// HALBORN

DRAFT

Jarvis Network Atomic-Swap V2

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 5th, 2021 - November 16th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/11/2021	Juned Ansari
0.2	Document Updates	11/12/2021	Juned Ansari
0.3	Document Review	11/12/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Juned Ansari	Halborn	Juned.Ansari@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Jarvis Network engaged Halborn to conduct a security assessment on their AtomicSwap V2 contracts beginning on November 5th, 2021 and ending November 16th, 2021. This security assessment was scoped to the AtomicSwap V2 contracts code in Solidity.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure development.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that all Atomic-Swap V2 Contract functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified several security risks that need to be addressed.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover

flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the Jarvis Network AtomicSwap V2 and Pool V5 contract solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE : Jarvis Network AtomicSwap V2 contracts
The security assessment was scoped to the following smart contract:

Listing 1: feature/atomic-swap-v2

```
1 contracts/v2/OCLRBase.sol
2 contracts/v2/implementations/OCLRKyber.sol
3 contracts/v2/implementations/OCLRUniswapV3.sol
4 contracts/v2/implementations/OCLRUniswapV2.sol
5 contracts/v2/implementations/interfaces/IKyberRouter.sol
6 contracts/v2/interfaces/IOnChainLiquidityRouter.sol
7 contracts/v2/interfaces/IOCLRBase.sol
8 contracts/v2/OnChainLiquidityRouter.sol
```

Commit-Id : e05d4aa807c8e170e8d457c2fae1bbc941a5fc27
OUT-OF-SCOPE : External libraries and economics attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	3	1

LIKELIHOOD

IMPACT

		(HAL-01)		
	(HAL-02)			
(HAL-05)				
(HAL-06)	(HAL-03) (HAL-04)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - CONTRACT ADMIN CAN REVOKE AND RENOUNCE HIMSELF	High	-
HAL02 - REGISTRATION MISSING IMPLEMENTATION EXISTENCE VALIDATION	Medium	-
HAL03 - OUTDATED DEPENDENCIES	Low	-
HAL04 - EXPERIMENTAL FEATURES ENABLED	Low	-
HAL05 - FLOATING PRAGMA	Low	-
HAL06 - PRAGMA TOO RECENT	Informational	-

FINDINGS & TECH DETAILS

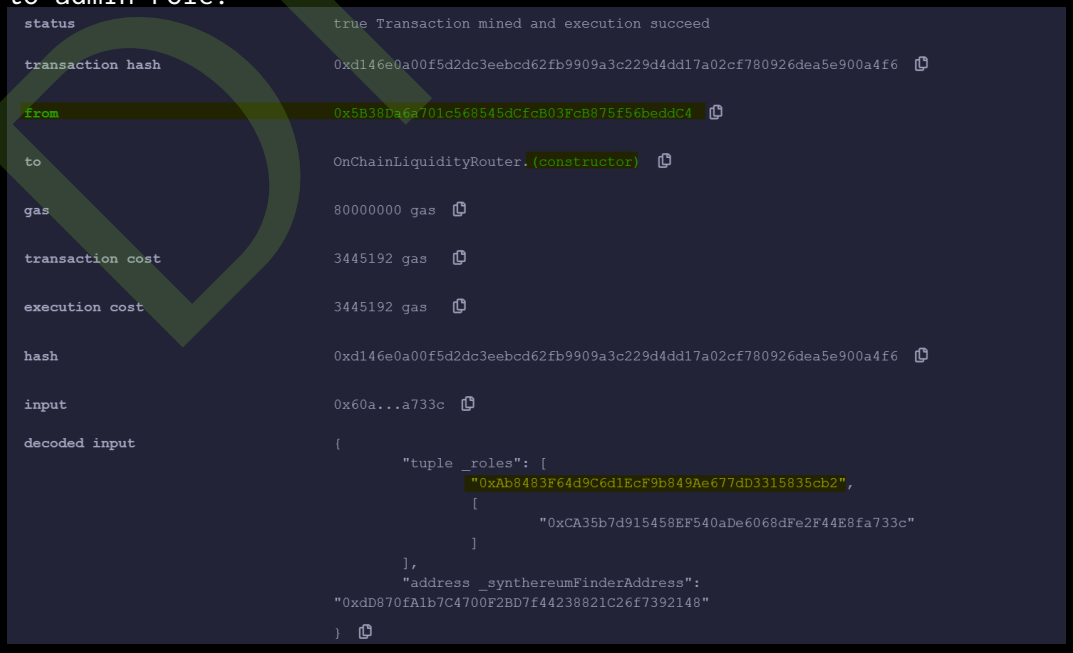
3.1 (HAL-01) CONTRACT ADMIN CAN REVOKE AND RENOUNCE HIMSELF - HIGH

Description:

The Owner of the contract is usually the account which deploys the contract. In the OnChainLiquidityRouter.sol smart contract, Only Admin can perform some privileged actions such as revokeRole, renounceRole, grantRole etc. . . , the revokeRole function is used to revoke a role, and the renounceRole function is used to renounce being a role owner. It was observed that admin can revoke his role via revokeRole, as well, can renounce ownership via renounceRole function. If an admin is mistakenly renounced/revoked, administrative access would result in the contract having no admin, eliminating the ability to call privileged functions. In such a case, contracts would have to be redeployed.

PoC Steps:

- Deploy the OnChainLiquidityRouter.sol contract using the account "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4". Supplied "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2" address as an address to admin role.



- Notice “0x5B38Da6a701c568545dCfcB03FcB875f56beddC4” (Address used to deploy the contract) missing admin role, thus couldn’t grant new roles. Admin account “0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2” can grant new roles, this implies admin account is an owner, only account that can perform privilege tasks.

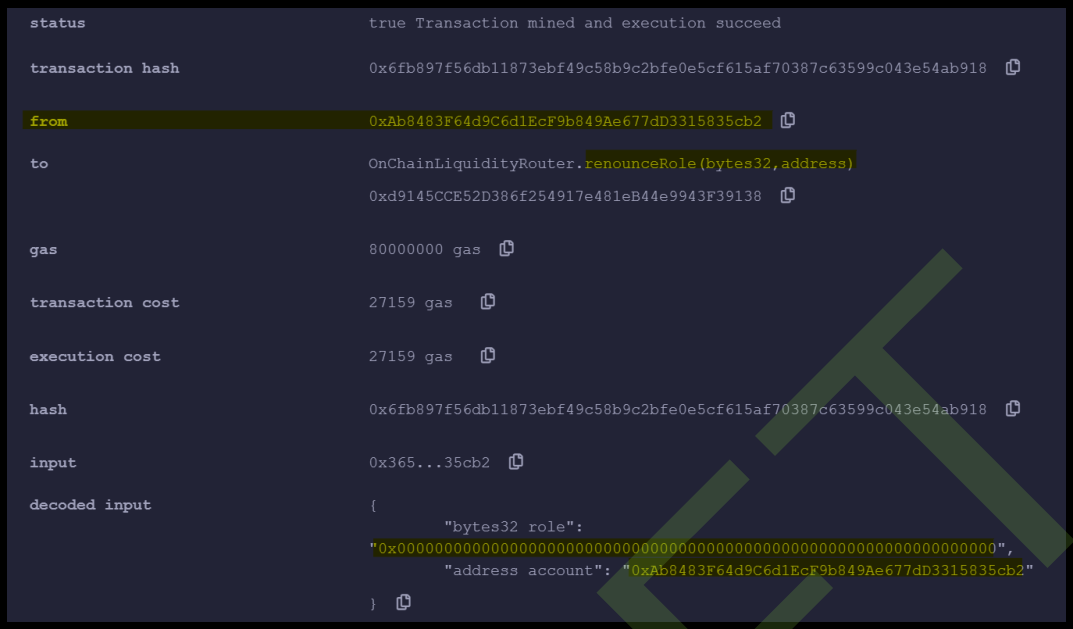
```
transact to OnChainLiquidityRouter.grantRole errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "AccessControl: account 0x5b38da6a701c568545dcfcb03fcb875f56beddc4 is missing role
0x0000000000000000000000000000000000000000000000000000000000000000".
```

