SwapExchange Audit Report

Prepared by Cyfrin

Version 1.1

Lead Auditors

Hans

Assisting Auditors

0kage

September 19, 2023

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

SwapExchange is a decentralized exchange that facilitates the swapping of ERC20 compatible tokens or Ethereum in a trustless manner.

5 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the SwapExchange smart contracts provided by SwapExchange. In this period, a total of 14 issues were found.

All Solidity source files (*.sol) in `contracts` directory except `test` were in scope for review.

Summary

Project Name	SwapExchange
Repository	Contracts
Commit	48d46306e7ad...
Audit Timeline	Sep 1st - Sep 8th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	1
Informational	2
Gas Optimizations	8
Total Issues	14

6 Findings

6.1 Medium Risk

6.1.1 Use safe transfer for ERC20 tokens

Severity: Medium

Description: The protocol intends to support all ERC20 tokens but the implementation uses the original transfer functions. Some tokens (like USDT) do not implement the EIP20 standard correctly and their transfer/transferFrom function return void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will revert.

```
TransferUtils.sol
    function _transferERC20(address token, address to, uint256 amount) internal {
        IERC20 erc20 = IERC20(token);
        require(erc20 != IERC20(address(0)), "Token Address is not an ERC20");
        uint256 initialBalance = erc20.balanceOf(to);
        require(erc20.transfer(to, amount), "ERC20 Transfer failed");//@audit-issue will revert for
↪   USDT
        uint256 balance = erc20.balanceOf(to);
        require(balance >= (initialBalance + amount), "ERC20 Balance check failed");
    }
```

Impact: Tokens that do not correctly implement the EIP20 like USDT, will be unusable in the protocol as they revert the transaction because of the missing return value.

Recommended Mitigation: We recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Protocol: Fixed in commit 564f711

Cyfrin: Verified.

6.1.2 Fee-on-transfer tokens are not supported

Severity: Medium

Description: The protocol intends to support all ERC20 tokens but does not support fee-on-transfer tokens. The protocol utilizes the functions `TransferUtils::_transferERC20()` and `TransferUtils::_transferFromERC20()` to transfer ERC20 tokens.

```
TransferUtils.sol
    function _transferERC20(address token, address to, uint256 amount) internal {
        IERC20 erc20 = IERC20(token);
        require(erc20 != IERC20(address(0)), "Token Address is not an ERC20");
        uint256 initialBalance = erc20.balanceOf(to);
        require(erc20.transfer(to, amount), "ERC20 Transfer failed");
        uint256 balance = erc20.balanceOf(to);
        require(balance >= (initialBalance + amount), "ERC20 Balance check failed");//@audit-issue
↪   reverts for fee on transfer token
    }
```

The implementation verifies that the transfer was successful by checking that the balance of the recipient is greater than or equal to the initial balance plus the amount transferred. This check will fail for fee-on-transfer tokens because the actual received amount will be less than the input amount. (Read here about fee-on-transfer tokens)

Although there are very few fee-on-transfer tokens, the protocol can't say it supports all ERC20 tokens if it doesn't support these weird ERC20 tokens.

Impact: Fee-on-transfer tokens can not be used for the protocol. Because of the rarity of these tokens, we evaluate this finding as a Medium risk.

Recommended Mitigation: The transfer utility functions can be updated to return the actually received amount. Or clearly document that only standard ERC20 tokens are supported.

Protocol: We are choosing not to implement this at this stage.

Cyfrin: Acknowledged. As recommended, please mention this in user documentation.

6.1.3 Centralization risk

Severity: Medium

Description: The protocol has an owner with privileged rights to perform admin tasks that can affect users. Especially, the owner can change the fee settings and reward handler address.

