// HALBORN

NewOrderDAO - veNewO

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 20th, 2022 - June 3rd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/27/2022	Alessandro Cara
0.2	Document Amended	06/03/2022	Alessandro Cara
0.3	Document Draft Review	06/06/2022	Gabi Urrutia
1.0	Remediation Plan	06/13/2022	Alessandro Cara
1.1	Remediation Plan Review	06/13/2022	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
Alessandro Cara	Halborn	Alessandro.Cara@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NewOrderDAO engaged Halborn to conduct a security audit on their smart contracts beginning on May 20th, 2022 and ending on June 3rd, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

The project included three main contracts:

- LpVault - A vault where users can stake their NEWO/USDC LP tokens for a reward
- Rewards - A reward contract that rewards users based on their NEWO deposit on VeVault
- VeVault - A vault where users can lock their tokens for 3 months to 3 years and achieve greater rewards based on the time they lock their tokens for

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 contracts. 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

Overall, the code was of good quality and thoroughly documented. Test cases were implemented and covered the majority of normal user flows of the platform. The code prevented overflows by making sure that operations were safe, and the locking system throughout all of the code functioned correctly, preventing users from withdrawing tokens before the lock time ended.

Halborn identified a number of issues which could impact the amount of rewards earned by a user.

In summary, Halborn identified some security risks and misconfigurations that were either solved or accepted by the NewOrderDAO 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 this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance coverage of the 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

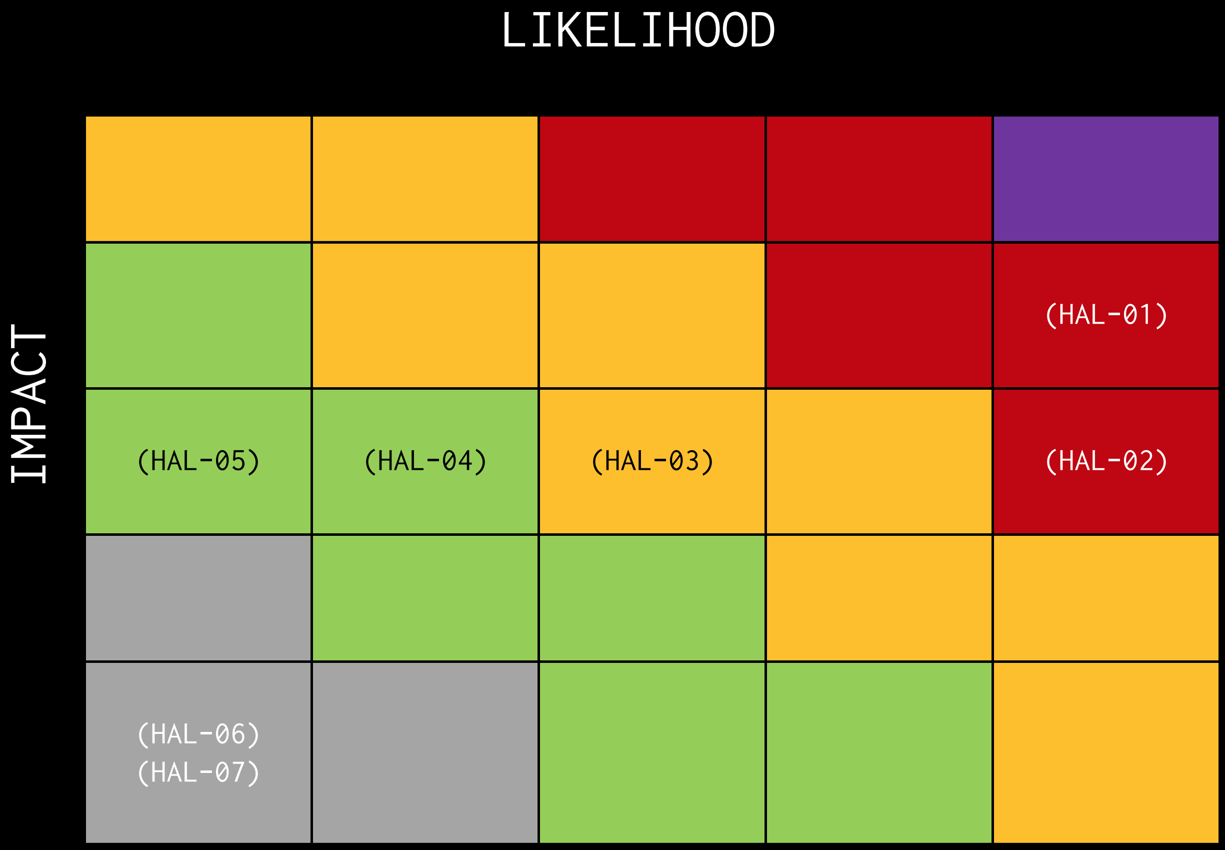
1.4 SCOPE

The assessment was scoped to the repository available in GitHub at commit **c941512baca195c0f54fc2a452c7229951da0e63**.

Remediation were reviewed up to the commit ID **ec22e6ab96f39d21f9104d256fe242cecbf19eb3**

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	1	2	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 WRONG MULTIPLIER CALCULATION ON RELOCK	High	SOLVED - 06/13/2022
HAL-02 LOSS OF PRECISION ON LPVAULT REWARDS	High	RISK ACCEPTED
HAL-03 INCOMPATIBILITY WITH INFLATIONARY/DEFLATIONARY/TRANSFER ON FEE TOKENS	Medium	RISK ACCEPTED
HAL-04 IMPROPER ROLE BASED ACCESS CONTROL	Low	RISK ACCEPTED
HAL-05 MISSING VALIDATION ON SETTERS	Low	SOLVED - 06/13/2022
HAL-06 RELOCKING TOKENS CAUSES PREVIOUS DEPOSIT TO BE LOCKED FOR AT LEAST THE MINIMUM LOCK TIME	Informational	SOLVED - 06/13/2022
HAL-07 MISSING WHITELIST CHECKS FOR ERC721 ASSETS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) WRONG MULTIPLIER CALCULATION ON RE-LOCK (VeVault) - HIGH

Description:

Testing revealed that in the VeVault smart contract, locking more tokens before the previous deposit has been withdrawn, would result in users locking all of their tokens for the new lock time, as well as recalculating the reward multiplier to adhere to the new lock time. While this is beneficial when users re-lock for more time; thus they should be granted a larger multiplier, it is an issue when a user locks tokens for a smaller amount of time.

The following scenario was used to identify the issue:

1. User1 locks 1000 NEWO tokens for one year, achieving a larger multiplier.
2. Just before the one year expires, they decide to lock an additional 100 NEWO tokens for three months.
3. Now their multiplier will be the multiplier for the lock time of three months, while all of their assets will be locked for an additional three months.

Proof of Concept:

The following Brownie script was used to simulate the aforementioned scenario:

Listing 1

```python
print("Depositing 200 NewO for users in VeVault and notifying
↳ Rewards of the deposit")
contract_newotoken.approve(VeVaultContract, web3.Web3.toWei(200_0,
↳  'ether'), {'from':user2})
contract_newotoken.approve(VeVaultContract, web3.Web3.toWei(200_0,
↳  'ether'), {'from': user3})
VeVaultContract.deposit(web3.Web3.toWei(200, 'ether'), user2,
↳ min_lock*3, {'from': user2})
VeVaultContract.deposit(web3.Web3.toWei(200, 'ether'), user3,
↳ min_lock*6, {'from': user3})
RewardsContract.notifyDeposit({'from': user2})
RewardsContract.notifyDeposit({'from': user3})

print("VeMult user2", VeVaultContract.veMult(user2)/1e12)
print("VeMult user3", VeVaultContract.veMult(user3)/1e2)
print("Fast forward for some time")
chain.sleep(min_lock)
chain.mine(1)
VeVaultContract.deposit(web3.Web3.toWei(200, 'ether'), user3,
↳ min_lock, {'from': user3})

print("VeMult user2", VeVaultContract.veMult(user2)/1e2)
print("VeMult user3", VeVaultContract.veMult(user3)/1e2)
```

The output of the above script is shown below:

Listing 2

```
VeMult user2 1.19
VeMult user3 1.55
[sleep]
VeMult user2 1.19
VeMult user3 1.0
```

As shown above, two deposits were made for each user at first. User2 locked his funds for three times the minimum lock period, whereas user3 locked for six times the minimum lock period. As such, user3 had a larger reward multiplier. However, after three months, user3 locked their funds for three more months, resulting in the downgrade of their multiplier.

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Halborn recommends that the multiplier calculations consider the length of the previous deposit in case of an additional deposit, as well as the multiplier that was applied before. This should be weighted against how long they held their tokens locked in the previous deposit.

Remediation Plan:

SOLVED: The NewOrderDAO team modified the contract to prevent users from re-locking their assets if the new unlock time is earlier than the current unlock time. Furthermore, they amended the documentation to reflect that users cannot repeatedly re-lock their funds for the minimum amount of time and keep a larger bonus indefinitely. Thus, when you lock the funds again, the multiplier will be that of the new lock time.

