deploy a smart contract wallet on the same address on Ethereum. Bob can do so by checking out the inputs that Alice used to create the wallet previously. Thus, Bob can technically make a request from L1 -> L2 to impersonate Alice's wallet (`0xABC`) and bypass the `onlyDelegate` modifier on ZkSync.

Additionally, Bob could make a L1 -> L2 request by calling the ZKSync's `BridgeFacet.xcall` directly to steal Alice's approved funds. Since the xcall relies on `msg.sender`, it will assume that the caller is Alice.

This issue is only specific to ZkSync chain due to the preservation of `msg.sender` for L1 -> L2 calls. For the other chains, the `msg.sender` is not preserved for L1 -> L2 calls and will always point to the L2's AMB forwarding the requests.

Recommendation: Due to the preservation of `msg.sender` for L1 -> L2 calls in ZkSync chain, any contracts deployed on ZkSync chain that relies on `msg.sender` for access control should be aware of the possibility that the same address on Ethereum and ZkSync chains might belong to two different owners.

This issue will only happen if contract-based accounts are involved. It does not affect EOA as only the owner who has the private key of the EOA can control the EOA on any EVM chain. If Connext plans to support ZkSync, it is recommended that only EOA can interact with ZkSync.

Otherwise, add a disclaimer/comment informing the users about the risks and asking them to verify that they have ownership of the address in both Ethereum and ZKSync before proceeding to interact with ZkSync.

Connext: Update from zkSync Team: We have a different address generation schema that would not allow address to be claimed on L2 by an adversary. Even if you deploy same address and same private key it would be different.

Spearbit: Acknowledged, since ZkSync L2 is using a different address generation schema as per the ZkSync team, this attack vector will not be possible.

5.1.7 No way to update a Stable Swap once assigned to a key

Severity: High Risk

Context: SwapAdminFacet.sol#L109

Description: Once a Stable Swap is assigned to a key (the hash of the canonical id and domain for token), it cannot be updated nor deleted. A Swap can be hacked or an improved version may be released which will warrant updating the Swap for a key.

Recommendation: Add a privileged `removeSwap()` function to remove a Swap already assigned to a key. In case a Swap has to be updated, it can be deleted and then initialized.

Connext: Solved in PR 2354.

Spearbit: Verified.

5.1.8 Renouncing ownership or admin role could affect the normal operation of Connext

Severity: High Risk

Context: WatcherClient.sol, WatchManager.sol, Merkle.sol, RootManager.sol, ConnextPriceOracle.sol, UpgradeBeaconController.sol, ProposedOwnableFacet.sol#L276-L285

Description: Consider the following scenarios.

- Instance 1 - Renouncing ownership

All the contracts that extend from `ProposedOwnable` or `ProposedOwnableUpgradeable` inherit a method called `renounceOwnership`. The owner of the contract can use this method to give up their ownership, thereby leaving the contract without an owner. If that were to happen, it would not be possible to perform any owner-specific functionality on that contract anymore.

The following is a summary of the affected contracts and their impact if the ownership has been renounced.

One of the most significant impacts is that Connext's message system cannot recover after a fraud has been resolved since there is no way to unpause and add the connector back to the system.

- Instance 2 - Renouncing admin role

All the contracts that extend from `ProposedOwnableFacet` inherit a method called `revokeRole`.

1. Assume that the Owner has renounced its power and the only Admin remaining used `revokeRole` to renounce its Admin role.
2. Now the contract is left with `Zero Owner & Admin`.
3. All swap operations collect adminFees via `SwapUtils.sol` contract. In absence of any Admin & Owner, these fees will get stuck in the contract with no way to retrieve them. Normally it would have been withdrawn using `withdrawSwapAdminFees|SwapAdminFacet.sol`.
4. This is simply one example, there are multiple other critical functionalities impacted once both Admin and Owner revoke their roles.

Recommendation:

1. Review if the `renounceOwnership` function is required for each of the affected contracts and remove them if they are not needed. Ensure that renouncing ownership in any of the affected contracts will not affect the normal operation of Connext.
2. Revise the `revokeRole` function to ensure that at least one Admin always remains in the system who will be responsible for managing all critical operations in case the owner renounces the role.

Connext: Solved in PR 2412.

Spearbit: Verified.

5.1.9 No way of removing Fraudulent Roots

Severity: High Risk

Context: RootManager.sol#L1

Description: Fraudulent Roots cannot be removed once fraud is detected by the Watcher. This means that Fraud Roots will be propogated to each chain.

Recommendation: Create a new method (callable only by Owner) which can be called when the contract is in paused state to remove the offending roots from the queue.

Connext: Ideas regarding resolution discussed in Pull Conversation linked with this issue.

Spearbit: Verified.

5.1.10 Large number of inbound roots can DOS the `RootManager`

Severity: High Risk

Context: RootManager.sol#L154-L163

Description: It is possible to perform a DOS against the `RootManager` by exploiting the `dequeueVerified` function or `insert` function of the `RootManager.sol`.

The following describes the possible attack path:

1. Assume that a malicious user calls the permissionless `GnosisSpokeConnector.send` function 1000 times (or any number of times that will cause an Out-of-Gas error later) within a single transaction/block on Gnosis causing a large number of Gnosis's outboundRoots to be forwarded to `GnosisHubConnector` on Ethereum.
2. Since the 1000 outboundRoots were sent at the same transaction/block earlier, all of them should arrive at the `GnosisHubConnector` within the same block/transaction on Ethereum.

3. For each of the 1000 outboundRoots received, the `GnosisHubConnector.processMessage` function will be triggered to process it, which will in turn call the `RootManager.aggregate` function to add the received outboundRoot into the `pendingInboundRoots` queue. As a result, 1000 outboundRoots with the same `commitBlock` will be added to the `pendingInboundRoots` queue.
4. After the delay period, the `RootManager.propagate` function will be triggered. The function will call the `dequeueVerified` function to dequeue 1000 verified outboundRoots from the `pendingInboundRoots` queue by looping through the queue. This might result in an Out-of-Gas error and cause a revert.
5. If the above `dequeueVerified` function does not revert, the `RootManager.propagate` function will attempt to insert 1000 verified outboundRoots to the aggregated Merkle tree, which might also result in an Out-of-Gas error and cause a revert.

If the `RootManager.propagate` function reverts when called, the latest aggregated Merkle root cannot be forwarded to the spokes. As a result, none of the messages can be proven and processed on the destination chains.

Note: the processing on the Hub (which is on mainnet) can also become very expensive, as the mainnet usually as a far higher gas cost than the Spoke.

Recommendation: Both solutions should be implemented to sufficiently mitigate this issue.

1. Place restrictions on the SpokeConnector's `send` function. The `send` function should be restricted so that the domain's outbound root will only be forwarded to the hub when the following conditions are met:
   - If the last root sent is different from the current root to be sent
   - After the execution interval has lapsed (e.g. only able to trigger the `send` function once every few minutes) - This is to prevent a malicious user from bypassing the first measure (lastRootSent != outboundRoot) by sending a cheap message to trigger the `dispatch` function to change the outboundRoot to a new one before calling the `send` function
2. Bound the number of outbound roots that can be aggregated per call, and allow them to be processed in batches (if needed)

Connext: Solved in PR 2199 and PR 2545.

Spearbit: Verified.

5.1.11 Missing `mirrorConnector` check on Optimism hub connector

Severity: High Risk

Context: OptimismHubConnector.sol#L69-L121

Description: `processMessageFromRoot()` calls `_processMessage()` to process messages for the "fast" path. But `_processMessage()` can also be called by the AMB in the slow path.

The second call to `_processMessage()` is not necessary (and could double process the message, which luckily is prevented via the `processed[]` mapping). The second call (from the AMB directly to `_processMessage()`) also doesn't properly verify the origin of the message, which might allow the insertion of fraudulent messages.

```
function processMessageFromRoot(...) ... {
    ...
    _processMessage(abi.encode(_data));
    ...
}
function _processMessage(bytes memory _data) internal override {
    // sanity check root length
    require(_data.length == 32, "!length");

    // get root from data
    bytes32 root = bytes32(_data);

    if (!processed[root]) {
        // set root to processed
        processed[root] = true;
        // update the root on the root manager
        IRootManager(ROOT_MANAGER).aggregate(MIRROR_DOMAIN, root);
    } // otherwise root was already sent to root manager
}
```

Recommendation: Remove the second path.

Connext: Solved in PR 2447.

Spearbit: Verified.

5.1.12 Add `_mirrorConnector` to `_sendMessage` of `BaseMultichain`

Severity: High Risk

Context: BaseMultichain.sol#L39-L47

Description: The function `_sendMessage()` of `BaseMultichain` sends the message to the address of the `_amb`. This doesn't seem right as the first parameter is the target contract to interact with according to multichain cross-chain. This should probably be the `_mirrorConnector`.

```
function _sendMessage(address _amb, bytes memory _data) internal {
    Multichain(_amb).anyCall(
        _amb, // Same address on every chain, using AMB as it is immutable
        ...
    );
}
```

Recommendation: Doublecheck the conclusion and change the code to:

```
- function _sendMessage(address _amb, bytes memory _data) ... {
+ function _sendMessage(address _amb, address _mirrorConnector, bytes memory _data) ... {
    Multichain(_amb).anyCall(
-       _amb,
+       _mirrorConnector
        ...
    );
}
```

Connext: Solved in PR 2386.

Spearbit: Verified.

5.1.13 Unauthorized access to change acceptanceDelay

Severity: High Risk

Context: DiamondInit.sol#L35-L40

Description: The `acceptanceDelay` along with `supportedInterfaces[]` can be set by any user without the need of any Authorization once the `init` function of DiamondInit has been called and set. This is happening since caller checks (`LibDiamond.enforceIsContractOwner();`) are missing for these fields.

Since `acceptanceDelay` defines the time post which certain action could be executed, setting a very large value could DOS the system (new owner cannot be set) and setting very low value could make changes without consideration time (Setting/Renounce Admin, Disable whitelisting etc at ProposedOwnableFacet.sol)

Recommendation: Change the function implementation as shown below:

```
LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();
-    ds.supportedInterfaces[type(IERC165).interfaceId] = true;
-    ds.supportedInterfaces[type(IDiamondCut).interfaceId] = true;
-    ds.supportedInterfaces[type(IDiamondLoupe).interfaceId] = true;
-    ds.supportedInterfaces[type(IProposedOwnable).interfaceId] = true;
-    ds.acceptanceDelay = _acceptanceDelay;

    if (!s.initialized) {
        ...
+       ds.supportedInterfaces[type(IERC165).interfaceId] = true;
+       ds.supportedInterfaces[type(IDiamondCut).interfaceId] = true;
+       ds.supportedInterfaces[type(IDiamondLoupe).interfaceId] = true;
+       ds.supportedInterfaces[type(IProposedOwnable).interfaceId] = true;
+       ds.acceptanceDelay = _acceptanceDelay;
        ...
    }
```

Connext: Fixed in PR 2393.

Spearbit: Verified.

5.1.14 Messages destined for ZkSync cannot be processed

Severity: High Risk

Context: ZkSyncHubConnector.sol#L49-L72

Description: For ZkSync chain, L2 to L1 communication is free, but L1 to L2 communication requires a certain amount of ETH to be supplied to cover the base cost of the transaction (including the _l2Value) + layer 2 operator tip.

The `_sendMessage` function of `ZkSyncHubConnector.sol` relies on the `IZkSync(AMB).requestL2Transaction` function to send messages from L1 to L2. However, the `requestL2Transaction` call will always fail because no ETH is supplied to the transaction (`msg.value` is zero).

As a result, the ZkSync's hub connector on Ethereum cannot forward the latest aggregated Merkle root to the ZkSync's spoke connector on ZkSync chain. Thus, any message destined for ZkSync chain cannot be processed since incoming messages cannot be proven without the latest aggregated Merkle root.

```
function _sendMessage(bytes memory _data) internal override {
  // Should always be dispatching the aggregate root
  require(_data.length == 32, "!length");
  // Get the calldata
  bytes memory _calldata = abi.encodeWithSelector(Connector.processMessage.selector, _data);

  // Dispatch message
  // [v2-docs.zksync.io/dev/developer-guides/Bridging/l1-l2.html#structure](https://v2-docs.zksync.io/d
↪  ev/developer-guides/Bridging/l1-l2.html#structure)
  // calling L2 smart contract from L1 Example contract
  // note: msg.value must be passed in and can be retrieved from the AMB view function
↪  `l2TransactionBaseCost`
  // [v2-docs.zksync.io/dev/developer-guides/Bridging/l1-l2.html#using-contract-interface-in-your-proje
↪  ct](https://v2-docs.zksync.io/dev/developer-guides/Bridging/l1-l2.html#using-contract-interface-in-
↪  your-project)
  IZkSync(AMB).requestL2Transaction{value: msg.value}(
    // The address of the L2 contract to call
    mirrorConnector,
    // We pass no ETH with the call
    0,
    // Encoding the calldata for the execute
    _calldata,
    // Ergs limit
    10000,
    // factory dependencies
    new bytes[]0
  );
}
```

Additionally, the ZkSync's hub connector contract needs to be loaded with ETH so that it can forward the appropriate amount of ETH when calling the ZkSync's `requestL2Transaction`. However, it is not possible to do so because no `receive()`, `fallback` or `payable` function has been implemented within the contract and its parent contracts for accepting ETH.

Recommendation: Ensure that ETH can be sent to the contract and appropriate amount of ETH is supplied when sending messages from L1 to L2 via the ZkSync's `requestL2Transaction` call.

Connext: Solved in PR 2407 and PR 2443.

Spearbit: Verified.

5.1.15 Cross-chain messaging via Multichain protocol will fail

Severity: High Risk

Context: BaseMultichain.sol#L39-L47

Description: Multichain v6 is supported by Connext for cross-chain messaging. The `_sendMessage` function of `BaseMultichain.sol` relies on Multichain's `anyCall` for cross-chain messaging.

Per the Anycall V6 documentation, a gas fee for transaction execution needs to be paid either on the source or destination chain when an `anyCall` is called. However, the `anyCall` is called without consideration of the gas fee within the connectors, and thus the `anyCall` will always fail. Since Multichain's hub and spoke connectors are unable to send messages, cross-chain messaging using Multichain within Connext will not work.

```
function _sendMessage(address _amb, bytes memory _data) internal {
    Multichain(_amb).anyCall(
        _amb, // Same address on every chain, using AMB as it is immutable
        _data,
        address(0), // fallback address on origin chain
        MIRROR_CHAIN_ID,
        0 // fee paid on origin chain
    );
}
```

Additionally, for the payment of the execution gas fee, a project can choose to implement either of the following methods:

- Pay on the source chain by depositing the gas fee to the caller contracts.
- Pay on the destination chain by depositing the gas fee to Multichain's anyCall contract at the destination chain.

If Connext decides to pay the gas fee on the source chain, they would need to deposit some ETH to the connector contracts. However, it is not possible because no `receive()`, `fallback` or `payable` function has been implemented within the contracts and their parent contracts for accepting ETH.

Recommendation: Ensure that the gas fee is paid when making an `anyCall`. If the execution gas fee should be paid on the source chain, ensure that the connector contracts can accept ETH.

Connext: Solved in PR 2421.

Spearbit: Verified.

5.2 Medium Risk

5.2.1 `_domainSeparatorV4()` not updated after `name`/`symbol` change

Severity: Medium Risk

Context: BridgeToken.sol#L58-L63, OZERC20.sol#L382-L388, OZERC20.sol#L348-L369, draft-EIP712.sol, EIP712.sol#L69-L75, EIP712.sol#L100-L102

Description: The `BridgeToken` allows updating the `name` and `symbol` of a token. However the `_CACHED_DOMAIN_SEPARATOR` (of EIP712) isn't updated. This means that `permit()`, which uses `_hashTypedDataV4()` and `_CACHED_DOMAIN_SEPARATOR`, still uses the old value. On the other hand `DOMAIN_SEPARATOR()` is updated.

Both and especially their combination can give unexpected results.

`BridgeToken.sol`

```
function setDetails(string calldata _newName, string calldata _newSymbol) external override onlyOwner {
   // careful with naming convention change here
   token.name = _newName;
   token.symbol = _newSymbol;
   emit UpdateDetails(_newName, _newSymbol);
 }
```

`OZERC20.sol`

```
function DOMAIN_SEPARATOR() external view override returns (bytes32) {
    // See {EIP712._buildDomainSeparator}
    return
      keccak256(
          abi.encode(_TYPE_HASH, keccak256(abi.encode(token.name)), _HASHED_VERSION, block.chainid,
↪   address(this))
      );
}
function permit(...) ... {
    ...
    bytes32 _hash = _hashTypedDataV4(_structHash);
    ...
}
```

draft-EIP712.sol

```
import "./EIP712.sol";
```

EIP712.sol

```
function _hashTypedDataV4(bytes32 structHash) internal view virtual returns (bytes32) {
    return ECDSA.toTypedDataHash(_domainSeparatorV4(), structHash);
}
function _domainSeparatorV4() internal view returns (bytes32) {
    if (address(this) == _CACHED_THIS && block.chainid == _CACHED_CHAIN_ID) {
        return _CACHED_DOMAIN_SEPARATOR;
    } else {
        return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
    }
}
```

Recommendation: Make the implementation of `DOMAIN_SEPARATOR()` and `_domainSeparatorV4()` the same. Decide on using cached versions of the domain separator. See also issue "EIP712 domain separator can be cached"

Connext: Solved in PR 2350.

Spearbit: Verified.

5.2.2 diamondCut() allows re-execution of old updates

Severity: Medium Risk

Context: LibDiamond.sol#L112-L115

Description: Once `diamondCut()` is executed, `ds.acceptanceTimes[keccak256(abi.encode(_diamondCut, _init, _calldata))]` is not reset to zero. This means the contract owner can rerun the old updates again without any delay by executing `diamondCut()` function.

Assume the following:

> `diamondCut()` function is executed to update the facet selector with version_2 A bug is found in version_2 and it is rolled back Owner can still execute `diamondCut()` function which will again update the facet selector to version 2 since `ds.acceptanceTimes[keccak256(abi.encode(_diamondCut, _init, _calldata))]` is still valid

Recommendation: Reset `ds.acceptanceTimes[keccak256(abi.encode(_diamondCut, _init, _calldata))]` to zero as shown below:

```
function diamondCut(
    IDiamondCut.FacetCut[] memory _diamondCut,
    address _init,
    bytes memory _calldata
  ) internal {
...
if (ds.facetAddresses.length != 0) {
      uint256 time = ds.acceptanceTimes[keccak256(abi.encode(_diamondCut, _init, _calldata))];
      require(time != 0 && time <= block.timestamp, "LibDiamond: delay not elapsed");
    }
+   ds.acceptanceTimes[keccak256(abi.encode(_diamondCut, _init, _calldata))] = 0;
...
}
```

Connext: Fix in PR 2222.

Spearbit: Verified.

5.2.3 User may not be able to override slippage on destination

Severity: Medium Risk

Context: BridgeFacet.sol#L741-L746

Description: If `BridgeFacet.execute()` is executed before `BridgeFacet.forceUpdateSlippage()`, user won't be able to update slippage on the destination chain. In this case, the slippage specified on the source chain is used. Due to different conditions on these chains, a user may want to specify different slippage values.

This can result in user loss, as a slippage higher than necessary will result the swap trade being sandwiched.

Recommendation: `xcall()` can take different parameters for source and destination slippage:

```
function xcall(
    uint32 _destination,
    address _to,
    address _asset,
    address _delegate,
    uint256 _amount,
-   uint256 _slippage,
+   uint256 _sourceSlippage,
+   uint256 _destinationslippage,
    bytes calldata _callData
) external payable returns (bytes32);
```

Then `TransferInfo params` should be encoded with `_destionationSlippage` and `_sourceSlippage` should be passed separately to `_xcall()`.

Connext: We previously had the slippage implemented as separate values, but decided to change it back for a couple of reasons:

1. Since the messages chains have variable latency, the `destinationChainSlippage` would be hard to predict at the time of `xcall`, especially for authenticated messages. users *can* override this using `forceUpdateSlippage`, but it requires an additional transaction
2. Adding two parameters to the interface clutters it, and we strove to make the `xcall` interface as minimal as possible so the developer experience has the lowest necessary overhead

Spearbit: Acknowledged. If you see this issue after launch, you can also consider adding an argument `forceUpdateSlippageOnDestination` in `xcall()`. Now unless user calls `forceUpdateSlippage()` on destination (identified by some state variable), transfer on destination is now allowed to be completed.

This is a complex solution so only recommended if you anticipate this being an issue in production. It also depends on how quickly the transfers on destination is completed.

5.2.4 Do not rely on token balance to determine when cap is reached

Severity: Medium Risk

Context: BridgeFacet.sol#L492-L495, RoutersFacet.sol#L556-L559

Description: Connext Diamond defines a cap on each token. Any transfer making the total token balance more than the cap is reverted.

```
uint256 custodied = IERC20(_local).balanceOf(address(this)) + _amount;
uint256 cap = s.caps[key];
if (cap > 0 && custodied > cap) {
    revert RoutersFacet__addLiquidityForRouter_capReached();
}
```

Anyone can send tokens to Connext Diamond to artificially increase the custodied amount since it depends on the token balance. This can be an expensive attack but it can become viable if price of a token (including next assets) drops.

Recommendation: Do not rely on token balance to determine custodied amount. Instead, consider maintaining the custodied amount whenever assets are transferred in or out via `AssetLogic`'s `handleIncomingAsset()` and `handleOutgoingAsset()`. With this change, any token transfer in and out of Connext should be done via these functions to properly maintain the custodied amount.

Note: Also consider the issue *"Malicious routers can temporarily DOS the bridge by depositing a large amount of liquidity"* when applying the recommendation.

Connext: Solved in PR 2444.

Spearbit: Verified.

5.2.5 Router recipient can be configured more than once

Severity: Medium Risk

Context: RoutersFacet.sol#L401

Description: The comments from the `setRouterRecipient` function mentioned that the router should only be able to set the recipient once. Otherwise, no problem is solved. However, based on the current implementation, it is possible for the router to set its recipient more than once.

```
File: RoutersFacet.sol
394:   /**
395:    * @notice Sets the designated recipient for a router
396:    * @dev Router should only be able to set this once otherwise if router key compromised,
397:    * no problem is solved since attacker could just update recipient
398:    * @param router Router address to set recipient
399:    * @param recipient Recipient Address to set to router
400:    */
401:   function setRouterRecipient(address router, address recipient) external onlyRouterOwner(router) {
```

Let's assume that during router setup, the `setupRouter` function is being called and the `owner` is set to Alice's first EOA (0x123), and the `recipient` is set to Alice's second EOA (0x456). Although the comment mentioned that the `setRouterRecipient` should only be set once, this is not true because this function will only revert if the `_prevRecipient == recipient`. As long as the new recipient is not the same as the previous recipient, the function will happily accept the new recipient.

Therefore, if the router's signing key is compromised by Bob (attacker), he could call the `setRouterRecipient` function to change the new recipient to his personal EOA and drain the funds within the router.

The `setRouterRecipient` function is protected by `onlyRouterOwner` modifier. Since Bob's has the compromised router's signing key, he will be able to pass this validation check.

```
File: RoutersFacet.sol
157:   /**
158:    * @notice Asserts caller is the router owner (if set) or the router itself
159:    */
160:   modifier onlyRouterOwner(address _router) {
161:     address owner = s.routerPermissionInfo.routerOwners[_router];
162:     if (!((owner == address(0) && msg.sender == _router) || owner == msg.sender))
163:       revert RoutersFacet__onlyRouterOwner_notRouterOwner();
164:     _;
165:   }
```

The second validation is at Line 404, which checks if the new recipient is not the same as the previous recipient. The `recipient` variable is set to Bob's EOA wallet, while `_prevRecipient` variable is set to Alice's second EOA (0x456). Therefore, the condition at Line 404 is `False`, and it will not revert. So Bob successfully set the recipient to his EOA at Line 407.

```
File: RoutersFacet.sol
401:   function setRouterRecipient(address router, address recipient) external onlyRouterOwner(router) {
402:     // Check recipient is changing
403:     address _prevRecipient = s.routerPermissionInfo.routerRecipients[router];
404:     if (_prevRecipient == recipient) revert RoutersFacet__setRouterRecipient_notNewRecipient();
405:
406:     // Set new recipient
407:     s.routerPermissionInfo.routerRecipients[router] = recipient;
```

Per the Github discussion, the motivation for such a design is the following:

> If a routers signing key is compromised, the attacker could drain the liquidity stored on the contract and send it to any specified address. This effectively means the key is in control of all unused liquidity on chain, which prevents router operators from adding large amounts of liquidity directly to the contract. Routers should be able to delegate the safe withdrawal address of any unused liquidity, creating a separation of concerns between router key and liquidity safety.

In summary, the team is trying to create a separation of concerns between router key and liquidity safety. With the current implementation, there is no security benefit of segregating the router owner role and recipient role unless the router owner has been burned (e.g. set to address zero). Because once the router's signing key is compromised, the attacker can change the recipient anyway. The security benefits of separation of concerns will only be achieved if the recipient can truly be set only once.

Recommendation: If the intention is to only allow the router owner to set the recipient once and not allow them to change it afterward, then the code should be as follows.

```
function setRouterRecipient(address router, address recipient) external onlyRouterOwner(router) {
    // Check recipient is changing
    address _prevRecipient = s.routerPermissionInfo.routerRecipients[router];
+    if (_prevRecipient != address(0)) revert RoutersFacet__setRouterRecipient_RecipientAlreadySet();
-    if (_prevRecipient == recipient) revert RoutersFacet__setRouterRecipient_notNewRecipient();

    // Set new recipient
    s.routerPermissionInfo.routerRecipients[router] = recipient;

    // Emit event
    emit RouterRecipientSet(router, _prevRecipient, recipient);
}
```

The above implementation will always revert as long as the recipient address has already been set.

Connext: Solved in PR 2413.

Spearbit: Verified.

5.2.6 The set of tokens in an internal swap pool cannot be updated

Severity: Medium Risk

Context: SwapAdminFacet.sol#L109-L119

Description: Once a swap is initialized by the `owner` or an `admin` (indexed by the `key` parameter) the `_pooledTokens` or the set of tokens used in this stable swap pool cannot be updated.

Now the `s.swapStorages[_key]` pools are used in other facets for assets that have the hash of their canonical token id and canonical domain equal to `_key`, for example when we need to swap between a local and adopted asset or when a user provides liquidity or interact with other `external` endpoints of `StableSwapFacet`.

If the submitted set of tokens to this pool `_pooledTokens` beside the local and adopted token corresponding to `_key` include some other bad/malicious tokens, users' funds can be at risk in the pool in question. If this happens, we need to pause the protocol, push an update, and `initializeSwap` again.

Recommendation: Document the procedure on how `_pooledTokens` is selected and submitted to `initializeSwap` to lower the risk of introducing potentially bad/malicious tokens into the system.

Connext: Solved in PR 2354.

Spearbit: Verified.

5.2.7 An incorrect decimal supplied to `initializeSwap` for a token cannot be corrected

Severity: Medium Risk

Context: SwapAdminFacet.sol#L109-L119

Description: Once a swap is initialized by the `owner` or an `admin` (indexed by the `key` parameter) the decimal precisions per tokens, and therefore `tokenPrecisionMultipliers` cannot be changed. If the supplied `decimals` also include a wrong value, it would cause incorrect calculation when a swap is being made and currently there is no update mechanism for `tokenPrecisionMultipliers` nor a mechanism for removing the `swapStorages[_key]`.

Recommendation: Add a restricted endpoint for updating the `tokenPrecisionMultipliers` for a token in an internal swap pool in case a mistake has been made when providing the decimals.

Connext: We will remove the swap in case we made a mistake when initializing the swap pool, because we have to update token `balances` and `adminFees` in swap object when updating `tokenPrecisionMultipliers`. Solved in PR 2354.

Spearbit: Verified.

5.2.8 Presence of `delegate` not enforced

Severity: Medium Risk

Context: BridgeFacet.sol#L395-L414, BridgeFacet.sol#L563-L567, BridgeFacet.sol#L337-L369

Description: A `delegate` address on the destination chain can be used to fix stuck transactions by changing the slippage limits and by re-executing transactions. However, the presence of a `delegate` address isn't checked in `_xcall()`.

Note: set to medium risk because tokens could get lost

```
function forceUpdateSlippage(TransferInfo calldata _params, uint256 _slippage) external
↪  onlyDelegate(_params) {
    ...
}
function execute(ExecuteArgs calldata _args) external nonReentrant whenNotPaused returns (bytes32) {
    (bytes32 transferId, DestinationTransferStatus status) = _executeSanityChecks(_args);
    ...
}
function _executeSanityChecks(ExecuteArgs calldata _args) private view returns (bytes32,
↪  DestinationTransferStatus) {
    // If the sender is not approved relayer, revert
    if (!s.approvedRelayers[msg.sender] && msg.sender != _args.params.delegate) {
      revert BridgeFacet__execute_unapprovedSender();
    }
}
```

Recommendation: Enforce the presence of a `delegate` address in `_xcall()`. Or at least document the behavior explicitly.

Connext: Yes, it's always going to be necessary to have a delegate if you want to have a strategy for handling destination-side slippage conditions being unfavorable. If you don't have one you are taking on the bet that unfavorable slippage conditions won't be maintained indefinitely. Some groups see having any EOA or multisig that can impact these parameters as a huge no-no, and requiring one to be defined would be a nonstarter for them. So we allow to not provide an option on that front, even if it is more risky and could lead to funds being frozen in transit.

We should add more clarity around this in the documentation though.

Spearbit: Acknowledged.

5.2.9 Relayer could lose funds

Severity: Medium Risk

Context: BridgeFacet.sol#L828-L835

Description: The `xReceive` function on the receiver side can contain unreliable code which Relayer is unaware of. In the future, more relayers will participate in completing the transaction.

Consider the following scenario:

1. Say that `Relayer A` executes the `xReceive` function on receiver side.
2. In the `xReceive` function, a call to withdraw function in a foreign contract is made where `Relayer A` is holding some balance.
3. If this foreign contract is checking `tx.origin` (say deposit/withdrawal were done via third party), then `Relayer A`'s funds will be withdrawn without his permission (since tx.origin will be the Relayer).

Recommendation: Relayers should be advised to use an untouched wallet address so that foreign code interaction cannot harm them.

Connext: To be documented, relayers EOA should be single-purpose.

Spearbit: Acknowledged.

5.2.10 `TypedMemView.sameType` does not use the correct right shift value to compare two `bytes29`s

Severity: Medium Risk

Context: TypedMemView.sol#L402

Description: The function `sameType` should shift `2 x 12 + 3` bytes to access the `type flag` (TTTTTTTTTT) when comparing it to `0`. This is due to the fact that when using `bytes29` type in bitwise operations and also comparisons to `0`, a paramater of type `bytes29` is zero padded from the right so that it fits into a `uint256` under the hood.

```
0x TTTTTTTTTT AAAAAAAAAAAAAAAAAAAAAAAA LLLLLLLLLLLLLLLLLLLLLLLL 00 00 00
```

Currently, `sameType` only shifts the `xor`ed value `2 x 12` bytes so the comparison compares the `type` flag and the `3` leading bytes of `memory address` in the packing specified below:

```
// First 5 bytes are a type flag.
// - ff_ffff_fffe is reserved for unknown type.
// - ff_ffff_ffff is reserved for invalid types/errors.
// next 12 are memory address
// next 12 are len
// bottom 3 bytes are empty
```

The function is not used in the codebase but can pose an important issue if incorporated into the project in the future.

```
function sameType(bytes29 left, bytes29 right) internal pure returns (bool) {
    return (left ^ right) >> (2 * TWELVE_BYTES) == 0;
}
```

Recommendation: Change `sameType()` to take the zero padding into account:

```
uint256 private constant TWENTY_SEVEN_BYTES = 8 * 27;
...
function sameType(bytes29 left, bytes29 right) internal pure returns (bool) {
    return (left ^ right) >> TWENTY_SEVEN_BYTES == 0;
}
```

See summa-tx/memview-sol/pull/10 for a solution.

We also recommend leaving a header comment indicating the source of libraries/contracts used.

Connext: Solved in PR 2394.

Spearbit: Verified.

5.2.11 Incorrect formula for the scaled amplification coefficient in NatSpec comments

Severity: Medium Risk

Context:

- StableSwapFacet.sol#L70
- SwapAdminFacet.sol#L103
- StableSwap.sol#L64
- StableSwap.sol#L143
- AmplificationUtils.sol#L24
- SwapUtils.sol#L62
- SwapUtils.sol#L248
- SwapUtils.sol#L304

- SwapUtilsExternal.sol#L54
- SwapUtilsExternal.sol#L127
- SwapUtilsExternal.sol#L285
- SwapUtilsExternal.sol#L341

Description: In the context above, the scaled amplification coefficient a is described by the formula $An(n-1)$ where A is the actual amplification coefficient in the stable swap invariant equation for n tokens.

```
* @param a the amplification coefficient * n * (n - 1) ...
```

The actual adjusted/scaled amplification coefficient would need to be An^{n-1} and not $An(n-1)$, otherwise, most of the calculations done when swapping between 2 tokens in a pool with more than 2 tokens would be wrong. For the special case of $n = 2$, those values are actually equal $2^{2-1} = 2 = 2 \cdot 1$. So for swaps or pools that involve only 2 tokens, the issue in the comment is not so critical. But if the number of tokens are more than 2, then we need to make sure we calculate and feed the right parameter to `AppStorage.swapStorages.{initial, future}A`

Recommendation: Fix the NatSpec comments to:

```
* @param a the amplification coefficient * n ** (n - 1) ...
```

And make sure any values of a for init functions or as an input to swap functions are scaled using n^{n-1} from the amplification coefficient.

Looks like the original source for `SwapUtils` is from here: SwapUtils.sol

which is a Solidity rewrite of the curve finance pool template contracts in Vyper: pool-templates

It is recommend adding header NatSpec comments and mentioning authors or original/modified sources for all files including the ones mentioned in this issue.

Connext: Solved in PR 2353.

Spearbit: Verified.

5.2.12 `RootManager.propagate` does not operate in a fail-safe manner

Severity: Medium Risk

Context: RootManager.sol#L147-L173

Description: A bridge failure on one of the supported chains will cause the entire messaging network to break down.

When the `RootManager.propagate` function is called, it will loop through the hub connector of all six chains (Arbitrum, Gnosis, Multichain, Optimism, Polygon, ZKSync) and attempt to send over the latest aggregated root by making a function call to the respective chain's AMB contract. There is a tight dependency between the chain's AMB and hub connector.

The problem is that if one of the function calls to the chain's AMB contract reverts (e.g. one of the bridges is paused), the entire `RootManager.propagate` function will revert, and the messaging network will stop working until someone figure out the problem and manually removes the problematic hub connector.

As Connext grows, the number of chains supported will increase, and the risk of this issue occurring will also increase.

Recommendation: The `RootManager.propagate` function should operate in a fail-safe manner (e.g. using try-catch or address.call). Chain's AMB contracts are considered external third-party and beyond Connext control. Thus, the `RootManager.propagate` function should not assume that function calls to these third-party bridge contracts will always succeed and will not revert.

Connext: Solved in PR 2430.

Spearbit: Verified.

5.2.13 Arborist once whitelisted cannot be removed

Severity: Medium Risk

Context: Merkle.sol#L93-L97

Description: Arborist has the power to write over the Merkle root. In case Arborist starts misbehaving (compromised or security issue) then there is no way to remove this Arborist from the whitelist.

