NewOrderDAO

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 17th, 2022 - January 24th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/17/2022	Roberto Reigada
0.2	Document Updates	01/21/2022	Roberto Reigada
0.3	Draft Review	01/25/2022	Gabi Urrutia
1.0	Remediation Plan	01/31/2022	Roberto Reigada
1.1	Remediation Plan Review	01/31/2022	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
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NewOrderDAO engaged Halborn to conduct a security audit on their fee collector smart contract beginning on January 17th, 2022 and ending on January 24th, 2022. The security assessment was scoped to the smart contracts provided in the following GitHub repositories:
- new-order-network/GovernanceTokenV2
- new-order-network/disbursement-contracts
- new-order-network/one-way-swap

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to audit 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

In summary, Halborn identified some security risks that were addressed by NewOrderDAO 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 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
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- 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

IN-SCOPE:

The security assessment was scoped to the following smart contracts:

- GovernanceTokenV2.sol
- DisbursementCliff.sol
- one-way-swap.sol

GovernanceTokenV2.sol Commit ID: e9cde694e53005e4504ae44d6462ee07e638a511

GovernanceTokenV2.sol Fixed Commit ID: 9a83d8dd7c02515ffd161b5356db90c0add5e8a0

DisbursementCliff.sol Commit ID: 4bc016a9daf9896c9bd602b132e2df70d8737c24

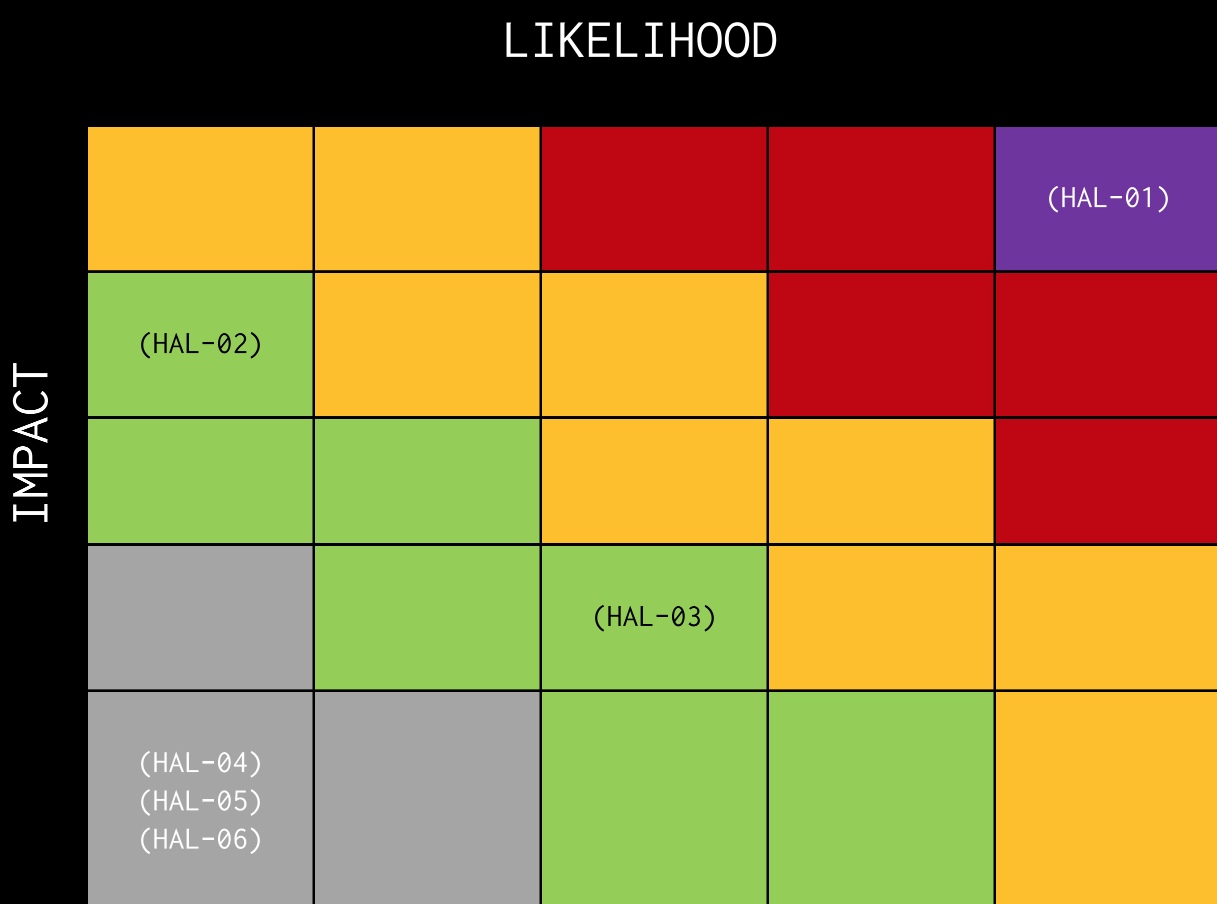
DisbursementCliff.sol Fixed Commit ID: d3c6d4a789dc370793a09a7d0d997c3cbf9fb073

