yAudit Chainswap CCIP Review

Review Resources:

- Chainswap CCIP codebase.

Auditors:

- Jackson
- panda

Table of Contents

Review Summary

Chainswap

Chainswap provides a cross-chain swapping service.

The contracts of the chainswap Repo were reviewed over 5 days. The code review was performed by 5 auditors between 1st April 2024 and 5th April 2024. The repository was under active development during the review, but the review was limited to the latest commit at the

start of the review. This was commit cd773c15eafc9fe03a9e0b70e73ecedf935a07c0 for the chainswap repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- CCIP.sol
- CCIP_AVAX.sol

After the findings were presented to the Chainswap team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Chainswap and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The access control mechanisms are properly implemented and follow best practices, ensuring only authorized entities can perform sensitive actions.

Category	Mark	Description
Mathematics	Good	The mathematics and algorithms used in the protocol appear to be correct and efficient, with no critical issues identified. Minor suggestions for improvement might be considered to optimize gas usage and ensure mathematical operations are safe from overflows and underflows.
Complexity	Average	Certain parts of the code could benefit from further simplification to enhance readability and maintainability.
Libraries	Good	The use of external libraries and contracts, such as OpenZeppelin and Chainlink, is appropriate and follows best practices.
Decentralization	Average	The protocol demonstrates a commitment to decentralization, though certain aspects, such as reliance on specific external services or contracts, could potentially introduce points of centralization. Future iterations could explore ways to increase decentralization further.
Code stability	Low	Code doesn’t function properly.
Documentation	Medium	Code is well documented. However, no additional documentation was provided.
Monitoring	Medium	Adequate monitoring mechanisms are in place. Events are emitted for key external functions.
Testing and verification	Low	No tests were provided to us with the repository to review.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact

    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

1. Critical - `CCIP`'s `sendMessagePayFirstStep()` always reverts when `tokenIn` is not USDC

Technical Details

In `swapInitialData()`, if the `tokenIn` is not USDC a v3 router is used to swap into USDC. A require statement is used to ensure that the swap results in a higher USDC balance than before the swap.

```
            uint256 beforeSendingUsdc = IERC20(usdc).balanceOf(address(this));
            IV3SwapRouter.ExactInputParams memory params =
IV3SwapRouter.ExactInputParams(
                _initialSwapData.v3InitialSwap, address(this), _initialSwapData.amountIn,
_initialSwapData.minAmountOutV3Swap
            );
            uint256 afterSendingUsdc = IERC20(usdc).balanceOf(address(this));
            require(afterSendingUsdc > beforeSendingUsdc, "Must swap into USDC");
```

However, this `require()` statement is placed after constructing the swap params and not after performing the swap. Since no balance has been added to the contract the require statement will always result in a revert.

Impact

Critical. `sendMessagePayFirstStep()` calls which must swap to USDC will always revert.

Recommendation

Check the USDC balances after the swap and not after constructing the params for the swap.

```
-           uint256 beforeSendingUsdc = IERC20(usdc).balanceOf(address(this));
            IV3SwapRouter.ExactInputParams memory params =
IV3SwapRouter.ExactInputParams(
                _initialSwapData.v3InitialSwap, address(this), _initialSwapData.amountIn,
_initialSwapData.minAmountOutV3Swap
            );
-           uint256 afterSendingUsdc = IERC20(usdc).balanceOf(address(this));
-           require(afterSendingUsdc > beforeSendingUsdc, "Must swap into USDC");
            // Swap ReceiverSwapData.finalToken to ETH via V3, then to USDC via uniswap
V3
+           uint256 beforeSendingUsdc = IERC20(usdc).balanceOf(address(this));
            USDCOut = v3Router.exactInput( params );
+           uint256 afterSendingUsdc = IERC20(usdc).balanceOf(address(this));
+           require(afterSendingUsdc > beforeSendingUsdc, "Must swap into USDC");
```

Also, check that the swap path ends in USDC, rather than checking the balances, as suggested in H1.

Developer Response

Fixed in 067717979f33a467d2afd92a43bc615d2672910b

2. Critical - USDC in the contract can be stolen using `sendMessagePayFirstStep()`

Issue Description

Exploiting the `path` used for swapping in `sendMessagePayFirstStep()` allows an attacker to use a token of minimal value to extract an amount of USDC equivalent to the total USDC balance in the contract.