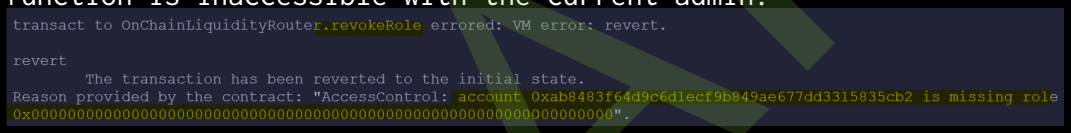
- Admin calls revokeRole function to revoke own admin role.

```
status                  true Transaction mined and execution succeed
transaction hash        0xf6c8f3d19bee3d5441f60fd37aa9964a3474d12ec0ba4b6b41c65d4491c51623
from                    0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to                      OnChainLiquidityRouter.revokeRole(bytes32,address)
                        0xd9145CCE52D386f254917e481eB44e9943F39138
gas                     80000000 gas
transaction cost        38942 gas
execution cost          38942 gas
hash                    0xf6c8f3d19bee3d5441f60fd37aa9964a3474d12ec0ba4b6b41c65d4491c51623
input                   0xd54...35cb2
decoded input           {
                              "bytes32 role":
                        "0x0000000000000000000000000000000000000000000000000000000000000000",
                              "address account": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2"
                        }
decoded output          {}
logs                    [
                              {
                                    "from": "0xd9145CCE52D386f254917e481eB44e9943F39138",
                                    "topic":
                        "0xf6391f5c32d9c69d2a47ea670b442974b53935d1edc7fd64eb21e047a839171b",
                                    "event": "RoleRevoked",
                                    "args": {
                                          "0":
                        "0x0000000000000000000000000000000000000000000000000000000000000000",
```

- Admin calls renounceRole function to renounce an admin of the contract.



- Now call to function such as revokeRole, renounceRole, grantRole function is inaccessible with the current admin.



Risk Level:

Likelihood - 3
Impact - 5

Recommendations:

It is recommended that the contract Admin cannot call renounceRole or revokeRole without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceRole or revokeRole function should be confirmed for two or more users.

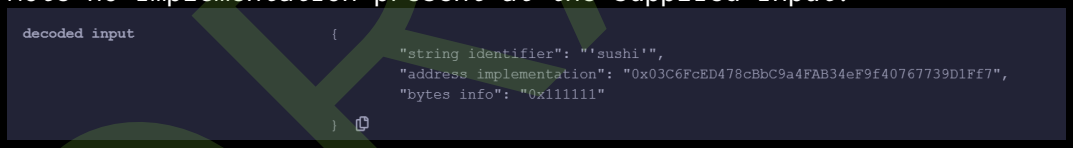
3.2 (HAL-02) REGISTRATION MISSING IMPLEMENTATION EXISTENCE VALIDATION - MEDIUM

Description:

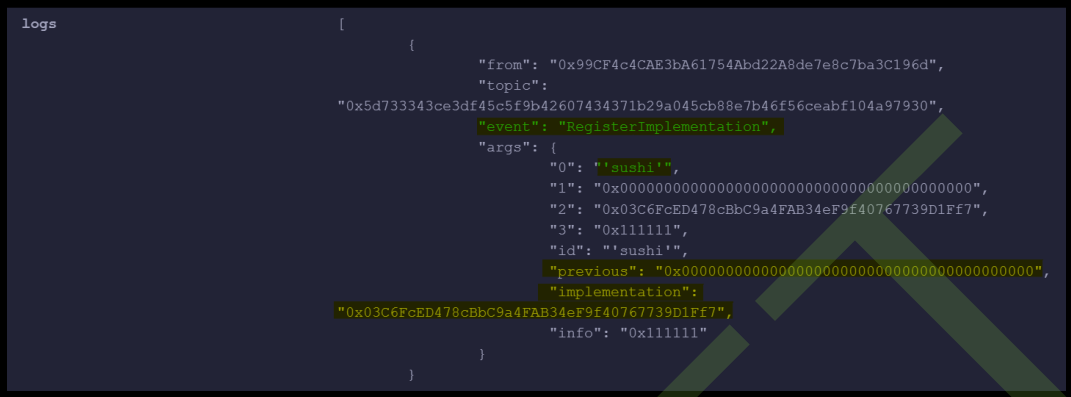
During the manual review, it was observed that in the contract OnChainLiquidityRouter.sol, the function registerImplementation doesn't validate whether the supplied implementation address exists or not. That leads to spurious mappings that may be changed in the future to that address or added to malicious implementations, and the events log will not generate at that time because the id to implementation mapping already exists.

POC:

- External call to registerImplementation with supplied input implementation address as "0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7". Note no implementation present at the supplied input.