Recommendation: Revise the `setArborist` function to tweak arborists whitelisting status

```
function setArborist(address newArborist, bool status) external onlyOwner {
    if (newArborist == address(0)) revert MerkleTreeManager__setArborist_zeroAddress();
    if (arborists[newArborist]) revert MerkleTreeManager__setArborist_alreadyArborist();
    arborists[newArborist] = status;
  }
```

Connext: Solved in PR 2426.

Spearbit: Verified.

5.2.14 WatcherManager is not set correctly

Severity: Medium Risk

Context: WatcherClient.sol#L36-L39

Description: The `setWatcherManager` function missed to actually update the `watcherManager`, instead it is just emitting an event mentioning that Watcher Manager is updated when it is not.

This could become a problem once new modules are added/revised in `WatcherManager` contract and `WatcherClient` wants to use this upgraded `WatcherManager`. `WatcherClient` will be forced to use the outdated WatcherManager contract code.

Recommendation: Revise the `setWatcherManager` function as shown below:

```
function setWatcherManager(address _watcherManager) external onlyOwner {
    require(_watcherManager != address(watcherManager), "already watcher manager");
+   watcherManager = WatcherManager(_watcherManager);
    emit WatcherManagerChanged(_watcherManager);
  }
```

Connext: Fixed in PR 2432.

Spearbit: Verified.

5.2.15 Check `__GAP`s

Severity: Medium Risk

Context: LPToken.sol#L16, OwnerPausableUpgradeable.sol#L16, StableSwap.sol#L39, Merkle.sol#L37, ProposedOwnable.sol#L193

Description: All `__GAP`s have the same size, while the different contracts have a different number of storage variables. If the `__GAP` size isn't logical it is more difficult to maintain the code.

Note: set to a risk rating of medium because the probably of something going wrong with future upgrades is low to medium, and the impact of mistakes would be medium to high.

```
LPToken.sol:                 uint256[49] private __GAP; // should probably be 50
OwnerPausableUpgradeable.sol: uint256[49] private __GAP; // should probably be 50
StableSwap.sol:              uint256[49] private __GAP; // should probably be 48
Merkle.sol:                  uint256[49] private __GAP; // should probably be 48
ProposedOwnable.sol:         uint256[49] private __GAP; // should probably be 47
```

Recommendation: Check and update the `__GAP`s of all the contracts. Perhaps the `__GAP` of `ProposedOwnableUpgradeable` should be moved to `ProposedOwnable`.

Connext: Solved in PR 2342.

Spearbit: Verified.

5.2.16 Message can be delivered out of order

Severity: Medium Risk

Context: SpokeConnector.sol#L330-L376

Description: Messages can be delivered out of order on the spoke. Anyone can call the permissionless `proveAndProcess` to process the messages in any order they want. A malicious user can force the spoke to process messages in a way that is beneficial to them (e.g., front-run).

Recommendation: Ensure that messages are always processed in order. Otherwise, if the risk is deemed acceptable, update the documentation to highlight that messages can be delivered out of order and that it is up to Connext users to protect themselves against this.

Connext: This problem reminds me of MEV around sequencing when building blocks. I think enforcing the message order would create additional complex problems. What if a certain message couldn't be processed, because the stableswap slippage had moved beyond the user-set slippage tolerance? In that case, all following messages would be beholden to that one user coming back to make sure they update their slippage tolerance (creating an isAlive assumption). What if that user doesn't want to increase their slippage tolerance - they'd rather wait out market conditions on their own time? They hold up the entire queue, a queue of quite possibly unrelated txs.

It's true that a relayer could choose to relay certain txs ahead of others. They could be biased towards their own transfers, and do some MEV/frontrunning this way... but any frontrunning that the relayer could do, anyone could do, simply by looking at the transaction in the mempool and frontrunning or sandwiching it for profit. Even if the relayer used flashbots, any observer could examine our subgraphs to see inbound txs. The relayer could definitely execute such MEV a bit more comfortably (by guaranteeing their frontrun goes through), but that slight benefit/advantage of their role in the system is not worth the UX tradeoffs involved in serializing here - which would almost guarantee occasional spots of traffic/congestion.

Spearbit: Acknowledged.

5.2.17 Extra checks in `_verifySender()` of `GnosisBase`

Severity: Medium Risk

Context: GnosisBase.sol#L16-L19

Description: According to the Gnosis bridge documentation the source chain id should also be checked using `messageSourceChainId()`. This is because in the future the same arbitrary message bridge contract could handle requests from different chains.

If a malicious actor would be able to have access to the contract at `mirrorConnector` on a to-be-supported chain that is not the `MIRROR_DOMAIN`, they can send an arbitrary root to this mainnet/L1 hub connector which the connector would mark it as coming from the `MIRROR_DOMAIN`. So the attacker can spoof/forge function calls and asset transfers by creating a payload root and using this along with their access to `mirrorConnector` on chain to send a cross-chain `processMessage` to the Gnosis hub connector and after they can use their payload root and proofs to forge/spoof transfers on the L1 chain.

Although it is unlikely that any other party could add a contract with the same address as `_amb` on another chain, it is safer to add additional checks.

```
function _verifySender(address _amb, address _expected) internal view returns (bool) {
  require(msg.sender == _amb, "!bridge");
  return GnosisAmb(_amb).messageSender() == _expected;
}
```

Recommendation: In function `_verifySender()` add a check to verify `messageSourceChainId() == MIRROR_DOMAIN`. Note: this will probably require adding an extra parameter to `_verifySender()`.

Connext: The `chainId != domain`, so has to be stored separately. Going to use `sourceChainId()` instead of the `messageSourceChainId()` as there is no documentation of what the `bytes32` should represent instead of `uint256`.

Solved in PR.

Spearbit: Verified.

5.2.18 Absence of Minimum delayBlocks

Severity: Medium Risk

Context: RootManager.sol#L102-106, SpokeConnector.sol#L218-L221

Description: Owner can accidentally set `delayBlocks` as `0` (or a very small delay block) which will collapse the whole fraud protection mechanism. Since there is no check for minimum delay before setting a new delay value so even a low value will be accepted by `setDelayBlocks` function

```
function setDelayBlocks(uint256 _delayBlocks) public onlyOwner {
    require(_delayBlocks != delayBlocks, "!delayBlocks");
    emit DelayBlocksUpdated(_delayBlocks, delayBlocks);
    delayBlocks = _delayBlocks;
  }
```

Recommendation: Introduce a variable `minDelay` which tells the minimum possible delay allowed by the contract. Any attempt to change delay value using `setDelayBlocks` function should ensure that new delay is larger/equal to `minDelay`

Connext: We could add a minimum for when delayBlocks is not 0, but that minimum will vary by chain / block time, so that minimum HAS to be configurable. We could add a separate configuration endpoint and make it so it takes 72 hours to change the delay blocks minimum, but that feels more like DAO functionality/responsibility. For that reason, going with "acknowledged". At the very least, users can visibly check what the delayBlocks are set to on-chain to make sure it's reasonable.

Spearbit: Acknowledged

5.2.19 Add extra `0` checks in `verifyAggregateRoot()` and `proveMessageRoot()`

Severity: Medium Risk

Context: SpokeConnector.sol#L403-L422, SpokeConnector.sol#L456-L481

Description: The functions `verifyAggregateRoot()` and `proveMessageRoot()` verify and confirm roots. A root value of `0` is a special case. If this value would be allowed, then the functions could allow invalid roots to be passed. Currently the functions `verifyAggregateRoot()` and `proveMessageRoot()` don't explicitly verify the roots are not `0`.

```
function verifyAggregateRoot(bytes32 _aggregateRoot) internal {
    if (provenAggregateRoots[_aggregateRoot]) {
        return;
    }
    ... // do several verifications
    provenAggregateRoots[_aggregateRoot] = true;
   ...
}
function proveMessageRoot(...) ... {
    if (provenMessageRoots[_messageRoot]) {
        return;
    }
    ... // do several verifications
    provenMessageRoots[_messageRoot] = true;
}
```

Recommendation: As an extra safety precaution do the following:

- In function `verifyAggregateRoot()` check `_aggregateRoot != 0`.
- In `proveMessageRoot()` check `_messageRoot != 0`.

Connext: Solved in PR 2442.

No check added in `proveMessageRoot()` because `calculateMessageRoot()` never results in `0`.

Spearbit: Verified.

5.3 Low Risk

5.3.1 `_removeAssetId()` should also clear `custodied`

Severity: Low Risk

Context: TokenFacet.sol#L483-L534

Description: In one of the fixes in PR 2530, `_removeAssetId()` doesn't clear `custodied` as it is assumed to be `0`.

```
function _removeAssetId(...) ... {
    // NOTE: custodied will always be 0 at this point
}
```

However `custodied` isn't always `0`. Suppose `cap` & `custodied` have a value (`!=0`), then `_setLiquidityCap()` is called to set the `cap` to `0`. The function doesn't reset the `custodied` value so it will stay at `!=0`.

Recommendation: Clear `custodied`.

Connext: The balance check of the `_removeAssetId` would cover this case. If the balance of the contract is `0` for the canonical asset, the entirety of the assets must be unbridged (meaning custodied would be `0`). Additionally, when `_setLiquidityCap` is called with an enforceable (nonzero) value, it is always set to the balance of the contract.

Spearbit: Acknowledged

5.3.2 Remove liquidity while paused

Severity: Low Risk

Context: StableSwapFacet.sol#L312-L360, StableSwap.sol#L394-L446

Description: The function `removeLiquidity()` in `StableSwapFacet.sol` has a `whenNotPaused` modifier, while the comment shows `Liquidity can always be removed, even when the pool is paused.`. On the other hand function `removeLiquidity()` in `StableSwap.sol` doesn't have this modifier.

`StableSwapFacet.sol#L394-L446`

```
// Liquidity can always be removed, even when the pool is paused.
function removeLiquidity(...) external ... nonReentrant whenNotPaused ... { ... }
function removeLiquidityOneToken(...) external ... nonReentrant whenNotPaused  ... { ... }
function removeLiquidityImbalance(...) external ... nonReentrant whenNotPaused  ... { ... }
```

`StableSwap.sol#L394-L446`

```
// Liquidity can always be removed, even when the pool is paused.
function removeLiquidity(...) external ... nonReentrant ... { ... }
function removeLiquidityOneToken(...) external ... nonReentrant whenNotPaused  ... { ... }
function removeLiquidityImbalance(...) external ... nonReentrant whenNotPaused  ... { ... }
```

Recommendation: Doublecheck if removal of liquidity is allowed while paused.

Connext: Solved in PR 2354.

Spearbit: Verified, all functions have `whenNotPaused` now.

5.3.3 Relayers can frontrun each other's calls to `BridgeFacet.execute`

Severity: Low Risk

Context:

- BridgeFacet.sol#L337
- BridgeFacet.sol#L366
- BridgeFacet.sol#L565-L567
- BridgeFacet.sol#L381

Description: Relayers can front run each other's calls to `BridgeFacet.execute`. Currently, there is no on-chain mechanism to track how many fees should be allocated to each relayer. All the transfer bump fees are funneled into one address `s.relayerFeeVault`.

Recommendation: If for example, off-chain agents keep track of the following event emitted by `execute` and use that to calculate the distribution of funds per relayer that can motivate the relayers to front run each other.

```
emit Executed(transferId, _args.params.to, asset, _args, local, amount, msg.sender);
```

Note: Only the `delegate` and allowed/approved relayers can call `execute`.

Connext: At the moment, fee disbursements for relayers are handled in a trusted manner. We're working with Gelato to figure out a better long term solution here.

More generally, if we have *adversarial* relayer networks (i.e. relayers are not beholden to a shared set of offchain incentives to penalize frontrunning), this issue will always exist. This is why we've chosen to only work with one decentralized relayer network for now.

Spearbit: Acknowledged.

5.3.4 `OptimismHubConnector.processMessageFromRoot` emits `MessageProcessed` for already processed messages

Severity: Low Risk

Context:

- OptimismHubConnector.sol#L119-L120
- OptimismHubConnector.sol#L76-L81

Description: Calls to `processMessageFromRoot` with an already processed `_data` still emit `MessageProcessed`. This might cause issues for off-chain agents like relayers monitoring this event.

Recommendation: We recommend moving `MessageProcessed` inside `_processMessage`'s `if` block:

```
function _processMessage(bytes memory _data) internal override {
  // sanity check root length
  require(_data.length == 32, "!length");

  // get root from data
  bytes32 root = bytes32(_data);

  if (!processed[root]) {
    // set root to processed
    processed[root] = true;
    // update the root on the root manager
    IRootManager(ROOT_MANAGER).aggregate(MIRROR_DOMAIN, root);
    emit MessageProcessed(_data, msg.sender); // <--- added line
  } // otherwise root was already sent to root manager
}
```

Note, if we inline `_processMessage` inside `processMessageFromRoot` we can avoid encoding and then decoding the `root`.

Connext: Solved in PR 2447.

Spearbit: Verified.

5.3.5 Add max cap for domains

Severity: Low Risk

Context: DomainIndexer.sol#L99

Description: Currently there isn't any cap on the maximum amount of `domains` which system can support. If the size of the domains and connectors grow, at some point due to out-of-gas errors in `updateHashes` function, both `addDomain` and `removeDomain` could DOS.

Recommendation: Place a cap for `MAX_DOMAINS`.

```
function addDomain(uint32 _domain, address _connector) internal {
...
// MAX_DOMAINS could be defined globally as `uint256 public constant MAX_DOMAINS = {value};`
require(domains.length < MAX_DOMAINS, "DomainIndexer at capacity");
...
}
```

Connext: Fixed in PR 2209.

Spearbit: Verified.

5.3.6 In certain scenarios calls to `xcall...` or `addRouterLiquidity...` can be DoSed

Severity: Low Risk

Context:

- TokenFacet.sol#L219-L222
- BridgeFacet.sol#L489-L497
- RoutersFacet.sol#L554-L561

Description: The `owner` or an `admin` can frontrun (or it can be by an accident) a call that:

- A router has made on a canonical domain of a canonical token to supply that token as liquidity OR
- A user has made to `xcall...` supplying a canonical token on its canonical domain.

The frontrunning call would set the `cap` to a low number (calling `updateLiquidityCap`). This would cause the calls mentioned in the bullet list to fail due to the checks against `IERC20(_local).balanceOf(address(this))`.

Recommendation: This is a low-risk issue due to the severity and the amount of privilege required to execute. But a delayed update mechanism can be applied for `updateLiquidityCap` or perhaps a governance body can vote for setting new caps.

Connext: This seems to me to be a pure griefing vector and in my opinion, can be ignored if that is the case.

Spearbit: Acknowledged.

5.3.7 Missing a check against `address(0)` in `ConnextPriceOracle`'s constructor

Severity: Low Risk

Context: ConnextPriceOracle.sol#L78-L81

Description: When `ConnextPriceOracle` is deployed an `address _wrapped` is passed to its constructor. The current codebase does not check whether the passed `_wrapped` can be an `address(0)` or not.

Recommendation: Add a check to make sure the `_wrapped` value provided cannot be `address(0)`.

Connext: Solved in PR 2371.

Spearbit: Verified.

5.3.8 `_executeCalldata()` can revert if insufficient gas is supplied

Severity: Low Risk

Context: BridgeFacet.sol#L785-L840

Description: The function `_executeCalldata()` contains the statement `gasleft() - 10_000`. This statement can revert if the available gas is less than `10_000`. Perhaps this is the expected behaviour.

Note: From the Tangerine Whistle fork only a maximum 63/64 of the available gas is sent to contract being called. Therefore, 1/64th is left for the calling contract.

```
function _executeCalldata(...) ... {
    ...
    (success, returnData) = ExcessivelySafeCall.excessivelySafeCall(_params.to,  gasleft() - 10_000,
↪   ... );
    ...
}
```

Recommendation: Double check the gas requirements. Consider giving a more clear error message if insuffient gas is supplied.

Connext: Solved in PR 2519.

Spearbit: Verified.

5.3.9 Be aware of precompiles

Severity: Low Risk

Context: BridgeFacet.sol#L785-L840

Description: The external calls by `_executeCalldata()` could call a precompile. Different chains have creative precompile implementations, so this could in theory pose problems. For example precompile 4 copies memory: what-s-the-identity-0x4-precompile

Note: precompiles link to dedicated pieces of code written in Rust or Go that can be called from the EVM. Here are a few links for documentation on different chains: moonbeam precompiles, astar precompiles

```
function _executeCalldata(...) ... {
    ...
    (success, returnData) = ExcessivelySafeCall.excessivelySafeCall( _params.to, ...);
    } else {
         returnData = IXReceiver(_params.to).xReceive(...);
    }
    ...
}
```

Recommendation: When deploying to new chains, verify that available precompiles can't be abused.

Connext: Added a comment in PR 2446 to reflect this, but this is a risk.

Spearbit: Acknowledged.

5.3.10 Upgrade to solidity 0.8.17

Severity: Low Risk

Context: nxtp contracts

Description: Solidity 0.8.17 released a bugfix where the optimizer could incorrectly remove storage writes if the code fit a certain pattern (see this security alert). This bug was introduced in 0.8.13. Since Connext is using the legacy code generation pipeline, i.e., compiling without the `via-IR` flag, the current code is not at risk. This is because assembly blocks dont write to storage.

However, if this changes and Connext compiles through `via-IR` code generation, the code is more likely to be affected. One reason to use this code generation pipeline could be to enable gas optimizations not available in legacy code generation.

Recommendation: Consider upgrading to solidity 0.8.17.

Connext: Fixed in PR 2436.

Spearbit: Verified.

5.3.11 Add domain check in `setupAssetWithDeployedRepresentation()`

Severity: Low Risk

Context: TokenFacet.sol#L192-L204

Description: The function `setupAssetWithDeployedRepresentation()` links a new `_representation` asset. However this should not be done on the canonical domain. So good to check this to prevent potential mistakes.

```
function setupAssetWithDeployedRepresentation(...) ... {
    bytes32 key = _enrollAdoptedAndLocalAssets(_adoptedAssetId, _representation, _stableSwapPool,
↪   _canonical);
    ...
  }
```

Recommendation: Add a check for `(_canonical.domain != s.domain)`.

Connext: Solved in PR 2298.

Spearbit: Verified.

5.3.12 If an adopted token and its canonical live on the same domain the cap for the custodied amount is applied for each of those tokens

Severity: Low Risk

Context: RoutersFacet.sol#L554-L561

Description: If `_local` is an adopted asset that lives on its canonical's original chain, then we are comparing the to-be-updated balance of `this` contract (`custodied`) with `s.caps[key]`. That means we are also comparing the balance of an adopted asset with the property above with the `cap`.

For example, if *A* is the canonical token and *B* the adopted, then `cap = s.caps[key]` is used to cap the custodied amount in this contract for both of those tokens. So if the `cap` is 1000, the contract can have a balance of 1000 *A* and 1000 *B*, which is twice the amount meant to be capped.

This is true basically for any approved asset with the above properties. When the `owner` or the `admin` calls `setupAsset`:

```
// File: https://github.com/connext/nxtp/blob/32a0370edc917cc45c231565591740ff274b5c05/packages/deploym
↪   ents/contracts/contracts/core/connext/facets/TokenFacet.sol#L164-L172
function setupAsset(
  TokenId calldata _canonical,
  uint8 _canonicalDecimals,
  string memory _representationName,
  string memory _representationSymbol,
  address _adoptedAssetId,
  address _stableSwapPool,
  uint256 _cap
) external onlyOwnerOrAdmin returns (address _local) {
```

such that `_canonical.domain == s.domain` and `_adoptedAssetId != 0`, then this asset has the property in question.

Recommendation: Extra attention must be paid to tokens with such properties and perhaps set the cap equal to half the amount one would have set otherwise.

Connext: Local on the canonical domain should always be the adopted assets, this assertion is represented in PR 2455.

Spearbit: Verified.

5.3.13 There are no checks/constraints against the `_representation` provided to `setupAssetWithDeployedRepresentation`

Severity: Low Risk

Context: TokenFacet.sol#L192-L198

Description: `setupAssetWithDeployedRepresentation` is similar to `setupAsset` in terms of functionality, except it does not deploy a `representation` token if necessary. It actually uses the `_representation` address given as the representation token. The `_representation` parameter given does not have any checks in terms of functionality compared to when `setupAsset` which deploys a new `BridgeToken` instance:

```
// File: packages\deployments\contracts\contracts\core\connext\facets\TokenFacet.sol#L399
_token = address(new BridgeToken(_decimals, _name, _symbol));
```

Basically, `representation` needs to implement `IBridgeToken` (`mint`, `burn`, `setDetails`, ...) and some sort of `IERC20`. Otherwise, if a function from `IBridgeToken` is not implemented or if it does not have `IERC20` functionality, it can cause failure/reverts in some functions in this codebase.

Another thing that is important is that the `decimals` for `_representation` should be equal to the `decimals` precision of the canonical token. And that `_representation` should not be able to update/change its `decimals`.

Also, this opens an opportunity for a bad owner or admin to provide a malicious `_representation` to this function. This does not have to be a malicious act, it can also happen by mistake from for example an admin.

Additionally the Connext Diamond must have the "right" to `mint()` and `burn()` the tokens.

Recommendation: We can require a call to a specific endpoint of `_representation` to return a magic value that would indicate that it supports some specific interfaces. This would only be a measure against possible benign mistakes that could happen by the `admin`s or the `owner` when calling `setupAssetWithDeployedRepresentation()` with a wrong value for `_representation`.

Also verify tokens can be `mint()`ed and `burn()`t.

Connext: In the following PR, NatSpec comments have been added for devs to warn them about this issue: PR 2472.

Also in `_enrollAdoptedAndLocalAssets`, `mint`ing and `burn`ing of a `IBridgeToken` asset has been tested when not on a canonical domain.

Spearbit: Verified.

5.3.14 In `dequeueVerified` when no verified items are found in the queue `last == first - 1`

Severity: Low Risk

Context: Queue.sol#L80

Description: The comment in `dequeueVerified` mentions that when no verified items are found in the queue, then `last == first`. But this is not true since the loop condition is `last >= first` and the loop only terminates (not considering the `break`) when `last == first - 1`.

It is important to correct this incorrect statement in the comment, since a dev/user can by mistake take this statement as true and modify/use the code with this incorrect assumption in mind.

Recommendation: Update the comment so that:

```
- first == last
+ last == first - 1
```

Connext: Solved in PR 2228.

Spearbit: Verified.

5.3.15 Dirty bytes in `_loc` and `_len` can override other values when packing a typed memory view in `unsafeBuildUnchecked`

Severity: Low Risk

Context:

- TypedMemView.sol#L323-L324
- TypedMemView.sol#L329-L330

Description: For a `TypeedMemView`, the location and the length are supposed to occupy `12` bytes (`uint96`), but the type used for these values for the input parameters for `unsafeBuildUnchecked` is `uint256`. This would allow those values to carry dirty bytes and when the following calculations are performed:

```
newView := shl(96, or(newView, _type)) // insert type
newView := shl(96, or(newView, _loc)) // insert loc
newView := shl(24, or(newView, _len)) // empty bottom 3 bytes
```

`_loc` can potentially manipulate the `type` section of the view and `_len` can potentially manipulate both the `_loc` and the `_type` section.

Recommendation: Either restrict the input parameter types or perform cleanup via masking in assembly to make sure those values would not be able to override other values in the packed view.

Connext: Solved in PR 2475.

Spearbit: Verified.

5.3.16 To use `sha2`, `hash160` and `hash256` of `TypedMemView` the hard-coded precompile addresses would need to be checked to make sure they return the corresponding hash values.

Severity: Low Risk

Context:

- TypedMemView.sol#L646
- TypedMemView.sol#L668-L669
- TypedMemView.sol#L685-L686

Description: `sha2`, `hash160` and `hash256` assume that the precompile contracts at `address(2)` and `address(3)` calculate and return the `sha256` and `ripemd160` hashes of the provided memory chunks. These assumptions depend on the chain that the project is going to be deployed on.

Recommendation: To use `sha2`, `hash160` and `hash256` of `TypedMemView` the hard-coded precompile addresses would need to be checked to make sure they are available on the specific chain and that they return the corresponding hash values.

For example `zkSync` recently added the precompile contracts for `sha245` and `keccak256` and the compiler `zksolc` inlines the precompiled addresses for those endpoints (`0x0002`, `0x8010`).

Connext: Functions removed in PR 2472.

Spearbit: Verified.

5.3.17 `sha2`, `hash160` and `hash256` of `TypedMemView.sha2` do not clear the memory after calculating the hash

Severity: Low Risk

Context:

- TypedMemView.sol#L646
- TypedMemView.sol#L662
- TypedMemView.sol#L679

Description: When a call to the precompile contract at `address(2)` (or at `address(3)`) is made, the returned value is placed at the slot pointed by the free memory pointer and then placed on the stack. The free memory pointer is not incremented to account for this used memory position nor the code tries to clean this memory slot of `32` bytes. Therefore after a call to `sha2`, `hash160` or `hash256`, we would end up with dirty bytes.

Recommendation: Either increment the free memory pointer by `0x20` to account for the dirty bytes or clean the dirty bytes. Cleaning would be cheaper since memory might not get expanded for the next memory operation in the codebase.

As a side note, it might be also best to call `sha2` function `sha256`.

Connext: Removed in PR 2474.

Spearbit: Verified.

5.3.18 Fee on transfer token support

Severity: Low Risk

Context: SwapUtils.sol#L902-L905

Description: It seems that only the `addLiquidity` function is currently supporting the fee on transfer token. All operations like swapping are prohibiting the fee on transfer token.

Note: The SwapUtilsExternal.sol contract allow fee on transfer token and as per product team, this is expected from this token

Recommendation: If fee on transfer token are not expected while adding liquidity then revert if token used is placing any fee on transfer

```
require(amounts[i] == token.balanceOf(address(this)) - beforeBalance, "Fee on transfer token not
↪  supported");
```

Connext: Solved in PR 2217.

SwapUtils is for internal StableSwap (implemented in StableSwapFacet). That doesn't support fee tokens. SwapUtilsExternal is for external StableSwap contract (StableSwap.sol) and it can support fee tokens as well. because all swap and add/remove liquidity will be executed via transferFrom function, and we can get real amount that we received.

Spearbit: Why not use `AssetLogic.handleIncomingAsset()` which already has this check? Reduces code duplication.

The fix should also be applied for all instances of `safeTransferFrom`:

```
core-libraries/SwapUtils.sol:712:      tokenFrom.safeTransferFrom(msg.sender, address(this), dx);
core-libraries/SwapUtils.sol:776:      tokenFrom.safeTransferFrom(msg.sender, address(this), dx);
core-libraries/SwapUtils.sol:902:        token.safeTransferFrom(msg.sender, address(this), amounts[i]);
core-libraries/SwapUtilsExternal.sol:749:      tokenFrom.safeTransferFrom(msg.sender, address(this),
↪  dx);
core-libraries/SwapUtilsExternal.sol:813:      tokenFrom.safeTransferFrom(msg.sender, address(this),
↪  dx);
core-libraries/SwapUtilsExternal.sol:868:        token.safeTransferFrom(msg.sender, address(this),
↪  amounts[i]);
```

can also use `handleOutgoingAsset()` for all outgoing transfers

```
core-libraries/SwapUtils.sol:732:    self.pooledTokens[tokenIndexTo].safeTransfer(msg.sender, dy);
core-libraries/SwapUtils.sol:782:    self.pooledTokens[tokenIndexTo].safeTransfer(msg.sender, dy);
core-libraries/SwapUtils.sol:988:      self.pooledTokens[i].safeTransfer(msg.sender, amounts[i]);
core-libraries/SwapUtils.sol:1034:    self.pooledTokens[tokenIndex].safeTransfer(msg.sender, dy);
core-libraries/SwapUtils.sol:1116:      self.pooledTokens[i].safeTransfer(msg.sender, amounts[i]);
core-libraries/SwapUtils.sol:1140:        token.safeTransfer(to, balance);
core-libraries/SwapUtilsExternal.sol:769:    self.pooledTokens[tokenIndexTo].safeTransfer(msg.sender,
↪  dy);
core-libraries/SwapUtilsExternal.sol:819:    self.pooledTokens[tokenIndexTo].safeTransfer(msg.sender,
↪  dy);
core-libraries/SwapUtilsExternal.sol:954:      self.pooledTokens[i].safeTransfer(msg.sender,
↪  amounts[i]);
core-libraries/SwapUtilsExternal.sol:1000:    self.pooledTokens[tokenIndex].safeTransfer(msg.sender,
↪  dy);
core-libraries/SwapUtilsExternal.sol:1082:      self.pooledTokens[i].safeTransfer(msg.sender,
↪  amounts[i]);
core-libraries/SwapUtilsExternal.sol:1106:        token.safeTransfer(to, balance);
```

5.3.19 Fee on transfer tokens can stuck the transaction

Severity: Low Risk

Context: AssetLogic.sol#L79

Description: Consider the following scenario.

1. Assume User has made a `xcall` with `amount A` of `token X` with `calldata C1`. Since their was no fee while transferring funds, transfer was a success.
2. Now, before this amount can be transferred on the destination domain,`token X` introduced a fee on transfer.
3. Relayer now executes this transaction on destination domain via `_handleExecuteTransaction` function on `BridgeFacet.sol#L756`.
4. This transfers the `amount A` of `token X` to destination domain but since now the fee on this token has been introduced, destination domain receives amount `A-delta`.
5. This calldata is called on destination domain but the amount passed is `A` instead of `A-delta` so if the `IXReceiver` has amount check then it will fail because it will now expect `A amount` when it really got `A-delta amount`.

Recommendation: Documentation should be updated in order to guide users about such scenarios so that they can design the IXReceiver contracts accordingly (handling calldata failures gracefully)

Connext: The only way a token can get used in the protocol is if it goes through the approval process. Vetting of tokens for use cross-chain will be done responsibly to ensure we aren't using fee-on-transfer tokens, since they aren't compatible with the protocol anyway: AssetLogic.sol#L55

If a token is upgradeable, and it upgrades its contracts to add the fee-on-transfer functionality, such an upgrade would break the flow at the step in the line above. For now, if the token is upgradeable, it will need to be carefully monitored by admins to handle unapproving the asset if it becomes a fee-on-transfer token.

Spearbit: Acknowedged.

5.3.20 Initial Liquidity Provider can trick the system

Severity: Low Risk

Context: SwapUtils.sol#L923-L924

Description: Since there is no cap on the amount which initial depositor can deposit, an attacker can trick the system in bypassing admin fees for other users by selling liquidity at half admin fees. Consider the following scenario.

1. `User A` provides the first liquidity of a huge amount.
2. Since there aren't any fees from initial liquidity, admin fees are not collected from `User A`.
3. Now `User A` can sell his liquidity to other users with half admin fees.
4. Other users can mint larger liquidity due to lesser fees and `User A` also get benefit of `adminFees/2`.

Recommendation: Add a cap until which admin fees are revoked for initial depositor. Once cap is breached, user will need to pay the admin fees.

Connext: The initial depositor will be owner or verified account. Protection from a malicious admin, is not worth implementing. At this point there are a myriad of ways malicious admins could take advantage of users, and this adds some extra complexity to address an edgecase that is relatively unlikely.

Spearbit: Acknowledged.

5.3.21 Ensure non-zero local asset in `_xcall()`

Severity: Low Risk

Context: BridgeFacet.sol#L481

Description: Local asset fetched in `_xcall()` is not verified to be a non-zero address. In case, if token mappings are not updated correctly and to future-proof from later changes, it's better to revert if a zero address local asset is fetched.

```
local = _getLocalAsset(key, canonical.id, canonical.domain);
```

Recommendation: Consider reverting if `local == address(0)`.

Connext: Solved in PR 2471.

Spearbit: Verified.

5.3.22 Use `ExcessivelySafeCall` to call `xReceive()`

Severity: Low Risk

Context: BridgeFacet.sol#L828

Description: For reconciled transfers, `ExcessivelySafeCall.excessivelySafeCall()` is used to call `xReceive()`. This is done to avoid copying large amount of return data in memory. This same attack vector exists for non-reconciled transfers, however in this case a usual function call is made for `xReceive()`.

However, in case non-reconciled calls fail due to this error, they can always be retried after reconciliation.

Recommendation: Consider replacing the direct function call `xReceive()` with `excessivelySafeCall()`. Store the success of the call in `success` and revert if it is `false`.

Connext: Solved in PR 2470.

Spearbit: Verified.

5.3.23 A `router`'s liquidity might get trapped if the `router` is removed

Severity: Low Risk

Context:

- RoutersFacet.sol#L297
- RoutersFacet.sol#L518
- RoutersFacet.sol#L498
- RoutersFacet.sol#L581

Description: If the `owner` or a `user` with `Role.Router` `Role` removes a `router` that does not implement calling `removeRouterLiquidity` or `removeRouterLiquidityFor`, then any liquidity remaining in the contract for the removed `router` cannot be transferred back to the `router`.

Recommendation: To be able to remove the `router`'s liquidity, the `owner` or a `user` with `Role.Router` `Role` needs to add the `router` back by calling `setupRouter` and make sure the `owner` parameter provided to `setupRouter` is not `address(0)`. Then the `router`'s `owner` would need to call `removeRouterLiquidity` for this `router`. Then the `owner` or a `user` with `Role.Router` `Role` of this `connext` instance can remove the `router` again.

A workaround would be when removing a `router`, it would instead be marked as `removed` and not clear the `router`'s `owner`. So even after marking it as removed the `router`'s `owner` can call `removeRouterLiquidityFor`.

Connext: Solved in PR 2413.

Spearbit: Verified.

5.3.24 In-flight transfers by the `relayer` can be reverted when `setMaxRoutersPerTransfer` is called beforehand by a lower number

Severity: Low Risk

Context:

- RoutersFacet.sol#L338
- BridgeFacet.sol#L586

Description: For in-flight transfers where an approved `sequencer` has picked and signed an x number of routers for a transfer, from the time a relayer or another 3rd party grabs this `ExecuteArgs _args` to the time this party submits it to the destination domain by calling `execute` on a `connext` instance, the `owner` or an `admin` can call `setMaxRoutersPerTransfer` with a number lower than x on purpose or not. And this would cause the call to `execute` to revert with `BridgeFacet__execute_maxRoutersExceeded`.

Recommendation: If this act is malicious, for example, a malicious `admin` doing this for a while to DoS the `execute` endpoint for fast liquidity routes, the `owner` would need to detect and remove the said `admin`.

A solution to prevent this type of DoS or accidental acts could be by adding delays between consecutive calls to `setMaxRoutersPerTransfer` for only `admin`s. Maybe the `owner` should be able to bypass this delay, since if we can't trust the `owner` there are more serious things that we should be worried about.

The delay range between when a sequencer has the `_args` ready and the time a relayer calling `execute` should be measured and documented for the involved parties.

Connext: Likely the max routers per transfer will be controlled by a DAO entity in the future, meaning the configuration changes will be subject to a wait period and can be anticipated. A failed `execute` would revert fairly early and likely could grief very little ETH from the relayer. The sequencer can just re-auction the transfer after the failed `execute` in short order.

Spearbit: Acknowledged.

5.3.25 All the privileged users that can call `withdrawSwapAdminFees` would need to trust each other

Severity: Low Risk

Context: SwapAdminFacet.sol#L183

Description: The `owner` needs to trust all the `admins` and also all `admins` need to trust each other. Since any `admin` can call `withdrawSwapAdminFees` endpoint to withdraw all the pool's admin fees into their account.