one-way-swap.sol Commit ID: d2d2f724f3ae1652c138423bbe794a8ec3535b18

OneWaySwap.sol Fixed Commit ID: 12b93877647ad63bac85aaa65b2b4243f81392e8

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	2	3



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - OVERFLOW IN CALCMAXTRANSFERRABLE FUNCTION	Critical	SOLVED - 01/28/2022
HAL02 - UNCHECKED TRANSFER	Low	SOLVED - 01/28/2022
HAL03 - MISSING ZERO ADDRESS CHECKS	Low	SOLVED - 01/28/2022
HAL04 - SOLC 0.8.3 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	SOLVED - 01/28/2022
HAL05 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 01/28/2022
HAL06 - TIMELOCKTOKEN IS NOT PAUSABLE	Informational	SOLVED - 01/28/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) OVERFLOW IN CALCMAXTRANSFERRABLE FUNCTION - CRITICAL

Description:

In the contract, TimeLockToken the function calcMaxTransferrable() is used to calculate the maximum amount of transferrable tokens for an address:

Listing 1: GovernanceTokenV2.sol (Lines 131)

```
112 /// @dev Calculates the maximum amount of transferrable tokens for
        address `who`
113 /// @return Number of transferrable tokens
114 function calcMaxTransferrable(address who)
115     public
116     view
117     returns (uint256)
118 {
119     if(timelockedTokens[who] == 0){
120         return balanceOf(who);
121     }
122     uint256 maxTokens;
123     if( vestTime[who] > block.timestamp || cliffTime[who] > block.
           timestamp){
124         maxTokens = 0;
125     } else {
126         maxTokens = timelockedTokens[who] * (block.timestamp -
               vestTime[who]) / disbursementPeriod[who];
127     }
128     if (timelockedTokens[who] < maxTokens){
129       return balanceOf(who);
130     }
131     return balanceOf(who) - timelockedTokens[who] + maxTokens;
132 }
```

This function is called with every transfer because of the _beforeTokenTransfer() hook:

Listing 2: GovernanceTokenV2.sol (Lines 105)

```
100 function _beforeTokenTransfer(
101     address from,
102     address to,
103     uint256 amount
104 ) internal virtual override {
105     uint maxTokens = calcMaxTransferrable(from);
106     if (from != address(0x0) && amount > maxTokens){
107       revert("amount exceeds available unlocked tokens");
108     }
109 }
```

An overflow can occur in the `return balanceOf(who)- timelockedTokens[who] + maxTokens;` line that will not allow the user to transfer any of his tokens, even if they are unlocked, until the end of the `disbursementPeriod`

Proof of Concept:

The Proof of Concept was executed using the following parameters:

- timelockedTokens -> 1000_000000000000000000
- vestTime -> chain.time() = now()
- cliffTime -> chain.time() + 15768000 = 6 months
- disbursementPeriod -> 31536000 seconds = 1 year

Then:

1. Waited 6 months: chain.sleep(15768000).
2. 6 months later 500_000000000000000000 of the user2 tokens were unlocked.
3. 200_000000000000000000 tokens were successfully transferred from user2 to user3.
4. user2 tried then to transfer another 200_000000000000000000 to user4. The transfer reverts with an Integer overflow error. All of the user2 tokens are totally locked now.
5. After this, user2 has to wait until the end of the disbursementPeriod to be able to transfer his tokens.

```
Deploying TimeLockToken (contract_TimeLockToken) -> owner.deploy(TimeLockToken, 'NewOrderDAOTKN', 'NODTKN', 1000000_000000000000000000, user1.address)
Transaction sent: 0xd4fcd753de745310daf51b12d6b4496e55af5fb9d151346e0921712f12107c23
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 3
  TimeLockToken.constructor confirmed   Block: 14050890   Gas used: 1091566 (0.14%)
  TimeLockToken deployed at: 0x683D6b2B72BDdc978cE1061fAa29e57727a8D4e9

contract_TimeLockToken.balanceOf(user1.address) -> 1000000000000000000000000
Calling -> contract_TimeLockToken.transfer(user2.address, 1000_000000000000000000, {'from': user1})
Transaction sent: 0x5a9eea2a64a45e5c026a9c50c4828517fe1da1d0e2410bc63a82ef69c23520eb
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  TimeLockToken.transfer confirmed   Block: 14050891   Gas used: 53001 (0.01%)

contract_TimeLockToken.balanceOf(user1.address) -> 999000000000000000000000
contract_TimeLockToken.balanceOf(user2.address) -> 1000000000000000000000

contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 1000000000000000000000
Calling -> contract_TimeLockToken.newTimeLock(1000_000000000000000000, chain.time() + 1, chain.time() + 15768000 + 1, 31536000, {'from': user2})
Transaction sent: 0x3927bad00d7f1113fab4e760951f8b27d9aa1636984d81de3e1656df5f4f9610
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  TimeLockToken.newTimeLock confirmed   Block: 14050892   Gas used: 106525 (0.01%)


contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 0


Sleeping 6 months...

contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 500000285388127853881
contract_TimeLockToken.balanceLocked(user2.address) -> 499999714611872146119
contract_TimeLockToken.balanceUnlocked(user2.address) -> 500000285388127853881
Calling -> contract_TimeLockToken.transfer(user3.address, 200_000000000000000000, {'from': user2})
Transaction sent: 0xf4be5afc12f727be19aafc32739e57dc2683a1c54ca56ce58e3691944fc87268
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 1
  TimeLockToken.transfer confirmed   Block: 14050894   Gas used: 59610 (0.01%)

Calling -> contract_TimeLockToken.transfer(user4.address, 200_000000000000000000, {'from': user2})
Transaction sent: 0x14bb7bbd8a7f26bdbf5102b2c0c1bd84063a225fcaf7097d5d717ee67f6daf03
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 2
  TimeLockToken.transfer confirmed (Integer overflow)   Block: 14050895   Gas used: 30493 (0.00%)

>>> tx.error()
Trace step 373, program counter 1539:
  File "contracts/GovernanceTokenV2.sol", line 131, in TimeLockToken.calcMaxTransferrable:
        if (timelockedTokens[who] < maxTokens){
          return balanceOf(who);
        }
        return balanceOf(who) - timelockedTokens[who] + maxTokens;
    }

    /// @dev Calculates the amount of locked tokens for address `who`
>>> output.greenn("contract_TimeLockToken.balanceOf(user2.address) -> " + str(contract_TimeLockToken.balanceOf(user2.address)))
maxTokens = contract_TimeLockToken.timelockedTokens(user2.address) * (chain.time() - contract_TimeLockToken.vestTime(user2.address)) / contract_TimeLockToken.disbursementPeriod(user2.address)
output.greenn("contract_TimeLockToken.timelockedTokens(user2.address) -> " + str(contract_TimeLockToken.timelockedTokens(user2.address)))
output.greenn("maxTokens -> " + str(int(maxTokens)))
contract_TimeLockToken.balanceOf(user2.address) -> 800000000000000000000
contract_TimeLockToken.timelockedTokens(user2.address) -> 1000000000000000000000
maxTokens -> 500001078132927430656
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to fix the overflow and the overall logic of the calcMaxTransferrable() function.

Remediation Plan:

SOLVED: NewOrderDAO team solved this issue in the commit ID: e7547837502f1e48151a52acaaa5c722dca4c253:

```
Deploying TimeLockToken (contract_TimeLockToken) -> owner.deploy(TimeLockToken, 'NewOrderDAOTKN', 'NODTKN', 1000000_000000000000000000, user1.address)
Transaction sent: 0xcdefd53f32d94b66e7a4e4d17b41a26aed24b5b337cd610a2e94a357e8129f17
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 2
  TimeLockToken.constructor confirmed   Block: 14108673   Gas used: 1091566 (0.14%)
  TimeLockToken deployed at: 0xefc5154f4c5619b9B1ffA706500186832FAe79C0

contract_TimeLockToken.balanceOf(user1.address) -> 1000000000000000000000000
Calling -> contract_TimeLockToken.transfer(user2.address, 1000_000000000000000000, {'from': user1})
Transaction sent: 0x3057a7092a0b6ca7a88126331d8321b653184c40fc867de6d21672a88e85e9ea
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  TimeLockToken.transfer confirmed   Block: 14108674   Gas used: 53001 (0.01%)

contract_TimeLockToken.balanceOf(user1.address) -> 999000000000000000000000
contract_TimeLockToken.balanceOf(user2.address) -> 1000000000000000000000

contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 1000000000000000000000
Calling -> contract_TimeLockToken.newTimeLock(1000_000000000000000000, chain.time() + 2, chain.time() + 15768000 + 1, 31536000, {'from': user2})
Transaction sent: 0xd26515d4121059fa35a27d6d1551531875ecc01216a4acfbeccc3cb6e4f87f1c
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 0
  TimeLockToken.newTimeLock confirmed   Block: 14108675   Gas used: 106525 (0.01%)


contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 0


Sleeping 6 months...

contract_TimeLockToken.calcMaxTransferrable(user1.address) -> 999000000000000000000000
contract_TimeLockToken.calcMaxTransferrable(user2.address) -> 500000253678335870116
contract_TimeLockToken.balanceLocked(user2.address) -> 499999746321664129884
contract_TimeLockToken.balanceUnlocked(user2.address) -> 500000253678335870116
Calling -> contract_TimeLockToken.transfer(user3.address, 200_000000000000000000, {'from': user2})
Transaction sent: 0x926bae1a5b304edfb11fe59f17525b101d1b3e34adad16743161f8f2d040a669
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 1
  TimeLockToken.transfer confirmed   Block: 14108677   Gas used: 59610 (0.01%)

Calling -> contract_TimeLockToken.transfer(user4.address, 200_000000000000000000, {'from': user2})
Transaction sent: 0xe2bb4e1c654c5cd0946c5373b93ee0dcd5db4ca565ce235e63b60587db7dcf13
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 2
  TimeLockToken.transfer confirmed   Block: 14108678   Gas used: 59610 (0.01%)


contract_TimeLockToken.balanceOf(user1.address) -> 999000000000000000000000
contract_TimeLockToken.balanceOf(user2.address) -> 600000000000000000000
contract_TimeLockToken.balanceOf(user3.address) -> 200000000000000000000
contract_TimeLockToken.balanceOf(user4.address) -> 200000000000000000000
```

3.2 (HAL-02) UNCHECKED TRANSFER - LOW

Description:

In the contracts DisbursementCliff and OneWaySwap the return value of some external transfer calls are not checked. Several tokens do not revert in case of failure and return false. If that happened, for example in the DisbursementCliff contract, the withdrawnTokens state variable would be incorrectly updated and the calculation of the amount of vested tokens would be wrong. It is also considered a best practice to check the return value of a ERC20.transfer() call.

Code Location:

DisbursementCliff.sol

Listing 3: DisbursementCliff.sol (Lines 76,86)

```
67 function withdraw(address _to, uint256 _value)
68     public
69     isReceiver
70 {
71     uint maxTokens = calcMaxWithdraw();
72     if (_value > maxTokens){
73       revert("Withdraw amount exceeds allowed tokens");
74     }
75     withdrawnTokens += _value;
76     token.transfer(_to, _value);
77 }
78
79 /// @dev Transfers all tokens to multisig wallet
80 function walletWithdraw()
81     public
82     isWallet
83 {
84     uint balance = token.balanceOf(address(this));
85     withdrawnTokens += balance;
86     token.transfer(wallet, balance);
87 }
```

one-way-swap.sol

Listing 4: one-way-swap.sol (Lines 36,37,44)

```
32 function swap(uint256 amount)
33     public
34     whenNotPaused
35 {
36     oldToken.transferFrom(msg.sender, burnAddress, amount);
37     newToken.transfer(msg.sender, amount);
38 }
39
40 function burn(uint256 amount, string memory why)
41     public
42     whenNotPaused
43 {
44     oldToken.transferFrom(msg.sender, burnAddress, amount);
45     emit Burned(msg.sender, amount, why);
46 }
```

Listing 5: one-way-swap.sol (Lines 70)

```
66 function walletWithdraw(ERC20 token, uint256 amount, address
       destination)
67     public
68     onlyOwner
69 {
70     token.transfer(destination, amount);
71 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to use SafeERC20.