- registerImplementation successfully registered with the fake implementation address.



```
logs [
        {
                "from": "0x99CF4c4CAE3bA61754Abd22A8de7e8c7ba3C196d",
                "topic":
"0x5d733343ce3df45c5f9b42607434371b29a045cb88e7b46f56ceabf104a97930",
                "event": "RegisterImplementation",
                "args": {
                        "0": "'sushi'",
                        "1": "0x0000000000000000000000000000000000000000",
                        "2": "0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7",
                        "3": "0x111111",
                        "id": "'sushi'",
                        "previous": "0x0000000000000000000000000000000000000000",
                        "implementation":
"0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7",
                        "info": "0x111111"
                }
        }
```


Code Location:

In the registration process, implementation existence validation is missing before `id` to `implementation` address mapping on line #88.

Listing 2: OnChainLiquidityRouter.sol (Lines 88)

```
82  function registerImplementation(
83    string calldata identifier,
84    address implementation,
85    bytes calldata info
86  ) external onlyMaintainer() {
87    address previous = idToAddress[keccak256(abi.encode(identifier
        ))];
88    idToAddress[keccak256(abi.encode(identifier))] =
        implementation;
89    dexImplementationInfo[implementation] = info;
90
91    emit RegisterImplementation(identifier, previous,
        implementation, info);
92  }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Consider adding a require check to validate implementation existence during the registration process before id to implementation address assignment.

3.3 (HAL-03) OUTDATED DEPENDENCIES - LOW

Description:

It was noticed that the 4.1.0 version of openzepplin-contracts is used in the in-scope Atomic-swap v2 contract. However, the latest version of those libraries is 4.3.2, which fixes a vulnerability in UUPSUpgradeable. Even though UUPSUpgradeable is not used directly within these contracts, but it is a security best practice keeping all libraries up-to-date.

Code Location:

Listing 3: atomic-swap-v2/libs/atomic-swap/package.json (Lines 27)

```
22  "devDependencies": {
23    "@jarvis-network/synthereum-contracts": "*",
24    "@nomiclabs/hardhat-etherscan": "^2.1.4",
25    "@nomiclabs/hardhat-truffle5": "^2.0.0",
26    "@nomiclabs/hardhat-web3": "^2.0.0",
27    "@openzeppelin/contracts": "4.1.0",
28    "hardhat-gas-reporter": "^1.0.4"
29  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Even though UUPSUpgradeable is not used directly within these contracts, it is always important to keep all libraries up-to-date.

References:

Open Zeppelin Advisory
UUPS Implementation Workaround

3.4 (HAL-04) EXPERIMENTAL FEATURES ENABLED - LOW

Description:

ABIEncoderV2 is enabled to be able to pass struct type into a function, both web3 and another contract. The use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(. . .) as arguments in external function calls or in event data without prior assignment to a local variable. Using return does not trigger the bug. The types bytesNN and bool will result in corrupted data, while enum might lead to an invalid revert.

Furthermore, arrays with elements shorter than 32 bytes may not be handled correctly, even if the base type is an integer type. Encoding such arrays in the way described above can lead to other data in the encoding being overwritten if the number of elements encoded is not a multiple of the number of elements that fit a single slot. If nothing follows the array in the encoding (note that dynamically sized arrays are always encoded after statically sized arrays with statically sized content), or if only a single array is encoded, no other data is overwritten. There are known bugs that are publicly released while using this feature. However, the bug only manifests itself when all the following conditions are met:

Storage data involving arrays or structs is sent directly to an external function call, to abi.encode or to event data without prior assignment to a local (memory) variable.
There is an array that contains elements with size less than 32 bytes or a struct that has elements that share a storage slot or members of type bytesNN shorter than 32 bytes.
In addition to that, in the following situations, the code is NOT affected:

All the structs or arrays only use uint256 or int256 types.

If only using integer types (that may be shorter) and only encode at most one array at a time.
If only returning such data and do not use it in abi.encode, external calls or event data.

