From: n-var CONSULTING

To: 0xSquid

Security Review

Coral

Disclaimer

The following is a legal disclaimer for n-var CONSULTING (“n-var”) regarding the analysis contained in this and any other associated or referenced reports (the “Reports”).

Please note that n-var may receive compensation from one or more clients (the “Clients”) for producing the Reports. The Reports may be distributed through other means and should not be considered as an endorsement or indictment of any particular project or team. Furthermore, the Reports do not guarantee the security of any particular project.

It is important to understand that the Reports do not provide any investment advice and should not be interpreted as considering or having any bearing on the potential economics of a token, token sale, or any other product, service, or other asset. Cryptographic tokens carry a high level of technical risk and uncertainty, and the Reports do not provide any warranty or representation to any third party in any respect, including regarding the bug-free nature of code, the business model, or proprietors of any such business model, and the legal compliance of any such business. It is required for any third party to understand that they should not rely on the Reports in any way, including to make any decisions to buy or sell any token, product, service, or other assets. n-var does not owe any duty to any third party by virtue of publishing these Reports.

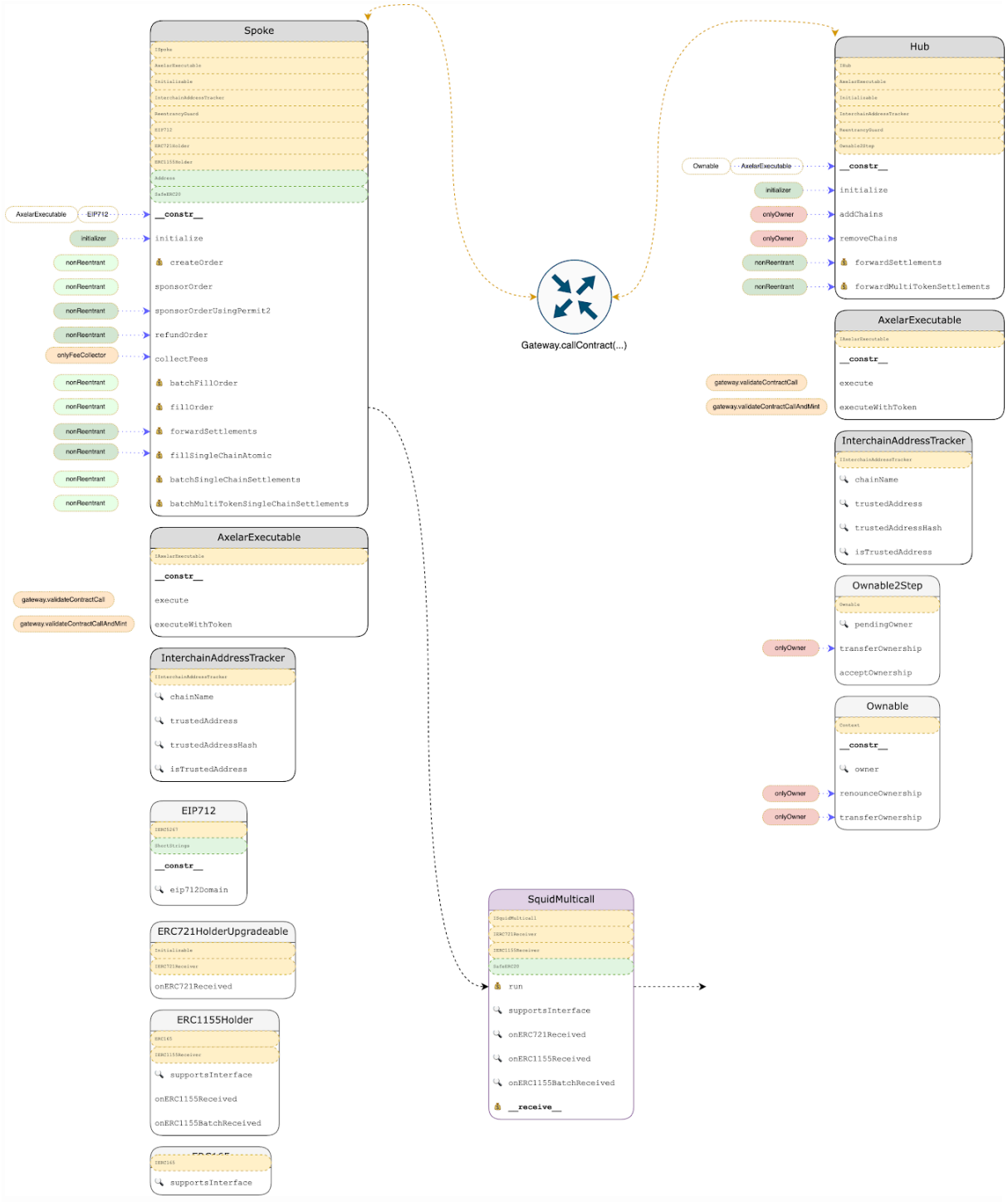
The Reports and the analysis described within are created solely for Clients and published with their consent. The scope of the analysis is limited to a review of code specified by the code repository and identified by a unique commit hash. Any Solidity code presents unique and unquantifiable risks as the Solidity language is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas or dependencies beyond the specified commit hash that could present security risks.

