SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: MRHB
Date: November 24th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for MRHB
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	DEX
Platform	EVM
Network	Ethereum, BSC
Language	Solidity
Methodology	Link
Changelog	07.10.2022 - Initial Review 02.11.2022 - Second Review 17.11.2022 - Third Review 24.11.2022 - Fourth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by MRHB (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/Marhaba-DeFi/commodity_swap
Commit:
d1fffd77b9ea57cb7e872b42913c31713c738a54
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Contracts:

```
File: ./contracts/interfaces/IERC20.sol
SHA3: 13f54b81ccda261985ba16cc17fd75a95d4afab9537e4b90969735959b2d7a6a

File: ./contracts/interfaces/ISwap.sol
SHA3: ee4193ad09e35c3ef7d3d4635bb2e7b671e411a47e01b106f213de493b1412e8

File: ./contracts/interfaces/ISwapExtras.sol
SHA3: 009a0c8732981bfee745b24161fe2abe2ca9b7f81f15d5c2247c68dfd372d9df

File: ./contracts/interfaces/ISwapFactoryExtended.sol
SHA3: 9fb0dd78f580053cf7bee0d07d20aff27faee89d343fb4a2301a9f257a90a930

File: ./contracts/lib/Lib.sol
SHA3: e529fd622c0d711b309b96e1c2c1b2e382599b60744aed67b0ef2f1bdf135752

File: ./contracts/swap/BaseSwap.sol
SHA3: 854d3d0d09a15c0b5e170199bab51755f60df840a428388de95b3d44a29e48ee

File: ./contracts/swap/ChainlinkSwap.sol
SHA3: 28d591e09973074a64462d9b9fd3bafec04e4dd1b47bc29bfdaccd56d0128eb2

File: ./contracts/swap/Swap.sol
SHA3: 3e1841c657c38108f3f45e295d38716a6634120bc8513556cc2bd7a7f0c56893

File: ./contracts/SwapFactory.sol
SHA3: dcdd9c06c751251eca2eceaa7d29ca252e7a75716ba14429602fa842c45fc52e

File: ./contracts/SwapFactoryExtended.sol
SHA3: b7fd41015f14d06604c77096688aec6d453f346dc9e1f6f782255a6874e99a0e

File: ./contracts/utils/Pausable.sol
SHA3: 7f4e50ff45cf2b975a20fff9ad3f2c059f35f4bb2ce12cb05d6b281c6322342b

File: ./contracts/utils/TransferHelper.sol
SHA3: cbe42435dcd8e8d510a95d6952057f08ba4910df7d657482c408b13522d7ab90
```

Second review scope

Repository:
 https://github.com/Marhaba-DeFi/commodity_swap
Commit:
 b8c140886e262eaf7b6d167474fee30066627e7d
Documentation:
 Functional requirements
 Technical description

Integration and Unit Tests: Yes
Contracts:

```
File: ./contracts/interfaces/IERC20.sol
SHA3: b732cf9ce2d2d0b944aef2ce28db21c85cbc40e5cee0fe8d537559064554d533

File: ./contracts/interfaces/IPool.sol
SHA3: 968b188cc6503066ece236323aaeeadb1cd0f351d4907e68af9092b8624ffd46

File: ./contracts/interfaces/IPoolExtras.sol
SHA3: 13feed663c37cb96929c70a80b05a3818a8e86719808920c858c135ad948bcbc

File: ./contracts/interfaces/IPoolFactoryExtended.sol
SHA3: 1849f34dff82c4821ca9fd9e8d8fa38e3ca1b82a49f093f2903e7833c255a471

File: ./contracts/lib/Lib.sol
SHA3: a783b06785c1e0f1cdc433d6cbb526e16cd0bcdd06c69a427307b851ed3f75ed

File: ./contracts/PoolFactory.sol
SHA3: 6fda1e49c2aac36e22f0c20372d63926f18fbed9ea4a17bf87cb79d1bfce8297

File: ./contracts/PoolFactoryExtended.sol
SHA3: e92d6937d9623efda633a31e520c34215f59357af12d89e535e28a37379cd69a

File: ./contracts/swap/BasePool.sol
SHA3: 6f9da708f52f44d24685a70781d867d49846563eb80f43528903e3d3f08f18b2

File: ./contracts/swap/PoolApi.sol
SHA3: 419c41e496a6da399b983bdd4fac28398cf718e3ec969e9785e757ae69d1ee2e