3.2 (HAL-02) LOSS OF PRECISION ON LPREWARDS - HIGH

Description:

Testing revealed that due to the use of different decimal precision throughout the contracts, there was a loss of precision and therefore of rewards for users. In more details, the share of rewards on the LpRewards contract was divided between the LP token stakers. However, upon exiting the pool for all staked user, an amount of reward tokens were left in the contract. Due to this rounding mistake, users would lose parts of their rewards, and funds would be locked into the vault contract.

Proof of Concept:

The following Brownie script shows the loss of precision and thus rewards to the users.

Listing 3

```
contract_newotoken.transfer(LpRewardContract, web3.Web3.toWei(200
↳ _000, 'ether'), {'from': treasury_account})
LpRewardContract.notifyRewardAmount(web3.Web3.toWei(200_000, '
↳ ether'), {'from': governance_token_owner})

contract_newotoken.approve(VeVaultContract, web3.Web3.toWei(200_0,
↳  'ether'), {'from':user2})
contract_newotoken.approve(VeVaultContract, web3.Web3.toWei(200_0,
↳  'ether'), {'from': user3})
VeVaultContract.deposit(web3.Web3.toWei(20, 'ether'), user2,
↳ min_lock*3, {'from': user2})
VeVaultContract.deposit(web3.Web3.toWei(20, 'ether'), user3,
↳ min_lock*3, {'from': user3})
RewardsContract.notifyDeposit({'from': user2})
RewardsContract.notifyDeposit({'from': user3})

usdc.approve(sushiswap_router, usdc.balanceOf(user2), {'from':
↳ user2})
usdc.approve(sushiswap_router, usdc.balanceOf(user3), {'from':
↳ user3})
```

```
contract_newotoken.approve(sushiswap_router, contract_newotoken.balanceOf(user2), {'from': user2})
contract_newotoken.approve(sushiswap_router, contract_newotoken.balanceOf(user3), {'from': user3})

sushiswap_router.addLiquidity(
    contract_newotoken,
    usdc,
    web3.Web3.toWei(100, 'ether'),
    50 * 10 ** 6,
    1,
    1,
    user3,
    999999999999,
    {'from': user3}
)
sushiswap_router.addLiquidity(
    contract_newotoken,
    usdc,
    web3.Web3.toWei(100, 'ether'),
    50 * 10 ** 6,
    1,
    1,
    user2,
    999999999999,
    {'from': user2}
)

contract_lp_sushi.approve(LpRewardContract, contract_lp_sushi.balanceOf(user2), {'from': user2})
LpRewardContract.deposit(200, user2, {'from': user2})
contract_lp_sushi.approve(LpRewardContract, contract_lp_sushi.balanceOf(user3), {'from':user3})
LpRewardContract.deposit(200, user3, {'from':user3})

print(”Fast forwarding for 7 days”)
chain.sleep(86400*7)
chain.mine(1)

print(”Rewards earned after 7 days travel user2”, LpRewardContract.earned(user2)/1e18)
print(”Rewards earned after 7 days travel user3”, LpRewardContract.earned(user3)/1e18)

```

```python
LpRewardContract.exit({'from':user2})
LpRewardContract.exit({'from': user3})

print("NewOBalance of user2", contract_newotoken.balanceOf(user2)
↳ /1e18)
print("NewOBalance of user3", contract_newotoken.balanceOf(user3)
↳ /1e18)
print("NewOBalance of LpRewards contract", contract_newotoken.
↳ balanceOf(LpRewardContract)/1e18)
```

The following code snippet shows the output of the test case:

Listing 4

```
Balance of NewO for user2 199999.99999999997
Balance of NewO for user3 199999.99999999997
[wait 7 days]
NewOBalance of user2 299878.67724867724
NewOBalance of user3 299878.01587301586
NewOBalance of LpRewards contract 3.3068783068788092
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Halborn recommends that the appropriate decimal checks are implemented to ensure that no precision is lost while rewarding users. This is caused by the multiplier calculations on the VeVault contract, which are applied to the LpRewards calculations.

Remediation Plan:

RISK ACCEPTED: The NewOrderDAO team considered this to be a limitation of smart contracts, and the team accepts that minor rounding errors will occur.

3.3 (HAL-03) INCOMPATIBILITY WITH INFLATIONARY/DEFLATIONARY/TRANSFER ON FEE TOKENS - MEDIUM

Description:

The contracts Rewards and VeVault use OpenZeppelin's safeTransferFrom and safeTransfer to handle token transfers. These functions call transferFrom and transfer internally in the token contract to actually execute the transfer. However, the balance is not verified before and after the transfer and the actual amount transferred may not be the same as the amount received in the case of a fee applied in the token contract. In the case of using a token of this kind, users may receive more tokens than what they actually deserve.

Code Location:

This applies to all transfers within the Reward and VeVault contracts.

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Whenever tokens are transferred, the delta of the previous (before transfer) and current (after transfer) token balance should be verified to match the user-declared token amount.

Remediation Plan:

RISK ACCEPTED: The NewOrderDAO team has confirmed that it will not use any of the tokens listed above; therefore this issue does not apply.

3.4 (HAL-04) IMPROPER ROLE BASED ACCESS CONTROL - LOW

Description:

Testing revealed that the smart contracts within scope did not implement a granular access control. All the privileged functionality was assigned to one account, the owner of the smart contract. This could lead to serious consequences should the ownership of this account be lost, or a malicious admin decided to take over the platform.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Halborn recommends that a more granular access control policy is enforced. For instance, the following user roles could be set:

- Pauser - user who can pause the contract
- Owner - admin of the contract which can access the most sensitive functionality
- ChangeWhitelist role - user who can add or remove tokens from the whitelist

Additionally, it is recommended that privileged users make use of a multi-signature wallet.

Remediation Plan:

RISK ACCEPTED: The NewOrderDAO team will be managing the smart contracts via a multi-signature wallet.

3.5 (HAL-05) MISSING VALIDATION ON SETTERS - LOW

Description:

During the manual review of the code, it was observed that the setters function did not properly validate that new values for sensitive parameters are valid. For instance, an admin might be able to set different grace periods, epochs and penalties to invalid values which would affect the result of calculations throughout the smart contracts.

Code Location:

The following extract was taken from the VeVault.sol contract to show how an admin could set all of these values to zero or an invalid value, which would result in unexpected calculations. The same applies to all other contracts in scope.

Listing 5

```
    function changeGracePeriod(uint256 newGracePeriod) external
↳ onlyOwner {
        _penalty.gracePeriod = newGracePeriod;
    }

    /**
     * @notice Owner can change state variabes which controls the
↳ penalty system
     */
    function changeEpoch(uint256 newEpoch) external onlyOwner {
        _lockTimer.epoch = newEpoch;
    }

    /**
     * @notice Owner can change state variabes which controls the
↳ penalty system
     */
    function changeMinPenalty(uint256 newMinPenalty) external
↳ onlyOwner {
        _penalty.minPerc = newMinPenalty;
    }

    /**
     * @notice Owner can change state variabes which controls the
↳ penalty system
     */
    function changeMaxPenalty(uint256 newMaxPenalty) external
↳ onlyOwner {
        _penalty.maxPerc = newMaxPenalty;
    }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Halborn recommends that validation is added to the setter functions, throughout all the smart contracts. At a minimum, NewOrderDAO should ensure that these values cannot be set to zero.

Remediation Plan:

SOLVED: The NewOrderDAO team amended the smart contracts to add checks that would prevent invalid settings values from being accepted.

3.6 (HAL-06) RELOCKING TOKENS CAUSES PREVIOUS DEPOSIT TO BE LOCKED FOR AT LEAST THE MINIMUM LOCK TIME - INFORMATIONAL

Description:

Testing revealed that re-locking tokens on the VeVault contract enforces the new lock time on the previously deposited amount. While this might be a design feature, it is not clear on the code description that this effect would happen.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Halborn recommends that this feature is reviewed, and should this be expected, the description of the protocol should be updated to reflect this.

Remediation Plan:

SOLVED: Updated documentation to properly inform contract users of this feature.

3.7 (HAL-07) MISSING WHITELIST CHECKS FOR ERC721 ASSETS - INFORMATIONAL

Description:

It was observed that while for ERC20 tokens there was a whitelist of allowed tokens for recovery, ERC721 tokens did not enforce any whitelist. It might be possible that untrusted ERC721 tokens are passed as parameters to the recover function, and that the transfer might not happen as expected.

Code Location:

VeVault.sol Lines# 619-622

Listing 6

```
function recoverERC721(address tokenAddress, uint256 tokenId)
↳ external onlyOwner {
    IERC721(tokenAddress).safeTransferFrom(address(this), owner,
↳ tokenId);
    emit RecoveredNFT(tokenAddress, tokenId);
}
```

The same applies to Rewards.sol and LpRewards.sol.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Halborn recommends that a whitelist for allowed ERC721 tokens is enforced to prevent untrusted assets from being handled by the vault contracts.

Remediation Plan:

ACKNOWLEDGED: Contracts are not intended to work with ERC721 tokens, and this function will only be used to retrieve ERC721 assets in case they are mistakenly sent to the contracts.

THANK YOU FOR CHOOSING

HALBORN