Recommendation: There is no accounting per different `admin`s and all admin funds are pooled into the same place. We need to make sure they are all aware of this.

Also document how the `admin`s are picked and how the trust is established between them and the `owner`.

To avoid this issue, Connext could also implement a governance voting structure where `admin`s would vote on how the funds are distributed.

Connext: Safe to assume that all admins are trusted as much as owner in their privileges on-chain for the time being (this will change under DAO management, of course). Minus upgrading, but withdrawing these fees is one of several ways the owner trusts admins to not be malicious. Others include updating fees, setting the relayer vault to a different address, removing assets, etc.

Spearbit: Acknowledged.

5.3.26 The supplied `_a` to `initializeSwap` cannot be directly updated but only ramped

Severity: Low Risk

Context:

- SwapAdminFacet.sol#L109-L119
- SwapAdminFacet.sol#L216

Description: Once a swap is initialized by the `owner` or an `admin` (indexed by the `key` parameter) the supplied `_a` (the scaled amplification coefficient, An^{n-1}) to `initializeSwap` cannot be directly updated but only ramped. The `owner` or the `admin` can still call `rampA` to update `_a`, but it will take some time for it to reach the desired value.

This is mostly important if by mistake an incorrect value for `_a` is provided to `initializeSwap`.

Recommendation: We need to make sure this is documented and the users/devs/routers are all aware of this issue.

Connext: Documented in PR 2469.

Spearbit: Verified.

5.3.27 Inconsistent behavior when xcall with a non-existent `_params.to`

Severity: Low Risk

Context: BridgeFacet.sol#L802

Description: A xcall with a non-existent `_params.to` behaves differently depending on the path taken.

1. Fast Liquidity Path - Use `IXReceiver(_params.to).xReceive`. The `_executeCalldata` function will revert if `_params.to` is non-existent. Which technically means that the execution has failed.
2. Slow Path - Use `ExcessivelySafeCall.excessivelySafeCall`. This function uses the low-level `call`, which will not revert and will return true if the `_params.to` is non-existent. The `_executeCalldata` function will return with `success` set to `True`, which means the execution has succeeded.

Recommendation: For consistency, if the `_params.to` points to a non-existent contract in the Slow Path, skip the `ExcessivelySafeCall.excessivelySafeCall` call, return, and set `success` as `False`.

Connext: It the call `params.to` is empty (meaning `address(0)`), then the `xcall` will revert: BridgeFacet.sol#L454

If the `execute` receives an empty `to`, then we can assume the execution is invalid. An invalid execution (via fast path) can only result in a loss of funds for those involved, since there is no hope of reimbursement. An invalid execution for slow path can only mean that fraud was committed (there was no corresponding xcall on sending chain) - concerns around that kind of invalid execution should be directed to the security model for the messaging layer.

Spearbit: Acknowledged.

5.3.28 The `lpToken` cloned in `initializeSwap` cannot be updated

Severity: Low Risk

Context:

- SwapAdminFacet.sol#L109-L119
- SwapAdminFacet.sol#L157

Description: Once a swap is initialized by the `owner` or an `admin` (indexed by the `key` parameter) an `LPToken lpToken` is created by cloning the provided `lpTokenTargetAddress` to the `initializeSwap` endpoint. There is no restriction on `lpTokenTargetAddress` except that it would need to be of `LPToken` like, but it can be malicious under the hood or have some security vulnerabilities, so it can not be trusted.

Recommendation: The procedure of picking and submitting the `lpTokenTargetAddress` to the `initializeSwap` would need to be thorough and documented. Also, one can avoid cloning `lpTokenTargetAddress` and instead create `LPToken` from a known implementation in the codebase that has been audited.

Connext: Some checks have been added. It does still allow admins to supply malicious values, but it would be much more easily detectable (as it wouldn't have to be checked against previously used values, you could simply search for the update event). Solved in PR 2389.

Spearbit: Verified.

5.3.29 Lack of zero check

Severity: Low Risk

Context: BridgeFacet.sol#L216, BridgeFacet.sol#L247, TokenFacet.sol#L272

Description: Consider the following scenarions.

- Instance 1 - `BridgeFacet.addSequencer`

The `addSequencer` function of `BridgeFacet.sol` does not check that the sequencer address is not zero before adding them.

```
function addSequencer(address _sequencer) external onlyOwnerOrAdmin {
    if (s.approvedSequencers[_sequencer]) revert BridgeFacet__addSequencer_alreadyApproved();
    s.approvedSequencers[_sequencer] = true;

    emit SequencerAdded(_sequencer, msg.sender);
}
```

If there is a mistake during initialization or upgrade, and set `s.approvedSequencers[0] = true`, anyone might be able to craft a payload to execute on the bridge because the attacker can bypass the following validation within the `execute` function.

```
if (!s.approvedSequencers[_args.sequencer]) {
    revert BridgeFacet__execute_notSupportedSequencer();
}
```

- Instance 2 - `BridgeFacet.enrollRemoteRouter`

The `enrollRemoteRouter` function of `BridgeFacet.sol` does not check that the domain or router address is not zero before adding them.

```
function enrollRemoteRouter(uint32 _domain, bytes32 _router) external onlyOwnerOrAdmin {
    // Make sure we aren't setting the current domain as the connextion.
    if (_domain == s.domain) {
        revert BridgeFacet__addRemote_invalidDomain();
    }
    s.remotes[_domain] = _router;
    emit RemoteAdded(_domain, TypeCasts.bytes32ToAddress(_router), msg.sender);
}
```

- Instance 3 - `TokenFacet._enrollAdoptedAndLocalAssets`

The `_enrollAdoptedAndLocalAssets` function of `TokenFacet.sol` does not check that the `_canonical.domain` and `_canonical.id` are not zero before adding them.

```
function _enrollAdoptedAndLocalAssets(
    address _adopted,
    address _local,
    address _stableSwapPool,
    TokenId calldata _canonical
) internal returns (bytes32 _key) {
    // Get the key
    _key = AssetLogic.calculateCanonicalHash(_canonical.id, _canonical.domain);

    // Get true adopted
    address adopted = _adopted == address(0) ? _local : _adopted;

    // Sanity check: needs approval
    if (s.approvedAssets[_key]) revert TokenFacet__addAssetId_alreadyAdded();

    // Update approved assets mapping
    s.approvedAssets[_key] = true;

    // Update the adopted mapping using convention of local == adopted iff (_adooted == address(0))
    s.adoptedToCanonical[adopted].domain = _canonical.domain;
    s.adoptedToCanonical[adopted].id = _canonical.id;
```

These two values are used for generating the key to determine if a particular asset has been approved. Additionally, zero value is treated as a null check within the `AssetLogic.getCanonicalTokenId` function:

```
// Check to see if candidate is an adopted asset.
_canonical = s.adoptedToCanonical[_candidate];
    if (_canonical.domain != 0) {
    // Candidate is an adopted asset, return canonical info.
    return _canonical;
}
```

Recommendation: Check that sequencer address is not zero before adding it.

```
function addSequencer(address _sequencer) external onlyOwnerOrAdmin {
+    require(_sequencer != address(0), "sequencer with zero address");

    if (s.approvedSequencers[_sequencer]) revert BridgeFacet__addSequencer_alreadyApproved();
    s.approvedSequencers[_sequencer] = true;

    emit SequencerAdded(_sequencer, msg.sender);
}
```

Check that the domain or router address is not zero before adding it.

```
function enrollRemoteRouter(uint32 _domain, bytes32 _router) external onlyOwnerOrAdmin {
+   require(_domain != 0, "zero domain");
+   require(_router != 0, "router with zero address");

    // Make sure we aren't setting the current domain as the connextion.
    if (_domain == s.domain) {
        revert BridgeFacet__addRemote_invalidDomain();
    }
    s.remotes[_domain] = _router;
    emit RemoteAdded(_domain, TypeCasts.bytes32ToAddress(_router), msg.sender);
}
```

Check that the `_canonical.domain` and `_canonical.id` are not zero before adding them.

```
function _enrollAdoptedAndLocalAssets(
    address _adopted,
    address _local,
    address _stableSwapPool,
    TokenId calldata _canonical
) internal returns (bytes32 _key) {
+   require (_canonical.domain != 0, "_canonical.domain is zero");
+   require (_canonical.id != 0, "_canonical.id is zero");

    // Get the key
    _key = AssetLogic.calculateCanonicalHash(_canonical.id, _canonical.domain);

    // Get true adopted
    address adopted = _adopted == address(0) ? _local : _adopted;

    // Sanity check: needs approval
    if (s.approvedAssets[_key]) revert TokenFacet__addAssetId_alreadyAdded();

    // Update approved assets mapping
    s.approvedAssets[_key] = true;

    // Update the adopted mapping using convention of local == adopted iff (_adooted == address(0))
    s.adoptedToCanonical[adopted].domain = _canonical.domain;
    s.adoptedToCanonical[adopted].id = _canonical.id;
```

Connext: Solved in PR 2297.

Spearbit: Verified.

5.3.30 When initializing Connext bridge make sure `_xAppConnectionManager` domain matches the one provided to the initialization function for the bridgee

Severity: Low Risk

Context:

- DiamondInit.sol#L59
- DiamondInit.sol#L62
- SpokeConnector.sol#L250-L252
- SpokeConnector.sol#L282-L296
- BridgeFacet.sol#L238-L240

Description: The only contract that implements `IConnectorManager` fully is `SpokeConnector` (through inheriting `ConnectorManager` and overriding `localDomain`):

```
// File: SpokeConnector.sol
function localDomain() external view override returns (uint32) {
  return DOMAIN;
}
```

So a `SpokeConnector` or a `IConnectorManager` has its own concept of the local domain (the domain that it lives / is deployed on). And this domain is used when we are hashing messages and inserting them into the `SpokeConnector`'s merkle tree:

```
// File: SpokeConnector.sol
bytes memory _message = Message.formatMessage(
  DOMAIN,
  bytes32(uint256(uint160(msg.sender))),
  _nonce,
  _destinationDomain,
  _recipientAddress,
  _messageBody
);

// Insert the hashed message into the Merkle tree.
bytes32 _messageHash = keccak256(_message);

// Returns the root calculated after insertion of message, needed for events for
// watchers
(bytes32 _root, uint256 _count) = MERKLE.insert(_messageHash);
```

We need to make sure that this local domain matches the `_domain` provided to this `init` function. Otherwise, the message hashes that are inserted into `SpokeConnector`'s merkle tree would have 2 different origin domains linked to them. One from `SpokeConnector` in this message hash and one from connext's `s.domain = _domain` which is used in calculating the transfer id hash.

The same issue applies to `setXAppConnectionManager`.

Recommendation: We suggest comparing these 2 domains to avoid potential pitfalls for cross-chain transfers.

```
// File: DiamondInit.sol
error DiamondInit__init_domainsDontMatch();
...

// Connext
s.LIQUIDITY_FEE_NUMERATOR = 9995;
s.maxRoutersPerTransfer = 5;

if( _domain != IConnectorManager(_xAppConnectionManager).localDomain()) {
   revert DiamondInit__init_domainsDontMatch();
}

s.domain = _domain;
s.xAppConnectionManager = IConnectorManager(_xAppConnectionManager);
```

The same check can be applied to `setXAppConnectionManager`:

```
// File: BridgeFacet.sol
function setXAppConnectionManager(address _xAppConnectionManager) external onlyOwnerOrAdmin {
  if( s.domain != IConnectorManager(_xAppConnectionManager).localDomain()) {
    revert DiamondInit__init_domainsDontMatch();
  }
  s.xAppConnectionManager = IConnectorManager(_xAppConnectionManager);
}
```

Connext: Solved in PR 2465.

Spearbit: Verified.

5.3.31 The stable swap pools used in Connext are incompatible with tokens with varying decimals

Severity: Low Risk

Context:

- TokenFacet.sol#L164
- TokenFacet.sol#L399
- SwapAdminFacet.sol#L140
- SwapAdminFacet.sol#L143
- SwapAdminFacet.sol#L169
- StableSwap.sol#L100
- SwapUtilsExternal.sol#L407
- SwapUtils.sol#L370

Description: The stable swap functionality used in Connext calculates and stores for each token in a pool, the token's precision relative to the pool's precision. The token precision calculation uses the token's decimals. And since this precision is only set once, for a token that can have its decimals changed at a later time in the future, the precision used might not be always accurate in the future. And so in the event of a token decimal change, the swap calculations involving this token would be inaccurate.

For exmpale in `_xp(...)`:

```
function _xp(uint256[] memory balances, uint256[] memory precisionMultipliers)
  internal
  pure
  returns (uint256[] memory)
{
  uint256 numTokens = balances.length;
  require(numTokens == precisionMultipliers.length, "mismatch multipliers");
  uint256[] memory xp = new uint256[]numTokens;
  for (uint256 i; i < numTokens; ) {
    xp[i] = balances[i] * precisionMultipliers[i];

    unchecked {
      ++i;
    }
  }
  return xp;
}
```

We are multiplying in `xp[i] = balances[i] * precisionMultipliers[i];` and cannot use division for tokens that have higher precision than the pool's default precision.

Recommendation: Document the procedure of what tokens are allowed to be included in stable swap pools and what actions would Connext take when a decimal change happens.

Leave a comment for users/devs whether only fixed decimal tokens are allowed or not in the protocol.

Connext: The intention with the current construction is to manually update assets whose decimals change (by removing them and setting them up again)

Added a comment for devs in PR 2453.

Spearbit: This issue needs off-chain monitoring and enforcement. The Connext team would need to monitor all tokens used in the protocol for this issue.

5.3.32 When Connext reaches the cap allowed custodied, race conditions can be created

Severity: Low Risk

Context:

- BridgeFacet.sol#L489-L497
- RoutersFacet.sol#L554-L561

Description: When `IERC20(local).balanceOf(address(this))` is close to `s.caps[key]` (this can be relative/subjective) for a canonical token on its canonical domain, a race condition gets created where users might try to frontrun each others calls to `xcall` or `xcallIntoLocal` to be included in a cross chain transfer.

This race condition is actually between all users and all liquidity `router`s. Since there is a same type of check when `router`s try to add liquidity.

```
uint256 custodied = IERC20(_local).balanceOf(address(this)) + _amount;
uint256 cap = s.caps[key];
if (cap > 0 && custodied > cap) {
  revert RoutersFacet__addLiquidityForRouter_capReached();
}
```

Recommendation: Connext team would need to monitor the custodied amount for each allowed asset and adjust the caps accordingly before such race conditions are created.

Connext: Doesn't fix this core race issue between users, but at the very least we can document the issue and provide resources for making sure there's plenty of liquidity headroom for a user's transfer.

Spearbit: Acknowledged.

5.3.33 Prevent `sequencer`s from signing multiple routes for the same cross-chain transfer

Severity: Low Risk

Context: BridgeFacet.sol#L622

Description: Liquidity routers only sign the hash of `(transferId, pathLength)` combo. This means that each `router` does not have a say in:

1. The ordering of `routers` provided/signed by the `sequencer`.
2. What other `routers` are used in the sequence.

If a sequencer signs 2 different routes (set of routers) for a cross-chain transfer, a relayer can decide which set of routers to use and provide to `BridgeFacet.execute` to make sure the liquidity from a specific set of routers' balances is used (also the same possibility if 2 different sequencers sign 2 different routes for a cross-chain transfer).

Recommendation: The implementation of the `sequencer` or a set of `sequencers` should prevent signing a transfer with a different set of routers.

Also, it would be great to document how the on/off-chain bidding of routers/sequencers work. Are approved routers/sequencers contracts/EOAs or a mix?

Connext: A check for this is implemented offchain, but doing this in an onchain-verifiable way is out of scope for this iteration of the network.

Spearbit: Acknowledged.

5.3.34 Well-funded malicious actors can DOS the bridge

Severity: Low Risk

Context: BridgeFacet.sol#L28

Description: A malicious actor (e.g. a well-funded cross-chain messaging competitor) can DOS the bridge cheaply. Assume Ethereum <-> Polygon bridge and the liquidity cap is set to 1m for USDC.

1. Using a slow transfer to avoid router liquidity fees, Bob (attacker) transferred 1m USDC from Ethereum to Polygon. 1m USDC will be locked on Connext's Bridge. Since the liquidity cap for USDC is filled, no one will be able to transfer any USDC from Ethereum to Polygon unless someone transfers POS-USDC from Polygon to Ethereum to reduce the amount of USDC held by the bridge.
2. On the destination chain, nextUSDC (local bridge asset) will be swapped to POS-USDC (adopted asset). The swap will incur low slippage because it is a stablewap. Assume that Bob will receive 999,900 POS-USDC back on Polygon. A few hundred or thousand loss is probably nothing for a determined competitor that wants to harm the reputation of Connext.
3. Bob bridged back the 999,900 POS-USDC using Polygon's Native POS bridge. Bob will receive 999,900 USDC in his wallet in Ethereum after 30 minutes. It is a 1-1 exchange using a native bridge, so no loss is incurred here.
4. Whenever the liquidity cap for USDC gets reduced on Connext's Bridge, Bob will repeat the same trick to keep the bridge in an locked state.
5. If Bob is well-funded enough, he could perform this against all Connext's bridges linked to other chains for popular assets (e.g. USDC), and normal users will have issues transferring popular assets when using xcall.

Recommendation: Consider implementing off-chain components that automatically rebalance the assets by deploying them elsewhere if a certain asset on the bridge is reaching the liquidity cap. However, this might need to be done carefully, otherwise, some tokens cannot bridge back.

Alternatively, prepare a continuity plan to deal with this issue manually (e.g. methods for reducing the liquidity on the bridge) if it happens.

Connext: Introducing transfers to rebalance the network is something we are familiar with how to do manually, as it is a requirement of our current system as well so that is the approach we will take. These conditions are introduced by the nature of a cap, but limiting the amount at stake while the protocol is new is higher priority to us.

Spearbit: Acknowledged.

5.3.35 `calculateTokenAmount` is not checking whether `amounts` provided has the same length as `balances`

Severity: Low Risk

Context:

- SwapUtils.sol#L636-L645
- StableSwap.sol#L280
- StableSwapFacet.sol#L167

Description: There is no check to make sure `amounts.length == balances.length` in `calculateTokenAmount`:

```
function calculateTokenAmount(
  Swap storage self,
  uint256[] calldata amounts,
  bool deposit
) internal view returns (uint256) {
  uint256 a = _getAPrecise(self);
  uint256[] memory balances = self.balances;
  ...
```

There are 2 bad cases:

1. `amounts.length > balances.length`, in this case, we have provided extra data which will be ignored silently and might cause miscalculation on or off chain.
2. `amounts.length < balances.length`, the loop in `calculateTokenAmount` would/should revert becasue of an index-out-of-bound error. In this case, we might spend more gas than necessary compared to if we had performed the check and reverted early.

Recommendation: Check for `amounts.length == balances.length` and thus avoid the pitfalls mentioned above.

```
function calculateTokenAmount(
  Swap storage self,
  uint256[] calldata amounts,
  bool deposit
) internal view returns (uint256) {
  uint256[] memory balances = self.balances;
  uint256 numBalances = balances.length;
  if(amounts.length != numBalances) {
    revert SomeCustomError(); // <-- this error needs to be defined
  }
  ...
```

Connext: Solved in PR 2388.

Spearbit: Verified.

5.3.36 Rearrange an expression in `_calculateSwapInv` to avoid underflows

Severity: Low Risk

Context: SwapUtils.sol#L581

Description: In the following expression used in `SwapUtils_calculateSwapInv`, if `xp[tokenIndexFrom] = x + 1` the expression would underflow and revert. We can arrange the expression to avoid reverting in this edge case.

```
dx = x - xp[tokenIndexFrom] + 1;
```

Recommendation: If the following rearrangement is used, the edge case discussed would not revert.

```
dx = (x + 1) - xp[tokenIndexFrom];
```

Note: We need to check whether the ranges of asset balances can be high such that `x+1` would overflow.

Connext: Solved in PR 2382.

Spearbit: Verified.

5.3.37 The pre-image of `DIAMOND_STORAGE_POSITION`'s storage slot is known

Severity: Low Risk

Context: LibDiamond.sol#L14

Description: The preimage of the hashed storage slot `DIAMOND_STORAGE_POSITION` is known.

Recommendation: It might be best to subtract `1` so that the preimage would not be easily attainable. As an example, this is the technique that OpenZeppelin uses.

Connext: Solved by PR 2224.

Spearbit: Verified.

5.3.38 The `@param` NatSpec comment for `_key` in `AssetLogic._swapAsset` is incorrect

Severity: Low Risk

Context: AssetLogic.sol#L221

Description: The `@param` NatSpec for `_key` indicates that this parameter is a canonical token id where instead it should mention that it is a hash of a canonical id and its corresponding domain. We need to make sure the correct value has been passed down to `_swapAsset`.

Recommendation: Change the NatSpec to:

```
- * @param _key - The canonical token id
+ * @param _key - Hash of a canonical id and its corresponding domain
```

Connext: Solved in PR 2457.

Spearbit: Verified.

5.3.39 Malicious routers can temporarily DOS the bridge by depositing a large amount of liquidity

Severity: Low Risk

Context: BridgeFacet.sol#L489-L497

Description: Both router and bridge share the same liquidity cap on the Connext bridge.

Assume that the liquidity cap for USDC is 1 million on Ethereum. Shortly after the Connext Amarok launch, a router adds 1 million USDC liquidity. No one would be able to perform a `xcall` transfer with USDC from Ethereum to other chains as it will always revert because the liquidity cap has exceeded.

The DOS is temporary because the router's liquidity on Ethereum will be reduced if there is USDC liquidity flowing in the opposite direction (e.g., From Polygon to Ethereum)

Recommendation: Consider having a separate liquidity cap for the router and bridge.

Connext: The routers were removed from tracking towards the cap. The cap will track funds moving through the bridge, and that will impact AMM prices, which will impact slippage, which should naturally tamp demand for routers to provide mainnet exit liquidity. Since capping only value through the bridge seems to have network-wide impacts, we decided to only count those funds towards the cap.

Spearbit: Verified.

5.3.40 Prevent deploying a representation token twice

Severity: Low Risk

Context: TokenFacet.sol#L164-L190, TokenFacet.sol#L272-L313, TokenFacet.sol#L354-L383

Description: The function `setupAsset()` is protected by `_enrollAdoptedAndLocalAssets()` which checks `s.approvedAssets[_key]` to prevent accidentally setting up an asset twice.

However the function `_removeAssetId()` is rather thorough and removes the `s.approvedAssets[_key]` flag. After a call to `_removeAssetId()`, an asset can be recreated via `setupAsset()`. This will deploy a second representation token which will be confusing to users of Connext.

Note: The function `setupAssetWithDeployedRepresentation()` could be used to connect a previous presentation token again to the canonical token.

Note: All these functions are authorized so it would only be a problem if mistakes are made.

```
function setupAsset(...) ... onlyOwnerOrAdmin ... {
    if (_canonical.domain != s.domain) {
        _local = _deployRepresentation(...);   // deploys a new token
    } else {
        ...
    }
    bytes32 key = _enrollAdoptedAndLocalAssets(_adoptedAssetId, _local, _stableSwapPool, _canonical);
    ...
}
function _enrollAdoptedAndLocalAssets(...) ... {
    ...
    if (s.approvedAssets[_key]) revert TokenFacet__addAssetId_alreadyAdded();
    s.approvedAssets[_key] = true;
    ...
}
function _removeAssetId(...) ... {
    ...
    delete s.approvedAssets[_key];
    ...
}
```

Recommendation: Consider checking that a token has previously been deployed. Probably a separate mapping is required to store previously deployed contracts.

Connext: Solved in PR 2462. The following still works after removing the token:

- `getLocalAsset` / `_getLocalAsset` / `_getRepresentationAsset` because it would be important to allow the stored value to be retrieved, regardless of the allowlist status. If we do delist an asset, this not working would make things a bit more complicated.
- `execute` / `_handleExecuteLiquidity` / `_creditTokens` because otherwise you run into a position where a user could have started a transfer, and the asset is removed. imagine you are transferring between two remote domains, using the slow path. in this case, while the transfer is in flight the `IBridgeToken.totalSupply` could be `0` on the destination, allowing the asset to be removed. when the transfer arrives to be reconciled, it would not be able to as it would no longer be able to use the `_creditTokens` or `execute` code paths.
- `repayAavePortal` because if the asset is removed, using `repayAavePortalFor` (which uses the adopted asset flow) would fail, and routers have to be able to pay down portal debt.

Spearbit: Verified.

5.3.41 Extra safety checks in `_removeAssetId()`

Severity: Low Risk

Context: TokenFacet.sol#L354-L383

Description: The function `_removeAssetId()` deletes assets but it doesn't check if the passed parameters are a consistent set. This allows for mistakes where the wrong values are accidentally deleted.

```
function _removeAssetId(bytes32 _key, address _adoptedAssetId, address _representation) internal {
    ...
    delete s.adoptedToCanonical[_adoptedAssetId];
    delete s.representationToCanonical[_representation];
    ...
}
```

Recommendation: Consider adding the following checks in the beginning of function `_removeAssetId()`:

```
if (s.adoptedToCanonical[_adoptedAssetId] != key) revert ....
if (s.representationToCanonical[_representation] != key) revert ...
```

Connext: Solved in PR 2454.

Spearbit: Verified.

5.3.42 Data length not validated

Severity: Low Risk

Context: GnosisSpokeConnector.sol#L54-L61, BaseMultichain.sol#L39-L47, OptimismSpokeConnector.sol#L51-L54

Description: The following functions do not validate that the input `_data` is 32 bytes.

- `GnosisSpokeConnector._sendMessag`
- `GnosisSpokeConnector._processMessage`
- `BaseMultichain.sendMessage`
- `OptimismSpokeConnector._sendMessage`

The input `_data` contains the outbound Merkle root or aggregated Merkle root, which is always 32 bytes. If the root is not 32 bytes, it is invalid and should be rejected.

Recommendation: Validate the input `_data` to ensure that it is 32 bytes.

`GnosisSpokeConnector._sendMessag`

```
function _sendMessage(bytes memory _data) internal override {
+   require(_data.length == 32, "!length");

    // send the message to the l1 connector by calling `processMessage`
    GnosisAmb(AMB).requireToPassMessage(
        mirrorConnector,
        abi.encodeWithSelector(Connector.processMessage.selector, _data),
        mirrorGas
    );
}

function _processMessage(bytes memory _data) internal override {
+   require(_data.length == 32, "!length");

    // ensure the l1 connector sent the message
    require(_verifySender(mirrorConnector), "!mirrorConnector");
    // ensure it is headed to this domain
    require(GnosisAmb(AMB).destinationChainId() == block.chainid, "!destinationChain");
    // update the aggregate root on the domain
    receiveAggregateRoot(bytes32(_data));
  }
```

`BaseMultichain.sendMessage`

```
function _sendMessage(address _amb, bytes memory _data) internal {
+   require(_data.length == 32, "!length");

    Multichain(_amb).anyCall(
        _amb, // Same address on every chain, using AMB as it is immutable
        _data,
        address(0), // fallback address on origin chain
        MIRROR_CHAIN_ID,
        0 // fee paid on origin chain
    );
}
```

`OptimismSpokeConnector._sendMessage`

```
function _sendMessage(bytes memory _data) internal override {
+    require(_data.length == 32, "!length");

    bytes memory _calldata = abi.encodeWithSelector(Connector.processMessage.selector, _data);
    OptimismAmb(AMB).sendMessage(mirrorConnector, _calldata, uint32(mirrorGas));
}
```

Connext: Solved in PR 2452.

Spearbit: Verified.

5.3.43 Verify timestamp reliability on L2

Severity: Low Risk

Context: StableSwap.sol#L136, ProposedOwnableFacet.sol, RoutersFacet.sol#L442, StableSwapFacet.sol#L46, SwapUtilsExternal.sol, ProposedOwnableFacet.sol#L263, AmplificationUtils.sol,

Description: Timestamp information on rollups can be less reliable than on mainnet. For instance, Arbitrum docs say:

> As a general rule, any timing assumptions a contract makes about block numbers and timestamps should be considered generally reliable in the longer term (i.e., on the order of at least several hours) but unreliable in the shorter term (minutes). (It so happens these are generally the same assumptions one should operate under when using block numbers directly on Ethereum!)

Uniswap docs mention this for Optimism:

> The block.timestamp of these blocks, however, reflect the block.timestamp of the last L1 block ingested by the Sequencer.

Recommendation: Before deploying on a rollup, consider checking their documentation on timestamp and check if the time dependent functionality is is safe. It can be the case that the deviation is small enough to not matter. Increase the deadline threshold, if the deviation is high enough.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.3.44 MirrorConnector cannot be changed once set

Severity: Low Risk

Context: PolygonHubConnector.sol#L51-L55 PolygonSpokeConnector.sol#L78-L82

Description: For chains other than Polygon, it is allowed to change mirror connector any number of times. For Polygon chain, the `_setMirrorConnector` is overridden.

1. Let's take `PolygonHubConnector` contract example:

```
function _setMirrorConnector(address _mirrorConnector) internal override {
   super._setMirrorConnector(_mirrorConnector);
   setFxChildTunnel(_mirrorConnector);
}
```

2. Since `setFxChildTunnel(PolygonHubConnector)` can only be called once due to below require check, this also restricts the number of time mirror connector can be altered.

```
function setFxChildTunnel(address _fxChildTunnel) public virtual {
    require(fxChildTunnel == address(0x0), "FxBaseRootTunnel: CHILD_TUNNEL_ALREADY_SET");
    ...
}
```

Recommendation: If it is required to allow changing of Mirror connector then remove the require condition in both `setFxChildTunnel(PolygonHubConnector)` and `setFxRootTunnel(PolygonSpokeConnector)`

Connext: Fixed in pull 2546 & 2520.

Spearbit: Verified.

5.3.45 Possible infinite loop in `dequeueVerified()`

Severity: Low Risk

Context: Queue.sol#L59-L102

Description: The loop in function `dequeueVerified()` doesn't end if `queue.first == queue.last == 0`. In this situation, at `unchecked { --last; }` the following happens: `last` wraps to `type(uint128).max`. Now `last` is very large and is surely `>=first` and thus the loop keeps running.

This problem can occur when `queue` isn't initialized.

```
function dequeueVerified(Queue storage queue, uint256 delay) internal returns (bytes32[] memory) {
    uint128 first = queue.first;
    uint128 last = queue.last;
    require(last >= first, "queue empty");
    for (last; last >= first; ) {
        ...
        unchecked { --last; } // underflows when last == 0 (e.g. queue isn't initialized)
    }
}
```

Recommendation: Document the risk of an infinite loop and the importance of initializing `queue`. Or do one of the following:

- Check `queue` is initialized (e.g. `queue.first >= 1` as `last >= first` is already checked).
- Remove `unchecked`.
- Change the type of `last` to a signed type : `int256`, this way it is no problem if its negative.

Connext: Solved in PR 2384.

Spearbit: Verified.

5.3.46 Do not ignore `staticcall`'s return value

Severity: Low Risk

Context: TypedMemView.sol#L652, TypedMemView.sol#L668-L669, TypedMemView.sol#L685-L686, TypedMemView.sol#L761,

Description: `TypedMemView` calls several precompiles through `staticcall` opcode and never checks its return value assuming it is a success. For instance:

```
// use the identity precompile to copy
// guaranteed not to fail, so pop the success
pop(staticcall(gas(), 4, _oldLoc, _len, _newLoc, _len))
```

However, there are rare cases when call to precompiles can fail. For example, when the call runs out of gas (since 63/64 of the gas is passed, the remaining execution can still have gas). Generally, not checking for success of calls is dangerous and can have unintended consequences.

Recommendation: `staticcall` returns `1` on success and `0` on failure. Consider reverting if it returns `0`.

See summa-tx/memview-sol/pull/9 for a solution.

Connext: Solved in PR 2451.

Spearbit: Verified.

5.3.47 Renounce wait time can be extended

Severity: Low Risk

Context: ProposedOwnable.sol#L109-L118

Description: The `_proposedOwnershipTimestamp` updates everytime on calling `proposeNewOwner` with `newlyProposed` as zero address. This elongates the time when owner can be renounced.

Recommendation: Add a check which rejects duplicate owner renounce request.

```
if (_proposed == newlyProposed && newlyProposed == address(0) && _proposedOwnershipTimestamp!=0){
    revert ProposedOwnable__proposeNewOwner_invalidProposal();
}
```

Connext: Solved in PR 2450.

Spearbit: Verified.

5.3.48 Extra parameter in function `checker()` at `encodeWithSelector()`

Severity: Low Risk

Context: SendOutboundRootResolver.sol#L32-L42

Description: The function `checker()` sets up the parameters to call the function `sendMessage()`. However, it adds an extra parameter `outboundRoot`, which isn't necessary.

```
function sendMessage() external {
    ...
}
function checker() external view override returns (bool canExec, bytes memory execPayload) {
    ...
    execPayload = abi.encodeWithSelector(this.sendMessage.selector, outboundRoot); // extra parameter
    ...
}
```

Recommendation: Remove `outboundRoot` and preferably use `abi.encodeCall(...)` which explicitly checks the provided parameters with the function definition.

```
- execPayload = abi.encodeWithSelector(this.sendMessage.selector, outboundRoot);
+ execPayload = abi.encodeCall(this.sendMessage, ());
```

Connext: `SendOutboundRootResolver.sol` is removed in PR 2199.

Spearbit: Verified.

5.4 Gas Optimization

5.4.1 `MerkleLib.insert()` can be optimized

Severity: Gas Optimization

Context: Merkle.sol#L76-L105, Merkle.sol#L115

Description: `MerkleTreeManager.insert()` calls `MerkleLib.insert()` repeatedly on `tree` stored in storage. Each call to `MerkleLib.insert()` reads the entire tree from storage, and writes 2 (`tree.count` and `tree.branch[i]`) back to storage.

These storage operations can be done only once at the beginning, by loading them in memory. The updated count and branches can be written back to the storage at the end saving expensive `SSTORE` and `SLOAD` operations.

Recommendation: Consider the following:

- Load the entire `tree` in memory at the beginning in `MerkleTreeManager.insert()`. Pass this memory `tree` to `MerkleLib.insert()`.
- `MerkleLib.insert()` now takes `tree` in memory.
- At the end of the function, it writes the updated `tree` back to storage.

Note that if size of the `leaves` array is much smaller than `TREE_DEPTH == 32`, there's a chance this increases the gas consumption since now there is an additional cost of memory expansion, and 33 slots are always written regardless of the size of the `leaves` array.

Connext: Solved in PR 2211.

Spearbit: Verified.

5.4.2 EIP712 domain separator can be cached

Severity: Gas Optimization

Context: OZERC20.sol#L386

Description: The domain separator can be cached for gas-optimization.

Recommendation: Consider caching the EIP712 domain separator. Following is the caching mechanism from the latest OpenZeppelin implementation for reference.

```
/**
* @dev Returns the domain separator for the current chain.
*/
function _domainSeparatorV4() internal view returns (bytes32) {
    if (address(this) == _CACHED_THIS && block.chainid == _CACHED_CHAIN_ID) {
        return _CACHED_DOMAIN_SEPARATOR;
    } else {
        return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
    }
}
```

Connext: Solved in PR 2350.

Spearbit: Verified.

5.4.3 `stateCommitmentChain` can be made immutable

Severity: Gas Optimization

Context: OptimismHubConnector.sol#L25

Description: Once assigned in constructor, `stateCommitmentChain` cannot be changed.

