// HALBORN

DRAFT

Orient Protocol

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 30th, 2023 - April 21th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/30/2023	Gokberk Gulgun
0.2	Document Edits	03/10/2023	Luis Bendia
0.3	Draft Review	03/13/2023	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Orient Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on March 30th, 2023 and ending on May 21th, 2023 .
The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a two 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.

Currently, the Multiplyr platform is still under development. Although the document address some findings, as established with the Alpine team, other audits must be performed once the project is finished.

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

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing

techniques help enhance coverage of smart contracts 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.
- Dynamic Analysis (ganache-cli, brownie, hardhat).

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.

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

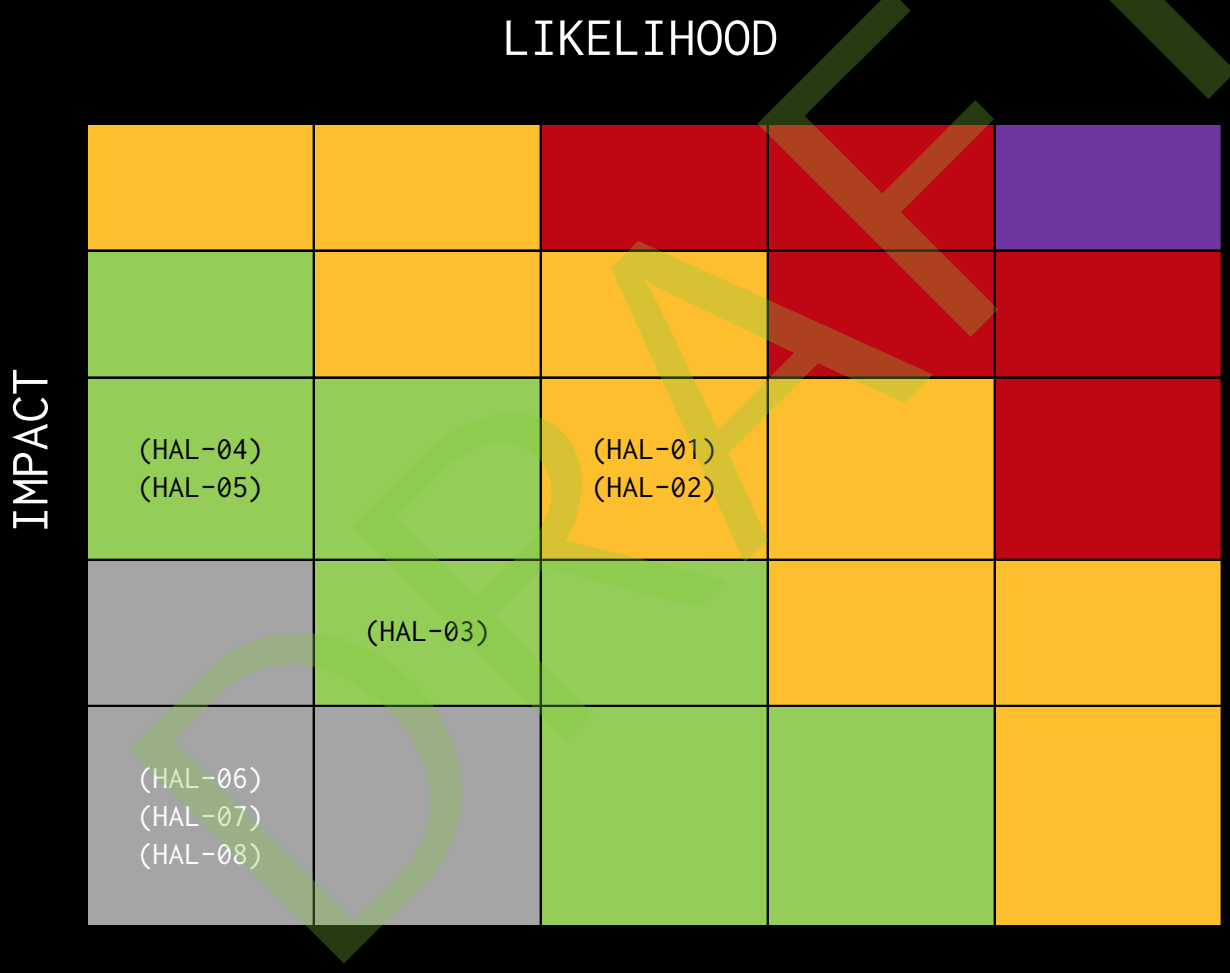
1.4 SCOPE

1. Orient Protocol Token Sale Contracts
    (a) Repository: Token Sale
    (b) Commit ID: 726dcbaef18670d344fa5621c23c4db0e403583a
2. Out-of-Scope
    (a) test/*.sol

Out-of-scope: External contract, libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	3	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING SLIPPAGE/MIN-RETURN CHECK IN THE TwoAssetBasket CONTRACT	Medium	-
(HAL-02) WORMHOLE MESSAGES ARE MISSING CRITICAL CHECKS	Medium	-
(HAL-03) USE SafeErc20.Safeapprove	Low	-
(HAL-04) THE CONTRACT SHOULD safeApprove(0) FIRST	Low	-
(HAL-05) LACK OF PAUSE/UNPAUSE FUNCTIONALITY	Low	-
(HAL-06) INCOMPATIBILITY WITH REBASING/DEFLATIONARY/INFLATIONARY TOKENS	Informational	-
(HAL-07) COMMENTED TODO’s IDENTIFIED	Informational	-
(HAL-08) GAS OPTIMIZATIONS	Informational	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING SLIPPAGE/MIN-RETURN CHECK IN THE TwoAssetBasket CONTRACT - MEDIUM

Description:

Trades can happen at a bad price and lead to receiving fewer tokens than at a fair market price. The attacker's profit is the protocol's loss.

The contract is missing slippage checks, which can lead to being vulnerable to sandwich attacks.

Code Location:

Listing 1: TwoAssetBasket.sol

```
173     uint256[] memory btcAmounts = uniRouter.
 ↳ swapExactTokensForTokens(
174         _tokensFromDollars(token1, amountInputFromBtc),
175         0,
176         pathBtc,
177         address(this),
178         block.timestamp
179     );
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Add minimum return amount checks. Accept a function parameter that can be chosen by the transaction sender, then check that the actually received amount is above this parameter. Alternatively, check if it's feasible

to send these transactions directly to a miner such that they are not visible in the public mempool.

3.2 (HAL-02) WORMHOLE MESSAGES ARE MISSING CRITICAL CHECKS - MEDIUM

Description:

During the code review, It has been observed wormhole messages are missing several important checks.

Code Location:

Listing 2: L2Vault.sol

```
232     function receiveTVL(bytes calldata message) external {
233             (IWormhole.VM memory vm, bool valid, string memory
 ↳ reason) = wormhole.parseAndVerifyVM(message);
234             require(valid, reason);
235
236             // TODO: check chain ID, emitter address
237             // Get tvl from payload
238             (uint256 tvl, bool received) = abi.decode(vm.payload,
 ↳ (uint256, bool));
```

Listing 3: L1Vault.sol

```
77    function receiveMessage(bytes calldata message) external {
78          (IWormhole.VM memory vm, bool valid, string memory reason)
 ↳  = wormhole.parseAndVerifyVM(message);
79          require(valid, reason);
80
81          // TODO: check chain ID, emitter address
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Ensure that all necessary checks are placed in the vaults, as in the example below. This can prevent possible misbehaves using the wormhole infrastructure.

Listing 4

```
    require(
        incomingTokenTransferInfoVM.emitterChainId ==
↳ ALPINE_CHAIN_ID,
        "message does not come from l2/l1 vaults"
    );
    require(
        incomingTokenTransferInfoVM.emitterAddress ==
            ALPINE_ADDRESS,
        "message does not come from vaults"
    );

    require(
        !completedTokenTransfers[incomingTokenTransferInfoVM.hash
↳ ],
        "transfer info already processed"
    );
```

3.3 (HAL-03) USE SafeErc20.Safeapprove - LOW

Description:

The approve() function will fail for certain token implementations that do not return a boolean value. Hence, it is recommended to use safeApprove().

Code Location:

Listing 5: L1Vault.sol (Lines 18,37)

```
91     function _transferFundsToL2(uint256 amount) internal {
92         token.approve(predicate, amount);
93         chainManager.depositFor(address(staging), address(token),
↳ abi.encodePacked(amount));
94
95         // Let L2 know how much money we sent
96         uint64 sequence = wormhole.nextSequence(address(this));
97         bytes memory payload = abi.encodePacked(amount);
98         wormhole.publishMessage(uint32(sequence), payload, 4);
99     }
```

Recommendation:

Update to _token.safeApprove(spender, type(uint256).max) in the function.

3.4 (HAL-04) THE CONTRACT SHOULD safeApprove(0) FIRST - LOW

Description:

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

Code Location:

Listing 6: L1Vault.sol

```
91      function _transferFundsToL2(uint256 amount) internal {
92          token.approve(predicate, amount);
93          chainManager.depositFor(address(staging), address(token),
↳ abi.encodePacked(amount));
94
95          // Let L2 know how much money we sent
96          uint64 sequence = wormhole.nextSequence(address(this));
97          bytes memory payload = abi.encodePacked(amount);
98          wormhole.publishMessage(uint32(sequence), payload, 4);
99      }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

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

Listing 7

```
IERC20(token).safeApprove(address(operator), 0);
IERC20(token).safeApprove(address(operator), amount);
```

3.5 (HAL-05) LACK OF PAUSE/UNPAUSE FUNCTIONALITY - LOW

Description:

L2Vault is already inherited from the PausableUpgradeable. However, the pause/unpause functionality has not been used.
In case a hack occurs, or an exploit is discovered, the team should be able to pause functionality until the necessary changes are made to the system. The deposits should be paused with Pause modifier.

Code Location:

Listing 8: L2Vault.sol

```
131     function deposit(uint256 amountToken) external {
132         _deposit(msg.sender, amountToken);
133      }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Pause functionality on the contract can help to secure the funds quickly.

3.6 (HAL-06) INCOMPATIBILITY WITH REBASING/DEFLATIONARY/INFLATIONARY TOKENS - INFORMATIONAL

Description:

The Orient Protocol does not appear to support rebasing/deflationary/inflationary tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred to contracts before and after the actual transfer to infer any fees/interest.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The following measures can help to mitigate the issue:

- Make sure the check balance/after balance is equal to the amount of any rebasing/inflation/deflation
- Add support in contracts for such tokens before accepting user-supplied tokens
- Consider supporting deflationary/rebasing /etc. tokens by extra check of balances before/after or strictly inform your users not to use such tokens if they do not want to lose them

3.7 (HAL-07) COMMENTED TODO's IDENTIFIED - INFORMATIONAL

Description:

Multiplyr project is currently under development. In the source code, there are many comments marking TODO. Although it is true that evidently this has been already noticed by the Alpine Team, as agreed they are marked down on a list for easy fix and focus on the coming audits.

The audit team identified the TODO comments on the next files:
- TwoAssetBasket.sol: L26, L209, L447
- BridgeScrow.sol: L56, L75
- BaseVault.sol: L32, L213, L214
- L2Vault.sol: L172, L236, L254, L288, L307
- L2AAVEStrategy.sol: L106, L187
- L1Vault.sol: L66, L79
- L1CompoundStrategy: L89, L110
- L1AnchorStrategy: L78

Please note that as this audit has been performed during several commits, some lines may change across different versions.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Implement the code left having security in mind to make the platform functional, robust and resilient.

3.8 (HAL-08) GAS OPTIMIZATIONS - INFORMATIONAL

Description:

Gas optimizations and additional safety checks are available for free when using newer compiler versions and the optimizer.

In the loop below, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`.
- Forwarder.sol: L12
- BaseVault.sol: L197, L217, L274, L341

Caching variables whenever inside a loop helps to save gas, as the most gas expensive access are the ones to storage.

- Forwarder.sol: L12 (requests.length variable)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an `uint` variable inside a loop. This is not applicable outside of loops.

It is recommended to create a memory variable that caches the storage variable to avoid accessing the storage more times than required, since it is more expensive than memory accesses.

Note that these two mechanisms can be applied in different context in the source code.

THANK YOU FOR CHOOSING

// HALBORN