Technical Details

The exploit involves using a token of negligible value without validating the swap path. This approach enables the conversion of tokens for an amount of USDC that matches the contract's total USDC balance. Given that USDC has 6 decimals, the exploit can be executed using DAI if no valueless token is available, thereby multiplying the value extracted by 12 times the exploit cost. We have disclosed details of the exploit to the team, including access to our findings at exploit code utilizing the `CCIP_AVAX` contract. While the exploit is straightforward with the `CCIP_AVAX.sol` contract, it can also be executed using the `CCIP.sol` contract, which includes an additional verification to ensure that the USDC amount is incremented. The attacker must create a custom ERC20 token and a custom Uniswap pool. The swap path will utilize this custom token, and during the swap, the custom ERC20 will transfer 1 wei of USDC to the CCIP contract to satisfy the condition that the USDC balance must increase.

Impact

Critical. This vulnerability allows for the potential theft of funds from the contract.

Recommendation

Ensure that the swap path ends with USDC to prevent exploitation. Check should be set up before:

- CCIP.sol#L594
- CCIP_AVAX.sol#L499-L505

Also, only the USDC necessary to complete the swap on the receive side should be approved in `CCIP_AVAX`, as is done in `CCIP` here, to ensure no balance aside from the USDC received by the contract via the user's bridge can be utilized in the swap.

Developer Response

Fixed in 44045936752eb223e49ae689758c9d1dfbb6d555 and c5d31e69b6759716292b28af9f9e492c739b21b0

High Findings

1. High - A token other than USDC can be swapped into before bridging

Technical Details

In `swapInitialData()` any path can be supplied by the user.

```
            USDCOut = lbRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
                _realAmountIn,
                _initialSwapData.minAmountOut, // Amount out min
                _initialSwapData.path,
                address(this),
                99999999999999999999
            );
```

This means a token other than USDC can be swapped into before bridging. Depending on whether there is a USDC balance in the contract or not, this balance can be stolen by a user using a different swap path than USDC.

Impact

High.

Recommendation

Ensure that the swap path ends with USDC. In CCIP an attempt was made to fix the issue by checking the balances before and after the swap.

```
            uint256 beforeSendingUsdc = IERC20(usdc).balanceOf(address(this));
            IV3SwapRouter.ExactInputParams memory params =
IV3SwapRouter.ExactInputParams(
                _initialSwapData.v3InitialSwap, address(this), _initialSwapData.amountIn,
_initialSwapData.minAmountOutV3Swap
            );
            uint256 afterSendingUsdc = IERC20(usdc).balanceOf(address(this));
            require(afterSendingUsdc > beforeSendingUsdc, "Must swap into USDC");
```

However, this is insufficient since USDC can be transferred to the contract before the swap to inflate its balance before issuing the draining swap.

Developer Response

Fixed in 44045936752eb223e49ae689758c9d1dfbb6d555

2. High - Potential Misuse of Residual Native Tokens

The native token is utilized for transaction fees and can also serve as the asset swapped for USDC. Given that the CCIP fee is typically lower than the amount of the native token sent, a residue of tokens may remain in the contract.

Technical Details

A user can leverage the remaining native tokens by assigning this surplus to `_initialSwapData.amountIn`. This action enables the exchange of native tokens for USDC, which can then be redeemed on the target chain. See CCIP.sol#L617-L621 and CCIP_AVAX.sol#L529-L532 for more details.

Impact

High. Users must be prevented from claiming the residual ETH of others.

Recommendation

- Ensure that `msg.value` is at least equal to `amountIn`.
- Process refunds for any ETH surplus not utilized for the CCIP fee.

Developer Response

Fixed in ad2dea5c177076d2db93676b23cbcb743c31aa70

Medium Findings

1. Medium - Updating `lbQuoter` will not update it

The `lbQuoter` gets reassigned to itself.

Technical Details

```
File: CCIP_AVAX.sol
220 |    function changeRouters(address _lbRouter, address _lbQuoter) external onlyOwner
{
221 |        lbRouter = ILBRouter(_lbRouter);
222 |        lbQuoter = IQuoter(lbQuoter);
```

https://github.com/MainCross123/yearn-audit/blob/main/CCIP_AVAX.sol#L220-L223

Impact

Medium. lbQuoter can't be changed.

Recommendation

```
- 222 |         lbQuoter = IQuoter(lbQuoter);
+ 222 |         lbQuoter = IQuoter(_lbQuoter);
```

Developer Response

Fixed in 067717979f33a467d2afd92a43bc615d2672910b

2. Medium - Non-USDC ERC20 left in the contract can be drained.

If a token is sent by mistake to the contract, a user can forge a message spending USDC to drain it. This attack is especially interesting for tokens with a small number of decimals or if the value of the token is greatly superior to ETH.

Technical Details

Let's consider USDT as an example and imagine there are 1000 USDT that have been mistakenly sent to the `CCIP.sol` contract.

An attacker sends a transaction using `sendMessagePayNative` or `sendMessagePayLINK` with an amount of USDC equivalent to 10**9 ETH (about 0.000005 USDC) but includes a carefully crafted `_text` message. In this message, the attacker manipulates the `receiverData.v2Path` to specify `USDT` as path[0].

When `_ccipReceive` is executed, the contract swaps the small amount of USDC for 10**9 ETH, and this amount is used as input for `swapExactTokensForTokensSupportingFeeOnTransferTokens`. With the manipulated path, the attacker can then swap the USDT for another token and redirect it to themselves.

Impact

Medium. Funds sent by mistake can be stolen.

Recommendation

Check the first path before swapping on line 878

Developer Response

Fixed in 85a60ea08bd8abbe110f7d6964484 3aa6c54eaf1.

Low Findings

1. Low - Fees should be capped.

It's recommended to cap fees to be capped to the denominator value: `(feeBps * 100)`.

Technical Details

```
File: CCIP_AVAX.sol
230 |  swapFee = _swapFee;


266 |  function changeFeeAndAddress(uint256 _fee, address _feeReceiver) external
onlyOwner {
267 |          swapFee = _fee;
268 |          feeReceiver = _feeReceiver;
269 |      }
```

CCIP_AVAX.sol#L230 CCIP_AVAX.sol#L266-L269

Impact

Gas savings.

Recommendation

Cap the fee to a reasonable value.

Developer Response

Fixed in 34ed97130958f1b905fbd518b5473abbef783a66

2. Low - Users can overpay `msg.value`, leading to locked native tokens

Technical Details

When a user is sending a message and paying with native tokens, the fee is calculated, and the balance of the smart contract is checked to ensure there is sufficient balance to pay the fee. However, this does not check whether the user has sent more `msg.value` than is expected to pay the fee.

Impact

Low. Presumably, this case will not arise often and when it does, it should not involve a large amount of funds.

Recommendation

Return the difference between the necessary fee and `msg.value` back to the user.

Developer Response

Fixed in ad2dea5c177076d2db93676b23cbcb743c31aa70

3. Low - Validate `_receiver` based on `_destinationChainSelector`

The recipient is validated to be non-zero, a user could use the wrong `_receiver` corresponding to a different chain from the one he was planning to use. It would be preferred to check the `_receiver` based on the `_destinationChainSelector` to prevent funds from being sent to a non-existing contract.

Technical Details

```
File:
305 |    modifier validateReceiver(address _receiver) {
306 |        if (_receiver == address(0)) revert InvalidReceiverAddress();
307 |        _;
308 |    }
```

CCIP.sol#L305-L308 CCIP_AVAX.sol#L254-L257

Impact

Low. Frontend should always use the right `_receiver`.

Recommendation

Check `_receiver` based on the `_destinationChainSelector`. You will need to add an admin function to set those.

Developer Response

Acknowledged, we won't implement this change for gas costs issues.

4. Low - Incorrect timelock implementation

The timelock mechanism is implemented via a simple conditional check within functions that require a delay between when an action is initiated and when it can be executed (e.g., `transferOwnership()`, `changeFeeAndAddress()`, `changeRouters()`). The contract sets a timeLockTime state variable to a future timestamp upon calling `activateTimelock()`. This variable is then checked against the current block timestamp in subsequent function calls to enforce a delay.

However, this implementation does not queue transactions or their parameters for future execution. Instead, it only restricts the execution of certain functions until after a specified time has passed. This approach cannot specify or queue specific actions and their parameters ahead of time, which is a **key feature** of a comprehensive timelock mechanism.

Technical Details

CCIP.sol#L317-L344 A timelock contract that could be used is for example the one from compound Timelock.sol.

Impact

Low. The timelock is not working as a real timelock.

Recommendation

Remove the timelock on the contract and use an external timelock.

Developer Response

We'll keep it as it is for simplicity by watching and monitoring timelock activations.

5. Low - Wrapped native tokens can't be used to swap.

When `_initialSwapData.tokenIn` is set to a wrapped native token (e.g., WETH or WAVAX), the system wraps the native token sent. Consequently, it is not feasible to utilize the wrapped token directly without first unwrapping it.

Technical Details

```
File: CCIP.sol
618 |    if (_initialSwapData.tokenIn == weth) {
619 |        IWETH(weth).deposit{value: _initialSwapData.amountIn}();
620 |    }
```

- CCIP.sol#L619
- CCIP_AVAX.sol#L530

Impact

Low. Users are unable to employ wrapped native tokens without unwrapping them first.

Recommendation

Introduce an additional parameter to discern whether the token should be wrapped or transferred directly if a wrapped native token is specified as `_initialSwapData.tokenIn`.

Developer Response

Fixed in 346e5758ec021d2494b2e54173b0061e91c0a910.

6. Low - `_ccipReceive()` will revert if `v2Path` does not start with `weth` or `path` does not end with `weth` and `isV2` is true

Technical Details

In `_ccipReceive()` the `v2Router` is approved to spend `weth`. However, there is no check that the `v2Path` swaps `weth` to another token, or `path` swaps into `weth`. If any other paths are supplied by the user, the receive will revert due to insufficient allowance of the sell token or lack of `weth` balance.

Impact

Low. Since the tokens can be recovered by calling `recoverFailedTransfer()`.

Recommendation

Ensure that the `ReceiverSwapData`'s `v2Path` begins with `weth` and `path` ends with `weth` if `isV2` is true before sending the message, or approve the first token in the path, rather than `weth` if different sell tokens should be allowed.

Developer Response

We want it to revert in case the path is wrong so we can execute the try-catch block and recover the message.

Gas Saving Findings

1. Gas - Cache array length outside the loop

When iterating over an array, it is recommended to cache the array length outside the loop to avoid unnecessary gas costs.

Technical Details

```
File: CCIP_AVAX.sol

400 | for (uint256 i = 0; i < _path.length; i++) {
```

CCIP_AVAX.sol#L451

Impact

Gas savings.

Recommendation

Cache the _path length outside the loop.

Developer Response

Fixed in 33fc98c82edffe22a8ae493ae183e61bdec2b777

2. Gas - Do not use `this.getRouter()`

Calling an external function internally, through the use of `this` wastes the gas overhead of calling an external function (100 gas).

Technical Details

To access the router address, it's possible to use `i_ccipRouter` which is accessible as an internal variable.

CCIP.sol : 410, 480, 669

CCIP_AVAX.sol: 340, 410, 586

Impact

Gas savings.

Recommendation

Use `i_ccipRouter` instead of `this.getRouter()`.

Developer Response

Team Acknowledged. Won't fix.

3. Gas - State variables only set in the constructor should be declared `immutable`

Optimizes gas utilization by bypassing a `GSSET` (costing 20,000 gas) during the constructor phase, and substitutes the initial access in each transaction, which incurs a `GCOLDSLOAD` cost of 2,100 gas, and subsequent accesses (`GWARMACCESS` at 100 gas each), with a PUSH32 operation, significantly reducing the gas expenditure to just 3 gas.

Technical Details

```
File: CCIP.sol


242 | IERC20 private s_linkToken;
223 | address public weth;
244 | address public usdc;
```

CCIP.sol#L242-L244

```
File: CCIP_AVAX.sol


193 |    IERC20 private s_linkToken;
194 |    address public wAVAX;
195 |    address public usdc;
```

CCIP_AVAX.sol#L193-L195

Impact

Gas savings.

Recommendation

Set the variables to immutable.

Developer Response

Fixed in 123c5a2b578a71cf60b1ecdc17ad7088a8748b05

4. Gas - Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a `GCOLDSLOAD` (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declaring the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables, will be much cheaper.

Technical Details

```
File: CCIP.sol

823  | Client.Any2EVMMessage memory message = s_messageContents[messageId];

979  | FailedMessagesUsers memory f = failedMessagesUsers[tokenReceiver][index];

1020 | AddressNumber memory an = failedMessageByMessageId[_messageId];
```

823 979 1020

```
File: CCIP_AVAX.sol

740 | Client.Any2EVMMessage memory message = s_messageContents[messageId];

886 | FailedMessagesUsers memory f = failedMessagesUsers[tokenReceiver][index];

927 | AddressNumber memory an = failedMessageByMessageId[_messageId];
```

740 886 927

Impact

Gas savings.

Recommendation

Use storage variables.

Developer Response

Fixed in b78e94d1929ce091d32ce9382d0854396ef4e728

5. Gas - Use `checkAndApproveAll` to change allowance to max.

You have created a function to check if the allowance is high enough and approve to max uint256 if it's not, but you also have this same code that can be replaced by a call to `checkAndApproveAll`.

Technical Details

CCIP.sol#L489-L492 CCIP.sol#L422-L425 CCIP_AVAX.sol#L352-L355 CCIP_AVAX.sol#L419-L422

Impact

Gas savings.

Recommendation

Make use of `checkAndApproveAll`.

Developer Response

Fixed in 0b10eb6bdf34541906ac709cfbce050213fbf910

6. Gas - Superfluous modifier can be removed

Technical Details

When a message is received `ccipReceive()` is called, which has the following modifiers:

```
        onlyRouter
        onlyAllowlisted(
            any2EvmMessage.sourceChainSelector,
            abi.decode(any2EvmMessage.sender, (address))
        )
```

`ccipReceive()` then calls `processMessage()`, which has the following modifiers:

```
        onlySelf
        onlyAllowlisted(
            any2EvmMessage.sourceChainSelector,
            abi.decode(any2EvmMessage.sender, (address))
        )
```

Since `processMessage()` contains the `onlySelf()` modifier, and the only place it's called is from `ccipReceive()`, which already contains an identical `onlyAllowlisted()` modifier, the one in `processMessage()` can be removed to save gas.

Impact

Gas savings.

Recommendation

Remove the extraneous modifier.

Developer Response

Fixed in 6e2d358f15149a83457afdd362e56fcbacd7fc77

7. Gas - Extraneous `require()` can be removed

Technical Details

`sendMessagePayFirstStep()` has the `onlyAllowlistedDestinationChain()` modifier, which checks

```
        if (!allowlistedDestinationChains[_destinationChainSelector])
            revert DestinationChainNotAllowlisted(_destinationChainSelector);
        _;
```

This mapping is then checked again on L617.

Impact

Gas savings.

Recommendation

Remove the extraneous require statement.

Developer Response

Fixed in 833ed33960c6d9003b37942b37788c23e1b21e73

8. Gas - Unused variable `finalToken` in `ReceiverSwapData`

Technical Details

`finalToken` is present in the `ReceiverSwapData` struct, but it is unused.

Impact

Gas savings.

Recommendation

Remove the unused variable from the struct.

Developer Response

It is used for checking if the user wants to receive USDC at the end of the updated code.

Informational Findings

1. Informational - swapping deadline doesn't need to be set in the future.

Using an arbitrary number like 99999999999999999999 is unnecessary, the transaction is executed on the current block timestamp, and using `block.timestamp` instead of an arbitrary value is recommended.

Technical Details

CCIP.sol#L577 CCIP.sol#L883 CCIP_AVAX.sol#L504 CCIP_AVAX.sol#L788 CCIP_AVAX.sol#L796

Impact

Informational.

Recommendation

`block.timestamp` can be used instead of an arbitrary number.

Developer Response

Fixed in 19ec0297d0012fc71150b6fef99586ed5eae03c3

2. Informational - Protocol fees can be bypassed

Technical Details

The protocol fee is calculated as

```
uint256 feeAmount = USDCOut * swapFee / (feeBps * 100);
```

In `swapInitialData()` the `swapFee` is set to 1000 in the on-chain deployments. This means that a `USDCOut` amount < 100 will cause the `feeAmount` to round down to 0. Depending on L2 fees, this may or may not be economically beneficial to an attacker.

Impact

Informational. L2 fees have to be sufficiently low enough, and an attacker motivated enough to exploit this.

Recommendation

Round the `feeAmount` calculation up rather than down so that a fee is always charged to the user.

Developer Response

With outputs so small the fee makes sense that it's zero.

3. Informational - The swap will revert if `tokenIn` is `weth` and `swapTokenInV2First` is true

Technical Details

The path in `swapInitialData()` is hardcoded to swap to `weth` if `swapTokenInV2First` is true. However, the `tokenIn` could also be `weth`. In this case, the swap will fail and revert.

Impact

Informational. The swap can be re-tried with appropriate inputs.

Recommendation

Revert with a useful error message if `swapTokenInV2First` is true and `tokenIn` is `weth`

Developer Response

Fixed in dd6648d083baf940c2a1ab9e89873f6943657678

Final remarks

Our review has identified significant areas for improvement within the contract code which was then corrected by the team. We recommend implementing a comprehensive testing strategy to ensure contract reliability. Users are advised to proceed with an appropriate level of caution until thorough testing has been completed.