The Reports are made available to third parties on n-var’s website for informational purposes only. n-var does not endorse nor is responsible for the content or operation of any third-party websites linked in the Reports, and shall have no liability to any person or entity for the use of linked third-party websites or their content.

Please note that the content of the Reports is current as of the date appearing on the Report and is subject to change without notice.

By accessing and reading the Reports, you acknowledge and agree to the above disclaimer.

Overview



Note:

- Contracts are deployed via CREATE2 factories and initialized immediately

Executive Summary

- **Review Period**: 5 person days (review + report)
- **Start**: Wed 28 Sep 2024
- **Delivery**: Wed 02 Oct 2024
- **Mitigations Review**:
    - Thu 03 Oct 2024 - https://github.com/0xsquid/squid-coral/compare/fix/audit...5d1e503def43040266f4b2569e26b5e765977dac#
    - Fri 04 Oct 2024 - https://github.com/0xsquid/squid-coral/commit/e477fc5dfa2970cb6c50217a64fbeb59e2f6afe5

Timeline & Scope

- **Scope**: https://github.com/0xsquid/squid-coral/tree/reaudit `@9994db5e2d9a8ba49c8fe6c4fc98012519b6a6cc`

[...] the tip of reaudit has the latest contracts that are frozen on our end.

Here's a compare link from the last time you saw the contracts to the most recent:

https://github.com/0xsquid/squid-coral/compare/old...9994db5e2d9a8ba49c8fe6c4fc98012519b6a6cc

There's a bunch of other stuff that was removed related to a poc that was built, but essentially the main contracts are:

Spoke.sol

ISpoke.sol

Hub.sol

IHub.sol

Utils.sol

- **Engagement Start**: Wed 28 Oct 2024

- **Delivery**: Wed 02 Oct 2024 - Initial Report
- **Mitigations Review**: Thu 03 Oct 2024 - Fixes: https://github.com/0xsquid/squid-coral/compare/fix/audit…5d1e503def43040266f4b2569e26b5e765977dac#
- **Mitigations Review Update**: Fri 04 Oct 2024 - Fixes: https://github.com/0xsquid/squid-coral/commit/e477fc5dfa2970cb6c50217a64fbeb59e2f6afe5

Files in Scope:

- `Spoke.sol`
- `ISpoke.sol`
- `Hub.sol`
- `IHub.sol`
- `Utils.sol`

Findings

Severity High

[HIGH][✔ fixed] Spoke - `refundOrder` refunds wrong address

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/b59f0a709bccfb4c592f6abaf174d92e0f90f71f and https://github.com/0xsquid/squid-coral/commit/e477fc5dfa2970cb6c50217a64fbeb59e2f6afe5 by changing the refund address to `order.fromAddress`.

According to the natspec documentation of `refundOrder`, funds refunded should be transferred to `order.fromAddress`. However, in code it is refunded to `order.toAddress` which seems wrong. Especially, as in the `sponsorOrder` case, funds are escrowed from `order.fromAddress` on order creation and one would expect funds be returned to that address on refund.

contracts/interfaces/ISpoke.sol (Lines 226-234):

```
/// @notice Refunds an eligible Order, transferring the order.fromAmount of order.fromToken from
/// the Spoke.sol contract to the order.fromAddress.
/// @dev Orders can be instantly refunded by the order.filler if the filler decides they don't wish
/// to fill the order.
/// @dev Orders can be refunded by anyone as long is the block.timestamp of the target chain is
/// greater than the order.expiry + 24 hours.
/// @param order Order to be refunded by the Spoke.sol contract in the format of the Order struct.
function refundOrder(Order calldata order) external;
```

contracts/Spoke.sol (Lines 196-214):

```
function refundOrder(Order calldata order) external nonReentrant {
    bytes32 orderHash = keccak256(abi.encode(order));

    if (orderHashToStatus[orderHash] != OrderStatus.CREATED) revert
OrderStateNotCreated();
    if (msg.sender != order.filler) {
        if (!(block.timestamp > (order.expiry + 1 days))) revert
OrderNotExpired();
    }
    if (order.fromChain != _getChainId()) revert InvalidSourceChain();

    orderHashToStatus[orderHash] = OrderStatus.REFUNDED;

    if (order.fromToken == Utils.NATIVE_TOKEN) {
        payable(order.toAddress).sendValue(order.fromAmount);
    } else {
        IERC20(order.fromToken).safeTransfer(order.toAddress,
order.fromAmount);
    }

    emit OrderRefunded(orderHash);
}
```

contracts/Spoke.sol (Lines 121-132):

```
function sponsorOrder(Order calldata order, bytes calldata signature)
external {
    if (order.fromToken == Utils.NATIVE_TOKEN) revert
NativeTokensNotAllowed();
    if (
        !SignatureChecker.isValidSignatureNow(
            order.fromAddress,
            _hashTypedDataV4(_hashOrderTyped(order)),
            signature
        )
    ) revert InvalidUserSignature();

    _createOrder(order, order.fromAddress);
}
```

Funds should be returned to `order.fromAddress`. However, the `createOrder` path allows anyone to escrow funds in the first place because `msg.sender == order.fromAddress` is not checked. This might still lead to funds being returned to an address they were not escrowed from which seems unexpected.

Severity Medium

[MEDIUM][✔ acknowledged] Spoke - fill order should verify that `SpokeMulticall.run()` spent all tokens

[Update] Remediation Note: Acknowledged. The client provided the following statement:

This is something that we're comfortable handling on our backend to ensure that all tokens sent to multicall are spent and anything left over will be handled by including an extra call to forward the amounts to the intended recipient. We have call types to handle full balances as well, if there are leftover tokens we'll address it appropriately in the calls encoding for post hooks.

During filling of orders a hook might be executed. This hook is executed via `SpokeMultiCall` which can execute arbitrary calls. Any tokens left on that contract (i.e. due to wrong parameterization) are up for grabs by anyone. In order to safeguard from such scenarios it is suggested to have `_fillOrder` verify that the amount of `ETH` or `ERC20` was actually consumed by the recipient of `SquidMultiCall` and that the token balance is not left at `SquidMultiCall`.

For example, this can be achieved by recording the token or `ETH` balance diff, before and after the `SquidMultiCall.run()` method, and check if the balance before and after the call for the tokens match.

[MEDIUM][✔ fixed] Spoke - `sponsorOrderUsingPermit2` should check `order.fromAmount != 0`

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/b1562d28b7f3c2f343a25add0375056609c7f357 by checking for non-zero order amounts.

Similar to `_createOrder()`, sponsoring an order using `permit2` should check that `order.fromAmount != 0`

contracts/Spoke.sol (Lines 141-148):

```
function _createOrder(Order calldata order, address fromAddress) private
nonReentrant {
    bytes32 orderHash = keccak256(abi.encode(order));

    if (orderHashToStatus[orderHash] != OrderStatus.EMPTY) revert
OrderAlreadyExists();
    if (block.timestamp > order.expiry) revert OrderExpired();
    if (order.fromChain != _getChainId()) revert InvalidSourceChain();
    if (order.fromToken != Utils.NATIVE_TOKEN && msg.value != 0) revert
UnexpectedNativeToken();
    if (order.fromAmount == 0) revert InvalidAmount();
```

contracts/Spoke.sol (Lines 162-191):

```
function sponsorOrderUsingPermit2(
    Order calldata order,
    IPermit2.PermitTransferFrom calldata permit,
    bytes calldata signature
) external nonReentrant {
    bytes32 orderHash = keccak256(abi.encode(order));

    if (orderHashToStatus[orderHash] != OrderStatus.EMPTY) revert
OrderAlreadyExists();
    if (block.timestamp > order.expiry) revert OrderExpired();
    if (order.fromChain != _getChainId()) revert InvalidSourceChain();
    if (order.fromToken == Utils.NATIVE_TOKEN) revert
NativeTokensNotAllowed();

    orderHashToStatus[orderHash] = OrderStatus.CREATED;

    orderHashToStatus[orderHash] = OrderStatus.CREATED;

    IPermit2.SignatureTransferDetails memory transferDetails;
    transferDetails.to = address(this);
    transferDetails.requestedAmount = order.fromAmount;
```

```
    bytes32 witness = _hashOrderTyped(order);
    permit2.permitWitnessTransferFrom(
        permit,
        transferDetails,
        order.fromAddress,
        witness,
        Utils.ORDER_WITNESS_TYPE_STRING,
        signature
    );
```

[MEDIUM][✔ fixed] Spoke - `sponsorOrderUsingPermit2` duplicate state update (copy paste)

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/b59f0a709bccfb4c592f6abaf174d92e0f90f71f by removing the duplicate state update.