ABIEncoderV2 is enabled to be able to pass struct type into a function, both web3 and another contract. Naturally, any bug can have wildly varying consequences depending on the program control flow, but we expect that this is more likely to lead to malfunction than exploitability. The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

Code Location:

Listing 4

```
1 contracts/v2/interfaces/IOCLRBase.sol:4:pragma experimental
      ABIEncoderV2;
2 contracts/v2/OnChainLiquidityRouter.sol:4:pragma experimental
      ABIEncoderV2;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Reference:

Solidity Optimizer and ABIEncoderV2 Bug

3.5 (HAL-05) FLOATING PRAGMA - LOW

Description:

Jarvis-Network in-scope Atomic-swap v2 contract uses the floating pragma ^0.8.4. Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Code Location:

Listing 5

```
contracts/v2/OCLRBase.sol:3:pragma solidity ^0.8.4;
contracts/v2/implementations/OCLRKyber.sol:3:pragma solidity
    ^0.8.4;
contracts/v2/implementations/OCLRUniswapV3.sol:3:pragma solidity
    ^0.8.4;
contracts/v2/implementations/OCLRUniswapV2.sol:3:pragma solidity
    ^0.8.4;
contracts/v2/implementations/interfaces/IKyberRouter.sol:1:pragma
    solidity ^0.8.4;
contracts/v2/interfaces/IOnChainLiquidityRouter.sol:3:pragma
    solidity ^0.8.4;
contracts/v2/interfaces/IOCLRBase.sol:3:pragma solidity ^0.8.4;
contracts/v2/OnChainLiquidityRouter.sol:3:pragma solidity ^0.8.4;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider locking the pragma version with known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

3.6 (HAL-06) PRAGMA TOO RECENT - INFORMATIONAL

Description:

Jarvis-Network in-scope Atomic-swap v2 Contract uses one of the latest pragma version (0.8.4) which was released on April 21, 2021. The latest pragma version (0.8.9) was released in September 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 8 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy GitHub repository, there is a json file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Code Location:

Listing 6

```
contracts/v2/OCLRBase.sol:3:pragma solidity ^0.8.4;
contracts/v2/implementations/OCLRKyber.sol:3:pragma solidity ^0.8.4;
contracts/v2/implementations/OCLRUniswapV3.sol:3:pragma solidity ^0.8.4;
contracts/v2/implementations/OCLRUniswapV2.sol:3:pragma solidity ^0.8.4;
contracts/v2/implementations/interfaces/IKyberRouter.sol:1:pragma solidity ^0.8.4;
contracts/v2/interfaces/IOnChainLiquidityRouter.sol:3:pragma solidity ^0.8.4;
contracts/v2/interfaces/IOCLRBase.sol:3:pragma solidity ^0.8.4;
contracts/v2/OnChainLiquidityRouter.sol:3:pragma solidity ^0.8.4;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities, such as pragma between 0.6.12 - 0.7.6, or the latest pragma for example, after the Solidity version 0.8.0 arithmetic operations revert to underflow and overflow by default, by using this version, utility contracts like SafeMath.sol would not be needed.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
Contract locking ether found:
        Contract OCLRBase (contracts/v2/OCLRBase.sol#20-46) has payable functions:
         - IOCLRBase.swapToCollateralAndMint(bytes,IOnChainLiquidityRouter.SwapMintParams,IOnChainLiquidityRouter.SynthereumMintParams) (contracts/v2/interfac
es/IOCLRBase.sol#20-27)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

OCLRUniswapV3.swapToCollateralAndMint(bytes,IOnChainLiquidityRouter.SwapMintParams,IOnChainLiquidityRouter.SynthereumMintParams) (contracts/v2/implementations
/OCLRUniswapV3.sol#37-194) sends eth to arbitrary user
        Dangerous calls:
        - (success) = msg.sender.call{value: address(this).balance}() (contracts/v2/implementations/OCLRUniswapV3.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Reentrancy in OnChainLiquidityRouter.redeemAndSwap(string,IOnChainLiquidityRouter.RedeemSwapParams,ISynthereumPoolOnChainPriceFeed,ISynthereumPoolOnChainPrice
Feed.RedeemParams,address) (contracts/v2/OnChainLiquidityRouter.sol#169-212):
        External calls:
        - result = implementation.functionDelegateCall(abi.encodeWithSignature(functionSig,dexImplementationInfo[implementation],inputParams,synthereumParams,
recipient)) (contracts/v2/OnChainLiquidityRouter.sol#190-199)
        Event emitted after the call(s):
        - Swap(returnValues.inputToken,returnValues.outputToken,returnValues.collateralToken,returnValues.inputAmount,returnValues.outputAmount,returnValues.c
ollateralAmountRefunded,implementation) (contracts/v2/OnChainLiquidityRouter.sol#203-211)
Reentrancy in OnChainLiquidityRouter.swapAndMint(string,IOnChainLiquidityRouter.SwapMintParams,ISynthereumPoolOnChainPriceFeed,ISynthereumPoolOnChainPriceFeed
.MintParams) (contracts/v2/OnChainLiquidityRouter.sol#124-166):
        External calls:
        - result = implementation.functionDelegateCall(abi.encodeWithSignature(functionSig,dexImplementationInfo[implementation],inputParams,synthereumParams)
) (contracts/v2/OnChainLiquidityRouter.sol#145-153)
        Event emitted after the call(s):
        - Swap(returnValues.inputToken,returnValues.outputToken,returnValues.collateralToken,returnValues.inputAmount,returnValues.outputAmount,returnValues.c
ollateralAmountRefunded,implementation) (contracts/v2/OnChainLiquidityRouter.sol#157-165)
Low level call in OCLRKyber.swapToCollateralAndMint(bytes,IOnChainLiquidityRouter.SwapMintParams,IOnChainLiquidityRouter.SynthereumMintParams) (contracts/v2/i
mplementations/OCLRKyber.sol#27-191):
        - (success) = msg.sender.call{value: inputParams.minOutOrMaxIn - amountsOut[0]}() (contracts/v2/implementations/OCLRKyber.sol#131-134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Pragma version^0.8.4 (contracts/v2/OCLRBase.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/implementations/OCLRUniswapV3.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/interfaces/IOCLRBase.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/interfaces/IOnChainLiquidityRouter.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0
.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in OCLRUniswapV3.swapToCollateralAndMint(bytes,IOnChainLiquidityRouter.SwapMintParams,IOnChainLiquidityRouter.SynthereumMintParams) (contracts/
v2/implementations/OCLRUniswapV3.sol#37-194):
        - (success) = msg.sender.call{value: address(this).balance}() (contracts/v2/implementations/OCLRUniswapV3.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Pragma version^0.8.4 (contracts/v2/OCLRBase.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/implementations/OCLRUniswapV2.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/interfaces/IOCLRBase.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (contracts/v2/interfaces/IOnChainLiquidityRouter.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0
.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in OCLRUniswapV2.swapToCollateralAndMint(bytes,IOnChainLiquidityRouter.SwapMintParams,IOnChainLiquidityRouter.SynthereumMintParams) (contracts/
v2/implementations/OCLRUniswapV2.sol#29-173):
        - (success) = msg.sender.call{value: inputParams.minOutOrMaxIn - amountsOut[0]}() (contracts/v2/implementations/OCLRUniswapV2.sol#120-123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

According to the test results, findings found by these tools were considered as false positives. All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

OnChainLiquidityRouter.sol, IOnChainLiquidityRouter.sol, IOCLRBase.sol