1. Validation is missing for admin fee setter functions.

```
FeeData.sol
31:     function setFeeValue(uint256 feeValue) external onlyOwner {
32:         require(feeValue < _feeDenominator, "Fee percentage must be less than 1");
33:         _feeValue = feeValue;
34:     }

43:
44:     function setFixedFee(uint256 fixedFee) external onlyOwner {//@audit-issue validate min/max
45:         _fixedFee = fixedFee;
46:     }
```

2. Important changes initiated by admin should be logged via events.

```
File: helpers/FeeData.sol

31:     function setFeeValue(uint256 feeValue) external onlyOwner {

36:     function setMaxHops(uint256 maxHops) external onlyOwner {

40:     function setMaxSwaps(uint256 maxSwaps) external onlyOwner {

44:     function setFixedFee(uint256 fixedFee) external onlyOwner {

48:     function setFeeToken(address feeTokenAddress) public onlyOwner {

53:     function setFeeTokens(address[] memory feeTokenAddresses) public onlyOwner {

60:     function clearFeeTokens() public onlyOwner {
```

```
File: helpers/TransferHelper.sol

86:     function setRewardHandler(address rewardAddress) external onlyOwner {

92:     function setRewardsActive(bool _rewardsActive) external onlyOwner {
```

Impact: While the protocol owner is regarded as a trusted party, the owner can change the fee settings and reward handler address without any validation or logging. This can lead to unexpected results and users can be affected.

Recommended Mitigation:

- Specify the owner's privileges and responsibilities in the documentation.
- Add constant state variables that can be used as the minimum and maximum values for the fee settings.
- Add proper validation for the admin functions.

- Log the changes in the important state variables via events.

Protocol:

- setFeeNumerator changes fixed in commit f8f07c5
- setFeeNumerator maximum reduced in commit 7874a8f
- setFixedFee changes fixed in commit af760b4
- events for setFeeToken/clearFeeToken/setFeeTokens as well as separating initialization setter to save emitting a heap of events when deploying, fixed in commit 927c102
- changing array arg type to calldata rather than memory, fixed in commit e615cb2
- events for RewardHandler and RewardsActive, fixed in commit 3068a2e
- events for MaxHops and MaxSwaps, fixed in commit 077577b.

Cyfrin: Verified.

6.2 Low Risk

6.2.1 Validation is missing for tokenA in `SwapExchange::calculateMultiSwap()`

Severity: Low

Description: The protocol supports claiming a chain of swaps and the function `SwapExchange::calculateMultiSwap()` is used to do some calculations including the amount of tokenA that can be received for a given amount of tokenB. Looking at the implementation, the protocol does not validate that the tokenA of the last swap in the chain is actually the same as the tokenA of `multiClaimInput`. Because this view function is supposed to be used by the frontend to 'preview' the result of a `MultiSwap`, this does not imply a direct security risk but can lead to unexpected results. (It is notable that the actual swap function `SwapExchange::_claimMultiSwap()` implemented a proper validation.)

```solidity
SwapExchange.sol
    function calculateMultiSwap(SwapUtils.MultiClaimInput calldata multiClaimInput) external view returns (SwapUtils.SwapCalculation memory) {
        uint256 swapIdCount = multiClaimInput.swapIds.length;
        if (swapIdCount == 0 || swapIdCount > _maxHops) revert Errors.InvalidMultiClaimSwapCount(_maxHops, swapIdCount);
        if (swapIdCount == 1) {
            SwapUtils.Swap memory swap = swaps[multiClaimInput.swapIds[0]];
            return SwapUtils._calculateSwapNetB(swap, multiClaimInput.amountB, _feeValue, _feeDenominator, _fixedFee);
        }
        uint256 matchAmount = multiClaimInput.amountB;
        address matchToken = multiClaimInput.tokenB;
        uint256 swapId;
        bool complete = true;
        for (uint256 i = 0; i < swapIdCount; i++) {
            swapId = multiClaimInput.swapIds[i];
            SwapUtils.Swap memory swap = swaps[swapId];
            if (swap.tokenB != matchToken) revert Errors.NonMatchingToken();
            if (swap.amountB < matchAmount) revert Errors.NonMatchingAmount();
            if (matchAmount < swap.amountB) {
                if (!swap.isPartial) revert Errors.NotPartialSwap();
                matchAmount = MathUtils._mulDiv(swap.amountA, matchAmount, swap.amountB);
                complete = complete && false;
            }
            else {
                matchAmount = swap.amountA;
            }
            matchToken = swap.tokenA;
        }
        (uint8 feeType,) = _calculateFeeType(multiClaimInput.tokenA,
↪ multiClaimInput.tokenB);//@audit-issue no validation matchToken == multiClaimInput.tokenA
        uint256 fee = FeeUtils._calculateFees(matchAmount, multiClaimInput.amountB, feeType,
↪ swapIdCount, _feeValue, _feeDenominator, _fixedFee);
        SwapUtils.SwapCalculation memory calculation;
        calculation.amountA = matchAmount;
        calculation.amountB = multiClaimInput.amountB;
        calculation.fee = fee;
        calculation.feeType = feeType;
        calculation.isTokenBNative = multiClaimInput.tokenB == Constants.NATIVE_ADDRESS;
        calculation.isComplete = complete;
        calculation.nativeSendAmount = SwapUtils._calculateNativeSendAmount(calculation.amountB,
↪ calculation.fee, calculation.feeType, calculation.isTokenBNative);
        return calculation;
    }
```

