Seascape - NFT SWAP

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 11th, 2022 - April 13th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/14/2022	Alessandro Cara
0.2	Document Amended	04/14/2022	Alessandro Cara
1.0	Final Document Review	04/18/2022	Constantin Casmir

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
Alessandro Cara	Halborn	Alessandro.Cara@halborn.com
Constantin Casmir	Halborn	Constantin.Casmir@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Seascape engaged Halborn to conduct a security audit on their smart contracts beginning on April 11th, 2022 and ending on April 13th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

It should be noted that the assessment was timeboxed and only three days were provided to assess the security of the smart contracts before public release.

The smart contracts within the scope provided the functionality of swapping Seascape's NFTs via their web platform. Users are able to initiate a swap request by offering up to five NFTs plus an optional bounty, and request up to five NFTs. Users of the platform with the requested NFTs can accept the swap by transferring their NFTs in exchange for the offered ones plus the optional bounty.

1.2 AUDIT SUMMARY

The team at Halborn was provided three days 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

In summary, Halborn identified some security risks that were mostly addressed by the Seascape 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.
- 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.

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

The assessment was scoped to the public GitHub repository on the nft-marketplace branch. The smart contracts in scope were inside the contracts/nft_swap directory.

The audited commit was 14a4eb1969eefa582a63dc3805186be08a38972f and fixes were assessed on newer commits.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	3

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥	🟥
🟩	🟩	🟨	🟨	🟥
⬜	🟩 (HAL-01)	🟩	🟨	🟨
⬜ (HAL-02) (HAL-03) (HAL-04)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING REENTRANCY GUARD	Low	SOLVED - 04/12/2022
(HAL-02) NFT NUMBER 0 CAN NEVER BE TRADED	Informational	SOLVED - 04/12/2022
(HAL-03) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 04/12/2022
(HAL-04) UPGRADE TO AT LEAST PRAGMA 0.8.10	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING REENTRANCY GUARD - LOW

Description:

During the audit, it was discovered that the cancelOffer function did not implement protections against reentrancy attacks. To protect against reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that protects the function against re-entrancy attacks.

Halborn attempted to exploit a possible reentrancy vulnerability in the cancelOffer function. However, it was not possible to successfully exploit the issue due to the NFT transfer not being successful after deletion of the offer index on the second iteration of the re-entered function.

Code Location:

The affected function was cancelOffer on line 396.

Recommendation:

It is recommended to add the nonReentrant modifier to avoid introducing future reentrancy vulnerabilities.

Remediation Plan:

SOLVED: The Seascape team added the nonReentrant modifier.

3.2 (HAL-02) NFT NUMBER 0 CAN NEVER BE TRADED - INFORMATIONAL

Description:

While currently supported NFTs start with tokenId equal to one, should additional NFT series be supported, with the first tokenId being zero, they would not be able to be traded.

Code Location:

Location: NftSwap.sol - function createOffer

Listing 1

```
require(_offeredTokens[index].tokenId > 0, "nft id must be greater
↳  than 0");
```

