HALBORN

Seascape - NFT Multisend

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 10th, 2022 - May 10th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 2
CONTACTS 2
1 EXECUTIVE OVERVIEW 3
1.1 INTRODUCTION 4
1.2 AUDIT SUMMARY 4
1.3 TEST APPROACH & METHODOLOGY 5
RISK METHODOLOGY 5
1.4 SCOPE 7
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 8
3 FINDINGS & TECH DETAILS 9
3.1 (HAL-01) UPGRADE TO AT LEAST PRAGMA 0.8.10 - INFORMATIONAL 11
Description 11
Code Location 11
Risk Level 11
Recommendation 12
Remediation Plan 12
4 AUTOMATED TESTING 12
4.1 STATIC ANALYSIS REPORT 14
Description 14
Slither results 14

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/10/2022	Alessandro Cara
0.2	Document Amended	05/10/2022	Alessandro Cara
1.0	Remediation Plan	05/10/2022	Alessandro Cara
1.1	Remediation Plan	05/10/2022	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Seascape engaged Halborn to conduct a security audit on their NFT multisend smart contract beginning on May 10th, 2022 and ending on May 10th, 2022. The security assessment was scoped to the smart contract provided to the Halborn team.

The contract in scope was a simple contract with only one function which allowed to transfer multiple NFTs in one single call.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week 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

Halborn perform different test cases to validate the functionality of the smart contract. These included trying to transfer NFTs not owned by the caller, as well as providing invalid data. The smart contract was found to function correctly and did not present any security vulnerabilities.

In summary, Halborn identified one security risk that stemmed from the use of an old Solidity pragma version. Halborn recommends that the contract is deployed using at least pragma 0.8.10. the Seascape team applied the fix.

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 this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance coverage of the bridge 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
- Static Analysis of security for scoped contract, and imported functions (Slither)
- Testnet deployment (Brownie, 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. 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.

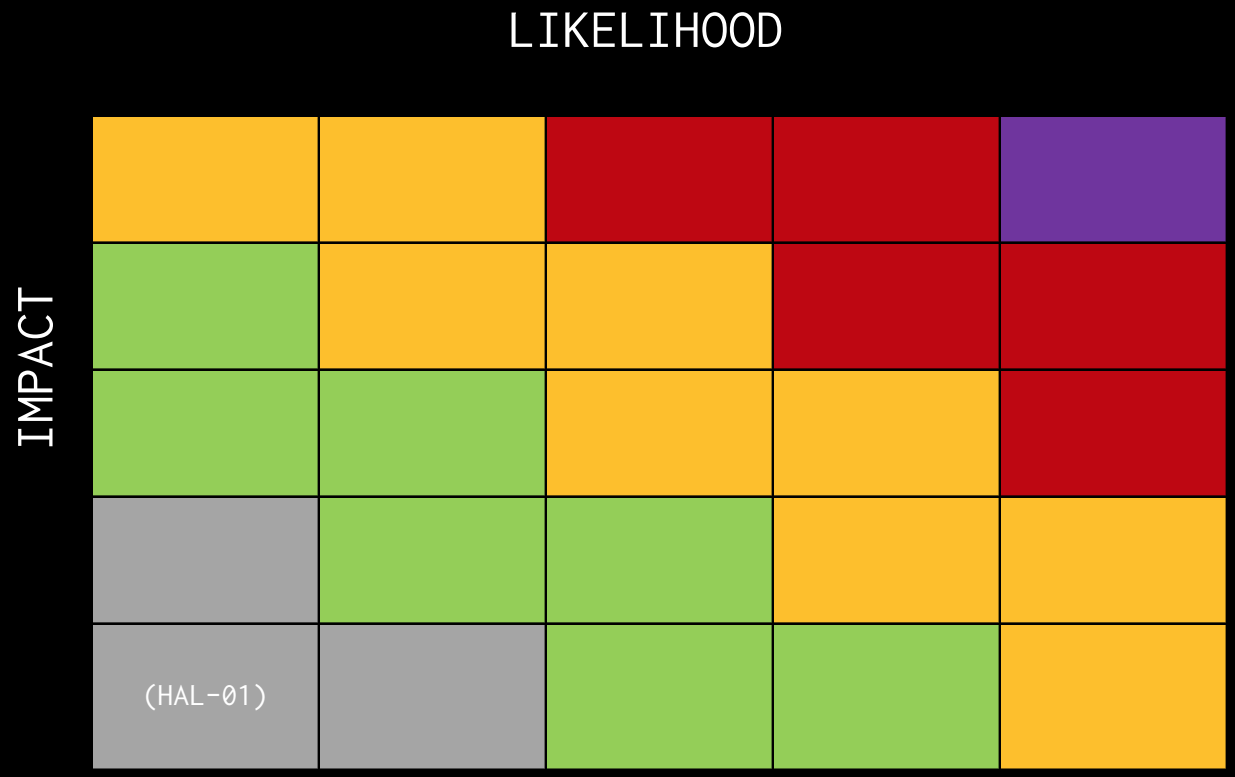
10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The assessment was scoped to the smart contract available in GitHub with commit ID **2305e92b4b3e3531b3649de21a8e12de385fc8d9**.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 UPGRADE TO AT LEAST PRAGMA 0.8.10	Informational	SOLVED - 05/10/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) 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 since 0.8.0 (can be more gas efficient than the SafeMath library)
- Low level inline: as of 0.8.2, leads to cheaper gas runtime. 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 built in, they cost an additional 20 to 40 gas due to the 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 contract 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:

SOLVED: Seascape upgraded the pragma version to 0.8.13.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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, Slither was run on the all-scoped contracts. 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.

Slither results:

Listing 1

```
MultiSend.sendNfts(uint256,address,address[],uint256[]) (contracts/multisend.sol#9-28) has external calls inside a loop: require(bool,string)(IERC721(nftAddresses[i]).ownerOf(nftIds[i]) == msg.sender,sender not owner of nft) (contracts/multisend.sol#21-22)
MultiSend.sendNfts(uint256,address,address[],uint256[]) (contracts/multisend.sol#9-28) has external calls inside a loop: IERC721(nftAddresses[i_scope_0]).safeTransferFrom(msg.sender,receiver,nftIds[i_scope_0]) (contracts/multisend.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Different versions of Solidity is used:
        - Version used: ['0.8.13', '^0.8.0']
        - 0.8.13 (contracts/multisend.sol#1)
        - ^0.8.0 (contracts/openzeppelin-contracts@4.5.0/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (contracts/openzeppelin-contracts@4.5.0/contracts/utils/introspection/IERC165.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.13 (contracts/multisend.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
```

```
13 Pragma version^0.8.0 (contracts/openzeppelin-contracts@4.5.0/
↳ contracts/token/ERC721/IERC721.sol#4) allows old versions
14 Pragma version^0.8.0 (contracts/openzeppelin-contracts@4.5.0/
↳ contracts/utils/introspection/IERC165.sol#4) allows old versions
15 solc-0.8.13 is not recommended for deployment
16 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#incorrect-versions-of-solidity
17 contracts/multisend.sol analyzed (3 contracts with 77 detectors),
↳ 7 result(s) found
```

All the findings are deemed to be false positives.

THANK YOU FOR CHOOSING

// HALBORN