Recommendation: `stateCommitmentChain` variable can be defined as immutable

Connext: Solved in PR 2480.

Spearbit: Verified.

5.4.4 Nonce can be updated in single step

Severity: Gas Optimization

Context: SpokeConnector.sol#L278-L279

Description: Nonce can be incremented in single step instead of using a second step which will save some gas

Recommendation: Kindly change the below lines:

```
function dispatch(
    uint32 _destinationDomain,
    bytes32 _recipientAddress,
    bytes memory _messageBody
  ) external onlyWhitelistedSender returns (bytes32) {
...
- uint32 _nonce = nonces[_destinationDomain];
- nonces[_destinationDomain] = _nonce + 1;

+ uint32 _nonce = nonces[_destinationDomain]++;

...
}
```

Connext: Solved in PR 2479.

Spearbit: Verified.

5.4.5 `ZkSyncSpokeConnector._sendMessage` encodes unnecessary data

Severity: Gas Optimization

Context:

- ZkSyncSpokeConnector.sol#L50-L54
- ZkSyncHubConnector.sol#L88

Description: Augmenting the `_data` with the `processMessage` function selector is unnecessary. Since on the mirror domain, we just need to provide the right parameters to `ZkSyncHubConnector.processMessageFromRoot` (which by the way anyone can call) to prove the `L2` message inclusion of the merkle root `_data`. Thus the current implementation is wasting gas.

Recommendation: We can rewrite `ZkSyncSpokeConnector._sendMessage` function as:

```
function _sendMessage(bytes memory _data) internal override {
  // Dispatch message through zkSync AMB
  L1_MESSENGER_CONTRACT.sendToL1(_data);
}
```

Also `ZkSyncHubConnector.processMessageFromRoot` would need to be changed accordingly and `TypedMemView` library import can be removed from `ZkSyncHubConnector`:

```
function processMessageFromRoot(
  // zkSync block number in which the message was sent
  uint32 _l2BlockNumber,
  // Message index, that can be received via API
  uint256 _l2MessageIndex,
  // The L2 transaction number in a block, in which the log was sent
  uint16 _l2TxNumberInBlock,
  // The message that was sent from l2
  bytes calldata _message,
  // Merkle proof for the message
  bytes32[] calldata _proof
) external {
  // sanity check root length (fn selector + 32 bytes root)
  require(_message.length == 32, "!length");

  IZkSync zksync = IZkSync(AMB);
  L2Message memory message = L2Message({
    txNumberInBlock: _l2TxNumberInBlock,
    sender: mirrorConnector,
    data: _message
  });

  bool success = zksync.proveL2MessageInclusion(_l2BlockNumber, _l2MessageIndex, message, _proof);
  require(success, "!proven");

  bytes32 _root = bytes32(_message);

  // NOTE: there are no guarantees the messages are processed once, so processed roots
  // must be tracked within the connector. See:
  // https://v2-docs.zksync.io/dev/developer-guides/Bridging/l2-l1.html#prove-inclusion-of-the-message-
↪   into-the-l2-block
  if (!processed[_root]) {
    // set root to processed
    processed[_root] = true;
    // update the root on the root manager
    IRootManager(ROOT_MANAGER).aggregate(MIRROR_DOMAIN, _root);
  } // otherwise root was already sent to root manager
}
```

Reference: Summary on L2->L1 messaging

Connext: Solved in PR 2505.

Spearbit: Verified.

5.4.6 `getD` can be optimized by removing an extra multiplication by `d` per iteration

Severity: Gas Optimization

Context:

- SwapUtils.sol#L327-L341
- SwapUtilsExternal.sol#L364-L378

Description: The calculation for the new `d` can be simplified by canceling a `d` from the numerator and denominator. Basically, we have :

$$f(D) = \frac{1}{n^{n+1} a \prod x_i} D^{n+1} + (1 - \frac{1}{na})D - \sum x_i$$

and having/assuming n, a, x_i are fixed, we are using Newton's method to find a solution for $f = 0$. The original implementation is using:

$$D' = D - \frac{f(D)}{f'(D)} = \frac{(na\sum x_i + \frac{D^{n+1}}{n^{n-1}\prod x_i})D}{(na-1)D + (n+1)\frac{D^{n+1}}{n^n \prod x_i}}$$

which can be simplified to:

$$D' = \frac{na\sum x_i + \frac{D^n}{n^{n-1}\prod x_i}D}{(na-1) + (n+1)\frac{D^n}{n^n \prod x_i}}$$

Recommendation: Lines 327-L341 can be changed to:

```
uint256 dP = 1;
for (uint256 j; j < numTokens; ) {
  dP = (dP * d) / (xp[j] * numTokens);
  // If we were to protect the division loss we would have to keep the denominator separate
  // and divide at the end. However this leads to overflow with large numTokens or/and D.
  // dP = dP * D * D * D * ... overflow!

  unchecked {
    ++j;
  }
}
prevD = d;
d =
  ((nA * s) / AmplificationUtils.A_PRECISION + dP * numTokens * d) /
  ((nA - AmplificationUtils.A_PRECISION) / AmplificationUtils.A_PRECISION + (numTokens + 1) * dP);
```

This simplification breaks some test cases:

```
Failing tests:
Encountered 9 failing tests in
↪  contracts_forge/core/connext/facets/StableSwapFacet.t.sol:StableSwapFacetTest
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__addSwapLiquidity_shouldWork() (gas: 208089)
[FAIL. Reason: Call did not revert as expected]
↪  test_StableSwapFacet__removeSwapLiquidityImbalance_failIfFrontRun() (gas: 382845)
[FAIL. Reason: Assertion failed.]
↪  test_StableSwapFacet__removeSwapLiquidityImbalance_failIfMoreThanLpBalance() (gas: 229659)
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__removeSwapLiquidityImbalance_failIfPaused()
↪  (gas: 245027)
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__removeSwapLiquidityImbalance_shouldWork() (gas:
↪  292194)
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__removeSwapLiquidityOneToken_failIfFrontRun()
↪  (gas: 390378)
[FAIL. Reason: Assertion failed.]
↪  test_StableSwapFacet__removeSwapLiquidityOneToken_failIfMoreThanLpBalance() (gas: 225984)
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__removeSwapLiquidityOneToken_shouldWork() (gas:
↪  278690)
[FAIL. Reason: Assertion failed.] test_StableSwapFacet__removeSwapLiquidity_shouldWork() (gas: 248608)

Encountered 3 failing tests in
↪  contracts_forge/core/connext/facets/SwapAdminFacet.t.sol:SwapAdminFacetTest
[FAIL. Reason: Assertion failed.] test_SwapAdminFacet__rampA_shouldWorkWithDownwards() (gas: 269074)
[FAIL. Reason: Assertion failed.] test_SwapAdminFacet__rampA_shouldWorkWithUpwards() (gas: 269057)
[FAIL. Reason: Assertion failed.]
↪  test_SwapAdminFacet__withdrawSwapAdminFess_shouldWorkWithExpectedAmount() (gas: 241254)

Encountered a total of 12 failing tests, 521 tests succeeded
```

Looking at the tests, it seems like some hardcoded constants have been used, like the following: SwapAdminFacet.t.sol#L604

Where do these constants come from?

Also, if this recommendation is taken into consideration, we would suggest adding fuzzing and differential tests for `getD`.

Connext: These constants come from saddle finance unit tests. Changing the code would break the tests and new constants have to be found to fix the test. We would rather keep the current code to stay as close to the Saddle original as possible.

Spearbit: Acknowledged.

5.4.7 `_recordOutputAsSpent` in `ArbitrumHubConnector` can be optimized by changing the require condition

Severity: Gas Optimization

Context: ArbitrumHubConnector.sol#L178

Description: In `_recordOutputAsSpent`, `_index` is compared with a literal value that is a power of `2`. The exponentiation in this statement can be completely removed to save gas.

Recommendation: Rewrite the `require` statement as:

```
require((_index >>_proof.length) == 0, "!minimal proof");
```

Gas saved according to test cases:

```
test_ArbitrumHubConnector__processMessageFromRoot_failsIfNot36Bytes() (gas: -215 (-0.059%))
test_ArbitrumHubConnector__processMessageFromRoot_works() (gas: -215 (-0.099%))
test_ArbitrumHubConnector__processMessageFromRoot_failsIfIncorrectProof() (gas: -215 (-0.121%))
test_ArbitrumHubConnector__processMessageFromRoot_failsIfProofNotMinimal() (gas: -215 (-0.122%))
test_ArbitrumHubConnector__processMessageFromRoot_failsIfAlreadyProcessed() (gas: -430 (-0.186%))
Overall gas change: -1290 (-0.587%)
```

A similiar recommendation should apply if this `require` statement is converted to a custom error pattern.

Connext: Solved in PR 2509.

Spearbit: Verified.

5.4.8 `Message.leaf`'s memory manipulation is redundant

Severity: Gas Optimization

Context: Message.sol#L97-L107

Description: The chunk of memory related to `_message` is dissected into pieces and then copied into another section of memory and hashed.

Recommendation: We can save gas if we hashed the original section.

```
function leaf(bytes29 _message) internal pure returns (bytes32 hash) {
  uint256 loc = _message.loc();
  uint256 len = _message.len();

  assembly {
    hash := keccak256(loc, len)
  }
}
```

Gas saved according to test cases:

```
test_MultichainSpokeConnector_processMessage_revertIfWrongDataLength(uint8) (gas: -163 (-0.419%))
Overall gas change: -163 (-0.419%)
```

Connext: Solved in PR 2484.

Spearbit: Verified.

5.4.9 `coerceBytes32` can be more optimized

Severity: Gas Optimization

Context: TypeCasts.sol#L10-L12

Description: It would be cheaper to not use `TypedMemView` in `coerceBytes32()`. We would only need to check the length and mask.

Note: `coerceBytes32` doesn't seem to be used. If that is the case it could also be removed.

Recommendation: As a rough sketch we could have:

```
error TheStringIsToLongForBytes32();
uint256 internal constant INT256_MIN = 1 << 255;

function coerceBytes32(string memory _s) internal pure returns (bytes32 result) {
  bytes memory b = bytes(_s);
  uint256 length = b.length;
  if(length == 0) {
    return bytes32(0);
  }

  if(length > 32) {
    revert TheStringIsToLongForBytes32();
  }

  assembly {
    // solhint-disable-previous-line no-inline-assembly
    let mask := sar(sub(shl(3, length), 1), INT256_MIN) // 2^(255) - 2^(255 - (8*(len) - 1))
    result:= and(mload(add(b, 0x20)), mask) // clean-up possible dirty bytes
  }
}
```

Connext: Removed function in PR 2502.

Spearbit: Verified.

5.4.10 Consider removing domains from `propagate()` arguments

Severity: Gas Optimization

Context: RootManager.sol#L147

Description: `propagate(uint32[] calldata _domains, address[] calldata _connectors)` only uses `_domains` to verify its hash against `domainsHash`, and to emit an event. Hence, its only use seems to be to notify off-chain agents of the supported domains.

Recommendation: Consider removing `_domains` from `propagate()`'s arguments. To still notify the off-chain agents about the supported domains, consider adding events to `DomainIndexer`'s `addDomain()` and `removeDomain()` function.

Connext: Fixed in commit 4c5821 and PR 2547.

Spearbit: Verified.

5.4.11 Loop counter can be made `uint256` to save gas

Severity: Gas Optimization

Context: SwapAdminFacet.sol#L132, StableSwap.sol#L90, Encoding.sol#L22-L45, TypedMemView.sol#L158

Description: There are several loops that use an `uint8` as the type for the loop variable. Changing that to `uint256` can save some gas.

Recommendation: Consider replacing `uint8` with `uint256` in loops.

Connext: Solved in PR 2372.

Spearbit: Verified.

5.4.12 Set owner directly to zero address in renounceOwnership

Severity: Gas Optimization

Context: ProposedOwnable.sol#L135 ProposedOwnableFacet.sol#L234

Description:

1. In `renounceOwnership` function, `_proposed` will always be address zero so instead of setting the variable `_proposed` as owner, we can directly set address(0) as the new owner.
2. Similarly for `renounceOwnership` function also set address(0) as the new owner.

Recommendation: Revise the `renounceOwnership` function

```
function renounceOwnership() public virtual onlyOwner {
    ...
    // Emit event, set new owner, reset timestamp
    _setOwner(address(0));
  }
```

Connext: Solved in PR 2482.

Spearbit: Verified.

5.4.13 Retrieve `decimals()` once

Severity: Gas Optimization

Context: BridgeFacet.sol#L6, BridgeFacet.sol#L514-L516, AssetLogic.sol#L5

Description: There are several locations where the number of `decimals()` of tokens are retrieved. As all tokens are whitelisted, it would also be possible to retrieve the `decimals()` once and store these to save gas.

`BridgeFacet.sol`

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
...
function _xcall(...) ... {
    ...
    ... AssetLogic.normalizeDecimals(ERC20(_asset).decimals(), uint8(18), _amount);
    ...
}
```

`AssetLogic.sol`

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
...
function swapToLocalAssetIfNeeded(...) ... {
    ...
    ... calculateSlippageBoundary(ERC20(_asset).decimals(), ERC20(_local).decimals(), _amount,
↪   _slippage) ...
    ...
}
function swapFromLocalAssetIfNeeded(...) ... {
    ...
    ... calculateSlippageBoundary(uint8(18), ERC20(adopted).decimals(), _normalizedIn, _slippage) ...
    ...
}
```

Recommendation: Consider to retrieve the `decimals()` of tokens once and store these.

Connext: Solved in PR 2530.

Spearbit: Verified.

5.4.14 The `root...` function in `Merkle.sol` can be optimized by using YUL, unrolling loops and using the scratch space

Severity: Gas Optimization

Context:

- Merkle.sol#L111
- Merkle.sol#L122

Description: We can use assembly, unroll loops, and use the scratch space to save gas. Also, `rootWithCtx` can be removed (would save us from jumping) since it has only been used here.

Recommendation:

```
library MerkleLib {
  ...

  /**
   * @notice Calculates and returns tree's current root.
   * @return _current bytes32 root.
   **/
  function root(Tree storage tree) internal view returns (bytes32 _current) {
    uint256 _index = tree.count;

    if(_index == 0) {
      return Z_32;
    }

    uint256 i;
    assembly {
      let TREE_SLOT := tree.slot

      for {} true {} {
        for {} true {} {
          if and(_index, 1) {
            mstore(0, sload(TREE_SLOT))
            mstore(0x20, Z_0)
            _current := keccak256(0, 0x40)
            break
          }

          if and(_index, shl(1, 1)) {
              mstore(0, sload(add(TREE_SLOT, 1)))
              mstore(0x20, Z_1)
              _current := keccak256(0, 0x40)
              i := 1
              break
          }

          if and(_index, shl(2, 1)) {
              mstore(0, sload(add(TREE_SLOT, 2)))
              mstore(0x20, Z_2)
              _current := keccak256(0, 0x40)
              i := 2
              break
          }

          if and(_index, shl(3, 1)) {
              mstore(0, sload(add(TREE_SLOT, 3)))
              mstore(0x20, Z_3)
              _current := keccak256(0, 0x40)
              i := 3
```

```
        break
    }

    if and(_index, shl(4, 1)) {
        mstore(0, sload(add(TREE_SLOT, 4)))
        mstore(0x20, Z_4)
        _current := keccak256(0, 0x40)
        i := 4
        break
    }

    if and(_index, shl(5, 1)) {
        mstore(0, sload(add(TREE_SLOT, 5)))
        mstore(0x20, Z_5)
        _current := keccak256(0, 0x40)
        i := 5
        break
    }

    if and(_index, shl(6, 1)) {
        mstore(0, sload(add(TREE_SLOT, 6)))
        mstore(0x20, Z_6)
        _current := keccak256(0, 0x40)
        i := 6
        break
    }

    if and(_index, shl(7, 1)) {
        mstore(0, sload(add(TREE_SLOT, 7)))
        mstore(0x20, Z_7)
        _current := keccak256(0, 0x40)
        i := 7
        break
    }

    if and(_index, shl(8, 1)) {
        mstore(0, sload(add(TREE_SLOT, 8)))
        mstore(0x20, Z_8)
        _current := keccak256(0, 0x40)
        i := 8
        break
    }

    if and(_index, shl(9, 1)) {
        mstore(0, sload(add(TREE_SLOT, 9)))
        mstore(0x20, Z_9)
        _current := keccak256(0, 0x40)
        i := 9
        break
    }

    if and(_index, shl(10, 1)) {
        mstore(0, sload(add(TREE_SLOT, 10)))
        mstore(0x20, Z_10)
        _current := keccak256(0, 0x40)
        i := 10
        break
    }

    if and(_index, shl(11, 1)) {
        mstore(0, sload(add(TREE_SLOT, 11)))
        mstore(0x20, Z_11)
```

```
            _current := keccak256(0, 0x40)
            i := 11
            break
        }

        if and(_index, shl(12, 1)) {
            mstore(0, sload(add(TREE_SLOT, 12)))
            mstore(0x20, Z_12)
            _current := keccak256(0, 0x40)
            i := 12
            break
        }

        if and(_index, shl(13, 1)) {
            mstore(0, sload(add(TREE_SLOT, 13)))
            mstore(0x20, Z_13)
            _current := keccak256(0, 0x40)
            i := 13
            break
        }

        if and(_index, shl(14, 1)) {
            mstore(0, sload(add(TREE_SLOT, 14)))
            mstore(0x20, Z_14)
            _current := keccak256(0, 0x40)
            i := 14
            break
        }

        if and(_index, shl(15, 1)) {
            mstore(0, sload(add(TREE_SLOT, 15)))
            mstore(0x20, Z_15)
            _current := keccak256(0, 0x40)
            i := 15
            break
        }

        if and(_index, shl(16, 1)) {
            mstore(0, sload(add(TREE_SLOT, 16)))
            mstore(0x20, Z_16)
            _current := keccak256(0, 0x40)
            i := 16
            break
        }

        if and(_index, shl(17, 1)) {
            mstore(0, sload(add(TREE_SLOT, 17)))
            mstore(0x20, Z_17)
            _current := keccak256(0, 0x40)
            i := 17
            break
        }

        if and(_index, shl(18, 1)) {
            mstore(0, sload(add(TREE_SLOT, 18)))
            mstore(0x20, Z_18)
            _current := keccak256(0, 0x40)
            i := 18
            break
        }

        if and(_index, shl(19, 1)) {
```

```
            mstore(0, sload(add(TREE_SLOT, 19)))
            mstore(0x20, Z_19)
            _current := keccak256(0, 0x40)
            i := 19
            break
        }

        if and(_index, shl(20, 1)) {
            mstore(0, sload(add(TREE_SLOT, 20)))
            mstore(0x20, Z_20)
            _current := keccak256(0, 0x40)
            i := 20
            break
        }

        if and(_index, shl(21, 1)) {
            mstore(0, sload(add(TREE_SLOT, 21)))
            mstore(0x20, Z_21)
            _current := keccak256(0, 0x40)
            i := 21
            break
        }

        if and(_index, shl(22, 1)) {
            mstore(0, sload(add(TREE_SLOT, 22)))
            mstore(0x20, Z_22)
            _current := keccak256(0, 0x40)
            i := 22
            break
        }

        if and(_index, shl(23, 1)) {
            mstore(0, sload(add(TREE_SLOT, 23)))
            mstore(0x20, Z_23)
            _current := keccak256(0, 0x40)
            i := 23
            break
        }

        if and(_index, shl(24, 1)) {
            mstore(0, sload(add(TREE_SLOT, 24)))
            mstore(0x20, Z_24)
            _current := keccak256(0, 0x40)
            i := 24
            break
        }

        if and(_index, shl(25, 1)) {
            mstore(0, sload(add(TREE_SLOT, 25)))
            mstore(0x20, Z_25)
            _current := keccak256(0, 0x40)
            i := 25
            break
        }

        if and(_index, shl(26, 1)) {
            mstore(0, sload(add(TREE_SLOT, 26)))
            mstore(0x20, Z_26)
            _current := keccak256(0, 0x40)
            i := 26
            break
        }
```

```
  if and(_index, shl(27, 1)) {
      mstore(0, sload(add(TREE_SLOT, 27)))
      mstore(0x20, Z_27)
      _current := keccak256(0, 0x40)
      i := 27
      break
  }

  if and(_index, shl(28, 1)) {
      mstore(0, sload(add(TREE_SLOT, 28)))
      mstore(0x20, Z_28)
      _current := keccak256(0, 0x40)
      i := 28
      break
  }

  if and(_index, shl(29, 1)) {
      mstore(0, sload(add(TREE_SLOT, 29)))
      mstore(0x20, Z_29)
      _current := keccak256(0, 0x40)
      i := 29
      break
  }

  if and(_index, shl(30, 1)) {
      mstore(0, sload(add(TREE_SLOT, 30)))
      mstore(0x20, Z_30)
      _current := keccak256(0, 0x40)
      i := 30
      break
  }

  if and(_index, shl(31, 1)) {
      mstore(0, sload(add(TREE_SLOT, 31)))
      mstore(0x20, Z_31)
      _current := keccak256(0, 0x40)
      i := 31
      break
  }

  _current := Z_32
  i := 32
  break
}

if gt(i, 30) {
  break
}

{
  if lt(i, 1) {
    switch and(_index, shl(1, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_1)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 1)))
      mstore(0x20, _current)
    }
```

```
    _current := keccak256(0, 0x40)
  }

  if lt(i, 2) {
    switch and(_index, shl(2, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_2)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 2)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 3) {
    switch and(_index, shl(3, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_3)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 3)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 4) {
    switch and(_index, shl(4, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_4)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 4)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 5) {
    switch and(_index, shl(5, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_5)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 5)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 6) {
    switch and(_index, shl(6, 1))
    case 0 {
```

```
      mstore(0, _current)
      mstore(0x20, Z_6)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 6)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 7) {
    switch and(_index, shl(7, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_7)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 7)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 8) {
    switch and(_index, shl(8, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_8)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 8)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 9) {
    switch and(_index, shl(9, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_9)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 9)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 10) {
    switch and(_index, shl(10, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_10)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 10)))
      mstore(0x20, _current)
```

```
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 11) {
    switch and(_index, shl(11, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_11)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 11)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 12) {
    switch and(_index, shl(12, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_12)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 12)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 13) {
    switch and(_index, shl(13, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_13)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 13)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 14) {
    switch and(_index, shl(14, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_14)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 14)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 15) {
```

```
      switch and(_index, shl(15, 1))
      case 0 {
        mstore(0, _current)
        mstore(0x20, Z_15)
      }
      default {
        mstore(0, sload(add(TREE_SLOT, 15)))
        mstore(0x20, _current)
      }

      _current := keccak256(0, 0x40)
    }

    if lt(i, 16) {
      switch and(_index, shl(16, 1))
      case 0 {
        mstore(0, _current)
        mstore(0x20, Z_16)
      }
      default {
        mstore(0, sload(add(TREE_SLOT, 16)))
        mstore(0x20, _current)
      }

      _current := keccak256(0, 0x40)
    }

    if lt(i, 17) {
      switch and(_index, shl(17, 1))
      case 0 {
        mstore(0, _current)
        mstore(0x20, Z_17)
      }
      default {
        mstore(0, sload(add(TREE_SLOT, 17)))
        mstore(0x20, _current)
      }

      _current := keccak256(0, 0x40)
    }

    if lt(i, 18) {
      switch and(_index, shl(18, 1))
      case 0 {
        mstore(0, _current)
        mstore(0x20, Z_18)
      }
      default {
        mstore(0, sload(add(TREE_SLOT, 18)))
        mstore(0x20, _current)
      }

      _current := keccak256(0, 0x40)
    }

    if lt(i, 19) {
      switch and(_index, shl(19, 1))
      case 0 {
        mstore(0, _current)
        mstore(0x20, Z_19)
      }
      default {
```

```
      mstore(0, sload(add(TREE_SLOT, 19)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 20) {
    switch and(_index, shl(20, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_20)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 20)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 21) {
    switch and(_index, shl(21, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_21)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 21)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 22) {
    switch and(_index, shl(22, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_22)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 22)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }

  if lt(i, 23) {
    switch and(_index, shl(23, 1))
    case 0 {
      mstore(0, _current)
      mstore(0x20, Z_23)
    }
    default {
      mstore(0, sload(add(TREE_SLOT, 23)))
      mstore(0x20, _current)
    }

    _current := keccak256(0, 0x40)
  }
```

```
if lt(i, 24) {
  switch and(_index, shl(24, 1))
  case 0 {
    mstore(0, _current)
    mstore(0x20, Z_24)
  }
  default {
    mstore(0, sload(add(TREE_SLOT, 24)))
    mstore(0x20, _current)
  }

  _current := keccak256(0, 0x40)
}

if lt(i, 25) {
  switch and(_index, shl(25, 1))
  case 0 {
    mstore(0, _current)
    mstore(0x20, Z_25)
  }
  default {
    mstore(0, sload(add(TREE_SLOT, 25)))
    mstore(0x20, _current)
  }

  _current := keccak256(0, 0x40)
}

if lt(i, 26) {
  switch and(_index, shl(26, 1))
  case 0 {
    mstore(0, _current)
    mstore(0x20, Z_26)
  }
  default {
    mstore(0, sload(add(TREE_SLOT, 26)))
    mstore(0x20, _current)
  }

  _current := keccak256(0, 0x40)
}

if lt(i, 27) {
  switch and(_index, shl(27, 1))
  case 0 {
    mstore(0, _current)
    mstore(0x20, Z_27)
  }
  default {
    mstore(0, sload(add(TREE_SLOT, 27)))
    mstore(0x20, _current)
  }

  _current := keccak256(0, 0x40)
}

if lt(i, 28) {
  switch and(_index, shl(28, 1))
  case 0 {
    mstore(0, _current)
    mstore(0x20, Z_28)
```

```
            }
            default {
              mstore(0, sload(add(TREE_SLOT, 28)))
              mstore(0x20, _current)
            }

            _current := keccak256(0, 0x40)
          }

          if lt(i, 29) {
            switch and(_index, shl(29, 1))
            case 0 {
              mstore(0, _current)
              mstore(0x20, Z_29)
            }
            default {
              mstore(0, sload(add(TREE_SLOT, 29)))
              mstore(0x20, _current)
            }

            _current := keccak256(0, 0x40)
          }

          if lt(i, 30) {
            switch and(_index, shl(30, 1))
            case 0 {
              mstore(0, _current)
              mstore(0x20, Z_30)
            }
            default {
              mstore(0, sload(add(TREE_SLOT, 30)))
              mstore(0x20, _current)
            }

            _current := keccak256(0, 0x40)
          }

          if lt(i, 31) {
            switch and(_index, shl(31, 1))
            case 0 {
              mstore(0, _current)
              mstore(0x20, Z_31)
            }
            default {
              mstore(0, sload(add(TREE_SLOT, 31)))
              mstore(0x20, _current)
            }

            _current := keccak256(0, 0x40)
          }
        }

        break
      }
    }
  }
  ...
  bytes32 internal constant Z_32 =
↪   hex"27ae5ba08d7291c96c8cbddcc148bf48a6d68c7974b94356f53754ef6171d757";
}
```

Gas savings according to test cases:

```
test_RootManager__propagate_shouldSendToAllSpokes(bytes32) (gas: -5323 (-0.201%))
test_RootManager__propagate_shouldSendToSpoke(bytes32) (gas: -6573 (-1.753%))
test_MultichainSpokeConnector_processMessage_revertIfWrongDataLength(uint8) (gas: 1470 (3.783%))
test_MainnetSpokeConnector__sendMessage_fromRootManagerWorks() (gas: -10657 (-18.110%))
test_messageFlowsWork() (gas: 650754 (21.106%))
test_Merkle__insert_shouldUpdateCount() (gas: -65402 (-25.713%))
test_GnosisSpokeConnector__sendMessage_shouldWork() (gas: -21314 (-34.402%))
test_OptimismSpokeConnector__sendMessage_works() (gas: -21314 (-34.472%))
test_MainnetSpokeConnector__sendMessage_failsIfCallerNotRootManager() (gas: -10657 (-40.132%))
test_MultichainSpokeConnector_sendMessage_sendMessageAndEmitEvent() (gas: -21314 (-40.516%))
test_PolygonSpokeConnector__sendMessage_works() (gas: -21314 (-42.397%))
test_ArbitrumSpokeConnector__sendMessage_works() (gas: -31971 (-45.808%))
Overall gas change: 436385 (-258.615%)
```

Note the extra gas in `test_messageFlowsWork()` is due to the `library` bytecode size getting really big and the number mostly represents the deployment gas overhead. The runtime gas saving for this test is actually around `-75582`.

We can also mix and match the techniques used here to avoid the big increase in deployment cost. For example, we can decide to not unroll loops.

It is recommended to add unit and differential tests for this function, especially if you are taking this suggestion into consideration.

Connext: Unrolling implemented. Deployment costs are negligible because this library should just be used in MerkleTreeManager, which should NOT be subject to redeployment (it's entire purpose in being separate contract is maintaining data permanence). Solved in PR 2211.

Spearbit: Verified.

5.4.15 The `insert` function in `Merkle.sol` can be optimized by using YUL, unrolling loops and using the scratch space

Severity: Gas Optimization

Context: Merkle.sol#L76

Description: If we use assembly. the scratch space for hashing and unrolling the loop, we can save some gas.

Recommendation:

```
/**
 * @notice Inserts a given node (leaf) into merkle tree.
 * @dev Reverts if the tree is already full.
 * @param node Element to insert into tree.
 * @return size uint256 Updated count (number of nodes in the tree).
 **/
function insert(Tree storage tree, bytes32 node) internal returns (uint256 size) {
  size = ++tree.count;
  if (size > MAX_LEAVES - 1) revert MerkleLib__insert_treeIsFull();

  assembly {
    let TREE_SLOT := tree.slot

    for {} true {} {

      switch and(size, 1)
      case 0 {
        mstore(0, sload(TREE_SLOT))
        mstore(0x20, node)
        node := keccak256(0, 0x40)
```

```
        }
        default {
          sstore(TREE_SLOT, node)
          break
        }

        switch and(size, shl(1, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 1)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 1), node)
          break
        }

        switch and(size, shl(2, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 2)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 2), node)
          break
        }

        switch and(size, shl(3, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 3)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 3), node)
          break
        }

        switch and(size, shl(4, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 4)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 4), node)
          break
        }

        switch and(size, shl(5, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 5)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 5), node)
          break
        }

        switch and(size, shl(6, 1))
```

```
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 6)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 6), node)
          break
        }

        switch and(size, shl(7, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 7)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 7), node)
          break
        }

        switch and(size, shl(8, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 8)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 8), node)
          break
        }

        switch and(size, shl(9, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 9)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 9), node)
          break
        }

        switch and(size, shl(10, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 10)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 10), node)
          break
        }

        switch and(size, shl(11, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 11)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 11), node)
```

```
      break
    }

    switch and(size, shl(12, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 12)))
      mstore(0x20, node)
      node := keccak256(0, 0x40)
    }
    default {
      sstore(add(TREE_SLOT, 12), node)
      break
    }

    switch and(size, shl(13, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 13)))
      mstore(0x20, node)
      node := keccak256(0, 0x40)
    }
    default {
      sstore(add(TREE_SLOT, 13), node)
      break
    }

    switch and(size, shl(14, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 14)))
      mstore(0x20, node)
      node := keccak256(0, 0x40)
    }
    default {
      sstore(add(TREE_SLOT, 14), node)
      break
    }

    switch and(size, shl(15, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 15)))
      mstore(0x20, node)
      node := keccak256(0, 0x40)
    }
    default {
      sstore(add(TREE_SLOT, 15), node)
      break
    }

    switch and(size, shl(16, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 16)))
      mstore(0x20, node)
      node := keccak256(0, 0x40)
    }
    default {
      sstore(add(TREE_SLOT, 16), node)
      break
    }

    switch and(size, shl(17, 1))
    case 0 {
      mstore(0, sload(add(TREE_SLOT, 17)))
      mstore(0x20, node)
```

```
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 17), node)
  break
}

switch and(size, shl(18, 1))
case 0 {
  mstore(0, sload(add(TREE_SLOT, 18)))
  mstore(0x20, node)
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 18), node)
  break
}

switch and(size, shl(19, 1))
case 0 {
  mstore(0, sload(add(TREE_SLOT, 19)))
  mstore(0x20, node)
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 19), node)
  break
}

switch and(size, shl(20, 1))
case 0 {
  mstore(0, sload(add(TREE_SLOT, 20)))
  mstore(0x20, node)
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 20), node)
  break
}

switch and(size, shl(21, 1))
case 0 {
  mstore(0, sload(add(TREE_SLOT, 21)))
  mstore(0x20, node)
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 21), node)
  break
}

switch and(size, shl(22, 1))
case 0 {
  mstore(0, sload(add(TREE_SLOT, 22)))
  mstore(0x20, node)
  node := keccak256(0, 0x40)
}
default {
  sstore(add(TREE_SLOT, 22), node)
  break
}
```

```
        switch and(size, shl(23, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 23)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 23), node)
          break
        }

        switch and(size, shl(24, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 24)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 24), node)
          break
        }

        switch and(size, shl(25, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 25)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 25), node)
          break
        }

        switch and(size, shl(26, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 26)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 26), node)
          break
        }

        switch and(size, shl(27, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 27)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 27), node)
          break
        }

        switch and(size, shl(28, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 28)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
```

```
          sstore(add(TREE_SLOT, 28), node)
          break
        }

        switch and(size, shl(29, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 29)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 29), node)
          break
        }

        switch and(size, shl(30, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 30)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 30), node)
          break
        }

        switch and(size, shl(31, 1))
        case 0 {
          mstore(0, sload(add(TREE_SLOT, 31)))
          mstore(0x20, node)
          node := keccak256(0, 0x40)
        }
        default {
          sstore(add(TREE_SLOT, 31), node)
          break
        }

        break
      }
    }
  }
```

Runtime gas saved according to test cases:

```
test_RootManager__propagate_shouldSendToSpoke(bytes32) (gas: -215 (-0.057%))
test_RootManager__propagate_shouldSendToAllSpokes(bytes32) (gas: -3176 (-0.120%))
test_Merkle__insert_shouldUpdateCount() (gas: -1750 (-0.688%))
test_MultichainSpokeConnector_processMessage_revertIfWrongDataLength(uint8) (gas: -1630 (-4.194%))
test_messageFlowsWork() (gas: 242592 (7.868%)) <--- due to deployment overhead cost mostly
Overall gas change: 235821 (2.808%)
```

Notes:

1. The above sketch does actually return the updated count, unlike the current implementation.
2. `SHL(a, b)` (where `a` and `b` are constants) in the sketch above (and also in the other comments) is a constant expression and the final result should be inlined by the compiler. You can check this fact or inline the final result yourself and save on compile time.

We can also mix and match the techniques used here to avoid the big increase in deployment cost. For example, we can decide to not unroll loops.

It is recommended to add unit and differential tests for this function, especially if you are taking this suggestion into consideration.

Connext: This is great work, but we'll definitely be converting the `insert` method to use an in-memory reference to the tree, not the tree in storage (for efficiency across multiple insertions, which should be the norm). In order to make this unrolling work, we'd need to switch to just mloading the tree instead in your assembly code. We don't have the internal capability to maintain Yul safely, so adding a large portion to core code paths seems inadvisable

Spearbit: Acknowledged.

5.4.16 `branchRoot` function in `Merkle.sol` can be more optimized by using YUL, unrolling the loop and using the scratch space

Severity: Gas Optimization

Context: Merkle.sol#L149

Description: We can use assembly, unroll the loop in `branchRoot`, and use the scratch space to save gas.