Location: NftSwap.sol - function acceptOffer

Listing 2

```
 require(_requestedTokenIds[i] > 0, "nft id must be greater than 0
↳ ");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove this check to allow different NFT sets being swapped.

Remediation Plan:

SOLVED: The Seascape team removed the require statements.

3.3 (HAL-03) UPGRADE TO AT LEAST PRAGMA 0.8.10 - INFORMATIONAL

Description:

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

- Safemath by default from 0.8.0 (can be more gas efficient than the SafeMath library)
- Low level inliner: from 0.8.2, leads to cheaper runtime gas. This is especially relevant when the contract has small functions. For example, OpenZeppelin libraries typically have a lot of small helper functions and if they are not inlined, they cost an additional 20 to 40 gas because of 2 extra jump instructions and additional stack operations needed for function calls.
- Optimizer improvements in packed structs: Before 0.8.3, storing packed structs, in some cases used an
  additional storage read operation. After EIP-2929, if the slot was already cold, this means unnecessary stack operations and extra deploy time costs. However, if the slot was already warm, this means
  additional cost of 100 gas alongside the same unnecessary stack operations and extra deploy time costs.
- Custom errors from 0.8.4, leads to cheaper deploy time cost and run time cost. Note: the run time cost is only relevant when the revert condition is met. In short, replace revert strings by custom errors.

Code Location:

The contracts within scope made use of the pragma version 0.6.7

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Halborn recommends that the project is upgraded to use at least **pragma** 0.8.10.

Remediation Plan:

ACKNOWLEDGED: The Seascape team decided not to upgrade to the latest pragma versions and instead deploy with the current one. They confirmed that future projects will use the newest versions.

3.4 (HAL-04) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

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++`. This does not only apply to the iterator variable. It also applies to variables declared within the loop code block.

Code Location:

NFTSwap.sol

- Line 236
- Line 246
- Line 260
- Line 287
- Line 290
- Line 333
- Line 355
- Line 360
- Line 402

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 applies to the iterator variable and to variables declared within the loop code block.

Remediation Plan:

SOLVED: The Seascape team now uses ++i to increment variables inside loops, saving some gas.

VERIFICATION OF MATHEMATICAL OPERATION'S SAFETY

Halborn reviewed the safety of the mathematical operations within the smart contract, as Seascape proposed removing the SafeMath library to reduce gas usage.

This review allowed Halborn to identify one integer overflow vulnerability which will not be able to be exploited due to the conditions of the smart contract and the Crowns token total supply.

Within the createOffer function, the following code dealt with calculating the amount of funds that would need to be transferred to the smart contract:

Listing 3

```
if (_bounty > 0) {
    if (address(crowns) == _bountyAddress) {
        require(crowns.balanceOf(msg.sender) >= fee + _bounty,
            "not enough CWS for fee & bounty");
    } else {
        require(supportedBountyAddresses[_bountyAddress],
            "bounty address not supported");

        if (_bountyAddress == address(0x0)) {
            require (msg.value >= _bounty, "insufficient transfer
↳ amount");
            uint256 returnBack = msg.value.sub(_bounty);
            if (returnBack > 0)
                msg.sender.transfer(returnBack);
        } else {
            IERC20 currency = IERC20(_bountyAddress);
            require(currency.balanceOf(msg.sender) >= _bounty,
                "not enough money to pay bounty");
        }
    }
```

Users can specify a bounty value, which is an additional sum of tokens that would be transferred in exchange for offered NFTs, against the requested NFTs. fee on the other hand, is a value decided by Seascape, which will initially be 1 - This represents the fee paid by the user creating an offer which is deposited in the smart contract, and burnt upon offer acceptance.

Should the bounty value be inserted as a large value which would allow overflowing the uint256 data type (2**256 - 1), attackers would be able to bypass the require statement detailed below as long as they hold the required amount of Crowns tokens to comply with the funds check.

Listing 4

```
require(crowns.balanceOf(msg.sender) >= fee + _bounty)
```

It should be noted that overflow is not possible when using different tokens.

After performing the integer overflow attack, the large _bounty value would be stored as it is, whereas the actual transferred token amount would be the result of the overflowing calculation. Should an attacker cancel their offer, they would be returned the value resulted by the overflowed calculation; therefore, this could not result in any fund loss.

On the other hand, should another user accept the offer, or the attacker accept its offer with a different wallet, the contract would be trying to send this large sum of Crowns token. Due to the large number needed to overflow this value, this would require the supply of the Crows token to multiply exponentially.

Listing 5

```
if(obj.bounty > 0) {
    if(obj.bountyAddress == address(0))
        msg.sender.transfer(obj.bounty);
    else
        IERC20(obj.bountyAddress).safeTransfer(msg.sender, obj.
↳ bounty);
}
```

The current max supply of Crowns is 10e17, which is exponentially below the amount required to overflow the value, even if Seascape increases the fixed fee value by a considerably large amount.

The other calculation which could be vulnerable to overflow after removing SafeMath was the following:

Listing 6

```
require (msg.value >= _bounty, "insufficient transfer amount");
uint256 returnBack = msg.value.sub(_bounty);
```

Due to the require statement before the calculation, it would not be possible to provide a value that would cause an integer underflow.

Seascape decided not to use SafeMath in their libraries and keep the Solidity compiler version to 0.6.7 which does not offer built-in mathematics safety.

THANK YOU FOR CHOOSING

HALBORN