File: ./contracts/swap/PoolChainlink.sol
SHA3: da08ef8ee85a7adf6334c906fae014fa14482a000f9c018471ece4a183a7ad9f

File: ./contracts/utils/TransferHelper.sol
SHA3: 9e8030d4205e07f8068f1bd72859391c92e639b6aaa3d6d0b51f8f7e7021ed59
```

Third review scope

Repository:
 https://github.com/Marhaba-DeFi/commodity_swap
Commit:
 1e01958657fe6e7bb8f05de1fb1a9293d9639041
Documentation:
 Functional requirements
 Technical description

Integration and Unit Tests: Yes
Contracts:

```
File: ./contracts/interfaces/IERC20.sol
SHA3: b732cf9ce2d2d0b944aef2ce28db21c85cbc40e5cee0fe8d537559064554d533
```

```
File: ./contracts/interfaces/IPool.sol
SHA3: 968b188cc6503066ece236323aaeeadb1cd0f351d4907e68af9092b8624ffd46

File: ./contracts/interfaces/IPoolExtras.sol
SHA3: 13feed663c37cb96929c70a80b05a3818a8e86719808920c858c135ad948bcbc

File: ./contracts/interfaces/IPoolFactoryExtended.sol
SHA3: 1849f34dff82c4821ca9fd9e8d8fa38e3ca1b82a49f093f2903e7833c255a471

File: ./contracts/lib/Lib.sol
SHA3: c9f91b72cbe1db189d7c3f38febfdcc6d8612566ffe7f9bee0185d0119d70acd

File: ./contracts/PoolFactory.sol
SHA3: 6fda1e49c2aac36e22f0c20372d63926f18fbed9ea4a17bf87cb79d1bfce8297

File: ./contracts/PoolFactoryExtended.sol
SHA3: e92d6937d9623efda633a31e520c34215f59357af12d89e535e28a37379cd69a

File: ./contracts/swap/BasePool.sol
SHA3: da43f3e3ce0adf982162b508e5fc8cacaa91af527182982add20bf85b6270e78

File: ./contracts/swap/PoolApi.sol
SHA3: b7660a43850e8c770ca0141d539a9535157b114796f73501879f6fed2d378964

File: ./contracts/swap/PoolChainlink.sol
SHA3: 2811f60d248061ef1766b340908b79d34aa68bbe0702bfb97b2f070185b46e6f

File: ./contracts/utils/TransferHelper.sol
SHA3: 9e8030d4205e07f8068f1bd72859391c92e639b6aaa3d6d0b51f8f7e7021ed59
```

Fourth review scope

Repository:
https://github.com/Marhaba-DeFi/commodity_swap

Commit:
c485ca5cc56d5078dea05ef3438ec273af751a1a

Documentation:
Functional requirements
Technical description

Integration and Unit Tests: Yes

Contracts:

```
File: ./contracts/interfaces/IERC20.sol
SHA3: b732cf9ce2d2d0b944aef2ce28db21c85cbc40e5cee0fe8d537559064554d533

File: ./contracts/interfaces/IPool.sol
SHA3: 968b188cc6503066ece236323aaeeadb1cd0f351d4907e68af9092b8624ffd46

File: ./contracts/interfaces/IPoolExtras.sol
SHA3: 4e598acdd8f07a0faf4be7e7639f1e6ac492eccb9f11d35e5d90a6efff3ca90a

File: ./contracts/interfaces/IPoolFactoryExtended.sol
SHA3: 1849f34dff82c4821ca9fd9e8d8fa38e3ca1b82a49f093f2903e7833c255a471

File: ./contracts/lib/Lib.sol
SHA3: 85857654fc02039e2199f04e15fb3b4bd1e862540a292d111108c1a220a62ab4

File: ./contracts/PoolFactory.sol
SHA3: e85f533c1c5ebd20d90040bdcb6c4e533d426621572e3470f4d3d7139ba762ad
```

```
File: ./contracts/PoolFactoryExtended.sol
SHA3: e92d6937d9623efda633a31e520c34215f59357af12d89e535e28a37379cd69a

File: ./contracts/swap/BasePool.sol
SHA3: 4b5953d0b9f05b5727a93c71871e80be2c02ba1d7e9caf4b67c2a802498e45fd

File: ./contracts/swap/PoolApi.sol
SHA3: b7f16b90bf419ea96e290aaa04e78b9a211e2740a08bf98932753785c76e7129

File: ./contracts/swap/PoolChainlink.sol
SHA3: 39597322e19d6fc189b8cfc01729efe470286c50af73a22b5b9a31f7fcb4b51e

File: ./contracts/utils/TransferHelper.sol
SHA3: 9e8030d4205e07f8068f1bd72859391c92e639b6aaa3d6d0b51f8f7e7021ed59
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are great.