The order update `orderHashToStatus[orderHash] = OrderStatus.CREATED` is called twice.

contracts/Spoke.sol (Lines 162-177):

```
function sponsorOrderUsingPermit2(
    Order calldata order,
    IPermit2.PermitTransferFrom calldata permit,
    bytes calldata signature
) external nonReentrant {
    bytes32 orderHash = keccak256(abi.encode(order));

    if (orderHashToStatus[orderHash] != OrderStatus.EMPTY) revert
OrderAlreadyExists();
    if (block.timestamp > order.expiry) revert OrderExpired();
    if (order.fromChain != _getChainId()) revert InvalidSourceChain();
    if (order.fromToken == Utils.NATIVE_TOKEN) revert
NativeTokensNotAllowed();

    orderHashToStatus[orderHash] = OrderStatus.CREATED;

    orderHashToStatus[orderHash] = OrderStatus.CREATED;
```

Remove the duplicate state update. Check code surrounding this area as this looks like a copy paste error that might have overwritten some logic.

[MEDIUM][✔ fixed] Spoke - `createOrder` frontrun griefing, due `fromAddress` not verified being `order.fromAddress`

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/4cdda567facd88ad6748e87d74e2b05b30884389 by enforcing that `msg.sender` is `order. fromAddress`.

There are two ways to create an order in the `Spoke` contract:

- by creating an order, with the caller providing liquidity (`order.fromAddress` is supposed to be `msg.sender`)
- by creating a sponsored order, with the sponsor (= signer of the message) providing liquidity (only for erc20 tokens) (`order.fromAddress` is `signer`)

contracts/Spoke.sol (Lines 115-119):

```
/// @inheritdoc ISpoke
function createOrder(Order calldata order) external payable {
    _createOrder(order, msg.sender);
}
```

contracts/Spoke.sol (Lines 121-132):

```
function sponsorOrder(Order calldata order, bytes calldata signature)
external {
    if (order.fromToken == Utils.NATIVE_TOKEN) revert
NativeTokensNotAllowed();
    if (
        !SignatureChecker.isValidSignatureNow(
            order.fromAddress,
            _hashTypedDataV4(_hashOrderTyped(order)),
```

```
            signature
        )
    ) revert InvalidUserSignature();

    _createOrder(order, order.fromAddress);
}
```

Both functions will ultimately call `_createOrder` which checks the order status by hashing it. Note, the order hash also serves as a `nonce`. The same order data cannot be used twice (which is fine, because the `expiry` field can vary, hence, creating different order hashes for logically same orders).

contracts/Spoke.sol (Lines 141-160):

```
function _createOrder(Order calldata order, address fromAddress) private
nonReentrant {
    bytes32 orderHash = keccak256(abi.encode(order));

    if (orderHashToStatus[orderHash] != OrderStatus.EMPTY) revert
OrderAlreadyExists();
    if (block.timestamp > order.expiry) revert OrderExpired();
    if (order.fromChain != _getChainId()) revert InvalidSourceChain();
    if (order.fromToken != Utils.NATIVE_TOKEN && msg.value != 0) revert
UnexpectedNativeToken();
    if (order.fromAmount == 0) revert InvalidAmount();

    orderHashToStatus[orderHash] = OrderStatus.CREATED;

    if (order.fromToken == Utils.NATIVE_TOKEN) {
        if (msg.value != order.fromAmount) revert InvalidNativeAmount();
    } else {
        IERC20(order.fromToken).safeTransferFrom(fromAddress,
address(this), order.fromAmount);
    }

    emit OrderCreated(orderHash, order);
}
```

However, the `createOrder(..., msg.sender) -> _createOrder(...)` path does not check that `msg.sender` is actually `order.fromAddress`. This allows a caller to frontrun other users `createOrder` calls, block their orders from being created and then refund

their own orders. The `refundOrder` mechanism, however, seems broken right now as it is refunding to `order.toAddress` which should be `order.fromAddress`

It is recommended to verify that `order.fromAddress` is `msg.sender` in the `createOrder(..., msg.sender)` path and check that `refundOrder` refunds to the correct address.

Severity Low

[LOW][✔ fixed] Spoke - `batchMultiTokenSingleChainSettlements` checks for impossible conditions

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/43c9c4c769d204fd8ca71c8f9fcc9e57d6c9c4a8 by removing the duplicate checks.

The following checks are either obsolete or should be placed in the beginning of the function:

- `orderHashes.length == 0` - can only be non zero based on the initial check in the beginning of the function
- `fromAmounts.length != fromTokens.length` - is guaranteed by the array initialization
- `fees.length != fromTokens.length` - is guaranteed by the array initialization
- `fromTokens.length < 2` should be at the beginning of the function

contracts/Spoke.sol (Lines 452-455):

```
if (orders.length == 0) revert InvalidArrayLength();
bytes32[] memory orderHashes = new bytes32[](orders.length);
uint256[] memory fromAmounts = new uint256[](fromTokens.length);
uint256[] memory fees = new uint256[](fromTokens.length);
```

contracts/Spoke.sol (Lines 476-481):

```
if (
    orderHashes.length == 0 ||
    fromAmounts.length != fromTokens.length ||
    fees.length != fromTokens.length ||
    fromTokens.length < 2
) revert InvalidArrayLength();
```

[LOW][✔ fixed] Spoke `fillSingleChainAtomic` - increment should be value asigment

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/471c40662aeb181a96392ee0c345ac3d054efdb2 by changing the code to value assignment.

`fromAmount` and `fee` should be set instead of incremented `+=` as there are no other arithmetic operations on these variables before the initial calculation. This is probably a copy paste from `batchSingleChainSettlements` logic.

contracts/Spoke.sol (Lines 354-375):

```
uint256 fromAmount = 0;
uint256 fee = 0;

if (
    !SignatureChecker.isValidSignatureNow(
        order.fromAddress,
        _hashTypedDataV4(_hashOrderTyped(order)),
        signature
    )
) revert InvalidUserSignature();
if (orderHashToStatus[orderHash] != OrderStatus.EMPTY) revert
OrderAlreadyExists();
if (settlementToStatus[orderHash] != SettlementStatus.EMPTY) revert
OrderAlreadySettled();
if (block.timestamp > order.expiry) revert OrderExpired();
if (order.fromChain != _getChainId()) revert InvalidSourceChain();
if (order.toChain != _getChainId()) revert InvalidDestinationChain();
if (order.fromToken == Utils.NATIVE_TOKEN) revert NativeTokensNotAllowed();
```

```
if (order.toToken != Utils.NATIVE_TOKEN && msg.value != 0) revert
UnexpectedNativeToken();
if (order.fromAmount == 0) revert InvalidAmount();
if (msg.sender != order.filler) revert OnlyFillerCanSettle();

fee += (order.fromAmount * order.feeRate) / 1000000;
fromAmount += order.fromAmount - fee;
```

```
fee = (order.fromAmount * order.feeRate) / 1000000;
 fromAmount = order.fromAmount - fee;
```

[LOW][✔ fixed] Spoke - `batchFillOrder` unnecessary check

[Update] Remediation Note: Addressed with

https://github.com/0xsquid/squid-coral/commit/43c9c4c769d204fd8ca71c8f9fcc9e57d6c9c4a8 by removing the duplicate checks.

The check for `calls.length != 0` can be omitted as it is implied by `orders.length != 0 && orders.length == calls.length`.

contracts/Spoke.sol (Lines 241-248):

```
function batchFillOrder(Order[] calldata orders, ISquidMulticall.Call[][]
calldata calls) external payable {
    if (
        orders.length == 0 ||
        calls.length == 0 ||
        orders.length != calls.length
    ) revert InvalidArrayLength();

    uint256 remainingNativeTokenValue = msg.value;
```

[LOW][✔ fixed] Hub - `forwardMultiTokenSettlements` allows duplicates

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/1ee1f6d1ae9175f448c0f7c0dcd8f2f5130ed43e by checking for duplicate tokens.

The `forwardMultiTokenSettlements` function is callable by anyone. The function does not check for duplicates in `fromTokens`. If duplicate token addresses are provided, `_findTokenIndex()` will only return the first hit although all tokens are later encoded and forwarded to the gateway. Since the token index is used to sum up all fees and amounts per tokens, this should not present a security issue.

contracts/Hub.sol (Lines 284-294):

```
function _findTokenIndex(
    address token,
    address[] calldata fromTokens
) private pure returns (uint256) {
    for (uint256 i = 0; i < fromTokens.length; i++) {
        if (fromTokens[i] == token) {
            return i;
        }
    }
    return type(uint256).max;
}
```

contracts/Hub.sol (Lines 182-196):

```
bytes32[] memory orderHashes = new bytes32[](orders.length);
uint256[] memory fromAmounts = new uint256[](fromTokens.length);
uint256[] memory fees = new uint256[](fromTokens.length);

for (uint256 i = 0; i < orders.length; i++) {
    bytes32 orderHash = keccak256(abi.encode(orders[i]));
    orderHashes[i] = orderHash;

    if (orderHashToStatus[orderHash] != OrderStatus.PROCESSED) revert
OrderNotProcessed();
    if (orders[i].fromChain != fromChain) revert InvalidSourceChain();
    if (orders[i].filler != filler) revert InvalidSettlementFiller();

    uint256 tokenIndex = _findTokenIndex(orders[i].fromToken, fromTokens);
```

```
    if (tokenIndex == type(uint256).max) revert
InvalidSettlementSourceToken();
```

However, it should be considered to strictly enforce that token addresses cannot repeat in the `fromTokens` call parameter as the array initializations suggest that tokens should be unique, or else, `fees` and `fromAmounts` may be initialized with too many items.

contracts/Hub.sol (Lines 181-185):

```
bytes32[] memory orderHashes = new bytes32[](orders.length);
uint256[] memory fromAmounts = new uint256[](fromTokens.length);
uint256[] memory fees = new uint256[](fromTokens.length);
```

[LOW][✔ fixed] Hub - `forwardTokenSettlements` checks for impossible conditions

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/43c9c4c769d204fd8ca71c8f9fcc9e57d6c9c4a8 by removing the duplicate checks.

The function `forwardSettlements` is hardcoded to expect

- `orders.length != 0`

These conditions are checked early in the function prologue:

contracts/Hub.sol (Lines 172-180):

```
function forwardMultiTokenSettlements(
    Order[] calldata orders,
    uint256 fromChain,
    address[] calldata fromTokens,
    address filler
```

```
) external payable nonReentrant {
    if (msg.value == 0) revert GasRequired();
    if (orders.length == 0) revert InvalidArrayLength();
    if (!activeChain[fromChain]) revert InvalidChainId();
```

Subsequently, the following arrays are initialized with fixed sizes:

contracts/Hub.sol (Lines 182-185):

```
bytes32[] memory orderHashes = new bytes32[](orders.length);
uint256[] memory fromAmounts = new uint256[](fromTokens.length);
uint256[] memory fees = new uint256[](fromTokens.length);
```

It should be noted that `fromTokens` is not checked at this time and may be zero, leading to zero length array initialization which may not be useful at all and should be checked early in the function. After building the order hashes, the following sanity checks are performed, they might be unnecessary or duplicate or at the wrong place in the function:

contracts/Hub.sol (Lines 206-211):

```
if (
    orderHashes.length == 0 ||
    fromAmounts.length != fromTokens.length ||
    fees.length != fromTokens.length ||
    fromTokens.length < 2
) revert InvalidArrayLength();
```

- `orderHashes.length != 0` - cannot be zero as `orders.length` is checked to be non-zero
- `fromAmounts.length == fromTokens.length` - is always true as `fromAmounts = new uint256[](fromTokens.length)`
- `fees.length == fromTokens.length` - is always true as `fees = new uint256[](fromTokens.length)`
- `fromTokens.length` - is a function call parameter and should be checked early in the function body. Note, that, `fromTokens.length == 0` will always revert in the function when calling `_findTokenIndex`.

It is recommended to remove unnecessary checks and move function call parameter checks to the beginning of the function.

[LOW][✔ fixed] Hub - `forwardSettlements` checks for impossible conditions

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/43c9c4c769d204fd8ca71c8f9fcc9e57d6c9c4a8 by removing the duplicate checks.

The function `forwardSettlements` is hardcoded to expect

- `fromTokens[].length == 1`
- and `orders[].length >= 1`

These conditions are checked early in the function prologue:

contracts/Hub.sol (Lines 110-118):

```
function forwardSettlements(
    Order[] calldata orders,
    uint256 fromChain,
    address[] calldata fromTokens,
    address filler
) external payable nonReentrant {
    if (msg.value == 0) revert GasRequired();
    if (orders.length == 0 || fromTokens.length != 1) revert
InvalidArrayLength();
    if (!activeChain[fromChain]) revert InvalidChainId();
```

After building the order hashes, another set of checks are peformed on the functions inputs, some of which are unnecessary or duplicate:

contracts/Hub.sol (Lines 142-147):

```
if (
    orderHashes.length == 0 ||
    fromTokens.length != 1 ||
```

```
    fromAmounts.length != 1 ||
    fees.length != 1
) revert InvalidArrayLength();
```

The function aborts if:

- `orderHashes.length == 0` - this cannot happen, as in the beginning of the function it is checked that `orders[].length >= 1`
- `fromTokens.length != 1` - this cannot happen, as the same check is performed in the beginning of the function.
- `fromAmounts.length != 1` - this cannot happen, as the array is hardcoded to length 1
- `fees.length != 1` - this cannot happen, as the array is hardcoded to length 1

contracts/Hub.sol (Lines 120-122):

```
bytes32[] memory orderHashes = new bytes32[](orders.length);
uint256[] memory fromAmounts = new uint256[](1);
uint256[] memory fees = new uint256[](1);
```

It is recommended to double check if these are indeed the checks to be performed again or else, remove them as they are unlikely to fire.

[LOW][✔ fixed] Hub - `addChains` allows duplicates leading to inconsistent state

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/75921bbed6c625a4b6868e97a1f88a3669da9aa1 by checking for duplicate chains.

The function `addChains` allows an owner to add chain to chainName mappings and set trusted addresses. The function does not check for duplicates and instead, overwrites the entries. This can lead to inconsistencies between `addChains` and `removeChains` when `addChains` is called with an existing `chainId` but with a different `chainName`. This will lead to the `chainIdToChainName` mapping being overwritten

which is also used in `removeChains` to resolve a `chainId` to `chainName`. Another trusted address for `chainName` is added, however, on remove, only one will be deleted.

contracts/Hub.sol (Lines 73-86):

```
function addChains(
    uint256[] calldata chainIds,
    string[] calldata chainNames
) external onlyOwner {
    if (chainIds.length == 0 || chainNames.length == 0) revert
InvalidArrayLength();
    if (chainIds.length != chainNames.length) revert InvalidArrayLength();
    for (uint256 i = 0; i < chainIds.length; i++) {
        chainIdToChainName[chainIds[i]] = chainNames[i];
        activeChain[chainIds[i]] = true;
        _setTrustedAddress(chainNames[i], spoke);
    }

    emit ChainsAdded(chainIds, chainNames);
}
```

Note: `chainIds[x]` is implicitly non-zero length-checked by `InterchainAddressTracker._setTrustedAddress()` and `InterchainAddressTracker._removeTrustedAddress()`.

Check for duplicate `chainId` to `chainName` mappings and revert if an existing chain is overwritten forcing the caller to first remove the chain and start for a clean state.

[LOW][✔ fixed] Unused Imports

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/bca80fd0353a6cca6d9ca97abe498c1b50cace71 by removing the unused imports.

The following contracts or interfaces are imported but not referenced within the source unit.

contracts/Spoke.sol (Lines 14-15):

```
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {IERC1155} from
"@openzeppelin/contracts/token/ERC1155/IERC1155.sol";
```

It is recommended, to remove the unused imports.

Severity Info

[INFO][✔ fixed] Hub - Consider declaring divisor as constant var

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/ccd924b2e27c5cc9a3260b662b9af9370f784d45 declaring the fee divisor constant.

Instead of inlining the fee divisor, consider declaring it as a named constant. This increases readability and may prevent inaccuracies due to typos.

contracts/Hub.sol (Lines 133-133):

```
uint256 fee = (orders[i].fromAmount * orders[i].feeRate) / 1000000;
```

contracts/Hub.sol (Lines 197-197):

```
uint256 fee = (orders[i].fromAmount * orders[i].feeRate) / 1000000;
```

[INFO][✔ fixed] Hub - `addChains` Unnecessary check for chainIds AND chainNames length == 0

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/91ba4b350be6cb0cdb360b0ee80c89ee53347383 by removing one zero length check.

The function `addChains` performs three checks:

- `chainIds.length != 0`
- `chainNames.length != 0`
- `chainIds.length == chainNames.length`

However, the check for `chainNames.length != 0` is unnecessary, `chainIds.length` cannot be zero and `chainIds` must match the length of `chainNames`.

contracts/Hub.sol (Lines 72-86):

```
/// @inheritdoc IHub
function addChains(
    uint256[] calldata chainIds,
    string[] calldata chainNames
) external onlyOwner {
    if (chainIds.length == 0 || chainNames.length == 0) revert
InvalidArrayLength();
    if (chainIds.length != chainNames.length) revert InvalidArrayLength();
    for (uint256 i = 0; i < chainIds.length; i++) {
        chainIdToChainName[chainIds[i]] = chainNames[i];
        activeChain[chainIds[i]] = true;
        _setTrustedAddress(chainNames[i], spoke);
    }

    emit ChainsAdded(chainIds, chainNames);
}
```

Consider removing the check for `chainNames.length == 0` as this is implicitly validated with the other two conditions.