Impact: The function will return an incorrect swap calculation result if the last swap in the chain has a different tokenA than the tokenA of `multiClaimInput` and it can lead to unexpected results.

Recommended Mitigation: Add a validation that the tokenA of the last swap in the chain is the same as the tokenA of `multiClaimInput`.

Protocol: Fixed in commit d3c758e.

Cyfrin: Verified.

6.3 Informational Findings

6.3.1 Not proper variable naming

Severity: Informational

Description: The contract `FeeData` has a internal variable `_feeValue` that is used to calculate the fee. Across the usage of this variable, it is used as a numerator while calculating the fee percentage. We recommend renaming this variable to `feeNumerator` to avoid confusion.

```
FeeUtils.sol
    function _calculateFees(uint256 amountA, uint256 amountB, uint8 feeType,  uint256 hops, uint256
↪ feeValue, uint256 feeDenominator, uint256 fixedFee)
    internal pure returns (uint256) {
        if (feeType == Constants.FEE_TYPE_TOKEN_B) {
            return MathUtils._mulDiv(amountB, feeValue, feeDenominator) * hops;
        }
        if (feeType == Constants.FEE_TYPE_TOKEN_A) {
            return MathUtils._mulDiv(amountA, feeValue, feeDenominator) * hops;
        }
        if (feeType == Constants.FEE_TYPE_ETH_FIXED) {
            return fixedFee * hops;
        }
        revert Errors.UnknownFeeType(feeType);
    }
```

Protocol: Fixed in commit f6154c9.

Cyfrin: Verified.

6.3.2 Unnecessary logical operation

Severity: Informational

Description: In the function `SwapExchange::calculateMultiSwap()` there is a logical operation that is not necessary in the for loop.

```
SwapExchange.sol
        for (uint256 i = 0; i < swapIdCount; i++) {
            swapId = multiClaimInput.swapIds[i];
            SwapUtils.Swap memory swap = swaps[swapId];
            if (swap.tokenB != matchToken) revert Errors.NonMatchingToken();
            if (swap.amountB < matchAmount) revert Errors.NonMatchingAmount();
            if (matchAmount < swap.amountB) {
                if (!swap.isPartial) revert Errors.NotPartialSwap();
                matchAmount = MathUtils._mulDiv(swap.amountA, matchAmount, swap.amountB);
                complete = complete && false;//@audit-issue INFO unnecessary operation, just set complete=false
            }
            else {
                matchAmount = swap.amountA;
            }
            matchToken = swap.tokenA;
        }
```

Protocol: Fixed in commit a079c11.

Cyfrin: Verified.

6.4 Gas Optimizations

6.4.1 Using bools for storage incurs overhead

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from `false` to `true`, after having been `true` in the past. Check more here.

```
File: helpers/FeeData.sol

18:     mapping(address => bool) public feeTokenMap;
```

```
File: helpers/TransferHelper.sol

16:     bool public rewardsActive;
```

Protocol: Fixed in commits 2d48a4b, 84dbd3a.