- Technical description is provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- Some values may be moved to specific constants like DIVISOR (10000).
- Swap functions are similar, consider moving common code to a reusable block.

Test coverage

Code coverage for the project is **94.66%**.

Security score

As a result of the audit, the code **does not** contain issues. The security score is **10** out of **10**.

All found issues are displayed in the "Findings" section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.

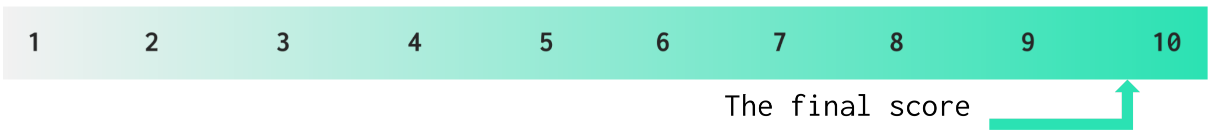

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
6 October 2022	9	5	4	3
2 November 2022	4	5	4	1
16 November 2022	2	2	2	0
24 November 2022	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Commodity swap is part of *TijarX* project, which is responsible for token swaps. It consists of the contracts:

- *BasePool* - inheritable contract, implement such features:
    - internal remove/add liquidity functions for inheriting contracts
    - set trade fee value
    - withdraw fee to dex admin and reserves to contract owner
    - destroy the contract
    - pausable and ownable functionality
- *PoolChainlink* - simple token swap contract
    - only owner can add/remove liquidity
    - anyone can swap tokens if the contract is not paused
- *PoolApi* - highly configurable token swap contract
    - only owner can add/remove liquidity
    - anyone can swap tokens if the contract is not paused
    - owner can specify where to fetch the price (API or Chainlink)
- *PoolFactory* - simple factory of *Swap* and *ChainlinkSwap* contracts
    - allows creating contracts by whitelisted users
    - it is possible to get all created contract addresses
- *PoolFactoryExtended* - sub factory contract for *PoolApi* contracts deployment

Privileged roles

Dex admin in any swap contract may:

- update swap fee from 0 to 1% value
- withdraw taken fee from swap contract
- set chainlink price feed address
- transfer the role to another user
- withdraw fee to dex admin, reserves to contract owner, and destroy contract
- pause/unpause swap functionality

Dex admin in *PoolApi* contract may:

- update Chainlink token and oracle addresses
- set price rate timeout duration

Contract owner of any swap contract may:

- withdraw all liquidity
- add/remove liquidity

Contract owner in *PoolApi* contract may:

- update price API link

Contract owner in *PoolChainlink* contract may:

- update *unitMultiplier* for commodity asset

Risks

- The admin may set an arbitrary price oracle address; it is recommended to manually check if the price oracle address is correct.
- Price API is specified by the contract owner and may be compromised.
- Sub factory address, responsible for creating *Swap* contracts, may be changed by factory admin; it is recommended to check if it is what exactly wanted to be deployed.
- The owner of the *PoolChainlink* contract may affect the price calculations by changing the unit multiplier used in the calculations.

Findings

■■■■ Critical

1. **Data Consistency**

   In some cases, when using Chainlink Oracle, outdated data may be received. The time when data received by the *latestRoundData* function was generated should be checked.

   In the current implementation, all fields except the price value are ignored, so outdated data may be received.

   This may lead to the wrong swap rate and loss of pool valuability.

   Path: ./contracts/swap/BaseSwap.sol : getChainLinkFeedPrice()

   Recommendation: check if the data is actual and revert the transaction if it is not so.

   Status: Fixed (second scope)

2. **Data Consistency**

   As the contract only stores 2 latest request ids, on receival, it should be carefully checked which one was received.

   Example: 2 requests for one trade side created; 1st is executed, but as *info.chainlinkRequestId[TRADE_SIDE]* already stores another request id the rate value is written to the wrong trade side.