```
Report for contracts/v2/interfaces/IOnChainLiquidityRouter.sol
https://dashboard.mythx.io/#/console/analyses/563ecc30-32d3-43af-969d-ddb8d77a6037
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
  [####################################]  100%
Report for contracts/v2/interfaces/IOCLRBase.sol
https://dashboard.mythx.io/#/console/analyses/bfd39625-0808-4def-89cd-bae67987aa0d
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
  [####################################]  100%
Report for contracts/v2/OnChainLiquidityRouter.sol
https://dashboard.mythx.io/#/console/analyses/2780863a-ba12-452c-883c-742cdbf31d46
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
39	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

OCLRKyber.sol, OCLRUniswapV3.sol, OCLRUniswapV2.sol, IKyberRouter.sol

```
Report for contracts/v2/OCLRBase.sol
https://dashboard.mythx.io/#/console/analyses/33546253-0a93-4de3-bf3c-6b7b1be65bfe
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
  [####################################]  100%
Report for contracts/v2/implementations/OCLRKyber.sol
https://dashboard.mythx.io/#/console/analyses/4e0e1073-cb30-41cd-a391-3d618eca41e0
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
  [####################################]  100%
Report for contracts/v2/implementations/OCLRUniswapV3.sol
https://dashboard.mythx.io/#/console/analyses/febd5f64-8a7c-43a3-ac98-b3c6b99689da
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
  [####################################]  100%
Report for contracts/v2/implementations/OCLRUniswapV2.sol
https://dashboard.mythx.io/#/console/analyses/b7645079-0698-4e24-b1c1-44c9c53be32a
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

All relevant valid findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN