// HALBORN

Reserve - Protocol

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 28th, 2022 - October 10th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/09/2022	Luis Buendia
0.2	Document Edit	09/29/2022	Luis Buendia
0.3	Document Edit	10/09/2022	Luis Buendia
0.4	Document Review	10/09/2022	Gabi Urrutia
1.0	Remediation Plan	11/15/2022	Luis Buendia
1.1	Remediation Plan Review	11/15/2022	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
Luis Buendía	Halborn	luis.buendia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Reserve engaged Halborn to conduct a security audit on their smart contracts beginning on August 28th, 2022 and ending on October 10th, 2022. The security audit was scoped to the smart contracts provided to the Halborn team.

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 is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

The audit did not reveal any critical flaws in the protocol. The Reserve team put significant effort into developing test cases and different scenarios to prevent unexpected code behavior. The issues identified correspond to the parameter sanitation in privileged functions and gas optimizations.

In summary, Halborn identified some security risks that were mostly addressed by the Reserve team.

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 contracts' 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.
- 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. (Hardhat).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Ganache).

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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were 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:

The security assessment was scoped to the following smart contracts on the prepared branch for the audit:

Commit ID: 9242d68bf070790272df2cd181836aaf044aa5dc.

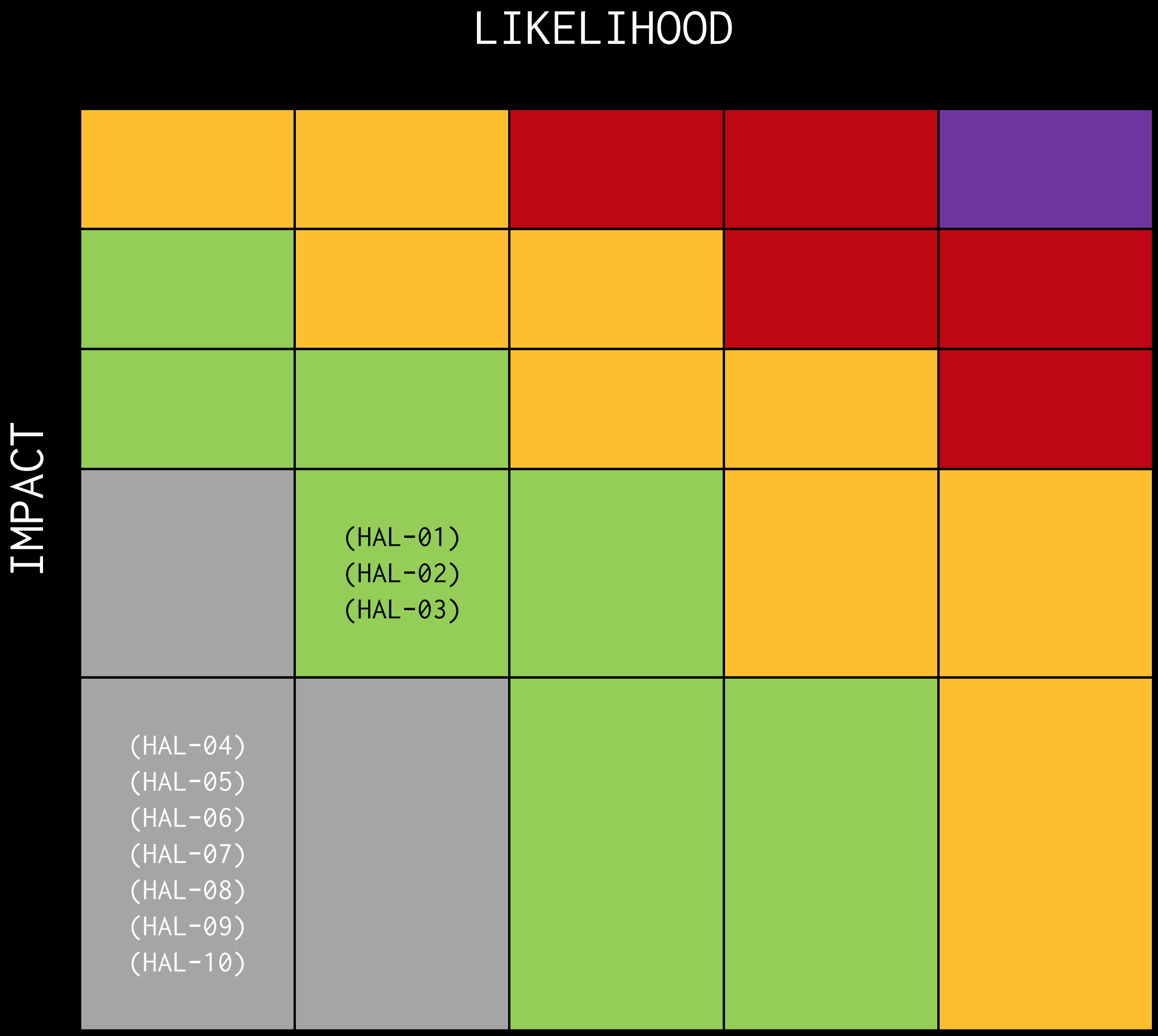
However, the development continued during the audit. The audit team has reviewed the latest version of the master branch during the last week of the audit, focusing on the added or modified code sections.

FIX Commit TREE/ID :

Main Branch

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	3	7



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - DEPLOYER CONTRACT DOES NOT SANITIZE OWNER ADDRESS	Low	SOLVED - 10/27/2022
HAL02 - PRIMER BASKET PARAMETER VALIDATION	Low	SOLVED - 10/08/2022
HAL03 - DESTINATION ADDRESSES ARE NOT MODIFIABLE	Low	SOLVED - 10/27/2022
HAL04 - USE CALLDATA INSTEAD OF MEMORY	Informational	SOLVED - 10/27/2022
HAL05 - CACHE ADDRESS TO SAVE GAS	Informational	SOLVED - 10/08/2022
HAL06 - USING RETURNED ADDRESS CAN SAVE GAS	Informational	SOLVED - 10/08/2022
HAL07 - REMOVE UNNECESSARY REQUIRE STATEMENT	Informational	SOLVED - 10/08/2022
HAL08 - THE CONTRACT SHOULD approve(0) FIRST	Informational	SOLVED - 10/08/2022
HAL09 - INCOMPLETE NATSPEC DOCUMENTATION	Informational	PARTIALLY SOLVED
HAL-10 - CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS	Informational	SOLVED - 10/27/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) DEPLOYER CONTRACT DOES NOT SANITIZE OWNER ADDRESS - LOW

Description:

The Deployer.sol contract does not check the owner address received as input parameter. If the address is set to zero or to the address of the deployer contract, the created contracts will no longer have a useful owner. This is not the expected behavior of the system, since in theory the FaceWrite.sol contract should call this function with the value of its address.

Code Location:

Location

Listing 1: Deployer.sol

```
    function deploy(
        string memory name,
        string memory symbol,
        string calldata mandate,
        address owner,
        DeploymentParams memory params
    ) external returns (address) {
        // Main - Proxy
        MainP1 main = MainP1(
            address(new ERC1967Proxy(address(implementations.main)
↳ , new bytes(0)))
        );
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Check that the owner address is non-zero and the address of the Deployer contract. Or create a require to ensure that the FacadeWrite.sol is the contract that calls this function.

Remediation Plan:

SOLVED: The Reserve team solved the above issue by creating the owner address sanitization. 83722ccb8cb4f659e9b9837919af5248b91765fe.

3.2 (HAL-02) PRIMER BASKET PARAMETER VALIDATION - LOW

Description:

The setPrimeBasket function of the BasketHandler.sol contract does not properly validate received input parameters. This function receives an array of erc20 tokens and a list of reference price amounts for the received addresses. However, the following three scenarios must be considered.

- The prime basket can be set to empty values
- The prime basket can have a duplicated erc20 in the basket. However, only the last set price takes the value
- The prime basket function accepts 0 as the rfAmt value

Under these conditions, especially the first and also second if the same erc20 is provided and the last price is zero, the RToken can be issued without depositing collateral. Although this can be a feature, in unexpected situations it can lead to undesired results.

Code Location:

Location

Listing 2: BasketHandler.sol

```
    function setPrimeBasket(IERC20[] calldata erc20s, uint192[]
 ↳ calldata targetAmts)
        external
        governance
    {
        require(erc20s.length == targetAmts.length, "must be same
 ↳ length");
        delete config.erc20s;
        IAssetRegistry reg = main.assetRegistry();
        bytes32[] memory names = new bytes32[](erc20s.length);
        IERC20 rToken = IERC20(address(main.rToken()));
        IERC20 rsr = main.rsr();
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is advised to sanitize the input parameters for:
- Avoid having the empty basket.
- Avoid having duplicated erc20
- Avoid accepting 0 as the reference amount value.

Remediation Plan:

SOLVED: The Reserve team solved the above issue by creating a function that checks for double items in the array, forcing the reference amount to be greater than zero and checking for empty values. 71617a5249a2137d1f39d05c8c991bef6b2bb84e.

3.3 (HAL-03) DESTINATION ADDRESSES ARE NOT MODIFIABLE - LOW

Description:

The Distributor.sol contract does not allow changing the state variable destination list. It only allows to add destinations. So, if an incorrect address is introduced, it cannot be removed.

Code Location:

Location

Listing 3: Distributor.sol

```
contract DistributorP1 is ComponentP1, IDistributor {
    using SafeERC20Upgradeable for IERC20Upgradeable;
    using FixLib for uint192;
    using EnumerableSet for EnumerableSet.AddressSet;

    EnumerableSet.AddressSet internal destinations;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Implement a mechanism to allow governance to modify destination addresses. Moreover, prevent this address from taking the zero address value.

Remediation Plan:

SOLVED: The Reserve team fixed the above issue by removing addresses if they have zero allocations. The fix was merged into the master branch.

3.4 (HAL-04) USE CALLDATA INSTEAD OF MEMORY - INFORMATIONAL

Description:

When a function with a memory array is called externally, the abi.decode() step has to use a for-loop to copy each index of the calldata to the memory index. Each iteration of this for-loop costs at least 60 gas (i.e. 60 * <mem_array>.length). Using calldata directly, obviates the need for such a loop in the contract code and runtime execution.
If the array is passed to an internal function which passes the array to another internal function where the array is modified and therefore memory is used in the external call, it's still more gas-efficient to use calldata when the external function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.

The current argument that calldata attribute can use is the required parameter in the Broker contract in the openTrade function.

Code Location:

Location

Listing 4: Broker.sol

```
    function openTrade(TradeRequest memory req) external
↳ notPausedOrFrozen returns (ITrade) {
        require(!disabled, "broker disabled");

        address caller = _msgSender();
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using **calldata** instead of **memory**.

Remediation Plan:

SOLVED: The Reserve team solved the above issue by refactoring the code and implementing the suggested changes on the master branch.

3.5 (HAL-05) CACHE ADDRESS TO SAVE GAS - INFORMATIONAL

Description:

The baskethandler contract is called multiples times in the same function. In order to get the address for the first time, the code must call the main contract to get the correct address. However, it is not possible to change the address during execution. Therefore, to reduce the gas cost, it is advised to cache the baskethandler address obtained from the main contract.

The identified places where caching can optimize the cost of gas are:
- In Rtoken.sol in the vest function
- In Rtoken.sol in the redeem function
- In BackingManager.sol in the manageTokens function

Code Location:

Location

Listing 5: RToken.sol

```
    function vest(address account, uint256 endId) external
 ↳ notPausedOrFrozen {
        // == Keepers ==
        main.assetRegistry().refresh();

        // == Checks ==
        CollateralStatus status = main.basketHandler().status();
        require(status == CollateralStatus.SOUND, "basket unsound"
 ↳ );

        // Refund old issuances if there are any
        IssueQueue storage queue = issueQueues[account];
        (uint256 basketNonce, ) = main.basketHandler().lastSet();

        // == Interactions ==
        // ensure that the queue models issuances against the
```

```
↳ current basket, not previous baskets
        if (queue.basketNonce != basketNonce) {
            refundSpan(account, queue.left, queue.right);
        } else {
            vestUpTo(account, endId);
        }
    }
```

Location

Listing 6: RToken.sol

```
    function redeem(uint256 amount) external notFrozen {
        require(amount > 0, "Cannot redeem zero");

        // == Refresh ==
        main.assetRegistry().refresh();

        // == Checks and Effects ==
        address redeemer = _msgSender();
        require(balanceOf(redeemer) >= amount, "not enough RToken"
↳ );
        // Allow redemption during IFFY + UNPRICED
        require(main.basketHandler().status() != CollateralStatus.
↳ DISABLED, "collateral default");
```

Location

Listing 7: BackingManager.sol

```
    function manageTokens(IERC20[] calldata erc20s) external
↳ notPausedOrFrozen {
        // == Refresh ==
        main.assetRegistry().refresh();

        if (tradesOpen > 0) return;
        // Only trade when all the collateral assets in the basket
↳  are SOUND
        require(main.basketHandler().status() == CollateralStatus.
↳ SOUND, "basket not sound");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use the same approach already implemented in other parts of the source code. Cache the baskethandler address of the main contract once and reuse it.

Remediation Plan:

SOLVED: The Reserve team fixed the above issue storing the addresses in the constructor. 71617a5249a2137d1f39d05c8c991bef6b2bb84e.

3.6 (HAL-06) USING RETURNED ADDRESS CAN SAVE GAS - INFORMATIONAL

Description:

The Deployer.sol contract deploys a proxy contract for all given implementations. Although the contract addresses are stored in a local variable in the function, to use the init functions of each, the main contract is called to retrieve the address of each deployed proxy.

Code Location:

Location

Listing 8: Deployer.sol

```
main.backingManager().init(
main,
params.tradingDelay,
params.backingBuffer,
params.maxTradeSlippage
);
// Init Basket Handler
main.basketHandler().init(main);
// Init Revenue Traders
main.rsrTrader().init(main, rsr, params.maxTradeSlippage);
main.rTokenTrader().init(main, IERC20(address(rToken)), params.
↳ maxTradeSlippage
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider changing the main contract call to the return address of the implementation variable in the Deployer contract.

Listing 9

```
// Init Backing Manager
components.backingManager.init(
  main,
  params.tradingDelay,
  params.backingBuffer,
  params.maxTradeSlippage
);
// Init Basket Handler
components.basketHandler.init(main);
// Init Revenue Traders
components.rsrTrader.init(main, rsr, params.maxTradeSlippage);
components.rTokenTrader.init(main, IERC20(address(rToken)),
↳ params.maxTradeSlippage);
```

Remediation Plan:

SOLVED: The Reserve team fixed the above issue following the recommendation. 71617a5249a2137d1f39d05c8c991bef6b2bb84e.

3.7 (HAL-07) REMOVE UNNECESSARY REQUIRE STATEMENT - INFORMATIONAL

Description:

There is a duplicate required statement in the Main contract. The contract checks that main is paused or frozen twice in the same function. However, it does not seem possible for it to change state from the beginning to the middle of the function.

Code Location:

Location

Listing 10: Main.sol

```
    function poke() external {
        require(!pausedOrFrozen(), "paused or frozen");
        // == Refresher ==
        assetRegistry.refresh();

        // == CE block ==
        require(!pausedOrFrozen(), "paused or frozen");
        furnace.melt();
        stRSR.payoutRewards();
    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing the second require statement.

Remediation Plan:

SOLVED: The Reserve team fixed the above issue by removing the require statement. 71617a5249a2137d1f39d05c8c991bef6b2bb84e.

3.8 (HAL-08) THE CONTRACT SHOULD approve(0) FIRST - INFORMATIONAL

Description:

Some tokens (such as USDT L199) do not work when the allowance is changed from an existing non-zero allowance value.
They must first be approved by zero, and then the actual allowance must be approved.

Code Location:

Location

Listing 11: BackingManager.sol

```
    function grantRTokenAllowance(IERC20 erc20) external
↳ notPausedOrFrozen {
        require(main.assetRegistry().isRegistered(erc20), "erc20
↳ unregistered");
        // == Interaction ==
        erc20.approve(address(main.rToken()), type(uint256).max);
    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Approve with a zero amount first before setting the actual amount.

Remediation Plan:

SOLVED: The Reserve team fixed the above issue by using the increase allowance function. 71617a5249a2137d1f39d05c8c991bef6b2bb84e.

3.9 (HAL-09) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Natspec documentation are useful for internal developers who need to work on the project, external developers who need to integrate with the project, auditors who need to review it but also end users since Snowtrace has officially integrated support for it directly on their site.

Although there are many functions that use the **Natspec** documentation, there is still work to be done to fully comply with the recommended style.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding the missing **Natspec** documentation.

Remediation Plan:

PARTIALLY SOLVED: The Reserve team continued, including detailed and exhaustive documentation. This is partially solved since it is an ongoing work as the project grows.

3.10 (HAL-10) CACHE ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL

Description:

Reading array length at each iteration of the loop takes 6 gas (3 for mload and 3 to place memory-offset) in the stack. Caching the array length in the stack saves around 3 gas per iteration.

Code Location:

Listing 12: BasketHandler.sol

```
        for (uint256 i = 0; i < config.erc20s.length; ++i) {
            IERC20 erc20 = config.erc20s[i];

            // Find collateral's targetName index
            uint256 targetIndex;
            for (targetIndex = 0; targetIndex < targetNames.length
 ↳ (); ++targetIndex) {
                if (targetNames.at(targetIndex) == config.
 ↳ targetNames[erc20]) break;
            }
            assert(targetIndex < targetNames.length());
```

Listing 13: BasketHandler.sol

```
            for (uint256 j = 0; j < backup.erc20s.length && size <
 ↳  backup.max; ++j) {
                if (goodCollateral(targetNames.at(i), backup.
 ↳ erc20s[j])) size++;
            }
```

Listing 14: BasketHandler.sol

```
            for (uint256 j = 0; j < backup.erc20s.length &&
 ↳ assigned < size; ++j) {
```

```
                    IERC20 erc20 = backup.erc20s[j];
```

Listing 15: BasketHandler.sol

```
        for (uint256 i = 0; i < basket.erc20s.length; ++i) {
            refAmts[i] = basket.refAmts[basket.erc20s[i]];
        }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

An array's length should be cached to save gas in for-loops.

Remediation Plan:

SOLVED: The Reserve team fixed the above issue by implementing the suggested optimizations 31e99f001111e534461c3203f525ddb0ee32efa0.

THANK YOU FOR CHOOSING

HALBORN