   This may lead to the wrong swap rate obtained and loss of pool valuability.

   Path: ./contracts/swap/Swap.sol : fulfill()

   Recommendation: carefully manage rates received for different trade sides.

   Status: Fixed (second scope)

3. **Data Consistency**

   The contract should not use outdated price values. It should be checked that price data was obtained recently.

   As the price is not reachable from API in the same transaction, it was requested, so the rate limitation for the old stored value should be checked. However, the checking mechanic is broken: *setPriceFeed* and *isRateTimeout* functions are called successively, and in the first one *priceInfo.lastTimeStamp[ID]* is updated to the current timestamp, and in the second one, it is checked that the value is not long ago from the current timestamp.

   This may lead to outdated swap price usage and loss of pool valuability.

Path: ./contracts/swap/Swap.sol : swap(), fulfill(), setPriceFeed(), isRateTimeout()

Recommendation: record the time when data was received, do not allow swaps using an outdated price value.

Status: Fixed (second scope)

4. **Data Consistency**

The contract should not use outdated price values. It should be checked that price data was obtained recently.

According to the implementation, if the price is under timeout, a fresh one is requested, but swap happens using outdated values.

This may lead to outdated swap price usage and loss of pool valuability.

Path: ./contracts/swap/PoolApi.sol : swap(), _setPriceFeed(), _isRateTimeout()

Recommendation: do not allow swaps using an outdated price value.

Status: Fixed (third scope)

■■■ High

1. **Requirements Violation**

The *ChainlinkSwap*, *Swap* contracts should emit events when reserves are low, and swap operation is impossible. In the *ChainlinkSwap* contract, *LowTokenBalance* event is emitted when *dexData.reserveB < amountB*, but if this condition is met, the transaction will be reverted according to the next required statement. The reverted transactions do not emit events.

This leads to an inability to emit low balance events notifications, which may affect the frontend part of the system and makes events declaration with the related logic redundant.

Paths:
./contracts/swap/ChainlinkSwap.sol : swap()
./contracts/swap/Swap.sol : swap()

Recommendation: remove redundant events which are never called, or rewrite the logic not to revert failed *swap* function calls.

Status: Fixed (second scope)

2. **Requirements Violation**

According to the documentation, initialization of Chainlink price feed and price info should be done only once for a contract, and only dex admin should be able to update some of the values.

However, in the current implementation, it is possible for an owner to call the function twice, resetting the values.

Path: ./contracts/swap/Swap.sol : initChainlinkAndPriceInfo()

Recommendation: prevent calling the function twice.

Status: Fixed (second scope)

3. Requirement Violation

According to documentation, contracts should not ask Chainlink for the current price if the last obtained price is not rate limited.

According to the current implementation, each time swap happens, the contract asks for a price update from the API.

Path: ./contracts/swap/Swap.sol : setPriceFeed()

Recommendation: implement code according to documentation or modify docs according to the implementation.

Status: Fixed (second scope)

4. Denial of Service Vulnerability

According to the implementation, absolute amounts should be processed to the functions. However, the price may be changed in case of high asset volatility, and the amounts may be considered wrong during function calls.

This may lead to inability of adding or removing liquidity.

Path: ./contracts/swap/PoolApi.sol : addLiquidity(), removeLiquidity()

Recommendation: provide slippage to be able to provide approximate values.

Status: Fixed (third scope)

5. Denial of Service Vulnerability

As oracle may return any value, the result should be validated.

The API may return a rate value equal to 0. If rate 0 is provided, no swaps may be executed on the contract as *setPriceFeed* and *isRateTimeout* functions are called in the *swap* function successively. In case the rate is 0, *priceInfo.lastTimeStamp[ID]* value is not updated, so *isRateTimeout* will never return *false* if rate limitation happens once.

It is not possible to request new rates except for a hard reset of the contract with *initChainlinkAndPriceInfo* which generally should not be accessible according to another issue.

This may lead to reverting of all swap transactions.

Path: ./contracts/swap/PoolApi.sol : setPriceFeed(), fulfill()

Recommendation: implement rate limitation mechanism safely, validate values obtained by oracle.

Status: Fixed (third scope)

6. **Denial of Service Vulnerability**

As price is requested each time *requestFreshPrice* function is invoked (in case of current value is under timeout), request may not be satisfied as callback from ChainLink is accepted only for the latest request.

In such a way, if spamming to *requestFreshPrice* happens, the price is not updated after requests fulfillment. Each request consumes some *LINK* tokens from the contract balance, and excessively frequent requests may lead to token loss.

This may lead to price being under timeout until users stop calling *requestFreshPrice* function for some time.

Path: ./contracts/swap/PoolApi.sol : _isRateTimeout(), requestFreshPrice()

Recommendation: do not send a repeat request to ChainLink until some time is gone.

Status: Fixed (fourth scope)

7. **Data Consistency**

According to the current implementation, price lifetime is started from receival. However, the price may be gotten from API some time earlier and not be consistent on receival by contract.

This may lead to outdated price usage.

Path: ./contracts/swap/PoolApi.sol : fulfill(), _requestVolumeData()

Recommendation: start counting price timeout from a request creation, storing it to cache and assigning it to *priceInfo.lastTimeStamp[ID]* on fulfillment.

Status: Fixed (fourth scope)

Medium

1. **Checks-Effects-Interactions Pattern Violation**

During functions, the state variables are being updated after the external calls, or checks are done after state variables are updated.

This can lead to reentrancies, race conditions, or denial of service vulnerabilities.

Paths:
./contracts/swap/BaseSwap.sol : _removeLiquidity()
./contracts/swap/ChainlinkSwap.sol : swap()
./contracts/swap/Swap.sol : swap()

Recommendation: implement function according to the Checks-Effects-Interactions pattern.

Status: Fixed (second scope)

2. **Double Event Emitting**

The *BaseSwap* contract has internal functions which emit events, but the *ChainlinkSwap* contract has functions which call internal *BaseSwap* functions and duplicates emitting such events: *LiquidityAdded*, *LiquidityRemoved*.

Double event emitting may confuse users and cause unexpected behavior of the off-chain logic and frontend interface.

Paths:
./contracts/swap/BaseSwap.sol : _addLiquidity(), _removeLiquidity()
./contracts/swap/ChainlinkSwap.sol : addLiquidity(), removeLiquidity()

Recommendation: remove double event emitting from *ChainlinkSwap* contract.

Status: Fixed (second scope)

3. **Requirement Violation**

According to documentation, contract owners should be able to set rate timeout between 60 and 300 seconds, but according to the implementation, they can set only between 120 and 300.

Paths:
./contracts/SwapFactory.sol : createLinkFeedWithApiSwap()
./contracts/swap/Swap.sol : setRateTimeOut()

Recommendation: implement code according to requirements.

Status: Fixed (second scope)

4. **Missing Parameters Validation**

According to implementation, rate timeout and trade fee values should be validated, but the validations are missed in the constructors.

However, the validations are done in the factories, the contracts could be deployed manually, so they are needed.

Paths:
./contracts/swap/ChainlinkSwap.sol : constructor()
./contracts/swap/Swap.sol : constructor()

Recommendation: validate parameters consciously.

Status: Fixed (second scope)

5. **Unnecessary Logic & Undocumented Behavior**

According to the implementation, there are 2 price feeds: BUY and SELL. Intuitively it is expected *BUY_PRICE* to be equal to *(1 / SELL_PRICE) * (10**16)*. However, in the function, the feeds are used like the same value is presented in both feeds.

If index is SELL, *(amount * SELL_PRICE) / (10**8)* formula is used.
If index is BUY, *(amount * (10**8)) / BUY_PRICE* formula is used.

And if *BUY_PRICE = (1 / SELL_PRICE) * (10**16)* =>
*(amount * (10**8)) / BUY_PRICE = (amount * (10**8)) / (1 / SELL_PRICE) / (10**16)* = *(amount * SELL_PRICE) / (10**8)* which means that there is no difference between the two formulas and only one feed may be kept.

Unsafe code is used in case of fetching price when adding/removing liquidity. Indexes are hardcoded both in fetch price and add/remove liquidity functions. Only SELL price feed is used. No documentation of the API was provided.

Path: ./contracts/swap/Swap.sol : fetchPrice(), convertPrice()

Recommendation: remove unnecessary code and variables and keep logic clear or provide documentation of why 2 different price feeds are needed.

Status: Fixed (second scope)

6. **Checks-Effects-Interactions Pattern Violation**

During functions, the state variables are being updated after the external calls, or checks are done after state variables are updated.

This may lead to reentrancies, race conditions, or denial of service vulnerabilities.

Path: ./contracts/swap/PoolApi.sol : swap()

Recommendation: implement function according to the Checks-Effects-Interactions pattern.

Status: Fixed (third scope)

7. **Requirement Violation**

The contract has the function *setApiInfo* which allows admin to set oracle details; after setting new values, it is supposed to update the prices by making requests to the oracles. Due to the type in the code, the function requests the price update for “buying” swaps twice.

This may lead to unexpected ChainLink tokens consumption and wrong calculations during the “selling” swaps.

Path: ./contracts/swap/PoolApi.sol : setApiInfo()

Recommendation: request price update for the sell side.

Status: Fixed (third scope)

8. **Requirement Violation**

According to documentation, contract owners should be able to set rate timeout between 60 and 120 seconds, but according to the implementation, they can set only between 60 and 300.

Path: ./contracts/lib/Lib.sol : _checkRateTimeout()

Recommendation: implement code according to requirements.

Status: Fixed (third scope)

9. **Requirement Violation**

According to documentation, dex admin should be the code supplier. However, it is not checked in factory contracts.

The pool creator may provide any address as dex admin (except 0x0). In such a way, pool contracts managed by the owner itself may be added to the factory pools list.

This may lead to unexpected admin of a pool and inability for receiving fees from swaps by the admin.

Path: ./contracts/PoolFactory.sol : createLinkFeedWithApiPool(), createChainlinkPool()

Recommendation: check the *dexSettings.dexAdmin* value or assign it manually.

Status: Fixed (fourth scope)

10. **Broken Code Design**

The computations done in the function *_convertUSDToStable(getCommodityPrice())* may lead to possible loss of precision as *_convertUSDToStable* relies on price value itself.

This may lead to new issues during further development and possible loss of precision.

Path: ./contracts/swap/PoolChainlink.sol : getAmountOut()

Recommendation: design functions to work with assets safely, separating price calculation logic; it is recommended to rewrite logic of the *_convertUSDToStable* function to add price requests from *getCommodityPrice*, this would allow to calculate price following “multiplication before division” rule, which will increase the precision of computations.

Status: Mitigated (the precision loss is quite small)

Low

1. **Default Variable Visibility**

The lack of variable visibility may cause unexpected variable visibility in derived contracts.

Path: ./contracts/SwapFactory.sol : factoryAdmin, subFactory

Recommendation: specify the needed visibility during the variable initialization.

Status: Fixed (second scope)

2. **Missing Interface Inheritance**

   It is recommended for contracts to inherit corresponding interfaces in order to ensure that the implementation matches the interface.

   Path: ./contracts/SwapFactoryExtended.sol : ISwapFactoryExtended

   Recommendation: provide inheritance consciously.

   Status: Fixed (second scope)

3. **Redundant Event Declaration**

   The declaration of unnecessary events will increase the Gas consumption of the code. Thus they should be removed from the code.

   Path: ./contracts/interfaces/ISwapExtras.sol : ChainlinkFeedAddrChanged

   Recommendation: remove or emit *ChainlinkFeedAddrChanged* event.

   Status: Fixed (second scope)

4. **Modification of a Well-Known Contracts**

   Imported or copy-pasted contracts (like SafeMath, Context, Ownable, etc.) should not be modified to keep the code clear for further development and to avoid unexpected issues.

   Paths:
   ./contracts/swap/BaseSwap.sol : ./../utils/Pausable.sol

   Recommendation: replace contracts with a trusted implementation (eg. OpenZeppelin).

   Status: Fixed (second scope)

5. **Missing Zero Address Validation**

   Address parameters are being used without checking against the possibility of *0x0*.

   However, the validations are done in the factories, the contracts could be deployed manually, so they are needed.

   Paths:
   ./contracts/swap/BaseSwap.sol : setChainlinkFeedAddress()
   ./contracts/swap/ChainlinkSwap.sol : constructor()
   ./contracts/swap/Swap.sol : constructor()

   Recommendation: add zero address validation.

   Status: Fixed (second scope)

6. Code Duplication

The contract has the function to set fees, the function argument is validated with the condition inside the *required* statement, but the project contains the library with the function for fee value validation.

Code duplications may lead to the redundant Gas usage during the contract deployment.

Path: ./contracts/swap/BaseSwap.sol : setTradeFee()

Recommendation: use the *SwapLib* *checkFee* function instead of the custom require statement.

Status: Fixed (second scope)

7. Wrong Natspec

The code of the project has comments which are not relevant to the implementation:

Swap admin to withdraw reserves in case of emergency - the function is accessible for owner, but not for admin.

Allows comm-dex admin to empty dex, sends reserves to comm-dex admin and fee to comm-dex admin - reserves and fees are withdrawn to the owner address, but not to the admin.

The comments and implementation mismatch may confuse developers in the future.

Path: ./contracts/swap/BaseSwap.sol

Recommendation: update the comments according to the implementation.

Status: Fixed (second scope)

8. Bug in a Deploy Script

It is required to do a pre-setup before running tests. To run tests from the *./test/2_priceFeedApi.test.js* file, it is required to run the *./scripts/deploySwapWithApi.js* file, but the file has an error on *line 44*. The address property of the contract ethers factory object *swapFactoryExtended.address* is passed as an argument to the constructor, but the ethers factory object does not have the *address* property.

This leads to an inability to run tests in *./test/2_priceFeedApi.test.js* file.

Path:
./scripts/deploySwapWithApi.js

Recommendation: replace *swapFactoryExtended.address* with *swapFactory.address* (swapFactory is an ethers object of the deployed contract).

Status: Fixed (second scope)

9. Redundant Library Usage

The contracts have the *using* library statements, which are applied to all variable types (*SwapLib for **, *ChainlinkLib for **, *SwapLib for **, *PriceLib for **), which is redundant.

Paths:
./contracts/swap/BaseSwap.sol
./contracts/swap/Swap.sol

Recommendation: specify the library using only for the required data types.

Status: Fixed (second scope)

10. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of *0x0*.

Paths:
./contracts/swap/BasePool.sol : withDrawAndDestory()
./contracts/swap/Swap.sol : initChainlinkAndPriceInfo(), setChainlinkOracleAddress(), setChainlinkTokenAddress()

Recommendation: add zero address validation.

Status: Fixed (third scope)

11. Default Variable Visibility

The lack of variable visibility may cause unexpected variable visibility in derived contracts.

Path: ./contracts/swap/PoolApi.sol : initialized

Recommendation: specify the needed visibility during the variable initialization.

Status: Fixed (third scope)

12. Missing Validation

Some variables may be set to unexpectedly high values by the admin.

This may lead to unexpected behavior.

Paths:
./contracts/swap/BasePool.sol : constructor()
./contracts/swap/PoolApi.sol : _setFeedSetting()
./contracts/swap/PoolChainlink.sol : _setFeedSetting()

Recommendation: add validation to prevent setting values out of the reasonable range.

Status: Fixed (third scope)

13. Redundant Calculations

The contract uses the custom library, which includes the *SUPPORTED_DECIMALS* contract. The constant is redundantly used during the calculations in *_convertUSDToStable* function.

This leads to unoptimized Gas consumption.

Path: ./contracts/swap/BasePool.sol : _convertUSDToStable()

Recommendation: rework the logic to remove the redundant calculations.

Status: Fixed (third scope)

14. Missing Validation

Some variables may be set to unexpectedly high values by the admin.

This may lead to unexpected behavior.

Path: ./contracts/swap/BasePool.sol : updateBuySpotDifference(), updateSellSpotDifference()

Recommendation: add validation to prevent setting values out of the reasonable range.

Status: Fixed (fourth scope)

15. Unreasonable Validation

Slippage is not allowed to be zero. Although slippage functionality is useful as it prevents failures due to high volatility, the ability to disable slippage may be wanted by users.

This may lead to the inability to provide the exact wanted result.

Path: ./contracts/swap/BasePool.sol : verifySlippageTolerance()

Recommendation: remove minimum slippage value.

Status: Fixed (fourth scope)

16. Variable Duplication

The *priceInfo.lastRequestTime* and *priceInfo.cachedRequestTimeStamp* store the same value.

Path: ./contracts/swap/PoolApi.sol : _requestVolumeData(), _setPriceFeed()

Recommendation: merge the variable to save storage.

Status: Mitigated (it is needed for further system scalability)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.