Cyfrin: Verified.

6.4.2 Cache array length outside of loop

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

```
File: SwapExchange.sol

403:         for (uint i = 0; i < swapIds.length; i++) {
```

```
File: helpers/FeeData.sol

55:         for (uint i = 0; i < feeTokenAddresses.length; i++) {
61:         for (uint i = 0; i < feeTokenKeys.length; i++) {
```

Protocol: Fixed in commit 9ba2a93

Cyfrin: Verified.

6.4.3 For Operations that will not overflow, you could use unchecked

```
File: SwapExchange.sol

209:    ++recordCount;

254:    totalNativeSendAmount += calculation.nativeSendAmount;

407:    total += swap.amountA;
```

```
File: libraries/SwapUtils.sol

93:             uint256 expectedValue = amount + fee;
```

Protocol: Fixed in commit f7a5dac.

Cyfrin: Verified.

6.4.4 Use Custom Errors

Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost. Read more here.

```
File: helpers/FeeData.sol

32:         require(feeValue < _feeDenominator, "Fee percentage must be less than 1");
```

```
File: helpers/TransferHelper.sol

20:         require(rewardAddress != address(0), "Reward Address is Invalid");

87:         require(rewardAddress != address(0), "Reward Address is Invalid");

98:         require(rewardHandler.logTokenFee(token, fee), "LogTokenFee failed");

103:         require(rewardHandler.logNativeFee(fee), "LogTNativeFee failed");
```

```
File: libraries/TransferUtils.sol

36:         require(erc20 != IERC20(address(0)), "Token Address is not an ERC20");

38:         require(erc20.transfer(to, amount), "ERC20 Transfer failed");

40:         require(balance >= (initialBalance + amount), "ERC20 Balance check failed");

45:         require(erc20 != IERC20(address(0)), "Token Address is not an ERC20");

47:         require(erc20.transferFrom(from, to, amount), "ERC20 Transfer failed");

49:         require(balance >= (initialBalance + amount), "ERC20 Balance check failed");

54:         require(flag == true, "ETH transfer failed");
```

Protocol: Fixed in commit ffe50aa and 9ba796f

Cyfrin: Verified.

6.4.5 Don't initialize variables with default value

```
File: SwapExchange.sol

130:         for (uint256 i = 0; i < length; i++) {

161:         for (uint256 i = 0; i < swapIdCount; i++) {

247:         for (uint256 i = 0; i < length; i++) {

346:         for (uint256 i = 0; i < swapIdCount; i++) {

403:         for (uint i = 0; i < swapIds.length; i++) {
```

```
File: helpers/FeeData.sol

55:         for (uint i = 0; i < feeTokenAddresses.length; i++) {

61:         for (uint i = 0; i < feeTokenKeys.length; i++) {
```

```
File: libraries/Constants.sol

6:     uint8 public constant FEE_TYPE_ETH_FIXED = 0;
```

Protocol: Fixed for for-loops in commit 83ffebcc099da256ec53644afc733eab2586c636.

Cyfrin: Verified.

6.4.6 `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

```
File: SwapExchange.sol

130:        for (uint256 i = 0; i < length; i++) {

161:        for (uint256 i = 0; i < swapIdCount; i++) {

247:        for (uint256 i = 0; i < length; i++) {

346:        for (uint256 i = 0; i < swapIdCount; i++) {

403:        for (uint i = 0; i < swapIds.length; i++) {
```

```
File: helpers/FeeData.sol

55:        for (uint i = 0; i < feeTokenAddresses.length; i++) {

61:        for (uint i = 0; i < feeTokenKeys.length; i++) {
```

Protocol: Fixed in commit 5662786.

Cyfrin: Verified.

6.4.7 Use != 0 instead of > 0 for unsigned integer comparison

```
File: helpers/FeeData.sol

64:        while (feeTokenKeys.length > 0) {
```

```
File: libraries/SwapUtils.sol

96:        else if (sentAmount > 0) {
```

Protocol: Fixed in commit 4679de1.

Cyfrin: Verified.