Recommendation:

```
/**
 * @notice Calculates and returns the merkle root for the given leaf `_item`,
 * a merkle branch, and the index of `_item` in the tree.
 * @param _item Merkle leaf
 * @param _branch Merkle proof
 * @param _index Index of `_item` in tree
 * @return _current Calculated merkle root
 **/
function branchRoot(
  bytes32 _item,
  bytes32[TREE_DEPTH] memory _branch,
  uint256 _index
) internal pure returns (bytes32 _current) {
  assembly {
    _current := _item
    let BRANCH_DATA_OFFSET := _branch
    let f

    f := shl(5, and(_index, 1))
    mstore(f, _current)
    mstore(sub(0x20, f), mload(BRANCH_DATA_OFFSET))
    _current := keccak256(0, 0x40)

    f := shl(5, iszero(and(_index, shl(1, 1))))
    mstore(sub(0x20, f), _current)
    mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 1))))
    _current := keccak256(0, 0x40)

    f := shl(5, iszero(and(_index, shl(2, 1))))
    mstore(sub(0x20, f), _current)
    mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 2))))
    _current := keccak256(0, 0x40)

    f := shl(5, iszero(and(_index, shl(3, 1))))
    mstore(sub(0x20, f), _current)
    mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 3))))
    _current := keccak256(0, 0x40)

    f := shl(5, iszero(and(_index, shl(4, 1))))
    mstore(sub(0x20, f), _current)
    mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 4))))
    _current := keccak256(0, 0x40)
```

```
f := shl(5, iszero(and(_index, shl(5, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 5))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(6, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 6))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(7, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 7))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(8, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 8))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(9, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 9))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(10, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 10))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(11, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 11))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(12, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 12))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(13, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 13))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(14, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 14))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(15, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 15))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(16, 1))))
mstore(sub(0x20, f), _current)
mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 16))))
_current := keccak256(0, 0x40)

f := shl(5, iszero(and(_index, shl(17, 1))))
```

```
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 17))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(18, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 18))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(19, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 19))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(20, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 20))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(21, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 21))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(22, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 22))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(23, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 23))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(24, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 24))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(25, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 25))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(26, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 26))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(27, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 27))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(28, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 28))))
        _current := keccak256(0, 0x40)

        f := shl(5, iszero(and(_index, shl(29, 1))))
        mstore(sub(0x20, f), _current)
        mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 29))))
```

```
      _current := keccak256(0, 0x40)

      f := shl(5, iszero(and(_index, shl(30, 1))))
      mstore(sub(0x20, f), _current)
      mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 30))))
      _current := keccak256(0, 0x40)

      f := shl(5, iszero(and(_index, shl(31, 1))))
      mstore(sub(0x20, f), _current)
      mstore(f, mload(add(BRANCH_DATA_OFFSET, shl(5, 31))))
      _current := keccak256(0, 0x40)
    }
  }
```

Gas saved according to test cases (runtime gas saving is even more since some deployment gas overhead has been added due to a bigger code size):

```
test_messageFlowsWork() (gas: -12332 (-0.400%))
test_MultichainSpokeConnector_processMessage_revertIfWrongDataLength(uint8) (gas: -2118 (-5.450%))
Overall gas change: -14450 (-5.850%)
```

It is recommended to add unit and differential tests for this function, especially if you are taking this suggestion into consideration.

Connext: Solved in PR 2211.

Spearbit: Verified.

5.4.17 Replace divisions by powers of `2` by right shifts and multiplications by left shifts

Severity: Gas Optimization

Context:

- Merkle.sol#L96

Description: When a variable `X` is divided (multiplied) by a power of `2` (`C = 2 ^ c`) which is a constant value, the division (multiplication) operation can be replaced by a right (left) shift to save gas.

Recommendation: Replace `X / C` by `X >> c` and `X * C` by `X << c`. Here `C` is a constant which equals to `2 ^ c`.

Connext: Solved in PR 2211.

Spearbit: Verified.

5.4.18 `TypedMemView.castTo` can be optimized by using bitmasks instead of multiple shifts

Severity: Gas Optimization

Context: TypedMemView.sol#L306-L307

Description: `TypedMemView.castTo` uses bit shifts to clear the type flag bits of a `memView`, instead masking can be used. Also an extra `OR` is used to calculate the final view.

Recommendation: Save gas by changing to:

```
newView := or(
   and(memView, LOW_27_BYTES_MASK),
   shl(_27_BYTES_IN_BITS, _newType)
)
```

where `LOW_27_BYTES_MASK` and `_27_BYTES_IN_BITS` should be defined as:

```
uint256 private constant _27_BYTES_IN_BITS = 8 * 27; // <--- also used this named constant where ever
↪  216 is used.
uint256 private constant LOW_27_BYTES_MASK = (1 << _27_BYTES_IN_BITS) - 1;
```

This would remove 1 `OR`, 1 `SHR` and 1 `SHL` and adds 1 `AND`.

Connext: Solved in PR 2510.

Spearbit: Verified.

5.4.19 Make `domain` immutable in Facets

Severity: Gas Optimization

Context: LibConnextStorage.sol#L145

Description: Domain in `Connector.sol` is an immutable variable, however it is defined as a storage variable in `LibConnextStorage.sol`. Also once initialized in `DiamondInit.sol`, it cannot be updated again.

To save gas, `domain` can be made an immutable variable to avoid reading from storage.

Recommendation: There is a tradeoff between maintaining domain only at `AppStorage` vs maintaining it in facets. To make domain immutable:

- Add domain as an immutable variable to `BaseConnextFacet` contract. Also add a constructor to initialize domain variable. Currently, these are the facets using `s.domain`: BridgeFacet.sol, RoutersFacet.sol#L554. Make sure to pass the correct domain value to their constructors.
- `AssetLogic.sol` library also uses `s.domain`. Pass in the domain as a function argument explicitly from Facets instead of reading it from `AppStorage`.

If applying this recommendation, note that the storage layout will change since now `domain` doesn't take a storage slot.

Connext: After finishing the core work involved in addressing this issue, we discovered that the modifications to the unit tests - which currently rely on a mutable domain value - would require a lot of overhaul in order to use an immutable domain value. The amount of work required here would not fit in the current timeline for launch (short-term).

Going to acknowledge this for now. It will be prioritized to be implemented via an upgrade in the future.

Spearbit: Acknowledged.

5.4.20 Cache router balance in `repayAavePortal()`

Severity: Gas Optimization

Context: PortalFacet.sol#L89-L115

Description: `repayAavePortal()` reads `s.routerBalances[msg.sender][local]` twice:

```
if (s.routerBalances[msg.sender][local] < _maxIn) revert
↪  PortalFacet__repayAavePortal_insufficientFunds();
...
s.routerBalances[msg.sender][local] -= amountDebited;
```

This can be cached to only read it once.

Recommendation: Cache `s.routerBalances[msg.sender][local]` at the beginning:

```
uint256 routerBalance = s.routerBalances[msg.sender][local];
if (routerBalance < _maxIn) revert PortalFacet__repayAavePortal_insufficientFunds();
...
s.routerBalances[msg.sender][local] = routerBalance - amountDebited;
```

Connext: Fixed in PR 2504.

Spearbit: Verified.

5.4.21 Unrequired if condition

Severity: Gas Optimization

Context: ConnextPriceOracle.sol#L97

Description: The below `if` condition is not required as price will always be 0. This is because if contract finds direct price for asset it returns early, otherwise if no direct price then tokenPrice is set to 0. This means for the code ahead tokenPrice will currently be 0.

```
function getTokenPrice(address _tokenAddress) public view override returns (uint256, uint256) {
...
    uint256 tokenPrice = assetPrices[tokenAddress].price;
    if (tokenPrice != 0 && ((block.timestamp - assetPrices[tokenAddress].updatedAt) <= VALID_PERIOD)) {
      return (tokenPrice, uint256(PriceSource.DIRECT));
    } else {
      tokenPrice = 0;
    }

    if (tokenPrice == 0) {
...
}
```

Recommendation: Remove the unrequired if condition

```
- if (tokenPrice == 0) {
    tokenPrice = getPriceFromOracle(tokenAddress);
    source = PriceSource.CHAINLINK;
- }
```

Connext: `getTokenPrice` has been rewritten in the following PR 2232 which takes this issue into consideration.

Spearbit: Verified. Related issue

5.4.22 Delete slippage for gas refund

Severity: Gas Optimization

Context: BridgeFacet.sol#L741

Description: Once `s.slippage[_transferId]` is read, it's never read again. It can be deleted to get some gas refund.

Recommendation: Consider adding this after BridgeFacet.sol#L741:

```
delete s.slippage[_transferId];
```

Connext: Fixed in PR 2501.

Spearbit: Verified.

5.4.23 Emit event at the beginning in `_setOwner()`

Severity: Gas Optimization

Context: ProposedOwnable.sol#L166

Description: `_setOwner()` maintains an extra variable `oldOwner` just to emit an event later:

```
function _setOwner(address newOwner) internal {
    address oldOwner = _owner;
    _owner = newOwner;
    _proposedOwnershipTimestamp = 0;
    _proposed = address(0);
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

If this emit is done at the beginning, `oldOwner` can be removed.

Recommendation: Apply this diff:

```
function _setOwner(address newOwner) internal {
+   emit OwnershipTransferred(_owner, newOwner);
-   address oldOwner = _owner;
    _owner = newOwner;
    _proposedOwnershipTimestamp = 0;
    _proposed = address(0);
-   emit OwnershipTransferred(oldOwner, newOwner);
}
```

Connext: Fixed in PR 2503.

Spearbit: Verified.

5.4.24 Simplify the assignment logic of `_params.normalizedIn` in `_xcall`

Severity: Gas Optimization

Context:

- BridgeFacet.sol#L504-L517
- BridgeFacet.sol#L444-L446

Description / Recommendation: When `amount > 0` we should have `asset != address(0)` since otherwise the call would revert:

```
if (_asset == address(0) && _amount != 0) {
  revert BridgeFacet__xcall_nativeAssetNotSupported();
}
```

and when `amount == 0` `_params.normalizedIn` is `0` which is the value passed to `_xcall` from `xcall` or `xcallIntoLocal`. So we can move the calculation for `_params.normalizedIn` into the `if (_amount > 0) {` block.

```
if (_amount > 0) {
  // Transfer funds of input asset to the contract from the user.
  AssetLogic.handleIncomingAsset(_asset, _amount);

  // Swap to the local asset from adopted if applicable.
  // TODO: drop the "IfNeeded", instead just check whether the asset is already local / needs swap here.
  _params.bridgedAmt = AssetLogic.swapToLocalAssetIfNeeded(key, _asset, local, _amount,
↪   _params.slippage);

  // Get the normalized amount in (amount sent in by user in 18 decimals).
  _params.normalizedIn = AssetLogic.normalizeDecimals(ERC20(_asset).decimals(), uint8(18), _amount);
}
```

gas saved according to test cases:

```
test_Connext__bridgeFastOriginLocalToDestinationAdoptedShouldWork() (gas: -39 (-0.001%))
test_Connext__bridgeFastAdoptedShouldWork() (gas: -39 (-0.001%))
test_Connext__unpermissionedCallsWork() (gas: -39 (-0.003%))
test_BridgeFacet__xcall_worksWithPositiveSlippage() (gas: -39 (-0.003%))
test_BridgeFacet__xcall_adoptedTransferWorks() (gas: -39 (-0.003%))
test_Connext__permissionedCallsWork() (gas: -39 (-0.003%))
test_BridgeFacet__xcallIntoLocal_works() (gas: -39 (-0.003%))
test_BridgeFacet__xcall_localTokenTransferWorksWithAdopted() (gas: -39 (-0.003%))
test_Connext__bridgeFastLocalShouldWork() (gas: -39 (-0.004%))
test_BridgeFacet__xcall_localTokenTransferWorksWhenNotAdopted() (gas: -39 (-0.004%))
test_Connext__bridgeSlowLocalShouldWork() (gas: -39 (-0.005%))
test_Connext__zeroValueTransferWithEmptyAssetShouldWork() (gas: -54 (-0.006%))
test_BridgeFacet__xcall_worksIfPreexistingRelayerFee() (gas: -39 (-0.013%))
test_BridgeFacet__xcall_localTokenTransferWorksWithoutAdopted() (gas: -39 (-0.013%))
test_BridgeFacet__xcall_zeroRelayerFeeWorks() (gas: -32 (-0.014%))
test_BridgeFacet__xcall_canonicalTokenTransferWorks() (gas: -39 (-0.014%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -3812 (-0.015%))
test_BridgeFacet__xcall_zeroValueEmptyAssetWorks() (gas: -54 (-0.034%))
test_BridgeFacet__xcall_worksWithoutValue() (gas: -795 (-0.074%))
test_Connext__zeroValueTransferShouldWork() (gas: -761 (-0.091%))
Overall gas change: -6054 (-0.308%)
```

Note, we need to make sure in future updates the value of `_params.normalizedIn == 0` for any invocation of `_xcall`.

Connext: Solved in PR 2511.

Spearbit: Verified.

5.4.25 Simplify `BridgeFacet._sendMessage` by defining `_token` only when needed

Severity: Gas Optimization

Context: BridgeFacet.sol#L895-L909

Description: In `BridgeFacet._sendMessage`, `_local` might be a canonical token that does not necessarily have to follow the `IBridgeToken` interface. But that is not an issue since `_token` is only used when `!_isCanonical`.

Recommendation: We suggest moving the casting `IBridgeToken(_local)` inside that `if` block:

```
// Get the formatted token ID
bytes29 _tokenId = BridgeMessage.formatTokenId(_canonical.domain, _canonical.id);

// Remove tokens from circulation on this chain if applicable.
if (_amount > 0) {
  if (!_isCanonical) {
    // If the token originates on a remote chain, burn the representational tokens on this chain.
    IBridgeToken(_local).burn(address(this), _amount);
  }
  // IFF the token IS the canonical token (i.e. originates on this chain), we lock the input tokens in
↪   escrow
  // in this contract, as an equal amount of representational assets will be minted on the destination
↪   chain.
  // NOTE: The tokens should be in the contract already at this point from xcall.
}
```

As a bonus we also save some gas according to test cases:

```
test_Connext__bridgeFastOriginLocalToDestinationAdoptedShouldWork() (gas: -11 (-0.000%))
test_Connext__bridgeFastAdoptedShouldWork() (gas: -11 (-0.000%))
test_Connext__unpermissionedCallsWork() (gas: -11 (-0.001%))
test_BridgeFacet__xcall_worksWithPositiveSlippage() (gas: -11 (-0.001%))
test_BridgeFacet__xcall_adoptedTransferWorks() (gas: -11 (-0.001%))
test_Connext__permissionedCallsWork() (gas: -11 (-0.001%))
test_BridgeFacet__xcallIntoLocal_works() (gas: -11 (-0.001%))
test_BridgeFacet__xcall_localTokenTransferWorksWithAdopted() (gas: -11 (-0.001%))
test_Connext__bridgeFastLocalShouldWork() (gas: -11 (-0.001%))
test_BridgeFacet__xcall_localTokenTransferWorksWhenNotAdopted() (gas: -11 (-0.001%))
test_BridgeFacet__xcall_worksWithoutValue() (gas: -11 (-0.001%))
test_Connext__zeroValueTransferWithEmptyAssetShouldWork() (gas: -11 (-0.001%))
test_Connext__bridgeSlowLocalShouldWork() (gas: -11 (-0.001%))
test_Connext__zeroValueTransferShouldWork() (gas: -11 (-0.001%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -600 (-0.002%))
test_BridgeFacet__xcall_worksIfPreexistingRelayerFee() (gas: -11 (-0.004%))
test_BridgeFacet__xcall_localTokenTransferWorksWithoutAdopted() (gas: -11 (-0.004%))
test_BridgeFacet__xcall_zeroRelayerFeeWorks() (gas: -9 (-0.004%))
test_BridgeFacet__xcall_canonicalTokenTransferWorks() (gas: -11 (-0.004%))
test_BridgeFacet__xcall_zeroValueEmptyAssetWorks() (gas: -11 (-0.007%))
Overall gas change: -807 (-0.037%)
```

Connext: Solved in PR 2508.

Spearbit: Verified.

5.4.26 Using `BridgeMessage` library in `BridgeFacet._sendMessage` can be avoid to save gas

Severity: Gas Optimization

Context: BridgeFacet.sol#L898-L918

Description: The usage of `BridgeMessage` library to calculate `_tokenId`, `_action`, and finally the formatted message involves lots of unnecessary memory writes, redundant checks, and overall complicates understanding the flow of the codebase.

The `BridgeMessage.formatMessage(_tokenId, _action)` value passed to `IOutbox(s.xAppConnectionManager.home()).dis` is at the end with the current implementation supposed to be:

```
abi.encodePacked(
  _canonical.domain,
  _canonical.id,
  BridgeMessage.Types.Transfer,
  _amount,
  _transferId
);
```

Also, it is redundant that the `BridgeMessage.Types.Transfer` has been passed to `dispatch`. it does not add any information to the message unless `dispatch` also accepts other types. This also adds extra gas overhead due to memory consumption both in the origin and destination domains.

Recommendation: Remove the `BridgeMessage.format...` lines from `_sendMessage` and supply the final message to `dispatch`:

```
function _sendMessage(
  bytes32 _transferId,
  uint32 _destination,
  bytes32 _connextion,
  TokenId memory _canonical,
  address _local,
  uint256 _amount,
  bool _isCanonical
) private returns (bytes32) {
  IBridgeToken _token = IBridgeToken(_local);

  // Remove tokens from circulation on this chain if applicable.
  if (_amount > 0) {
    if (!_isCanonical) {
      // If the token originates on a remote chain, burn the representational tokens on this chain.
      _token.burn(address(this), _amount);
    }
    // IFF the token IS the canonical token (i.e. originates on this chain), we lock the input tokens
↪   in escrow
    // in this contract, as an equal amount of representational assets will be minted on the
↪   destination chain.
    // NOTE: The tokens should be in the contract already at this point from xcall.
  }

  bytes memory _messageBody = abi.encodePacked(
    _canonical.domain,
    _canonical.id,
    BridgeMessage.Types.Transfer,
    _amount,
    _transferId
  );

  // Send message to destination chain bridge router.
  bytes32 _messageHash = IOutbox(s.xAppConnectionManager.home()).dispatch(
    _destination,
    _connextion,
    _messageBody
  );

  // return message hash
  return _messageHash;
}
```

And as we can see by the gas diff in test cases, we will save a lot of gas:

```
test_BridgeFacet__execute_worksWithAdopted() (gas: 21 (0.002%))
```

```
test_BridgeFacet__execute_worksWithNegativeSlippage() (gas: 21 (0.002%))
test_BridgeFacet__execute_worksWithPositiveSlippage() (gas: 21 (0.002%))
test_BridgeFacet__execute_respectsSlippageOverrides() (gas: 21 (0.002%))
test_BridgeFacet__execute_receiveLocalWorks() (gas: 21 (0.002%))
test_BridgeFacet__execute_calldataFailsLoudlyOnFast() (gas: 7 (0.002%))
test_BridgeFacet__execute_handleAlreadyReconciled() (gas: -6 (-0.002%))
test_BridgeFacet__execute_calldataFailureHandledOnSlow() (gas: -6 (-0.002%))
test_BridgeFacet__execute_failIfNoRoutersAndNotReconciled() (gas: -2 (-0.003%))
test_BridgeFacet__execute_failsIfRouterNotApprovedForPortal() (gas: 7 (0.003%))
test_BridgeFacet__execute_failsIfNoLiquidityAndAaveNotEnabled() (gas: 7 (0.004%))
test_BridgeFacet__execute_failIfSignatureInvalid() (gas: 7 (0.005%))
test_BridgeFacet__execute_failIfSequencerSignatureAndSequencerAddressMismatch() (gas: 7 (0.005%))
test_BridgeFacet__execute_failIfSequencerSignatureInvalid() (gas: 7 (0.005%))
test_BridgeFacet__execute_worksOnCanonical() (gas: 21 (0.005%))
test_BridgeFacet__execute_failIfAlreadyExecuted() (gas: 7 (0.005%))
test_BridgeFacet__execute_worksWithAave() (gas: 21 (0.005%))
test_BridgeFacet__execute_successfulCalldata() (gas: 21 (0.006%))
test_BridgeFacet__execute_worksWithDelegateAsSender() (gas: 21 (0.006%))
test_BridgeFacet__execute_worksWithLocalAsAdopted() (gas: 21 (0.006%))
test_BridgeFacet__execute_worksWithUnapprovedIfNoWhitelist() (gas: 21 (0.007%))
test_BridgeFacet__execute_noCalldataWorks() (gas: 21 (0.007%))
test_BridgeFacet__execute_worksWithEmptyCanonicalIfZeroValue() (gas: 21 (0.007%))
test_BridgeFacet__execute_failIfRouterHasInsufficientFunds() (gas: 16 (0.007%))
test_BridgeFacet__execute_failIfRouterNotApproved() (gas: 7 (0.007%))
test_BridgeFacet__execute_failIfSequencerNotApproved() (gas: 7 (0.008%))
test_BridgeFacet__execute_worksWith0Value() (gas: 21 (0.008%))
test_BridgeFacet__execute_failIfPaused() (gas: 7 (0.009%))
test_BridgeFacet__execute_failIfSenderNotApproved() (gas: 7 (0.009%))
test_BridgeFacet__execute_failIfAnyRouterHasInsufficientFunds() (gas: 43 (0.010%))
test_BridgeFacet__execute_failIfAnySignatureInvalid() (gas: 34 (0.014%))
test_BridgeFacet__execute_failIfSequencerSignatureAndRoutersMismatch() (gas: 34 (0.017%))
test_BridgeFacet__execute_failIfPathLengthGreaterThanMaxRouters() (gas: 52 (0.022%))
test_BridgeFacet__execute_multipath() (gas: 129 (0.023%))
test_Connext__bridgeFastOriginLocalToDestinationAdoptedShouldWork() (gas: -5039 (-0.104%))
test_Connext__bridgeFastAdoptedShouldWork() (gas: -5038 (-0.106%))
test_Connext__unpermissionedCallsWork() (gas: -5038 (-0.373%))
test_BridgeFacet__xcall_worksWithPositiveSlippage() (gas: -5038 (-0.431%))
test_BridgeFacet__xcall_adoptedTransferWorks() (gas: -5038 (-0.431%))
test_Connext__permissionedCallsWork() (gas: -5038 (-0.434%))
test_BridgeFacet__xcallIntoLocal_works() (gas: -5038 (-0.438%))
test_BridgeFacet__xcall_localTokenTransferWorksWithAdopted() (gas: -5038 (-0.439%))
test_Connext__bridgeFastLocalShouldWork() (gas: -5038 (-0.457%))
test_BridgeFacet__xcall_localTokenTransferWorksWhenNotAdopted() (gas: -5039 (-0.458%))
test_BridgeFacet__xcall_worksWithoutValue() (gas: -5038 (-0.470%))
test_Connext__zeroValueTransferWithEmptyAssetShouldWork() (gas: -5037 (-0.569%))
test_Connext__bridgeSlowLocalShouldWork() (gas: -5038 (-0.583%))
test_Connext__zeroValueTransferShouldWork() (gas: -5038 (-0.603%))
test_BridgeFacet__xcall_worksIfPreexistingRelayerFee() (gas: -5038 (-1.688%))
test_BridgeFacet__xcall_localTokenTransferWorksWithoutAdopted() (gas: -5038 (-1.696%))
test_BridgeFacet__xcall_zeroRelayerFeeWorks() (gas: -4031 (-1.722%))
test_BridgeFacet__xcall_canonicalTokenTransferWorks() (gas: -5038 (-1.857%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -576237 (-2.224%))
test_BridgeFacet__xcall_zeroValueEmptyAssetWorks() (gas: -5037 (-3.209%))
Overall gas change: -670287 (-18.077%)
```

Connext: Solved in PR 2512.

Spearbit: Verified.

5.4.27 `s.aavePool` can be cached to save gas in `_backLoan`

Severity: Gas Optimization

Context: PortalFacet.sol#L179-L184

Description: `s.aavePool` can be cached to save gas by only reading once from the storage.

Recommendation: Cache `s.aavePool` to save some gas:

```
address aPool = s.aavePool;

// increase allowance
SafeERC20Upgradeable.safeApprove(IERC20Upgradeable(_asset), aPool, 0);
SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable(_asset), aPool, _backing + _fee);

// back loan
IAavePool(aPool).backUnbacked(_asset, _backing, _fee);
```

gas saved according to test cases:

```
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -5610 (-0.022%))
test_PortalFacet__repayAavePortal_shouldWorkUsingSwap() (gas: -250 (-0.023%))
test_PortalFacet__repayAavePortalFor_shouldWork() (gas: -250 (-0.024%))
test_PortalFacet__repayAavePortal_works() (gas: -250 (-0.147%))
Overall gas change: -6360 (-0.215%)
```

Connext: Fixed in PR 2513.

Spearbit: Verified.

5.4.28 <= or >= when comparing a constant can be converted to < or > to save gas

Severity: Gas Optimization

Context: General

Description: In this context, we are doing the following comparison:

```
X <= C // or
X >= C
```

Where `X` is a variable and `C` is a constant expression.

But since the right-hand side of <= (or >=) is the constant expression `C` we can convert <= into < (or >= into >) to avoid extra opcode/bytecodes being produced by the compiler.

Recommendation: To turn <= into <, we just need to increment `C` by `1` and use `C+1` on the right-hand side instead (if `C` is `type(uint256).max` the comparison can be replaced by `true`):

```
X < (C+1)
```

or in the case of >= we need to decrement (when `C` is not `0`, when `C` is `0` the comparison can be replaced by `true` for unsigned values):

```
X > (C-1)
```

We can either calculate `C+1` (or `C-1`)and use it in our comparison or let the compiler inline this value.

Connext: Solved in PR 2514.

Spearbit: Verified.

5.4.29 Use memory's scratch space to `calculateCanonicalHash`

Severity: Gas Optimization

Context:

- AssetLogic.sol#L500-L502
- ArbitrumHubConnector.sol#L167

Description: `calculateCanonicalHash` uses `abi.encode` to prepare a memory chuck to calculate and return a hash value. Since only `2` words of memory are required to calculate the hash we can utilize the memory's scratch space `[0x00, 0x40)` for this regard. Using this approach would prevent from paying for memory expansion costs among other things.

Recommendation: `calculateCanonicalHash` can be changed to:

```
function calculateCanonicalHash(bytes32 _id, uint32 _domain) internal pure returns (bytes32 cHash) {
  assembly {
      mstore(0, _id)
      mstore(0x20, _domain)
      cHash := keccak256(0, 0x40)
  }
}
```

gas diff according to test cases:

```
test_BridgeFacet__execute_worksWithAdopted() (gas: -140 (-0.012%))
test_BridgeFacet__execute_worksWithNegativeSlippage() (gas: -140 (-0.012%))
test_BridgeFacet__execute_worksWithPositiveSlippage() (gas: -140 (-0.012%))
test_BridgeFacet__execute_respectsSlippageOverrides() (gas: -140 (-0.012%))
test_BridgeFacet__xcall_worksWithPositiveSlippage() (gas: -139 (-0.012%))
test_BridgeFacet__xcall_adoptedTransferWorks() (gas: -139 (-0.012%))
test_BridgeFacet__xcallIntoLocal_works() (gas: -139 (-0.012%))
test_BridgeFacet__xcall_localTokenTransferWorksWithAdopted() (gas: -139 (-0.012%))
test_BridgeFacet__execute_receiveLocalWorks() (gas: -139 (-0.012%))
test_PortalFacet__repayAavePortal_failsIfRepayTooMuch() (gas: -139 (-0.013%))
test_PortalFacet__repayAavePortal_shouldWorkUsingSwap() (gas: -140 (-0.013%))
test_BridgeFacet__xcall_localTokenTransferWorksWhenNotAdopted() (gas: -140 (-0.013%))
test_BridgeFacet__xcall_worksWithoutValue() (gas: -139 (-0.013%))
test_PortalFacet__repayAavePortalFor_shouldWork() (gas: -139 (-0.013%))
test_BridgeFacet__xcall_failIfAssetNotSupported() (gas: -139 (-0.014%))
test_PortalFacet__repayAavePortal_failsIfSwapFailed() (gas: -140 (-0.014%))
test_PortalFacet__repayAavePortalFor_failsIfZeroTotalAmount() (gas: -139 (-0.015%))
test_Connext__bridgeFastAdoptedShouldWork() (gas: -831 (-0.017%))
test_Connext__bridgeFastOriginLocalToDestinationAdoptedShouldWork() (gas: -971 (-0.020%))
test_BridgeFacet__execute_multipath() (gas: -139 (-0.025%))
test_BridgeFacet__execute_failIfAnyRouterHasInsufficientFunds() (gas: -129 (-0.031%))
test_BridgeFacet__execute_worksOnCanonical() (gas: -139 (-0.035%))
test_BridgeFacet__execute_worksWithAave() (gas: -140 (-0.036%))
test_BridgeFacet__execute_successfulCalldata() (gas: -140 (-0.039%))
test_BridgeFacet__execute_worksWithDelegateAsSender() (gas: -139 (-0.041%))
test_BridgeFacet__execute_worksWithLocalAsAdopted() (gas: -139 (-0.041%))
test_BridgeFacet__execute_calldataFailsLoudlyOnFast() (gas: -139 (-0.043%))
test_BridgeFacet__execute_worksWithUnapprovedIfNoWhitelist() (gas: -139 (-0.044%))
test_BridgeFacet__execute_noCalldataWorks() (gas: -139 (-0.044%))
test_BridgeFacet__execute_worksWithEmptyCanonicalIfZeroValue() (gas: -140 (-0.045%))
test_BridgeFacet__xcall_worksIfPreexistingRelayerFee() (gas: -139 (-0.047%))
test_BridgeFacet__xcall_localTokenTransferWorksWithoutAdopted() (gas: -139 (-0.047%))
test_BridgeFacet__xcall_zeroRelayerFeeWorks() (gas: -112 (-0.048%))
test_BridgeFacet__execute_handleAlreadyReconciled() (gas: -139 (-0.051%))
test_BridgeFacet__xcall_canonicalTokenTransferWorks() (gas: -139 (-0.051%))
test_BridgeFacet__execute_calldataFailureHandledOnSlow() (gas: -139 (-0.052%))
```

```
test_InboxFacet__reconcile_fastLiquidityMultipathWorks() (gas: -279 (-0.054%))
test_BridgeFacet__execute_worksWith0Value() (gas: -140 (-0.054%))
test_BridgeFacet__execute_failIfRouterHasInsufficientFunds() (gas: -129 (-0.055%))
test_Connext__permissionedCallsWork() (gas: -692 (-0.060%))
test_BridgeFacet__xcall_failIfCapReachedOnCanoncal() (gas: -138 (-0.060%))
test_Connext__unpermissionedCallsWork() (gas: -833 (-0.062%))
test_BridgeFacet__execute_failsIfRouterNotApprovedForPortal() (gas: -129 (-0.062%))
test_BridgeFacet__execute_failsIfNoLiquidityAndAaveNotEnabled() (gas: -129 (-0.072%))
test_RoutersFacet__addLiquidityForRouter_failsIfHitsCap() (gas: -138 (-0.072%))
test_InboxFacet__reconcile_fastLiquiditySingleRouterWorks() (gas: -279 (-0.074%))
test_Connext__zeroValueTransferWithEmptyAssetShouldWork() (gas: -693 (-0.078%))
test_Connext__bridgeSlowLocalShouldWork() (gas: -692 (-0.080%))
test_PortalFacet__repayAavePortal_works() (gas: -140 (-0.082%))
test_InboxFacet__reconcile_failsIfPortalAndNoRouter() (gas: -279 (-0.083%))
test_Connext__zeroValueTransferShouldWork() (gas: -693 (-0.083%))
test_RoutersFacet__addLiquidity_routerIsSender() (gas: -139 (-0.087%))
test_RoutersFacet__addLiquidityForRouter_worksForToken() (gas: -139 (-0.087%))
test_Connext__bridgeFastLocalShouldWork() (gas: -971 (-0.088%))
test_InboxFacet__reconcile_worksPreExecute() (gas: -279 (-0.091%))
test_InboxFacet__reconcile_worksWithLocal() (gas: -279 (-0.091%))
test_InboxFacet__reconcile_failIfCompleted() (gas: -279 (-0.096%))
test_InboxFacet__reconcile_failIfReconciled() (gas: -279 (-0.096%))
test_RoutersFacet__removeRouterLiquidity_worksWithRecipientSet() (gas: -138 (-0.102%))
test_InboxFacet__reconcile_worksWithCanonical() (gas: -279 (-0.113%))
test_RoutersFacet__removeRouterLiquidityFor_works() (gas: -138 (-0.121%))
test_RoutersFacet__removeRouterLiquidity_worksWithToken() (gas: -138 (-0.122%))
test_BridgeFacet__xcall_failInsufficientErc20Tokens() (gas: -139 (-0.123%))
test_BridgeFacet__xcall_failInsufficientErc20Approval() (gas: -138 (-0.125%))
test_PortalFacet__repayAavePortal_failsIfInsufficientAmount() (gas: -139 (-0.160%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -49695 (-0.192%))
test_RoutersFacet__addLiquidityForRouter_failsIfAssetUnapproved() (gas: -138 (-0.199%))
test_AssetLogic__swapToLocalAssetIfNeeded_worksWithAdopted() (gas: -166 (-0.270%))
test_RoutersFacet__addLiquidityForRouter_failsIfRouterUnapproved() (gas: -138 (-0.279%))
test_RoutersFacet__removeRouterLiquidity_failsIfNotEnoughFunds() (gas: -138 (-0.299%))
test_TokenFacet__addStableSwapPool_canDelete() (gas: -138 (-0.313%))
test_TokenFacet__addStableSwapPool_success() (gas: -138 (-0.332%))
test_AssetLogic__swapToLocalAssetIfNeeded_worksWithLocal() (gas: -156 (-1.021%))
test_AssetLogic__swapToLocalAssetIfNeeded_worksIfZero() (gas: -156 (-1.022%))
Overall gas change: -66221 (-7.431%)
```

Reference: View the opcode diff between the 2 different implementations here: godbolt.org

Note: We need to make sure that `uint32 _domain` does not have dirty bytes when passed to this function, otherwise those bytes can change the outcome for the calculated hash.

The same recommendation also applies to `ArbitrumHubConnector._confirmHash`.

Connext: Given the increased risk of generating incorrect canonical hashes if this function is used incorrectly in future upgrades, we've chosen not to implement this suggestion.

Spearbit: Acknowledged.

5.4.30 `isLocalOrigin` can be optimized by using a named return parameter

Severity: Gas Optimization

Context: AssetLogic.sol#L419-L446

Description: `isLocalOrigin` after getting the code size of `_token` returns a comparison result as a `bool`:

```
assembly {
  _codeSize := extcodesize(_token)
}
return _codeSize != 0;
```

This last comparison can be avoided if we use a named return variable since the cast to `bool` type would automatically does the check for us. Currently, the check/comparison is performed twice under the hood.

Note: also see issue "Use `contract.code.length`".

Recommendation: In practice if we had used `_token.code.length != 0` as the return value and omitted the `assembly` block, it should be `6` gas more expensive since the compiler (tested with `0.8.13`) cleans the `_token` address before getting the code length:

```
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
AND
EXTCODESIZE
ISZERO
ISZERO
```

whereas the original implementation would turn out to be:

```
EXTCODESIZE
ISZERO
ISZERO
```

We can actually shave off `6` gas from the original implementation by changing it to:

```
function isLocalOrigin(address _token, AppStorage storage s) internal view returns (bool result) {
  // If the token contract WAS deployed by the bridge, it will be stored in this mapping.
  // If so, the token is NOT of local origin.
  if (s.representationToCanonical[_token].domain != 0) {
    return false;
  }
  // If the contract was NOT deployed by the bridge, but the contract does exist, then it
  // IS of local origin. Returns true if code exists at `_addr`.
  // solhint-disable-next-line no-inline-assembly
  assembly {
    result := extcodesize(_token)
  }
}
```

which the part we care about compiles to:

```
EXTCODESIZE
```

Gas saved according to test cases if we go with my suggestion:

```
test_InboxFacet__reconcile_worksWithCanonical() (gas: -6 (-0.002%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -1600 (-0.006%))
Overall gas change: -1606 (-0.009%)
```

For reference: godbolt.org

Connext: Following the recommendation provided in "Use `contract.code.length`" issue for readability in PR 2486.

Spearbit: Acknowledged.

5.4.31 The branching decision in `AmplificationUtils._getAPrecise` can be removed.

Severity: Gas Optimization

Context:

- AmplificationUtils.sol#L49-L66
- SwapUtilsExternal.sol#L152-L169

Description: `_getAPrecise` uses `if/else` block to compare a_1 to a_0. This comparison is unnecessary if we use a more simplified formula to return the interpolated value of a.

Recommendation: We can gas optimize and simplify the code like the following:

```
/**
 * @notice Return A in its raw precision
 * @dev See the StableSwap paper for details
 * @param self Swap struct to read from
 * @return currentA A parameter in its raw precision form
 */
function _getAPrecise(SwapUtils.Swap storage self) internal view returns (uint256 currentA) {
  uint256 t1 = self.futureATime; // time when ramp is finished
  currentA = self.futureA; // final A value when ramp is finished

  if (block.timestamp < t1) {
    uint256 t0 = self.initialATime; // time when ramp is started
    uint256 a0 = self.initialA; // initial A value when ramp is started
    assembly {
      currentA := div(
        add(
          mul(a0, sub(t1, timestamp())),
          mul(currentA, sub(timestamp(), t0))
        ),
        sub(t1, t0)
      )
    }
  }
}
```

gas savings according to test files:

```
test_Connext__bridgeFastOriginLocalToDestinationAdoptedShouldWork() (gas: -48 (-0.001%))
test_Connext__bridgeFastAdoptedShouldWork() (gas: -64 (-0.001%))
test_StableSwapFacet__removeSwapLiquidityImbalance_failIfPaused() (gas: -8 (-0.003%))
test_StableSwapFacet__removeSwapLiquidity_shouldWork() (gas: -8 (-0.003%))
test_StableSwapFacet__removeSwapLiquidityOneToken_failIfMoreThanLpBalance() (gas: -8 (-0.004%))
test_StableSwapFacet__removeSwapLiquidityImbalance_failIfMoreThanLpBalance() (gas: -16 (-0.007%))
test_SwapAdminFacet__withdrawSwapAdminFess_shouldWorkWithExpectedAmount() (gas: -16 (-0.007%))
test_StableSwapFacet__addSwapLiquidity_shouldWork() (gas: -16 (-0.008%))
test_StableSwapFacet__removeSwapLiquidityOneToken_failIfFrontRun() (gas: -32 (-0.008%))
test_StableSwapFacet__removeSwapLiquidityImbalance_shouldWork() (gas: -24 (-0.008%))
test_StableSwapFacet__removeSwapLiquidityImbalance_failIfFrontRun() (gas: -32 (-0.008%))
test_StableSwapFacet__removeSwapLiquidityOneToken_shouldWork() (gas: -24 (-0.009%))
test_StableSwapFacet__swapExact_shouldWork() (gas: -16 (-0.011%))
test_StableSwapFacet__swap_shouldWork() (gas: -16 (-0.012%))
test_StableSwapFacet__swapExact_failIfNotMinDy() (gas: -24 (-0.013%))
test_StableSwapFacet__swap_failIfNotMinDy() (gas: -24 (-0.014%))
test_StableSwapFacet__removeSwapLiquidityImbalance_failIfNotMatchPoolTokens() (gas: -8 (-0.015%))
test_StableSwapFacet__calculateRemoveSwapLiquidityOneToken_shouldWork() (gas: -16 (-0.020%))
test_StableSwapFacet__calculateSwap_shouldWork() (gas: -16 (-0.027%))
test_AmplificationUtils__rampA_works() (gas: -8 (-0.032%))
test_StableSwapFacet__calculateSwapTokenAmount_shouldWork() (gas: -16 (-0.032%))
test_StableSwapFacet__getSwapVirtualPrice_shouldWork() (gas: -16 (-0.036%))
test_AmplificationUtils__rampA_revertIfFuturePriceTooLarge() (gas: -8 (-0.073%))
test_AmplificationUtils__rampA_revertIfFuturePriceTooSmall() (gas: -8 (-0.074%))
test_StableSwapFacet__getSwapAPrecise() (gas: -16 (-0.150%))
test_StableSwapFacet__getSwapA_shouldWork() (gas: -16 (-0.150%))
test_SwapAdminFacet__stopRampA_shouldWork() (gas: -405 (-0.266%))
test_LibDiamond__initializeDiamondCut_withZeroAcceptanceDelay_works() (gas: -107793 (-0.416%))
test_SwapAdminFacet__rampA_shouldWorkWithDownwards() (gas: -2410 (-0.904%))
test_SwapAdminFacet__rampA_shouldWorkWithUpwards() (gas: -2422 (-0.908%))
test_AmplificationUtils__stopRampA_works() (gas: -397 (-1.459%))
test_AmplificationUtils__getA_works() (gas: -397 (-3.239%))
test_AmplificationUtils__getAPrecise_works() (gas: -397 (-3.255%))
test_AmplificationUtils___getAPrecise_works() (gas: -800 (-5.366%))
Overall gas change: -115525 (-16.537%)
```

Note: one of the original tests `test_SwapAdminFacet__rampA_shouldWorkWithDownwards` actually fails and this is due to rounding errors and how that rounding error is handled using the more concise formula versus the original implementation using branched `if/else` blocks (`a0 ~ a1`). The original implementation would return `4794` for `this.getSwapAPrecise(_canonicalKey)` versus the new solution which returns `4793` and the actual value to a few decimal points is `4793.32027668628`.

A few assumptions that would need to be checked:

- `block.timestamp` is always greater than or equal to `t0`. This is true when `rampA` or `stopRampA` is used. We need to also check the initialization for `t0 = initialATime` and any other potential place that might set that value, the invariant `block.timestamp >= t0` would need to be checked. This is because in the `assembly` block the subtraction would not revert in case of an underflow. This would basically guarantee that $t \in [t_0, t_1)$ inside the outer `if` block.
- We need to check with the ranges of `a0`, `a1`, `t0`, `t1` and `block.timestamp` the numerator in our new concise formula would not overflow (t is `block.timestamp`):

$$\frac{a_0(t_1 - t) + a_1(t - t_0)}{t_1 - t_0}$$

Note, we can also check that $a_0 = a_1$ to return early. It would add gas overhead for cases where they are not equal.

Connext: Solved by PR 2355.

Spearbit: Verified.

5.4.32 Optimize increment in `insert()`

Severity: Gas Optimization

Context: Merkle.sol#L76-L81

Description: The increment `tree.count` in function `insert()` can be optimized.

```
function insert(Tree storage tree, bytes32 node) internal returns (uint256) {
    uint256 size = tree.count + 1;
    ...
    tree.count = size;
    ...
}
```

Recommendation: Consider changing the code to:

```
function insert(Tree storage tree, bytes32 node) internal returns (uint256) {
-    uint256 size = tree.count + 1;
+    uint256 size = ++tree.count;
    ...
-    tree.count = size;
    ...
}
```

Connext: Solved in PR 2211.

Spearbit: Verified.

5.4.33 Optimize calculation in loop of `dequeueVerified`

Severity: Gas Optimization

Context: Queue.sol#L59-L102

Description: The function `dequeueVerified()` can be optimized in the following way: `(block.number - commitBlock >= delay)` is the same as `(block.number - delay >= commitBlock )` And `block.number - delay` is constant so it can be calculated outside of the loop.

Also `(x >= y)` can be replaced by `(!(x < y))` or `(!(y > x))` to save some gas.

```
function dequeueVerified(Queue storage queue, uint256 delay) internal returns (bytes32[] memory) {
    ...
    for (last; last >= first; ) {
        uint256 commitBlock = queue.commitBlock[last];
        if (block.number - commitBlock >= delay) {
            ...
      }
   }
}
```

Recommendation: Consider changing the code to:

```
function dequeueVerified(Queue storage queue, uint256 delay) internal returns (bytes32[] memory) {
    ...
+   uint256 highestAcceptableCommitBlock = block.number - delay;
    for (last; last >= first; ) {
        uint256 commitBlock = queue.commitBlock[last];
-       if (block.number - commitBlock >= delay) {
+       if (!(commitBlock > highestAcceptableCommitBlock)) {
            ...
      }
   }
}
```

Connext: Solved in PR 2228.

Spearbit: Verified.

5.4.34 Cache array length for loops

Severity: Gas Optimization

Context: Diamond.sol#L35, Multicall.sol#L16, StableSwap.sol#L90, LibDiamond.sol#L116, LibDiamond.sol#L141, LibDiamond.sol#L159, SwapAdminFacet.sol#L109-L177,LibDiamond.sol#L174, SwapUtils.sol#L216, SwapUtilsExternal.sol#L253, Merkle.sol#L113, SpokeConnector.sol#L354, SpokeConnector.sol#L370, BytesUtils.sol#L108, BytesUtils.sol#L119, MerkleTrie.sol#L140, MerkleTrie.sol#L227, Merkle.sol#L20, MerklePatriciaProof.sol#L36, MerklePatriciaProof.sol#L99, MerklePatriciaProof.sol#L128, TypedMemView.sol#L813

Description: Fetching array length for each iteration generally consumes more gas compared to caching it in a variable.

Recommendation: All the highlighted code above can be made gas-efficient by caching the array length. For instance:

```
-for (uint256 i; i < xp.length; ) {
+uint256 len = xp.length;
+for (uint256 i; i < len; ) {
```

Connext: Solved in PR 2434.

Spearbit: Verified.

5.4.35 Use custom errors instead of encoding the error message

Severity: Gas Optimization

Context: TypedMemView.sol#L287-L290, TypedMemView.sol#L145, Encoding.sol#L35

Description: `TypedMemView.sol` replicates the functionality provided by custom error with arguments:

```
(, uint256 g) = encodeHex(uint256(typeOf(memView)));
(, uint256 e) = encodeHex(uint256(_expected));
string memory err = string(
    abi.encodePacked("Type assertion failed. Got 0x", uint80(g), ". Expected 0x", uint80(e))
);
revert(err);
```

`encodeHex()` is only used to encode a variable for an error message.

Recommendation:

- Use custom errors instead of encoding variables in revert string.
- Remove `encodeHex()` functions in favour of custom errors.

Connext: Solved in PR 2507.

Spearbit: Verified.

5.4.36 Avoid `OR` with a zero variable

Severity: Gas Optimization

Context: TypedMemView.sol#L306, TypedMemView.sol#L328, TypedMemView.sol#L132

Description: Boolean `OR` operation with a zero variable is a no-op. Highlighted code above perform a boolean `OR` operation with a zero variable which can be avoided:

```
newView := or(newView, shr(40, shl(40, memView)))
...
newView := shl(96, or(newView, _type)) // insert type
...
_encoded |= _nibbleHex(_byte >> 4); // top 4 bits
```

Recommendation: Apply this diff:

```
- newView := or(newView, shr(40, shl(40, memView)))
+ newView := shr(40, shl(40, memView))
...
- newView := shl(96, or(newView, _type)) // insert type
+ newView := shl(96, _type) // insert type
...
- _encoded |= _nibbleHex(_byte >> 4); // top 4 bits
+ _encoded = _nibbleHex(_byte >> 4); // top 4 bits
```

Connext: Solved in PR 2506.

Spearbit: Verified.

5.4.37 Use scratch space instead of free memory

Severity: Gas Optimization

Context: TypedMemView.sol#L651, TypedMemView.sol#L667, TypedMemView.sol#L684

Description: Memory slots 0x00 and 0x20 are scratch space. So any operation in assembly that needs at most 64 bytes of memory to write temporary data can use scratch space.

Functions sha2(), hash160() and hash256() use free memory to write the intermediate hash values. The scratch space can be used here since these values fit in 32 bytes. It saves gas spent on reading the free memory pointer, and memory expansion.

Recommendation: Consider apply this diff which replaces the use of free memory with scratch space:

```
function sha2(bytes29 memView) internal view returns (bytes32 digest) {
 ...
    assembly {
-       let ptr := mload(0x40)
-       pop(staticcall(gas(), 2, _loc, _len, ptr, 0x20)) // sha2
+       pop(staticcall(gas(), 2, _loc, _len, 0x00, 0x20)) // sha2
-       digest := mload(ptr)
+       digest := mload(0x00)
    }
}
...
function hash160(bytes29 memView) internal view returns (bytes20 digest) {
...
    assembly {
-       let ptr := mload(0x40)
-       pop(staticcall(gas(), 2, _loc, _len, ptr, 0x20)) // sha2
+       pop(staticcall(gas(), 2, _loc, _len, 0x00, 0x20)) // sha2
-       pop(staticcall(gas(), 3, ptr, 0x20, ptr, 0x20)) // rmd160
+       pop(staticcall(gas(), 3, 0x00, 0x20, 0x00, 0x20)) // rmd160
-       digest := mload(add(ptr, 0xc)) // return value is 0-prefixed.
+       digest := mload(0xc) // return value is 0-prefixed.
    }
}
...

function hash256(bytes29 memView) internal view returns (bytes32 digest) {
...
    assembly {
-       let ptr := mload(0x40)
-       pop(staticcall(gas(), 2, _loc, _len, ptr, 0x20)) // sha2
+       pop(staticcall(gas(), 2, _loc, _len, 0x00, 0x20)) // sha2
-       pop(staticcall(gas(), 2, ptr, 0x20, ptr, 0x20)) // sha2
+       pop(staticcall(gas(), 2, 0x00, 0x20, 0x00, 0x20)) // sha2
-       digest := mload(ptr)
+       digest := mload(0x00)
    }
}
```

Connext: Removed the above functions in PR 2474

Spearbit: Verified.

5.4.38 Redundant checks in `_processMessageFromRoot()` of `PolygonSpokeConnector`

Severity: Gas Optimization

Context: PolygonSpokeConnector.sol#L61-L74, PolygonSpokeConnector.sol#L78-L82, FxBaseChildTunnel.sol#L32-41

Description: The function `_processMessageFromRoot()` of `PolygonSpokeConnector` does two checks on `sender`, which are the same:

- `validateSender(sender)` checks `sender == fxRootTunnel`
- `_setMirrorConnector()` and `setFxRootTunnel()` set `fxRootTunnel = _mirrorConnector` and `mirrorConnector = _mirrorConnector`
- `require(sender == mirrorConnector, ...)` checks `sender == mirrorConnector` which is the same as `sender == fxRootTunnel`.

Note: the `require` in `_setMirrorConnector()` makes sure the values can't be updated later on. So one of the checks in function `_processMessageFromRoot()` could be removed to save some gas and to make the code easier

to understand.

```
contract PolygonSpokeConnector is SpokeConnector, FxBaseChildTunnel {
    function _processMessageFromRoot(...,  address sender, ... ) validateSender(sender) {
    ...
    require(sender == mirrorConnector, "!sender");
    ...
  }
    function _setMirrorConnector(address _mirrorConnector) internal override {
        require(fxRootTunnel == address(0x0), ...);
        setFxRootTunnel(_mirrorConnector);
   }
}
abstract contract FxBaseChildTunnel is IFxMessageProcessor {
    function setFxRootTunnel(address _fxRootTunnel) public virtual {
        ...
        fxRootTunnel = _fxRootTunnel; // == _mirrorConnector
    }
    modifier validateSender(address sender) {
        require(sender == fxRootTunnel, ...);
        _;
    }
}
```

Recommendation: Remove one of the redundant checks in `_processMessageFromRoot()`.

Connext: Solved in PR 2521.

Spearbit: Verified.

5.4.39 Consider using bitmaps in `_recordOutputAsSpent()` of `ArbitrumHubConnector`

Severity: Gas Optimization

Context: ArbitrumHubConnector.sol#L171-L196, Outbox.sol#L219-L235

Description: The function `_recordOutputAsSpent()` stores status via a mapping of booleans. However the equivalent function `recordOutputAsSpent()` of Arbritrum Nitro uses a mapping of bitmaps to store the status. Doing this saves gas. Note: this saving is possible because the `index` values are neatly ordered.

```
function _recordOutputAsSpent(..., uint256 _index, ...) ... {
    ...
    require(!processed[_index], "spent");
    ...
    processed[_index] = true;
}
```

Arbitrum version:

```
function recordOutputAsSpent(..., uint256 index, ... ) ... {
     ...
    (uint256 spentIndex, uint256 bitOffset, bytes32 replay) = _calcSpentIndexOffset(index);
    if (_isSpent(bitOffset, replay)) revert AlreadySpent(index);
    spent[spentIndex] = (replay | bytes32(1 << bitOffset));
 }
```

Recommendation: Consider using bitmaps in `_recordOutputAsSpent()`.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.4.40 Move `nonReentrant` from `process()` to `proveAndProcess()`

Severity: Gas Optimization

Context: SpokeConnector.sol#L330-L376, SpokeConnector.sol#L492-L553

Description: The function `process()` has a `nonReentrant` modifier. The function `process()` is also `internal` and is only called from `proveAndProcess()`, so it also possible to move the `nonReentrant` modifier to function `proveAndProcess()`. This would save repeatedly setting and unsetting the status of `nonReentrant`, which saves gas.

```
function proveAndProcess(...) ... {
    ...
    for (uint32 i = 0; i < _proofs.length; ) {
        process(_proofs[i].message);
        unchecked { ++i; }
    }
}
function process(bytes memory _message) internal nonReentrant returns (bool _success) {
    ...
}
```

Recommendation: Consider moving the `nonReentrant` from `process()` to `proveAndProcess()`. Note: if in the future a separation between `prove()` and `process()` is made, then the location of the `nonReentrant` modifier will have to be reconsidered.

Connext: Solved in PR 2516.

Spearbit: Verified.

5.5 Informational

5.5.1 OpenZeppelin libraries `IERC20Permit` and `EIP712` are final

Severity: Informational

Context: OZERC20.sol#L10-L11, draft-IERC20Permit.sol#L5-L7, draft-EIP712.sol#L5-L7

Description: The OpenZeppelin libraries have changed `IERC20Permit` and `EIP712` to a final version, so the final versions can be used.

`OZERC20.sol`

```
import "@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol";
import {EIP712} from "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";
```

`draft-IERC20Permit.sol`

```
// EIP-2612 is Final as of 2022-11-01. This file is deprecated.
import "./IERC20Permit.sol";
```

`draft-EIP712.sol`

```
// EIP-712 is Final as of 2022-08-11. This file is deprecated.
import "./EIP712.sol";
```

Recommendation: Consider changing the imports in `OZERC20` to:

```
- import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol";
+ import "@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol";
- import {EIP712} from "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";
+ import {EIP712} from "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
```

Connext: OZ latest release is 4.8.0 and it doesn't include ERC20Permit.sol. Changes to EIP712.sol in PR 2350.

Spearbit: Verified.

5.5.2 Use Foundry's multi-chain tests

Severity: Informational

Context: nxtp contracts

Description: Foundry supports multi-chain testing that can be useful to catch bugs earlier in the development process. Local multi-chain environment can be used to test many scenarios not possible on test chains or in production. Since Connectors are a critical part of NXTP protocol.

Recommendation: Consider integrating Foundry's multi-chain testing. See MakerDAO PR for inspiration.

Connext: Testing framework still a bit mixed, and would require more offchain changes. Ideally a complete migration to a single framework will happen in the future.

Spearbit: Acknowledged.

5.5.3 Risk of chain split

Severity: Informational

Context: Connector.sol#L37, LibConnextStorage.sol#L145

Description: Domains are considered immutable (unless implementation contracts are redeployed). In case of chain splits, both the forks will continue having the same domain and the recipients won't be able to differentiate which source chain of the message.

Recommendation: There are different ways to address this (assuming that the forks have different chain IDs):

- When a chain split is observed off-chain, delete all the facets on the chain you don't want to support.
- When a chain split is observed off-chain, remove all connectors on the chain you don't want to support through `DomainIndex.removeDomain()`.
- Instead of supplying domain information on deployment, consider chain ID for EVM compatible chains. Note that non-EVM chains need to be handled separately.

In the case both the forks use the same chain ID, there will generally be a risk of transaction/signature replay. However, it's highly likely one of them won't be economically successful (like in the case of the short-lived merge split).

In general, it will be good to warn users during these times.

Connext: Acknowledge this is a problem, but using the `block.chainId` is not a valid identifier in non-evm chains. Having an inconsistent identifier structure across domains seems like adding unnecessary complexity.

Spearbit: Acknowledged.

5.5.4 Use zkSync's custom compiler for compiling and (integration) testing

Severity: Informational

Context: General, compiler.

Description: The protocol needs to be deployed on zkSync. For deployment, the contracts would need to be compiled with zkSync's custom compiler. The bytecode generated by the custom Solidity compiler is quite different compared to the original compiler. One thing to note is that cryptographic functions in Solidity are being replaced/inlined to static calls to zkSync's set of system precompile contracts.

Recommendation: Introduce the hardhat tooling provided by the zkSync to compile and test the protocol (locally and on their testnet). Also, note the custom compiler is in development and does not have the full feature set of complication options provided by Solidity (For example, cannot use `--optimize-runs` or `--via-ir`)

`zksolc -h`:

```
The zkEVM Solidity compiler 1.2.0
Compiles the given Solidity input files (or the standard input if none given or "-" is used as a file
↪  name) and outputs
the components specified in the options at standard output or in files in the output directory, if
↪  specified. Imports
are automatically read from the filesystem.

Example: zksolc ERC20.sol --optimize --output-dir './build/'

USAGE:
    zksolc [FLAGS] [OPTIONS] [--] [input-files]...

FLAGS:
        --dump-assembly    Dump the zkEVM assembly of all contracts
        --dump-ethir       Dump the Ethereal Intermediate Representation (IR) of all contracts
        --dump-evm         Dump the EVM legacy assembly Intermediate Representation (IR) of all
        ↪  contracts
        --dump-llvm        Dump the LLVM Intermediate Representation (IR) of all contracts
        --dump-yul         Dump the Yul Intermediate Representation (IR) of all contracts
        --force-evmla      Sets the EVM legacy assembly pipeline forcibly
    -h, --help             Prints help information
        --optimize         Enable the LLVM bytecode optimizer
        --abi              Output ABI specification of the contracts
        --asm              Output zkEVM assembly of the contracts
        --bin              Output zkEVM bytecode of the contracts
        --hashes           Output function signature hashes of the contracts
        --overwrite        Overwrite existing files (used together with -o)
        --standard-json    Switch to Standard JSON input / output mode. Reads from stdin, result is
        ↪  written to stdout
    -V, --version          Prints version information
        --yul              Switch to Yul mode

OPTIONS:
        --allow-paths <allow-paths>          Allow a given path for imports. A list of paths can be
        ↪  supplied by
                                             separating them with a comma
        --base-path <base-path>              Use the given path as the root of the source tree instead
        ↪  of the root of
                                             the filesystem
        --combined-json <combined-json>      Output a single json document containing the specified
        ↪  information.
                                             Available arguments: abi, hashes Example: solc
                                             ↪  --combined-json abi,hashes
        --include-path <include-paths>...    Make an additional source directory available to the
        ↪  default import
```

```
                                        callback. Use this option if you want to import contracts
                                        ↪  whose location is
                                        not fixed in relation to your main source tree, e.g.
                                        ↪  third-party libraries
                                        installed using a package manager. Can be used multiple
                                        ↪  times. Can only be
                                        used if base path has a non-empty value
    -l, --libraries <libraries>...      Direct string or file containing library addresses. Syntax:
                                        <libraryName>=<address> [, or whitespace] ... Address is
                                        ↪  interpreted as a
                                        hex string prefixed by 0x
        --llvm-opt <llvm-options>       Sets the LLVM optimizer options
    -o, --output-dir <output-directory> If given, creates one file per component and contract/file
    ↪  at the specified
                                        directory
        --solc <solc>                   Path to the `solc` executable. By default, the one in
        ↪  $PATH is used

ARGS:
    <input-files>...    The input file paths
```

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.5 Shared logic in `SwapUtilsExternal` and `SwapUtils` can be consolidated or their changes would need to be synched.

Severity: Informational

Context:

- SwapUtilsExternal.sol#L17
- SwapUtils.sol#L18
- AmplificationUtils.sol#L13
- SwapUtils.sol#L715

Description: The `SwapUtilsExternal` library and `SwapUtils` share quite a lot of functions (events/...) logics . The main differences are:

- `SwapUtilsExternal.swap` does not have the following check but `SwapUtils.swap` does:

```
// File: connext/libraries/SwapUtils.sol#L715

require(dx == tokenFrom.balanceOf(address(this)) - beforeBalance, "no fee token support");
```

This is actually one of the big/important diffs between current `SwapUtils` and `SwapUtilsExternal`. Other differences are:

- Some functions are `internal` in `SwapUtils`, but they are `external/public` in `SwapUtilsExternal`.
- `AmplificationUtils` is basically copied in `SwapUtilsExternal` and its functions have been made `external`.
- `SwapUtilsExternal` does not implement `exists`.
- `SwapUtilsExternal` does not implement `swapInternal`.
- The `SwapUtils`'s `Swap` struct has an extra field `key` as do the events in this file.
- Some inconsistent formatting.

Recommendation: It is recommended to consolidate/refactor shared features or at least leave a note for the devs that the edits for `SwapUtils` and `SwapUtilsExternal` need to be synced. The same recommendation applies for `AplificationUtils` and `SwapUtilsExternal`. Related issue: 155

Connext: `Fee token support` fixed in pr 2217. And for `SwapUtilsExternal` and `SwapUtils`, they are actually used in different contracts, individual StableSwap contract and StableSwapFacet. so some functions are a bit different from each other.

Spearbit: Verified and acknowledged.

5.5.6 Document why `< 3s` was chosen as the timestamp deviation cap for price reporting in `setDirectPrice`

Severity: Informational

Context: ConnextPriceOracle.sol#L162-L163

Description: `setDirectPrice` uses the following `require` statement to filter direct price reports by the `owner`.

```
require(_timestamp - block.timestamp < 3, "in future");
```

Only prices with `_timestamp` within `3s` of the current block timestamp are allowed to be registered.

Recommendation: It would be best to document was to why the specific value of `3s` was chosen here. And also document how this reporting system works off-chain. Is the owner an agent or a bot that sources prices and sends transactions regularly to the `setDirectPrice` endpoint?

Connext: Introduced after the following `C4` audit in Issue 205.

Actually, we don't use `PriceOracle` in the current version. so that is out of the scope of the audit. and we don't have any off-chain infrastructure for updating price of the oracle.

Spearbit: Acknowledged.

5.5.7 Document what `IConnectorManager` entities would be passed to `BridgeFacet`

Severity: Informational

Context: BridgeFacet.sol#L239

Description: Document what type of `IConnectorManager` implementations would the `owner` or an `admin` set for the `s.xAppConnectionManager`.

The only examples in the codebase are `SpokeConnectors`.

Recommendation: Add documentation.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.8 Document what an internal swap pool would look like

Severity: Informational

Context:

- SwapAdminFacet.sol#L97-L98
- SwapAdminFacet.sol#L123

Description/Recommendation:

> `@param` _key the hash of the canonical id and domain for token

Document what `token` here refers to, whether if it is just a general `IERC20` token used in cross-chain transactions and whether it can have canonical, adopted, representation, ... versions. That would mean pools are indexed by these tokens (one pool per canonical token per chain/domain).

The number of pooled tokens (let's not confuse this with the `token` mentioned in the above paragraph) is capped at 32. Document what a general set of pooled tokens could look like and how big they could get. Leave a comment as to why the cap of 32 was chosen.

Connext: Solved and documented in PR 2488 & PR 2360.

Spearbit: Verified.

5.5.9 Second `nonReentrant modifier`

Severity: Informational

Context: BridgeFacet.sol#L258-L322, BridgeFacet.sol#L337-L369

Description: A previous version of `xcall()` had a `nonReentrant` modifier. This modifier was removed to enable `execute()` to call `xcall()` to return data to the originator chain. To keep a large part of the original protection it is also possible to use a separate `nonReentrant modifier` (which uses a different storage variable) for `xcall()`/`xcallIntoLocal()`. This way both execute and `xcall()`/`xcallIntoLocal()` can be called once at the most.

```
function xcall(...) ... {
}
function xcallIntoLocal(...) ... {
}
function execute(ExecuteArgs calldata _args) external nonReentrant whenNotPaused returns (bytes32) {
}
```

Recommendation: Consider adding a separate `nonReentrant modifier` to `xcall()`/`xcallIntoLocal()`.

Connext: Solved in PR 2485.

Spearbit: Verified.

5.5.10 Return `0` in `swapToLocalAssetIfNeeded()`

Severity: Informational

Context: AssetLogic.sol#L110-L136

Description: The `return` in function `swapToLocalAssetIfNeeded()` could also return `0`. Which is somewhat more readable and could save some gas. Note: after studying the compiler output it might not actually save gas.

```
function swapToLocalAssetIfNeeded(...) ... {
    if (_amount == 0) {
      return _amount;
    }
    ...
}
```

Recommendation: Double check is actually gas is saved and then consider changing the code to:

```
-return _amount;
+return 0;
```

Connext: Solved in PR 2212.

Spearbit: Verified.

5.5.11 Use `contract.code.length`

Severity: Informational

Context: LibDiamond.sol#L254-L260, AssetLogic.sol#L454-L468

Description: Retrieving the size of a contract is done in assembly, with `extcodesize()`. This can also be done in solidity which is more readable.

Note: assembly might be a bit more gas efficient, especially if optimized even further: see issue "`isLocalOrigin` can be optimized by using a named return parameter".

LibDiamond.sol

```
function enforceHasContractCode(address _contract, string memory _errorMessage) internal view {
    uint256 contractSize;
    assembly {
      contractSize := extcodesize(_contract)
    }
    require(contractSize != 0, _errorMessage);
  }
```

AssetLogic.sol

```
function isLocalOrigin(address _token, AppStorage storage s) internal view returns (bool) {
    ...
    uint256 _codeSize;
    // solhint-disable-next-line no-inline-assembly
    assembly {
      _codeSize := extcodesize(_token)
    }
    return _codeSize != 0;
  }
```

Recommendation: Consider using `contract.code.length`.

Connext: Fixed in PR 2486.

Spearbit: Verified.

5.5.12 `cap` and liquidity tokens

Severity: Informational

Context: SwapUtils.sol#L869-L958

Description: The function `addLiquidity()` also adds tokens to the Connext Diamond contract. If these tokens are the same as canonical tokens it wouldn't play nicely with the `cap` on these tokens. For others tokens it might also be relevant to have a `cap`.

```
function addLiquidity(...) ... {
    ...
    token.safeTransferFrom(msg.sender, address(this), amounts[i]);
    ...
}
```

Recommendation: Doublecheck the `cap` requirements in relation to liquidity tokens.

Connext: There are two primary places the canonical token to be added to the system:

1. Use of `xcall` on canonical domain (canonical assets locked for minting on destination).
2. Providing router liquidity on the canonical domain for fast-path transfers.

On the canonical domain, there is no AMM because the `adopted == canonical == local`.

The main goal of the `cap` is to limit the system-wide security by putting a ceiling on the total number of `next` assets that can be created from the funds provided to `xcall`. This limit should prevent AMMs and routers from providing excess liquidity on the remote domains, as there will be no way to get the representation asset once you reach the cap.

In earlier versions of this feature, we tracked the cap from both of the sources above. However, we removed the tracking from routers (2 above) since it seemed to overcomplicate things, and add to other DoS vectors (see "Malicious routers can temporarily DOS the bridge by depositing a large amount of liquidity "). This means the cap doesn't directly represent the value locked in the system (which it wouldn't have been able to do due to remote AMMs), but keeps the core supply restriction mechanism described above. That should mean that the `cap` and `custodied` values should only be updated on `execute` and `xcall`. Updated in PR 2463.

Spearbit: Verified.

5.5.13 Simplify `_swapAssetOut()`

Severity: Informational

Context:

- AssetLogic.sol#L277-L333
- AssetLogic.sol#L194-L216
- PortalFacet.sol#L79-L119

Description: The function `_swapAssetOut()` has relative complex logic where it first checks the tokens that will be received and then preforms a swap. It prevents reverting by setting the `success` flag. However function `repayAavePortal()` still reverts if this flag is set. The comments show this was meant for `reconcile()`, however repaying the Aave dept in the reconcile phase no longer exists. So `_swapAssetOut()` could just revert if insufficiently tokens are provided. This way it would also be more similar to `_swapAsset()`. This will make the code more readable and safe some gas.

AssetLogic.sol

```
function _swapAssetOut(...) ... returns ( bool success, ...) {
    ...
    if (ipool.exists()) {
        ...
        // Calculate slippage before performing swap.
        // NOTE: This is less efficient then relying on the `swapInternalOut` revert, but makes it
↪   easier
        // to handle slippage failures (this can be called during reconcile, so must not fail).
        ...
        if (_maxIn >= ipool.calculateSwapInv(tokenIndexIn, tokenIndexOut, _amountOut)) {
            success = true;
            amountIn = ipool.swapInternalOut(tokenIndexIn, tokenIndexOut, _amountOut, _maxIn);
        }
    } else {
        ...
        uint256 _amountIn = pool.calculateSwapOutFromAddress(_assetIn, _assetOut, _amountOut);
        if (_amountIn <= _maxIn) {
            success = true;
            ...
            amountIn = pool.swapExactOut(_amountOut, _assetIn, _assetOut, _maxIn, block.timestamp +
↪   3600);
        }
    }
}
```

```
 function swapFromLocalAssetIfNeededForExactOut(...) {
     ...
     return _swapAssetOut(_key, _asset, adopted, _amount, _maxIn);
  }
```

PortalFacet.sol

```
function repayAavePortal(...) {
    ...
    (bool success, ..., ...) = AssetLogic.swapFromLocalAssetIfNeededForExactOut(...);
    if (!success) revert PortalFacet__repayAavePortal_swapFailed();
    ...
  }
```

Recommendation: Consider simplifying `_swapAssetOut()`.

Connext: Solved in PR 2488.

Spearbit: Verified.

5.5.14 Return default false in the function end

Severity: Informational

Context: MerklePatriciaProof.sol#L16

Description: `verify` function is missing a default return value. A return value of `false` can be added on the function end

Recommendation: Return false at the end of function.

```
function verify(
    bytes memory value,
    bytes memory encodedPath,
    bytes memory rlpParentNodes,
    bytes32 root
  ) internal pure returns (bool) {
...
if (traversed == 0) {
          return false;
        }

        pathPtr += traversed;
        nodeKey = bytes32(RLPReader.toUintStrict(currentNodeList[1]));
      } else {
        return false;
      }
    }
return false;
}
```

Connext: This is in the polygon contracts, we will leave them as is.

Spearbit: Acknowledged.

5.5.15 Change occurances of `whitelist` to `allowlist`

Severity: Informational

Context: General

Description: In the codebase, `whitelist` is used to represent entities or objects that are allowed to be used or perform certain tasks. This word is not so accurate/suggestive and also can be offensive.

Recommendation: We can replace all occurrences of `whitelist` with `allowlist` which actually conveys its function more clearly.

For reference: draft-knodel-terminology-02#section-2.2

Connext: Solved in PR 2525.

Spearbit: Verified.

5.5.16 Incorrect comment on `_mirrorConnector`

Severity: Informational

Context: SpokeConnector.sol#L155

Description: The comment on _mirrorConnector is incorrect as this does not denote `address of the spoke connector`

Recommendation: Change the comment for _mirrorConnector to below:

```
- * @param _mirrorConnector The address of the spoke connector.
+ * @param _mirrorConnector The address of the hub connector.
```

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.17 `addStableSwapPool` can have a more suggestive name and also better documentation for the `_stableSwapPool` input parameter is recommended

Severity: Informational

Context:

- TokenFacet.sol#L210
- TokenFacet.sol#L164-L172
- TokenFacet.sol#L192-L198

Description:

1. The name suggests we are adding a new pool, although we are replacing/updating the current one.
2. `_stableSwapPool` needs to implement `IStableSwap` and it is supposed to be an external stable swap pool. It would be best to indicate that and possibly change the parameter input type to `IStableSwap _stableSwapPool`.
3. `_stableSwapPool` provided by the `owner` or an `admin` can have more than just 2 tokens as the `@notice` comment suggests. For example, the pool could have `oUSDC`, `nextUSDC`, `oDAI`, `nextDAI`, ... Also there are no guarantees that the pooled tokens are pegged to each other. There is also a potential of having these pools have malicious or worthless tokens.

What external pools does Connext team uses or is planning to use?

This comment also applies to `setupAsset` and `setupAssetWithDeployedRepresentation`.

Recommendation: Add documentation.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.18 Setting `_cap` on non-canonical domains of an asset is not necessary.

Severity: Informational

Context:

- TokenFacet.sol#L164-L172
- TokenFacet.sol#L192-L198

Description/Recommendation: When `_canonical.domain != s.domain`, setting/saving the supplied `_cap` is not necessary when calling `setupAsset` or `setupAssetWithDeployedRepresentation`. Since on non-canonical domains for an asset the check against `_cap` when adding liquidity for a router or when calling `xcall...` is never performed.

Connext: Decided to revert when supplying a positive `_cap` to `setupAsset` or `setupAssetWithDeployedRepresentation` on a non-canonical domain.

This solution is introduced in PR 2444.

Also note that in the above PR when `s.caps[_key]` is updated `s.custodied[canonical]` is synced to `IERC20(canonical).balanceOf(address(this))`.

Spearbit: Verified.

5.5.19 `_local` has a misleading name in `_addLiquidityForRouter` and `_removeLiquidityForRouter`

Severity: Informational

Context:

- RoutersFacet.sol#L548
- RoutersFacet.sol#L599
- RoutersFacet.sol#L520
- RoutersFacet.sol#L500
- RoutersFacet.sol#L486
- RoutersFacet.sol#L472

Description: The name for `_local` parameter is misleading, since it has been used in `_addLiquidityForRouter`

```
(TokenId memory canonical, bytes32 key) = _getApprovedCanonicalId(_local);
```

and in `_removeLiquidityForRouter`

```
TokenId memory canonical = _getCanonicalTokenId(_local);
```

and we have the following call flow path:

`AssetLogic.getCanonicalTokenId` uses the `adoptedToCanonical` mapping first then check if the input parameter is a canonical token for the current domain, then uses `representationToCanonical` mapping.

So here `_local` could be an adopted token.

Recommendation: It would be best to rename `_local` to `_asset` since in other places in the codebase having the word `local` hints that the token is either a canonical or a representation token.

Connext: `_addLiquidityForRouter` and `_removeLiquidityForRouter` are required to use only `local` tokens and not `adopted`. The changes in the following PR 2489 enforce that.

Also `AssetAllowlist` functionality has been removed from the protocol. Therefore routers only can provide/add liquidity for approved assets.

Spearbit: Verified.

5.5.20 Document `_calculateSwap`'s and `_calculateSwapInv`'s calculations

Severity: Informational

Context:

- SwapUtils.sol#L545
- SwapUtils.sol#L581
- SwapUtilsExternal.sol#L582
- SwapUtilsExternal.sol#L618

Description: In `_calculateSwap`, the `-1` in `dy = xp[tokenIndexTo] - y - 1` is actually important. This is because given no change in the asset balance of all tokens that already satisfy the stable swap invariant (`dx = 0`), `getY` (due to rounding errors) might return:

- `y = xp[tokenIndexTo]` which would in turn make `dy = -1` that would revert the call. This case would need to be investigated.
- `y = xp[tokenIndexTo] - 1` which would in turn make `dy = 0` and so the call would return `(0, 0)`.
- `y = xp[tokenIndexTo] + 1` which would in turn make `dy = -2` that would revert the call. This case would need to be investigated.

And similiarly in `_calculateSwapInv`, doing the same analysis for `+ 1` in `dx = x - xp[tokenIndexFrom] + 1`, if `getYD` returns:

- `xp[tokenIndexFrom] +1`, then `dx = 2`;
- `xp[tokenIndexFrom]`, then `dx = 1`;
- `xp[tokenIndexFrom] - 1`, then `dx = 0`;

Note, that the behavior is different and the call would never revert.

Recommendation: It would be great to analyze and document the decision as to why `+ 1` and `- 1` were chosen in those calculations.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.21 Providing the `from` amount the same as the pool's `from` token balance, one might get a different return value compared to the current pool's `to` balance

Severity: Informational

Context: SwapUtils.sol#L426-L432

Description: Note, due to some imbalance in the asset pool, given `x = xp[tokenIndexFrom]` (aka, no change in asset balance of `tokenIndexFrom` token in the asset pool), we might see a decrease or increase in the asset balance of `tokenIndexTo` to bring back the pool to satisfying the stable swap invariant. One source that can introduce an imbalance is when the scaled amplification coefficient is ramping.

Recommendation: It would be great to leave a comment/warning for the users/devs to make them aware of this fact.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.22 Document what `type 0` means for `TypedMemView`

Severity: Informational

Context: Message.sol#L94

Description: In the following line, `0` is passed as the new type for a `TypedMemView bytes29`

```
_message.slice(PREFIX_LENGTH, _message.len() - PREFIX_LENGTH, 0)
```

But there is no documentation as to what type `0` signifies.

Recommendation: Document what `type 0` means for `TypedMemView`.

Connext: Type `0` just reserves the first byte when you are manipulating the memview (since they should all be typed) -- kind of the same as a null type.

Spearbit: Acknowledged.

5.5.23 Mixed use of `require` statements and custom errors

Severity: Informational

Context: General

Description: The codebase includes a mix of `require` statements and custom errors.

Recommendation: For consistency and in some cases cheaper gas cost, it would be best to use custom errors throughout the whole project.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.24 `WatcherManager` can make `watchers` public instead of having a getter function

Severity: Informational

Context: WatcherManager.sol#L18, WatcherManager.sol#L47

Description: `WatcherManager` has a private mapping `watchers` and a getter function `isWatcher()` to query that mapping. Since `WatcherManager` is not inherited by any other contract, it is safe to make it public to avoid the need of an explicit getter function.

Recommendation: Consider removing `isWatcher()` function, making `watchers` public and renaming it to `isWatcher`. This change does not change `WatcherManager`'s interface.

If applying this recommendation, take into account if you plan to add any contract inheriting from `WatcherManager`. That contract will now have the ability to modify `isWatcher`.

Connext: Solved in PR 2449.

Spearbit: Verified.

5.5.25 Incorrect comment about relation between zero amount and asset

Severity: Informational

Context: BridgeFacet.sol#L514-L515

Description: At BridgeFacet.sol#L514, if `_amount == 0`, `_asset` is allowed to have any user-specified value. `_xcall()` reverts when zero address is specified for `_asset` on a non-zero `_amount`:

```
if (_asset == address(0) && _amount != 0) {
    revert BridgeFacet__xcall_nativeAssetNotSupported();
}
```

However, according to this comment if amount is 0, `_asset` also has to be the zero address which is not true (since it uses IFF):

```
_params.normalizedIn = _asset == address(0)
  ? 0 // we know from assertions above this is the case IFF amount == 0
  : AssetLogic.normalizeDecimals(ERC20(_asset).decimals(), uint8(18), _amount);
```

Recommendation: Update the comment to:

```
_params.normalizedIn = _asset == address(0)
  ? 0 // we know from assertions above that amount is 0 IF amount == 0
  : AssetLogic.normalizeDecimals(ERC20(_asset).decimals(), uint8(18), _amount);
```

Additionally, also think whether a 0 amount transfer should be allowed for a non-zero asset. ERC20 standard allows that, so an argument can be made to allow 0 amount cross-chain transfers. However, reverting in this case results in a cleaner code.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.26 New `Connector` needs to be deployed if `AMB` changes

Severity: Informational

Context: Connector.sol#L42

Description: The AMB address is configured to be immutable. If any of the chain's AMB changes, the `Connector` needs to be deployed.

```
/**
 * @notice Address of the AMB on this domain.
 */
address public immutable AMB;
```

Recommendation: Consider removing the `immutable` modifier and allowing the AMB address to be updated if needed.

Connext: The connectors are designed to be dropped and redeployed. Its likely a new AMB contract would require additional changes anyway.

Spearbit: Acknowledged.

5.5.27 Functions should be renamed

Severity: Informational

Context: ArbitrumHubConnector.sol#L86, OptimismHubConnector.sol#L87, ZkSyncHubConnector.sol#L88

Description: The following functions should be renamed to be aligned with the naming convention of the `fxPortal` contracts.

- `OptimismHubConnector.processMessageFromRoot` to `OptimismHubConnector.processMessageFromChild`
- `ArbitrumHubConnector.processMessageFromRoot` to `ArbitrumHubConnector.processMessageFromChild`
- `ZkSyncHubConnector.processMessageFromRoot` to `ZkSyncHubConnector.processMessageFromChild`

Recommendation: Update the function names accordingly.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.28 Twice function `aggregate()`

Severity: Informational

Context: Multicall.sol#L13, RootManager.sol#L185

Description: Both the contracts `Multicall` and `RootManager` have a function called `aggregate()`. This could be confusing. Contract `Multicall` doesn't seem to be used.

Multicall.sol

```
function aggregate(Call[] memory calls) public view returns (uint256 blockNumber, bytes[] memory
↪  returnData) {
    ...
}
```

RootManager.sol

```
function aggregate(uint32 _domain, bytes32 _inbound) external whenNotPaused onlyConnector(_domain) {
    ...
}
```

Recommendation: Consider renaming one of the functions or deprecate contract `Multicall` if it isn't used.

Connext: Contract `Multicall` is removed in PR 2373.

Spearbit: Verified.

5.5.29 Careful when using `_removeAssetId()`

Severity: Informational

Context: TokenFacet.sol#L354-L383

Description: The function `_removeAssetId()` removes an assets. Although it is called via authorized functions, mistakes could be made.

It there are any representation assets left, they are worthless as they can't be bridged back anymore (unless reinstated via `setupAssetWithDeployedRepresentation()`). The representation assets might also be present and allowed in the `StableSwap`. If so, the owners of the worthless tokens could quickly swap them for real tokens. The canonical tokens will also be locked.

```
function _removeAssetId(...) ... {
    ...
}
```

Recommendation: Check if representation tokens are still present/allowed in the `StableSwap`. Also consider checking how many of the tokens are in circulation.

Connext: Solved in PR 2483.

Spearbit: Verified.

5.5.30 Unused import `IAavePool` in `InboxFacet`

Severity: Informational

Context: InboxFacet.sol#L13

Description: Contract `InboxFacet` imports `IAavePool`, however it doesn't use it.

```
import {IAavePool} from "../interfaces/IAavePool.sol";
```

Recommendation: Remove the import

```
-import {IAavePool} from "../interfaces/IAavePool.sol";
```

Connext: Solved in PR 2491.

Spearbit: Verified.

5.5.31 Use `IERC20Metadata`

Severity: Informational

Context: BridgeFacet.sol#L6, BridgeFacet.sol#L514-L516, AssetLogic.sol#L5

Description: The contract `ERC20` is imported a few times, to be able to access the `decimals()` interface. This interface can also be retrieved from the OpenZeppelin interface `@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol`, which seems more logical.

`BridgeFacet.sol`

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
...
function _xcall(...) ... {
    ...
    ... AssetLogic.normalizeDecimals(ERC20(_asset).decimals(), uint8(18), _amount);
    ...
}
```

`AssetLogic.sol`

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
...
function swapToLocalAssetIfNeeded(...) ... {
    ...
    ... calculateSlippageBoundary(ERC20(_asset).decimals(), ERC20(_local).decimals(), _amount,
↪   _slippage) ...
    ...
}
function swapFromLocalAssetIfNeeded(...) ... {
    ...
    ... calculateSlippageBoundary(uint8(18), ERC20(adopted).decimals(), _normalizedIn, _slippage) ...
    ...
}
```

Recommendation: Consider using `@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol` instead of `ERC20.sol`.

Connext: Solved in PR 2491.

Spearbit: Verified.

5.5.32 Generic name of `proposedTimestamp()`

Severity: Informational

Context: ProposedOwnableFacet.sol#L106-L122

Description: The function `proposedTimestamp()` has a very generic name. As there are other `Timestamp` functions this might be confusing.

```
function proposedTimestamp() public view returns (uint256) {
    return s._proposedOwnershipTimestamp;
}
function routerWhitelistTimestamp() public view returns (uint256) {
    return s._routerWhitelistTimestamp;
}
function assetWhitelistTimestamp() public view returns (uint256) {
    return s._assetWhitelistTimestamp;
}
```

Recommendation: Consider renaming `proposedTimestamp()` to `proposedOwnershipTimestamp()`.

Connext: It is not quite as specific as it could be, but changing the name would create an annoying interface difference with `ProposedOwnable`.

Spearbit: Acknowledged.

5.5.33 Two different `nonce`s

Severity: Informational

Context: LibConnextStorage.sol#L139, SpokeConnector.sol#L131-L133

Description: Both `LibConnextStorage` and `SpokeConnector` define a `nonce`. As the names are very similar this could be confusing.

`LibConnextStorage.sol`

```
struct AppStorage {
    ...
    * @notice Nonce for the contract, used to keep unique transfer ids.
    * @dev Assigned at first interaction (xcall on origin domain).
   uint256 nonce;
    ...
}
```

`SpokeConnector.sol`

```
   * @notice domain => next available nonce for the domain.
   mapping(uint32 => uint32) public nonces;
```

Recommendation: Conder renaming one of the `nonce`s.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.34 Tips to optimize `rootWithCtx`

Severity: Informational

Context: Merkle.sol#L122-L126

Description: To help with the optimization mentioned in the comment of `rootWithCtx()`, here is a way to count the trailing `0`s: graphics.stanford.edu/~seander/bithacks.html#ZerosOnRightModLookup.

```
function rootWithCtx(Tree storage tree, bytes32[TREE_DEPTH] memory _zeroes) internal view returns
↪   (bytes32 _current) {
    ...
    // TODO: Optimization: skip the first N loops where the ith bits are all 0 - start at that
    // depth with zero hashes.
    ...
}
```

Recommendation: If you do want to optimize, the link above can be used. However it is unlikely it will be a large saving.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.35 Use `delete`

Severity: Informational

Context: ProposedOwnable.sol#L161-L167, RoutersFacet.sol#L315

Description: The functions `_setOwner()` and `removeRouter()` clear values by setting them to `0`. Other parts of the code use `delete`. So using `delete` here too would be more consistent.

`ProposedOwnable.sol`

```
function _setOwner(address newOwner) internal {
    ...
    _proposedOwnershipTimestamp = 0;
    _proposed = address(0);
    ...
}
```

`RoutersFacet.sol`

```
function removeRouter(address router) external onlyOwnerOrRouter {
    ...
    s.routerPermissionInfo.routerOwners[router] = address(0);
    ...
    s.routerPermissionInfo.routerRecipients[router] = address(0);
    ...
}
```

Recommendation: Consider changing the code to:

```
function _setOwner(address newOwner) internal {
    ...
-   _proposedOwnershipTimestamp = 0;
+   delete _proposedOwnershipTimestamp;
-   _proposed = address(0);
+   delete _proposed;
    ...
}

function removeRouter(address router) external onlyOwnerOrRouter {
    ...
-   s.routerPermissionInfo.routerOwners[router] = address(0);
+   delete s.routerPermissionInfo.routerOwners[router];
    ...
-   s.routerPermissionInfo.routerRecipients[router] = address(0);
+   delete s.routerPermissionInfo.routerRecipients[router];
    ...
}
```

Connext: Solved in PR 2492.

Spearbit: Verified.

5.5.36 replace usages of `abi.encodeWithSignature` and `abi.encodeWithSelector` with `abi.encodeCall` to ensure typo and type safety

Severity: Informational

Context:

- SpokeConnector.sol#L520-L526

Description:

- When `abi.encodeWithSignature` is used the compiler does not check for mistakes in the signature or the types provided.
- When `abi.encodeWithSelector` is used the compiler does not check for parameter type inconsistencies.

Recommendation: Create an interface for contracts that implement `handle(uint32,uint32,bytes32,bytes)`, for example `IHandle` and instead here use the following to be typo and type safe:

```
bytes memory _calldata = abi.encodeCall(
  IHandle.handle,
  (
    _m.origin(),
    _m.nonce(),
    _m.sender(),
    _m.body().clone()
  )
);
```

This suggestion also applies to other locations where `abi.encodeWithSignature` or `abi.encodeWithSelector` have been used.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.37 `setAggregators` is missing checks against `address(0)`

Severity: Informational

Context: ConnextPriceOracle.sol#L178

Description: `setAggregators` does not check if `tokenAddresses[i]` or `sources[i]` is `address(0)`.

Recommendation: Add checks to avoid potential pitfalls.

Connext: Solved in PR 2232.

Spearbit: Verified.

5.5.38 `setAggregators` can be simplified

Severity: Informational

Context: ConnextPriceOracle.sol#L178

Description: `setAggregators` does not check if `tokenAddresses` length is equal to `sources` to revert early.

Recommendation: Check if `tokenAddresses` length is equal to `sources`.

As an alternative, we can also pass a `struct` like below to avoid needing to check the equality if the length of these 2 arrays.

```
struct AggregatorData {
   address token;
   address source;
}
...
function setAggregators(AggregatorData[] calldata _aggregatorData) external onlyOwner {
```

A good side effect is that the encoded calldata for `setAggregators` would also be smaller.

Connext: Solved in PR 2232.

Spearbit: Verified.

5.5.39 Event is not emitted when an important action happens on-chain

Severity: Informational

Context:

- PortalFacet.sol#L57
- PortalFacet.sol#L64
- SpokeConnector.sol#L220
- SpokeConnector.sol#L418
- SpokeConnector.sol#L480
- ZkSyncHubConnector.sol#L124

Description: No event is emitted when an important action happens on-chain.

Recommendation: We event for the context mentioned in this issue.

- PortalFacet.sol#L57 : No event is emitted when the `aavePool` is changed:

```
event AavePoolChanged(address oldAavePool, address newAavePool, address caller);
```

- PortalFacet.sol#L64 : No event is emitted when `aavePortalFeeNumerator` is updated.
- SpokeConnector.sol#L220 : No event is emitted when `setDelayBlocks` is called.
- SpokeConnector.sol#L418 : No event is emitted when `provenAggregateRoots[_aggregateRoot]` is set.
- SpokeConnector.sol#L480 : No event is emitted when `provenMessageRoots[_messageRoot]` is set.
- ZkSyncHubConnector.sol#L124 : No event is emitted when a new root has been processed.

Connext: Solved in PR 2493.

Spearbit: Verified.

5.5.40 Add unit/fuzz tests to make sure edge cases would not cause an issue in `Queue._length`

Severity: Informational

Context: Queue.sol#L130

Description: It is always assumed `last + 1 >= first`.

Recommendation: It would be great to add unit/fuzz tests to check for this invariant. Adding these tests should/would ensure potential future mistakes.

Connext: Yes, if `first > last`, it should only ever be by 1 (and indicates that the queue is full). We should fuzz to check whether any invariant states are attainable. We could add a require (or custom error) case here where we check this assumption, but I think it's a waste of gas if it's provably unattainable state ('provable' via testing).

Spearbit: Acknowledged.

5.5.41 Consider using `prefix(...)` instead of `slice(0,...)`

Severity: Informational

Context:

- BridgeMessage.sol#L232
- TypedMemView.sol#L493

Description: `tokenId()` calls `TypedMemView.slice()` function to slice the first few bytes from `_message`:

```
return _message.slice(0, TOKEN_ID_LEN, uint40(Types.TokenId));
```

`TypedMemView.prefix()` can also be used here since it achieves the same goal.

Recommendation: Consider using `TypedMemView.prefix()` instead of `slice()`.

Connext: We think `slice` is slightly more readable, and consistent with other `BridgeMessage` implementations.

Spearbit: Acknowledged.

5.5.42 Elaborate `TypedMemView` encoding in comments

Severity: Informational

Context: TypedMemView.sol#L33-L35

Description: `TypedMemView` describes its encoding in comments as:

```
// next 12 are memory address
// next 12 are len
// bottom 3 bytes are empty
```

The comments can be elaborated to make them less ambiguous.

Recommendation:

```
// next 12 bytes are memory address from where the memory view starts
// next 12 bytes are length (in bytes) of the memory view
// bottom 3 bytes are empty
```

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.43 Remove Curve StableSwap paper URL

Severity: Informational

Context: SwapUtils.sol#L63, SwapUtilsExternal.sol#L55

Description: Link to Curve StableSwap paper referenced in comments is no longer active: www.curve.fi/stableswap-paper.pdf The current working URL is curve.fi/files/stableswap-paper.pdf.

Recommendation: Consider removing the URL from comments and just refer to "Curve stableswap paper".

Connext: Fixed in PR 2215.

Spearbit: Verified.

5.5.44 Missing Validations in AmplificationUtils.sol

Severity: Informational

Context: AmplificationUtils.sol#L85-L86

Description:

1. If `initialAPrecise=futureAPrecise` then there will not be any ramping.
2. In `stopRampA` function, `self.futureATime > block.timestamp` can be revised to `self.futureATime >= block.timestamp` since once current timestamp has reached futureATime, futureAprice will be returned always.

Recommendation: Add a check to ensure `initialAPrecise!=futureAPrecise`.

Connext: Issue number 1 is fixed by PR 2220. Issue number 2 was identified as false positive as mentioned in SpearBit section, Kindly reject the resolution pull PR 2219.

Spearbit: The current check in the audit repo is better since, when `self.futureATime == block.timestamp` calling or not calling `stopRampA` will not make any difference. Stopping the ramp is only effective just before we reach `self.futureATime` and $a_1 \neq a_0$.

Also `self.futureATime >= block.timestamp` check would incur more gas usage.

5.5.45 Incorrect PriceSource is returned

Severity: Informational

Context: ConnextPriceOracle.sol#L97-L106

Description: Price Source is returned incorrectly in case of stale prices as shown below

1. `getTokenPrice` function is called with `_tokenAddress T1`.
2. Assume the direct price is stale, so `tokenPrice` is set to 0.

```
uint256 tokenPrice = assetPrices[tokenAddress].price;
    if (tokenPrice != 0 && ((block.timestamp - assetPrices[tokenAddress].updatedAt) <= VALID_PERIOD)) {
      return (tokenPrice, uint256(PriceSource.DIRECT));
    } else {
      tokenPrice = 0;
    }
```

3. Now contract tries to retrieve price from oracle. In case the price is outdated, the returned price will again be 0 and source would be set to `PriceSource.CHAINLINK`.

```
if (tokenPrice == 0) {
      tokenPrice = getPriceFromOracle(tokenAddress);
      source = PriceSource.CHAINLINK;
    }
```

4. Assuming v1PriceOracle is not yet set, contract will simply return the price and source which in this case is 0, PriceSource.CHAINLINK. In this case amount is correct but source is not correct.

```
return (tokenPrice, uint256(source));
```

Recommendation: If the price remains 0 then source returned should be `PriceSource.NA`

Connext: Solved in PR 2232.

Spearbit: Verified.

5.5.46 PriceSource.DEX is never used

Severity: Informational

Context: ConnextPriceOracle.sol#L58

Description: The enum value `DEX` is never used in the contract and can be removed.

Recommendation: Remove `DEX` from `PriceSource` enum

```
enum PriceSource {
    NA,
    DIRECT,
    CHAINLINK,
    V1_ORACLE
  }
```

Connext: Solved in PR 2375.

Spearbit: Verified.

5.5.47 Incorrect comment about `handleOutgoingAsset`

Severity: Informational

Context: AssetLogic.sol#L61

Description: The comment is incorrect as this function does not transfer funds to msg.sender.

```
/**
* @notice Handles transferring funds from the Connext contract to msg.sender.
* @param _asset - The address of the ERC20 token to transfer.
* @param _to - The recipient address that will receive the funds.
* @param _amount - The amount to withdraw from contract.
*/
function handleOutgoingAsset(
    address _asset,
    address _to,
    uint256 _amount
) internal {
```

Recommendation: Update the comment accordingly.

Connext: Solved in PR 2221.

Spearbit: Verified.

5.5.48 SafeMath is not required for Solidity 0.8.x

Severity: Informational

Context: OZERC20.sol

Description: Solidity 0.8.x has a built-in mechanism for dealing with overflows and underflows, so there is no need to use the SafeMath library

Recommendation: Consider removing SafeMath.

Connext: Solved in PR 2350.

Spearbit: Verified.

5.5.49 Use a deadline check modifier in `ProposedOwnable`

Severity: Informational

Context:

- ProposedOwnable.sol#L127-L129
- ProposedOwnable.sol#L151-L153

Description: Any change in ownership through `acceptProposedOwner()` and `renounceOwnership()` has to go through a deadline check:

```
// Ensure delay has elapsed
if ((block.timestamp - _proposedOwnershipTimestamp) <= _delay)
    revert ProposedOwnable__acceptProposedOwner_delayNotElapsed();
```

This check can be extracted out in a modifier for readability.

Recommendation: Replace the highlighted code with a modifier `deadlineCheck`:

```
modifier deadlineCheck() {
    if ((block.timestamp - _proposedOwnershipTimestamp) <= _delay)
        revert ProposedOwnable__OwnershipChange_delayNotElapsed();
    _;
}
```

Connext: Solved in PR 2494.

Spearbit: Verified.

5.5.50 Use `ExcessivelySafeCall` in `SpokeConnector`

Severity: Informational

Context: SpokeConnector.sol#L527-L550

Description: The low-level call code highlighted code above looks to be copied from `ExcessivelySafeCall.sol`.

Recommendation: Consider replacing this low-level call with the function call `ExcessivelySafeCall.excessivelySafeCall()`.

Connext: Solved in PR 2523.

Spearbit: Verified.

5.5.51 `s.LIQUIDITY_FEE_NUMERATOR` might change while a cross-chain transfer is in-flight

Severity: Informational

Context: RoutersFacet.sol#L352

Description: `s.LIQUIDITY_FEE_NUMERATOR` might change while a cross-chain transfer is in-flight.

Recommendation: Leave a note/warning for users that when their cross-chain transfers are in-flight, the liquidity fee might change but it will always be less than or equal to `5%`.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.52 The constant expression for `EMPTY_HASH` can be simplified

Severity: Informational

Context: BaseConnextFacet.sol#L17

Description: `EMPTY_HASH` is a constant with a value equal to: `hex"c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfa` which is the `keccak256` of an empty `bytes`. We can replace this constant hex literal with a more readable alternative.

Recommendation: Use the following constant expression which is more readable that will also be inlined by the compiler

Connext: Solved in PR 2495.

Spearbit: Verified.

5.5.53 Simplify and add more documentation for `getTokenPrice`

Severity: Informational

Context: ConnextPriceOracle.sol#L83

Description: `getTokenPrice` can be simplified and it can try to return early whenever possible.

Recommendation: The following is a more simplified version of `getTokenPrice`

```
function getTokenPrice(address _tokenAddress) public view override returns (uint256, uint256) {
  address tokenAddress = _tokenAddress;

  if (_tokenAddress == address(0)) {
    tokenAddress = wrapped;
  }

  uint256 tokenPrice = assetPrices[tokenAddress].price;
  if (tokenPrice != 0 && ((block.timestamp - assetPrices[tokenAddress].updatedAt) <= VALID_PERIOD)) {
    return (tokenPrice, uint256(PriceSource.DIRECT));
  }

  tokenPrice = getPriceFromOracle(tokenAddress);
  if(tokenPrice != 0) {
    return (tokenPrice, uint256(PriceSource.CHAINLINK));
  }

  if (v1PriceOracle != address(0)) {
    tokenPrice = IPriceOracle(v1PriceOracle).getTokenPrice(tokenAddress);
    if(tokenPrice != 0) {
      return (tokenPrice, uint256(PriceSource.V1_ORACLE));
    }
  }

  return (0, uint256(PriceSource.NA));
}
```

It would be great to document/comment that this function tries to source the price from different sources according to the following order:

1. From the current contract storage / directly. Data set by the owner.
2. From a chainlink aggregator.
3. From the `v1PriceOracle` if set.

Connext: Solved in PR 2344.

Spearbit: Verified.

5.5.54 Remove unused code, files, interfaces, libraries, contracts, ...

Severity: Informational

Context: Mentioned in Recommendation **Description:** The codebase includes code, files, interfaces, libraries, and contracts that are no longer in use.

Recommendation: It would be best to remove unused code to declutter the codebase.

- IGasTokenOracle.sol#L4 : `IGasTokenOracle` is an unused interface.
- ITokenExchange.sol#L9 : `ITokenExchange` is an unused interface.
- ConnextPriceOracle.sol#L22 : `getRoundData` function is unused. Looks like this interface has been copied from AggregatorV3Interface.sol @ chainlink : Also, it might be best to move this interface `AggregatorV3Interface` to the interfaces folder and create a file for it.
- ConnextPriceOracle.sol#L72-L73 : Unsuded events `NewAdmin` and `PriceRecordUpdated`
- Multicall.sol#L7 : Unsued contract `Multicall`
- BridgeToken.sol#L7-L8 : Unused import of `BridgeMessage`
- BaseConnextFacet.sol#L21 : Unused error `BaseConnextFacet__onlyBridgeRouter_notBridgeRouter`

- BridgeFacet.sol#L43 : Unused error `BridgeFacet__xcall_notSupportedAsset`
- BridgeFacet.sol#L45 : Unused error `BridgeFacet__xcall_canonicalAssetNotReceived`
- InboxFacet.sol#L38 : Unused error `InboxFacet__reconcile_notConnext`
- TokenFacet.sol#L21 : Unused error `TokenFacet__addAssetId_nativeAsset`
- RootManager.sol#L8 : Unsued import of `Message`
- TypedMemView.sol#L401-L403 : Unused function `sameType`
- TypeCasts.sol#L10 : Unused function `coerceBytes32`
- TypeCasts.sol#L15 : Unused function `coerceString`

Connext: Solved in PR 2502.

Spearbit: Verified.

5.5.55 `_calculateSwapInv` and `_calculateSwap` can mirror each other's calculations

Severity: Informational

Context:

- SwapUtils.sol#L579-L580
- SwapUtils.sol#L543-L544

Description: `_calculateSwapInv` could have mirrored the implementation of `_calculateSwap`

```
uint256 y = xp[tokenIndexTo] - (dy * multipliers[tokenIndexTo]);
uint256 x = getY(_getAPrecise(self), tokenIndexTo, tokenIndexFrom, y, xp);
```

Or the other way around `_calculateSwap` mirror `_calculateSwapInv` and pick whatever is cheaper.

Recommendation: Consider simplifying the code by mirroring the implementation.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.56 Document that the virtual price of a stable swap pool might not be constant

Severity: Informational

Context: SwapUtils.sol#L403

Description: The virtual price of the LP token is not constant when the amplification coefficient is ramping even when/if token balances stay the same.

Recommendation: Document this issue for users and devs.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.57 Document the reason for picking `d` is the starting point for calculating `getYD` using the Newton's method.

Severity: Informational

Context: SwapUtils.sol#L285

Description: `d` the stable swap invariant passed to `getYD` as a parameter and it is used as the starting point of the Newton method to find a root. This root is the value/price for the `tokenIndex` token that would stabilize the pool so that it would statisfy the stable swap invariant equation.

Recommendation: It would be great to leave comment and document the reasoning behind why `d` was selected as the starting point.

Connext: Comment added in PR 2381.

Spearbit: Verified.

5.5.58 Document the max allowed tokens in stable swap pools used

Severity: Informational

Context: StableSwapFacet.sol, SwapUtilsExternal.sol

Description: Based on the `uint8` type for the indexes of tokens in different stable swap pools, it is inferred that the max possible number of tokens that can exist in a pool is `256`.

There is the following check when initializing internal pools:

```
if (_pooledTokens.length <= 1 || _pooledTokens.length > 32)
  revert SwapAdminFacet__initializeSwap_invalidPooledTokens();
```

This means the internal pools can only have number of pooled tokens in $2, \cdots, 32$.

Recommendation: It would be great if this maximum value was documented.

Note: In Curve pool templates, the token indexes, and the number of tokens are of the type `int128`.

Connext: Solved in PR 2360.

Spearbit: Verified.

5.5.59 Rename `adoptedToLocalPools` to better indicate what it represents

Severity: Informational

Context: LibConnextStorage.sol#L152

Description: `adoptedToLocalPools` is used to keep track of external pools where one can swap between different variations of a token. But one might confuse this mapping as holding internal stable swap pools.

Recommendation: To avoid confusion it would be best to rename this field to a name that better indicates what it represents (for example, `externalPools` or `adoptedToLocalExternalPools`). It is really important to mention that this is a mapping of external pools used by the protocol.

Note, these external pools are only used when there are no internal pools for a token variation.

Connext: Solved in PR 2358.

Spearbit: Verified.

5.5.60 Document the usage of commented mysterious numbers in `AppStorage`

Severity: Informational

Context: LibConnextStorage.sol#L120

Description: Before each `struct AppStorage`'s field definition there is a line comment consisting of only digits

```
// xx
```

One would guess they might be relative slot indexes in the storage (relative to `AppStorage`'s slot). But the numbers are not consistent.

Recommendation: It would be best to add further comments as to what they mean. And if they do represent relative storage slots, some of those would need to be corrected.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.61 `RouterPermissionsManagerInfo` can be packed differently for readability

Severity: Informational

Context: LibConnextStorage.sol#L111-L118

Description: `RouterPermissionsManagerInfo` has multiple fields that are each a mapping of `address` to a different value. The `address` here represents a liquidity router address. It would be more readable to pack these values such that only one mapping is used. This would also indicate how all these mapping have the same shared key which is the router.

Recommendation: The `RouterPermissionsManagerInfo` can be changed into `routerInfos`

```
// note this would pack tighter in storage
struct RouterInfos {
  bool approved;
  bool approvedForPortals;

  address recipient;

  address owner;
  address proposedOwner;
  uint256 proposedTimestamp;
}
...
// field in AppStorage
mapping(address => RouterInfo) routerInfos;
```

Note: This suggestion only applies to new deployments and not to the case when upgrading the contract.

Connext: This should be done in a separate PR, where we manage the configuration separately. See connext/nxtp@6fdac03.

Spearbit: Verified.

5.5.62 Consolidate `TokenId` struct into a file that can be imported in relevant files

Severity: Informational

Context:

- BridgeMessage.sol#L39-L42
- LibConnextStorage.sol#L38-L41

Description: `TokenId` struct is defined both in `BridgeMessage` and `LibConnextStorage` with the same structure/fields. If in future, one would need to update one struct the other one should also be updated in parallel.

Recommendation: To avoid syncing issues, it is recommended to either import `TokenId` from one of the files into the other or extract `TokenId` into a shared file that can be imported to both of the mentioned files.

Connext: Solved in PR 2496.

Spearbit: Verified.

5.5.63 Typos, grammatical and styling errors

Severity: Informational

Context: Mentioned in Recommendation

Description: There are a few typos and grammatical mistakes that can be corrected in the codebase.

Recommendation:

- AssetLogic.sol#L527 : `recieved` should be `received`
- LibConnextStorage.sol#L154 : missing `the`

```
- * @notice Mapping of whitelisted assets on same domain as contract.
+ * @notice Mapping of whitelisted assets on the same domain as the contract.
```

- LibDiamond.sol#L98 : typos, `befor` should be `before` and `elpases` should be `elapses`

```
- // period or befor the delay elpases
+ // period or before the delay elapses
```

- ConnextPriceOracle.sol#L126 : typo > should be <

```
- // answeredInRound > roundId ===> ChainLink Error: Stale price
+ // answeredInRound < roundId ===> ChainLink Error: Stale price
```

- TokenFacet.sol#L290 : Typo `_adooted` should be `_adopted`
- ProposedOwnableFacet.sol#L160 : typo `sounce` should be `source`
- ProposedOwnableFacet.sol#L273 : typo `assingned` should be `assigned`
- RelayerFacet.sol#L71-L72 : typo `router` should be `relayer fee vault`

```
- * @notice Updates the relayer fee router
- * @param _relayerFeeVault The address of the new router
+ * @notice Updates the relayer fee vault
+ * @param _relayerFeeVault The address of the new relayer fee vault
```

- IArbitrumInbox.sol#L5 : typo `messagesto` should be `messages to`
- IArbitrumOutbox.sol#L5 : typo `messagesto` should be `messages to`
- IArbitrumOutbox.sol#L32-L34 : usage of `///` is not consistent with other comment blocks.
- Connector.sol#L87 : Typo `HubConnector` should be `Connector`

- ArbitrumHubConnector.sol#L128 : Typo `none` should be `node's`
- GnosisHubConnector.sol#L49 : Typo `l1` should be `l2`
- RoutersFacet.sol#L194 : Confusing comment.

```
- * @notice Returns the approved router for the given router address
+ * @notice Returns whether a given router address is approved or not
```

- RoutersFacet.sol#L197 : `getRouterApproval` can be renamed to `isApprovedRouter`
- RoutersFacet.sol#L251 : `getRouterApprovalForPortal` can be renamed to `isRouterApprovedForAavePortal`
- PriceOracle.sol#L15 : add return named parameters for clarity and update the `@return` NatSpec accordingly:

```
function getTokenPrice(address token) external view virtual returns (uint256 price, uint256 source);
```

Connext: Partly solved in PR 2297. We'll fix the rest later on.

Spearbit: Acknowledged.

5.5.64 Keep consistent return parameters in `calculateSwapToLocalAssetIfNeeded`

Severity: Informational

Context: AssetLogic.sol#L392-L394

Description: All return paths in `calculateSwapToLocalAssetIfNeeded` except one return `_local` as the 2nd return parameter. It would be best for readability and consistency change the following code to follow the same pattern

```
if (_asset == _local) {
  return (_amount, _asset);
}
```

Recommendation: Change the above few lines to

```
if (_local == _asset) {
  return (_amount, _local);
}
```

Connext: Solved in PR 2357.

Spearbit: Verified.

5.5.65 Fix/add or complete missing NatSpec comments.

Severity: Informational

Context: Mentioned in Recommendation

Description: Some NatSpec comments are either missing or are incomplete.

Recommendation: Add or complete missing NatSpec comments.

- AssetLogic.sol#L184-L205, missing the `@return` NatSpec for the 1st return parameter of type `bool`.
- LibConnextStorage.sol#L113 : Missing NatSpec comment for `approvedForPortalRouters`. Note this field is used to query for approved liquidity routers that can borrow from `Aave` Portal.
- SwapUtils.sol#L137-L145 : It would be best to have the NatSpec `@param` comments in the same order as the supplied parameters to the function.
- IXReceiver.sol#L5 : IXReceiver is missing a NatSpec and it is a crucial component of the protocol. The following should be specially documented:

  - Having a way to recover the tokens send to the receiving contract, when the call to `xReceive` fails.
  - Being able to do an `xcall` to send back results.
  - Limits on the amount of gas that can be used.
- IDiamondCut.sol#L51 : We are not proposing here. This is a `@notice` for `rescindDiamondCut`.
- ProposedOwnableFacet.sol#L125-L127 : The `delay()` is also used for other proposals, not only changing the ownership (removing asset or router whitelists).
- RelayerFacet.sol#L26-L27 : Make the comments more specific

```
- * @notice Emitted when a relayer is added or removed from whitelists
- * @param relayer - The relayer address to be added or removed
+ * @notice Emitted when a relayer is added to the whitelists
+ * @param relayer - The relayer address to be added
```

- RelayerFacet.sol#L33-L34 : Make the comments more specific

```
- * @notice Emitted when a relayer is added or removed from whitelists
- * @param relayer - The relayer address to be added or removed
+ * @notice Emitted when a relayer is removed from the whitelists
+ * @param relayer - The relayer address to be removed
```

- StableSwapFacet.sol#L242: Missing NatSpec comments for `minAmountOut` and `deadline` parameters.
- StableSwapFacet.sol#L266: Missing NatSpec comments for `maxAmountIn` and `deadline` parameters.
- StableSwap.sol#L347: Missing NatSpec comments for `deadline` parameter.
- StableSwap.sol#L366: Missing NatSpec comments for `deadline` parameter.
- IConnectorManager.sol#L18-L21 : `home()` returns an `IOutbox` interface, but in the NatSpec comments there is only a mention of local inbox contract.
- BridgeFacet.sol#L147-L153 : Clarifiy that in the NatSpec, `instance` refers to a `connext remote xApp router` instance. And perhaps the event name `RemoteAdded` and its second input parameter's name `address remote` can be renamed accordingly.
- BridgeMessage.sol#L178-L183 : Document whether `evmId` is being called by some off-chain agent. Otherwise this function has not being used in the codebase and can possibly be removed.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.66 Define and use constants for different literals used in the codebase.

Severity: Informational

Context:

- AssetLogic.sol#L180
- AssetLogic.sol#L260
- AssetLogic.sol#L330
- AssetLogic.sol#L528

Description: Throughout the project, a few literals have been used. It would be best to define a named constant for those. That way it would be more clear the purpose of those values used and also the common literals can be consolidated into one place.

Recommendation:

- AssetLogic.sol#L180 : Define a named constant for `uint8(18)`.

- AssetLogic.sol#L260, AssetLogic.sol#L330 : `3600` could be a constant `ONE_HOUR`.
- AssetLogic.sol#L528 : define a named constant for `10_000`.

Connext: Solved in PR 2497.

Spearbit: Verified.

5.5.67 Enforce using `adopted` for the returned parameter in `swapFromLocalAssetIfNeeded...` for consistency.

Severity: Informational

Context:

- AssetLogic.sol#L162
- AssetLogic.sol#L212
- AssetLogic.sol#L355

Description: The other return paths in `swapFromLocalAssetIfNeeded`, `swapFromLocalAssetIfNeededForExactOut` and `calculateSwapFromLocalAssetIfNeeded` use the `adopted` parameter as one of the return value components. It would be best to have all the return paths do the same thing.

Note `swapFromLocalAssetIfNeeded` and `calculateSwapFromLocalAssetIfNeeded` should always return `(_, adopted)` and `swapFromLocalAssetIfNeededForExactOut` should always return `(_, _, adopted)`.

Recommendation: Change

```
// swapFromLocalAssetIfNeeded
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (_amount, _asset);
}

...

// swapFromLocalAssetIfNeededForExactOut
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (true, _amount, _asset);
}

...

// calculateSwapFromLocalAssetIfNeeded
// If the adopted asset is the local asset, no need to swap.
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (_amount, _asset);
}
```

To

```
// swapFromLocalAssetIfNeeded
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (_amount, adopted); // <--- changed line
}

...

// swapFromLocalAssetIfNeededForExactOut
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (true, _amount, adopted); // <--- changed line
}

...

// calculateSwapFromLocalAssetIfNeeded
// If the adopted asset is the local asset, no need to swap.
address adopted = s.canonicalToAdopted[_key];
if (adopted == _asset) {
  return (_amount, adopted); // <--- changed line
}
```

Connext: Solved in PR 2356.

Spearbit: Verified.

5.5.68 Use interface types for parameters instead of casting to the interface type multiple times

Severity: Informational

Context:

- AssetLogic.sol#L38-L58

Description: Sometimes casting to the interface type has been performed multiple times. It will be cleaner if the parameter is defined as having that interface and avoid unnecessary casts.

Recommendation: Change the input parameter type of `_asset` from `address` to `IERC20` in `handleIncomingAsset`:

```
using SafeERC20 for IERC20;
...
function handleIncomingAsset(IERC20 _asset, uint256 _amount) internal {
  // Sanity check: if amount is 0, do nothing.
  if (_amount == 0) {
    return;
  }
  // Sanity check: asset address is not zero.
  if (address(_asset) == address(0)) {
    revert AssetLogic__handleIncomingAsset_nativeAssetNotSupported();
  }

  // Record starting amount to validate correct amount is transferred.
  uint256 starting = _asset.balanceOf(address(this));

  // Transfer asset to contract.
  _asset.safeTransferFrom(msg.sender, address(this), _amount);

  // Ensure correct amount was transferred (i.e. this was not a fee-on-transfer token).
  if (_asset.balanceOf(address(this)) - starting != _amount) {
    revert AssetLogic__handleIncomingAsset_feeOnTransferNotSupported();
  }
}
```

Also, note that we added `using SafeERC20 for IERC20;` which can be used for all other `transfer` functions `IERC20` types in this library.

Connext: Solved in PR 2498.

Spearbit: Verified.

5.5.69 Be aware of tokens with multiple addresses

Severity: Informational

Context: RoutersFacet.sol#L536-L571

Description: If a token has multiple addresses (see weird erc20) then the token cap might have an unexpected effect, especially if the two addresses have a different cap.

```
function _addLiquidityForRouter(...) ... {
    ...
    if (s.domain == canonical.domain) {
        // Sanity check: caps not reached
        uint256 custodied = IERC20(_local).balanceOf(address(this)) + _amount;
        uint256 cap = s.caps[key];
        if (cap > 0 && custodied > cap) {
            revert RoutersFacet__addLiquidityForRouter_capReached();
        }
    }
    ...
}
```

Recommendation: For tokens with multiple addresses: whitelist only one address or have the same caps is multiple addresses are whitelistes.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.70 Remove old references to `claim`s

Severity: Informational

Context: RelayerFacet.sol#L13-L14, RelayerFacet.sol#L46-L54

Description: The contract `RelayerFacet` still has some references to `claim`s. These are a residue from a previous version and are not used currently.

```
error RelayerFacet__initiateClaim_emptyClaim();
error RelayerFacet__initiateClaim_notRelayer(bytes32 transferId);
event InitiatedClaim(uint32 indexed domain, address indexed recipient, address caller, bytes32[]
↪  transferIds);
event Claimed(address indexed recipient, uint256 total, bytes32[] transferIds);
```

Recommendation: Remove the following:

```
-error RelayerFacet__initiateClaim_emptyClaim();
-error RelayerFacet__initiateClaim_notRelayer(bytes32 transferId);
-event InitiatedClaim(uint32 indexed domain, address indexed recipient, address caller, bytes32[]
↪  transferIds);
-event Claimed(address indexed recipient, uint256 total, bytes32[] transferIds);
```

Connext: Removed.

Spearbit: Verified.

5.5.71 Doublecheck references to `Nomad`

Severity: Informational

Context: Connext contracts

Description: The code refers to `nomad` several times in a way that is currently not accurate. This could be confusing to the maintainers and readers of the code. This includes the following examples:

```
BridgeFacet.sol:419:   * @notice Initiates a cross-chain transfer of funds, calldata, and/or various
↪  named properties using the nomad
BridgeFacet.sol:423:   * assets will be swapped for their local nomad asset counterparts (i.e.
↪  bridgeable tokens) via the configured AMM if
BridgeFacet.sol:424:   * necessary. In the event that the adopted assets *are* local nomad assets, no
↪  swap is needed. The local tokens will
InboxFacet.sol:87:   * @notice Only accept messages from an Nomad Replica contract.
RoutersFacet.sol:533:   * @param _local - The address of the nomad representation of the asset
AssetLogic.sol:102:   * @notice Swaps an adopted asset to the local (representation or canonical) nomad
↪  asset.
AssetLogic.sol:139:   * @notice Swaps a local nomad asset for the adopted asset using the stored stable
↪  swap
AssetLogic.sol:185:   * @notice Swaps a local nomad asset for the adopted asset using the stored stable
↪  swap
AssetLogic.sol:336:   * @notice Calculate amount of tokens you receive on a local nomad asset for the
↪  adopted asset
AssetLogic.sol:375:   * @notice Calculate amount of tokens you receive of a local nomad asset for the
↪  adopted asset
LibConnextStorage.sol:54: * @param receiveLocal - If true, will use the local nomad asset on the
↪  destination instead of adopted.
LibConnextStorage.sol:148:   * @dev Swaps for an adopted asset <> nomad local asset (i.e. POS USDC <>
↪  madUSDC on polygon).
LibConnextStorage.sol:204:   * this domain (the nomad local asset).
LibConnextStorage.sol:268:   * @dev Swaps for an adopted asset <> nomad local asset (i.e. POS USDC <>
↪  madUSDC on polygon)
ConnectorManager.sol:11: * @dev Each nomad router contract has a `XappConnectionClient`, which
↪  references a
```

Recommendation: Doublecheck the references to `Nomad`.

Connext: Solved in PR 2216.

Spearbit: Verified.

5.5.72 Document usage of Nomad domain schema

Severity: Informational

Context: LibConnextStorage.sol#L49-L50, BridgeFacet.sol#L258, BridgeFacet.sol#L291, IXReceiver.sol#L5

Description: The library `LibConnextStorage` specifies that the domains are compatible with the nomad domain schema. However other locations don't mention this. This is especially important during the enrollment of new domains.

```
* @param originDomain - The originating domain (i.e. where `xcall` is called). Must match nomad domain
↪  schema
* @param destinationDomain - The final domain (i.e. where `execute` / `reconcile` are called). Must
↪  match nomad domain schema
struct TransferInfo {
    uint32 originDomain;
    uint32 destinationDomain;
    ...
}
```

Recommendation: Where relevant document the domain schema. This should include `xcall()`, `xcallIntoLocal()`, `xReceive()`.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.73 Router has multiple meanings

Severity: Informational

Context: RoutersFacet.sol#L16, LibConnextStorage.sol#L18, InboxFacet.sol#L217

Description: The term `router` is used for three different concepts. This is confusing for maintainers and readers of the code:

A) The router that provides Liquidity and signs bids

```
* `router` - this is the address that will sign bids sent to the sequencer
```

B) The router that can add new routers of type A (B is a role and the address could be a multisig)

```
/// @notice Enum representing address role
enum Role {
  None,
  Router,
  Watcher,
  Admin
}
```

C) The router that what previously was `BridgeRouter` or `xApp Router`:

```
   * @param _router The address of the potential remote xApp Router
```

Recommendation: Change the names to make them more clear, for example:

- A Router -> LiquidityRouter
- B Router -> LiquidityRouterWhitelistManager

- C Router -> XappRouter

Connext: Solved in PR 2500.

Spearbit: Verified.

5.5.74 Robustness of receiving contract

Severity: Informational

Context: BridgeFacet.sol#L785-L840, BridgeFacet.sol#L756-L770

Description: In the `_reconciled` branch of the code, the functions `_handleExecuteTransaction()`, `_executeCalldata()` and `excessivelySafeCall()` don't revert when the underlying call reverts This seems to be intentional. This underlying revert can happen if there is a bug in the underlying call or if insufficient gas is supplied by the relayer.

Note: if a `delegate` address is specified it can retry the call to try and fix temporary issues.

The receiving contract already has received the tokens via `handleOutgoingAsset()` so must be prepared to handle these tokens. This should be explicitly documented.

```
function _handleExecuteTransaction(...) ... {
    AssetLogic.handleOutgoingAsset(_asset, _args.params.to, _amountOut);
    _executeCalldata(_transferId, _amountOut, _asset, _reconciled, _args.params);
    ...
}
function _executeCalldata(...) ... {
    if (_reconciled) {
        ...
        (success, returnData) = ExcessivelySafeCall.excessivelySafeCall(...);
    } else {
       ...
    }
}
```

Recommendation: Document that the called contract must be so robust it can handle received tokens, even when its function reverts.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.75 Functions can be combined

Severity: Informational

Context: BridgeFacet.sol#L258-L322

Description: Both `xcall` and `xcallIntoLocal` have same code except `receiveLocal` (which is set false for xcall and true for xcallIntoLocal) value. Instead of having these as separate function, a single function can be created which can tweak the functionalities of `xcall` and `xcallIntoLocal` on basis of `receiveLocal` value

Recommendation: Remove the `xcallIntoLocal` function and revise the `xcall` function as below:

```
function xcall(
    uint32 _destination,
    address _to,
    address _asset,
    address _delegate,
    uint256 _amount,
    uint256 _slippage,
    bytes calldata _callData,
    bool _receiveLocal
  ) external payable returns (bytes32) {
...
receiveLocal: _receiveLocal,
...
}
```

Connext: We chose to not clutter the `xcall` interface with this param.

Spearbit: Acknowledged.

5.5.76 Document source of `zeroHashes`

Severity: Informational

Context: Merkle.sol#L173-L241

Description: The hashes which are used in function `zeroHashes()` are not explained, which makes it more difficult to understand and verify the code.

```
function zeroHashes() internal pure returns (bytes32[TREE_DEPTH] memory _zeroes) {
    ...
    // keccak256 zero hashes
    bytes32 internal constant Z_0 =
↪   hex"0000000000000000000000000000000000000000000000000000000000000000";
    ...
    bytes32 internal constant Z_31 =
↪   hex"8448818bb4ae4562849e949e17ac16e0be16688e156b5cf15e098c627c0056a9";
}
```

Recommendation: Document where the constants are derived from. Here is an algorithm that generates them:

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.13;
import "hardhat/console.sol";

contract Generate {
    uint constant DEPOSIT_CONTRACT_TREE_DEPTH = 32;
    bytes32[DEPOSIT_CONTRACT_TREE_DEPTH] zero_hashes;

 constructor()  {
        console.logBytes32(zero_hashes[0]);
        // Compute hashes in empty sparse Merkle tree
        for (uint height = 0; height < DEPOSIT_CONTRACT_TREE_DEPTH - 1; height++) {
            zero_hashes[height + 1] = keccak256(abi.encodePacked(zero_hashes[height],
↪   zero_hashes[height]));
            console.logBytes32(zero_hashes[height + 1]);
        }
    }
}
```

Or use this as documentation:

```
// keccak256 zero hashes
bytes32 internal constant  Z_0 = bytes32(0);
bytes32 internal constant  Z_1 = keccak256(abi.encodePacked( Z_0,  Z_0));
...
bytes32 internal constant Z_31 = keccak256(abi.encodePacked(Z_30, Z_30));
}
```

Or this:

```
Z_i represent the hash values at different heights for a binary tree with leaf values equal to `0`.
Z_i = keccak256(abi.encodePacked(Z_{i-1}, Z_{i-1}));
```

Also add a reference in the comments to the verification of the original code.

Connext: Solved in PR 2211.

Spearbit: Verified.

5.5.77 Document underflow/overflows in `TypedMemView`

Severity: Informational

Context: TypedMemView.sol#L559-L582, TypedMemView.sol#L204-L210

Description: The function `index()` has an overflow when `_bytes == 32` and function `leftMask()` has an underflow when `_len == 0`. These two compensate each other so the end result of `index()` is as expected. As the special case for `_bytes == 0` is also handled, we assume this is intentional.

However this behavior isn't mentioned in the comments, while other underflow/overflows are documented.

```
library TypedMemView {
    function index(
        bytes29 memView,
        uint256 _index,
        uint8 _bytes
    ) internal pure returns (bytes32 result) {
        ...
        unchecked {
            uint8 bitLength = _bytes * 8;
        }
        ...
    }
    function leftMask(uint8 _len) private pure returns (uint256 mask) {
        ...
        mask := sar( sub(_len, 1),  ...   )
        ...
    }
}
```

Recommendation: Add comments about underflow/overflows, for example on the following locations:

```
library TypedMemView {
    function index(
        bytes29 memView,
        uint256 _index,
        uint8 _bytes
    ) internal pure returns (bytes32 result) {
        ...
        unchecked {
            uint8 bitLength = _bytes * 8;  // is 0 when _bytes  == 32
        }
        ...
    }
    function leftMask(uint8 _len) private pure returns (uint256 mask) {
        ...
        mask := sar( sub(_len, 1), ... )  // underflows when _len == 0
        ...
    }
}
```

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.78 Use `while` loops in `dequeueVerified()`

Severity: Informational

Context: Queue.sol#L59-L102

Description: Within function `dequeueVerified()` there are a few `for` loops that mention a variable as there first element. This is a `null` statement and can be removed. After removing, only a `while` condition remains. Replacing the `for` with a `while` would make the code more readable. Also `(x >= y)` can be replaced by `(!(x < y))` or `(!(y > x))` to save some gas.

```
 function dequeueVerified(Queue storage queue, uint256 delay) internal returns (bytes32[] memory) {
    ...
    for (last; last >= first; ) {
        ...
    }
    ...
    for (first; first <= last; ) {
        ...
    }
}
```

Recommendation: Replace the `for` loops with `while` loops:

```
-for (last; last >= first; ) { ... }
+while (!(first > last)) { ... }
```

Connext: Solved in PR PR 2228 and PR 2550.

Spearbit: Verified.

5.5.79 Duplicate functions in `Encoding.sol`

Severity: Informational

Context: Encoding.sol#L35-L74, TypedMemView.sol#L72-L168

Description: `Encoding.sol` defines a few functions already present in `TypedMemView.sol`: `nibbleHex()`, `byteHex()`, `encodeHex()`.

Recommendation: Consider removing `nibbleHex()`, `byteHex()` and `encodeHex()` from both the files as custom errors can be used instead.

Connext: Fixed in PR 2499.

Spearbit: Verified.

5.5.80 Document about two `MerkleTreeManager`'s

Severity: Informational

Context: messaging/Merkle.sol

Description: On the hub domain (e.g. mainnet) there are two `MerkleTreeManager`s, one for the hub and one for the `MainnetSpokeConnector`. This might not be obvious to the casual readers of the code. Accidentally confusing the two would lead to weird issues.

Recommendation: Add a comment in the code explaining there are two `MerkleTreeManager`s on the hub domain.

Connext: Added a comment in PR 2438.

Spearbit: Verified.

5.5.81 Match filename to contract name

Severity: Informational

Context: messaging/Merkle.sol#L11, messaging/libraries/Merkle.sol#L9, ProposedOwnable.sol#L28, ProposedOwnable.sol#L176 OZERC20.sol#L48

Description: Sometimes the name of the `.sol` file is different than the contract name. Also sometimes multiple contracts are defined in the same file. Additionally there are multiple `.sol` files with the same name. This makes it more difficult to find the file containing the contract.

File: `messaging/Merkle.sol`

```
contract MerkleTreeManager is ProposedOwnableUpgradeable {
    ...
}
```

File: `messaging/libraries/Merkle.sol`

```
library MerkleLib {
    ...
}
```

File: `ProposedOwnable.sol`

```
abstract contract ProposedOwnable is IProposedOwnable {
    ...
}
abstract contract ProposedOwnableUpgradeable is Initializable, ProposedOwnable {
    ...
}
```

File: `OZERC20.sol`

```
contract ERC20 is IERC20, IERC20Permit, EIP712 {
    ...
}
```

Recommendation: Split up files containing multiple contracts and match the filename to the contract name. For example:

- `messaging/Merkle.sol` => `MerkleTreeManager.sol`
- `messaging/libraries/Merkle.sol` => `MerkleLib.sol`
- `ProposedOwnable.sol` => split in `ProposedOwnable.sol` and `ProposedOwnableUpgradeable.sol`
- `OZERC20.sol` => perhaps keep it because this based on an OpenZeppelin contract

Connext: Solved in PR 2441.

Spearbit: Verified.

5.5.82 Use `uint40` for type in `TypedMemView`

Severity: Informational

Context: TypedMemView.sol#L347

Description: All internal functions in `TypedMemView` use `uint40` for `type` except `build()`. Since internal functions can be called by inheriting contracts, it's better to provide a consistent interface.

Recommendation: Change `type` from `uint256` to `uint40` in `build()`

```
function build(
-   uint256 _type,
+   uint40 _type,
    uint256 _loc,
    uint256 _len
```

Connext: Leaving as is in original library.

Spearbit: Acknowledged.

5.5.83 Comment in function `typeOf()` is inaccurate

Severity: Informational

Context: TypedMemView.sol#L391

Description: A comment in function `typeOf()` is inaccurate. It says it is shifting 24 bytes, however it is shifting `216 / 8 = 27` bytes.

```
    function typeOf(bytes29 memView) internal pure returns (uint40 _type) {
        assembly {
            ...
            // 216 == 256 - 40
            _type := shr(216, memView) // shift out lower 24 bytes
        }
    }
```

Recommendation: Change the comment to

```
-_type := shr(216, memView) // shift out lower 24 bytes
+_type := shr(216, memView) // shift out lower 27 bytes
```

Connext: Solved in PR 2533.

Spearbit: Verified.

5.5.84 Missing Natspec documentation in `TypedMemView`

Severity: Informational

Context: TypedMemView.sol#L802

Description: `unsafeJoin()`'s Natspec documentation is incomplete as the second argument to function is not documented.

Recommendation: Add Natspec for the second argument `_location`. Here is a suggestion:

```
@param _location  The memory location from where the joined view begins.
```

Connext: Solved in PR 2533.

Spearbit: Verified.

5.5.85 Remove irrelevant comments

Severity: Informational

Context: TypedMemView.sol#L770, SpokeConnector.sol#L499, BridgeFacet.sol#L419

Description:

- Instance 1 - `TypedMemView.sol#L770`

`clone()` has this comment that seems to be copied from `equal()`. This is not applicable to `clone()` and can be deleted.

```
* @dev Shortcuts if the pointers are identical, otherwise compares type and digest.
```

- Instance 2 - `SpokeConnector.sol#L499`

The function `process` of `SpokeConnector` contains comments that are no longer relevant.

```
// check re-entrancy guard
// require(entered == 1, "!reentrant");
// entered = 0;
```

Instance 3 - `BridgeFacet.sol#L419`

Nomad is no longer used within Connext. However, they are still being mentioned in the comments within the codebase.

```
   * @notice Initiates a cross-chain transfer of funds, calldata, and/or various named properties using
↪  the nomad
   * network.
```

Recommendation: Remove or update comments that are no longer relevant.

Connext: Solved in PR 2533.

Spearbit: Verified.

5.5.86 Incorrect comment about TypedMemView encoding

Severity: Informational

Context: TypedMemView.sol#L414

Description: A `TypedMemView` variable of type `bytes29` is encoded as follows:

- First 5 bytes encode a type flag.
- Next 12 bytes point to a memory address.
- Next 12 bytes encode the length of the memory view (in bytes).
- Next 3 bytes are empty.

When shifting a `TypedMemView` variable to the right by 15 bytes (120 bits), the encoded length and the empty bytes are removed. Hence, this comment is incorrect:

```
// 120 bits = 12 bytes (the encoded loc) + 3 bytes (empty low space)
 _loc := and(shr(120, memView), _mask)
```

Recommendation: Apply this diff:

```
-// 120 bits = 12 bytes (the encoded loc) + 3 bytes (empty low space)
+// 120 bits = 12 bytes (the encoded len) + 3 bytes (empty low space)
```

Connext: Solved in PR 2533.

Spearbit: Verified.

5.5.87 Constants can be used in `assembly` blocks directly

Severity: Informational

Context: ExcessivelySafeCall.sol#L127-L133, TypedMemView.sol#L411-L415, TypedMemView.sol#L443-L446

Description: Yul cannot read global variables, but that is not true for a constant variable as its value is embedded in the bytecode. Highlighted code above have the following pattern:

```
uint256 _mask = LOW_12_MASK;  // assembly can't use globals
assembly {
    // solium-disable-previous-line no-inline-assembly
    _len := and(shr(24, memView), _mask)
}
```

Here, `LOW_12_MASK` is a constant which can be used directly in `assembly` code.

Recommendation: Replace mask variables with the constant variables.

Connext: Leaving the same as in source libraries: summa-tx-TypedMemView.sol#L396, summa-tx-TypedMemView.sol#L364, nomad-ExcessivelySafeCall.sol#L127

Spearbit: Acknowledged. Although the reason to variables for constants in the original library is because they support a wide range of solidity versions. Older versions did not support using constants directly in assembly.

5.5.88 Document source of `processMessageFromRoot()`

Severity: Informational

Context: ArbitrumHubConnector.sol#L86-L117

Description: The function `processMessageFromRoot()` of `ArbitrumHubConnector` doesn't contain a comment where it is derived from. Most other functions have a link to the source. Linking to the source would make the function easier to verify and maintain.

Recommendation: Add a comment where `processMessageFromRoot()` is derived from. This seems to be:

- `confirmNode()` RollupCore.sol#L269-L285.
- `executeTransaction()` Outbox.sol#L123-L147.

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.89 Be aware of zombies

Severity: Informational

Context: ArbitrumHubConnector.sol#L119-L138, Node.sol#L20-L23

Description: The function `_validateSendRoot()` of `ArbitrumHubConnector` check that `stakerCount` and `childStakerCount` are larger than `0`. The definition of `stakerCount` and `childStakerCount` document that they could include `zombies`. Its not immediately clear what `zombies` are, but it might be relevant to consider them.

```
contract ArbitrumHubConnector is HubConnector {
    function _validateSendRoot(...) ... {
        ...
        require(node.stakerCount > 0 && node.childStakerCount > 0, "!staked");
      }
}
```

```
// Number of stakers staked on this node. This includes real stakers and zombies
uint64 stakerCount;
// Number of stakers staked on a child node. This includes real stakers and zombies
uint64 childStakerCount;
```

Recommendation: Double check if zombies have to be taken into account.

Connext: Zombies are stakers that have lost their disputes source

We can check for zombies, but because the challenge periods don't align it doesn't fully protect us (i.e. because we don't wait the 7 days, a staker who wasn't a zombie could become one). Instead, we take the weaker assumption that someone has built on the node.

Spearbit: Acknowledged.

5.5.90 Readability of `proveAndProcess()`

Severity: Informational

Context: SpokeConnector.sol#L330-L365

Description: The function `proveAndProcess()` is relatively difficult to understand because it first processes for the case of `i==0` and then does a loop over `i==1..._proofs.length`.

```
function proveAndProcess(...) ... {
    ...
    bytes32 _messageHash = keccak256(_proofs[0].message);
    bytes32 _messageRoot = calculateMessageRoot(_messageHash, _proofs[0].path, _proofs[0].index);
    proveMessageRoot(_messageRoot, _aggregateRoot, _aggregatePath, _aggregateIndex);
    messages[_messageHash] = MessageStatus.Proven;
    for (uint32 i = 1; i < _proofs.length; ) {
        _messageHash = keccak256(_proofs[i].message);
        bytes32 _calculatedRoot = calculateMessageRoot(_messageHash, _proofs[i].path, _proofs[i].index);
        require(_calculatedRoot == _messageRoot, "!sharedRoot");
        messages[_messageHash] = MessageStatus.Proven;
        unchecked { ++i; }
    }
    ...
}
```

Recommendation: Consider integrating the `0` case within the loop to make the code more readable. This example costs slightly more gas. Other variations that cost less gas are also possible but will likely make it less readable:

```
function proveAndProcess(...) ... {
    ...
    bytes32 _messageHash;
    bytes32 _messageRoot;
    for (uint32 i = 0; i < _proofs.length; ) {
        _messageHash = keccak256(_proofs[i].message);
        bytes32 _calculatedRoot = calculateMessageRoot(_messageHash, _proofs[i].path, _proofs[i].index);
        if (i == 0) {
            proveMessageRoot(_calculatedRoot, _aggregateRoot, _aggregatePath, _aggregateIndex);
            _messageRoot = _calculatedRoot;
        }
        require(_calculatedRoot == _messageRoot, "!sharedRoot");
        messages[_messageHash] = MessageStatus.Proven;
        unchecked { ++i; }
    }
    ...
}
```

Connext: Acknowledged.

Spearbit: Acknowledged.

5.5.91 Readability of `checker()`

Severity: Informational

Context: SendOutboundRootResolver.sol#L32-L42

Description: The function `checker()` is relatively difficult to read due to the `else if` chaining of `if` statements. As the `if` statements call `return()`, the `else` isn't necessary and the code can be made more readable.

```
function checker() external view override returns (bool canExec, bytes memory execPayload) {
    bytes32 outboundRoot = CONNECTOR.outboundRoot();
    if ((lastExecuted + EXECUTION_INTERVAL) > block.timestamp) {
      return (false, bytes("EXECUTION_INTERVAL seconds are not passed yet"));
    } else if (lastRootSent == outboundRoot) {
      return (false, bytes("Sent root is the same as the current root"));
    } else {
      execPayload = abi.encodeWithSelector(this.sendMessage.selector, outboundRoot);
      return (true, execPayload);
    }
}
```

Recommendation: Consider changing the code to something like this:

```
function checker() external view override returns (bool canExec, bytes memory execPayload) {
    bytes32 outboundRoot = CONNECTOR.outboundRoot();
    if ((lastExecuted + EXECUTION_INTERVAL) > block.timestamp)
        return (false, bytes("EXECUTION_INTERVAL seconds are not passed yet"));

    if (lastRootSent == outboundRoot)
        return (false, bytes("Sent root is the same as the current root"));

    execPayload = abi.encodeWithSelector(this.sendMessage.selector, outboundRoot);
    return (true, execPayload);
}
```

Connext: `SendOutboundRootResolver.sol` is removed in PR 2199.

Spearbit: Verified.

5.5.92 Use function `addressToBytes32`

Severity: Informational

Context: SpokeConnector.sol#L282-L289, TypeCasts.sol#L31-L33

Description: The function `dispatch()` of `SpokeConnector` contains an explicit conversion from `address` to `bytes32`. There is also a function `addressToBytes32()` that does the same and is more readable.

```
function dispatch(...) ... {
    bytes memory _message = Message.formatMessage(
      ...
      bytes32(uint256(uint160(msg.sender))),
      ...
    );
```

Recommendation: Use function `addressToBytes32`:

```
-bytes32(uint256(uint160(msg.sender))),
+addressToBytes32(msg.sender),
```

Connext: Fixed in PR 2522.

Spearbit: Verified.

6 Appendix: Architecture