[INFO][✔ fixed] Remove unused source units

[Update] Remediation Note: Addressed with https://github.com/0xsquid/squid-coral/commit/f7a73b0a92762c4ba119c658fd8bca05495ca05a by removing unused source units.

`SpokeMulticall.sol` only contains non functional commented code.

contracts/SpokeMulticall.sol (Lines 1-19):

```
// // SPDX-License-Identifier: MIT

// pragma solidity ^0.8.0;

// import {Initializable} from
'@openzeppelin/contracts/proxy/utils/Initializable.sol';
// import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
// import {SafeERC20} from
"@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
// import {ISpokeMulticall} from "./interfaces/ISpokeMulticall.sol";

// contract SpokeMulticall is ISpokeMulticall, Initializable {
//     using SafeERC20 for IERC20;

//     address public constant nativeToken =
0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
//     address public spoke;

//     modifier onlySpoke() {
//         if (msg.sender != spoke) revert OnlySpoke();
//         _;
//     }
```

It is recommended, to remove this file from the code base.

Additional Notes

[HIGH] [out-of-scope] SquidMultiCall - `_setCallDataParameter` out of bounds mem write; integer overflow

The function takes a `memory` reference type and low-level assembly modifies the callers memory data without performing any checks on it.

- no bounds checks that `(parameterPosition *32 + 36) < callData.length` - this allows out of bounds memory write to any other memory struct in the calls context
- no integer overflow protection - allows to provide a sufficiently large enough `parameterPosition` to wrap the low-level arithmetic operation (integer overflow)

There being no bound check for `parameterPosition` to be within the `callData` means, that, if you fill an order, whoever provides the `calls[]` can write anything anywhere in current mem (including modifying the calls struct itself).

This pattern looks bad as it is often seen with backdoors, but it is also hard to maintain a codebase which such an unclear side-effect rich pattern and it is clear to say that this will pop up in every future code review because it is a vulnerable pattern. If not already, it will eventually lead to security problems, increase the attack surface for malicious activity or in the best case hide errors.

contracts/SquidMulticall.sol (Lines 20-35):

```
Call memory call = calls[i];

if (call.callType == CallType.FullTokenBalance) {
    (address token, uint256 amountParameterPosition) = abi.decode(
        call.payload,
        (address, uint256)
    );
    uint256 amount = IERC20(token).balanceOf(address(this));
    // Deduct 1 from amount to keep hot balances and reduce gas cost
    if (amount > 0) {
        // Cannot underflow because amount > 0
        unchecked {
            amount -= 1;
        }
    }
    _setCallDataParameter(call.callData, amountParameterPosition, amount);
```

contracts/SquidMulticall.sol (Lines 50-59):

```
function _setCallDataParameter(
    bytes memory callData,
    uint256 parameterPosition,
    uint256 value
) private pure {
    assembly {
        // 36 bytes shift because 32 for prefix + 4 for selector
        mstore(add(callData, add(36, mul(parameterPosition, 32))), value)
    }
}
```

Here's a very simple PoC that illustrate that this pattern can be used to overwrite data from the unrelated `MyMemoryType` from within `_setCallDataParameter` even though the passed memory location is `callData`.

```
contract Storage {
    event logbytes(uint index, bytes a);
    event Log( uint256 left);

    struct MyMemoryType {
            uint256 value;
    }

    function _setCallDataParameter(
        bytes memory callData,
        uint256 parameterPosition,
        uint256 value
    ) private pure  {
        assembly {
            mstore(add(callData, add(36, mul(parameterPosition, 32))),
value)
        }
    }

    function doSomethingWithCalldata(bytes memory callData) public  returns
(uint256){
        MyMemoryType memory mem; // dummy struct on mem, we'll overflow
into ths
        mem.value = 1000; // initial value

        emit logbytes(1, callData);
        _setCallDataParameter(callData, 4, 0xce);
        emit logbytes(4, callData);

        return mem.value;
    }

    function callSomething(address target) public returns(bool) {
```

```
        (bool success, bytes memory data) = target.call("1111"); //@audit
contract exists
        emit logbytes(1, data);
        require(success);
        return success;
    }
}
```

[INFO] [out-of-scope][Note] Any token stuck in `SquidMulticall` can be swept by anyone by crafting an appropriate order

SquidMulticall is permissionless. Any token stuck in the contract after a `SquidMulticall.run()` invocation (i.e. from `Spoke._fillOrder`) may be swept by anyone. Therefore, it is important to either safeguard the Multicall functionality itself or in the `Spoke` to ensure that no tokens/`ETH` are left in the Multicall utility.