Remediation Plan:

SOLVED: NewOrderDAO team now makes use of SafeERC20.safeTransfer() and SafeERC20.safeTransferFrom() in all their token transfers.

3.3 (HAL-03) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The constructor of the OneWaySwap contract is missing address validation. Every address should be validated and checked that is different from zero. This is also considered a best practice.

Code location:

Listing 6: one-way-swap.sol (Lines 24)

```
constructor(ERC20 oldToken_, ERC20 newToken_, address burnAddress_
    )
{
    oldToken = oldToken_;
    newToken = newToken_;
    burnAddress = burnAddress_;
    _pause();
}
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

SOLVED: NewOrderDAO team added the zero address checks.

3.4 (HAL-04) SOLC 0.8.3 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

Solidity compiler version 0.8.3, 0.8.4 and 0.8.9 fixed important bugs in the compiler. The version 0.8.3 set in the truffle-config.js file of the GovernanceTokenV2 project is missing all these fixes:

- 0.8.4
- 0.8.9

Code Location:

Listing 7: GovernanceTokenV2.sol

```
pragma solidity ^0.8.3;
```

Listing 8: truffle-config.js (Lines 102)

```
100 compilers: {
101   solc: {
102     version: "0.8.3", // Fetch exact version from solc-bin (default: truffle's version)
103     // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
104     // settings: {          // See the solidity docs for advice about optimization and evmVersion
105     //  optimizer: {
106     //    enabled: false,
107     //    runs: 200
108     //  },
109     //  evmVersion: "byzantium"
110     // }
111   },
112 },
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the most tested and stable versions, such as 0.6.12 or 0.7.6. Otherwise, if you still want to use ^0.8.0, because of the new functionality it provides, it is recommended to use 0.8.9 version.

Remediation Plan:

SOLVED: NewOrderDAO team set in the truffle-config.js file the 0.8.9 version for the contracts GovernanceTokenV2 and OneWaySwap and the 0.6.12 version for the DisbursementCliff contract.

3.5 (HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the following contracts there are functions marked as public but they are never directly called within the same contract or in any of their descendants:

GovernanceTokenV2.sol
- newTimeLock() (GovernanceTokenV2.sol#70-85)
- balanceUnlocked() (GovernanceTokenV2.sol#158-160)

DisbursementCliff.sol
- withdraw() (DisbursementCliff.sol#67-77)
- walletWithdraw() (DisbursementCliff.sol#80-87)

one-way-swap.sol
- swap() (onewayswap.sol#32-38)
- burn() (onewayswap.sol#40-46)
- walletWithdraw() (onewayswap.sol#66-71)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If the functions are not intended to be called internally or by their descendants, it is better to mark all of these functions as external to reduce gas costs.

Remediation Plan:

SOLVED: NewOrderDAO team declared the mentioned functions as external to reduce the gas costs.

3.6 (HAL-06) TIMELOCKTOKEN IS NOT PAUSABLE - INFORMATIONAL

Description:

The contract TimeLockToken is not pausable/ownable. Even if this addition would add centralization it could be useful in case of an emergency, for example, the token could be paused in case of a cross-chain bridge hack.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In case of wanting to add an extra security layer, Halborn recommends making the contract pausable as a mitigation against external contract hacks.

Remediation Plan:

SOLVED: NewOrderDAO team created a Pausable variant of the TimeLockToken contract called GovernanceTokenPausable. NewOrderDAO team will decide which variant to deploy.

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:

GovernanceTokenV2.sol

```
TimeLockToken.newTimeLock(uint256,uint256,uint256,uint256) (contracts/GovernanceTokenV2.sol#70-85) uses a dangerous strict equality:
        - require(bool,string)(balanceLocked(msg.sender) == 0,Cannot timelock additional tokens while tokens already locked) (contracts/GovernanceTokenV2.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

TimeLockToken.newTimeLock(uint256,uint256,uint256,uint256) (contracts/GovernanceTokenV2.sol#70-85) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(balanceLocked(msg.sender) == 0,Cannot timelock additional tokens while tokens already locked) (contracts/GovernanceTokenV2.sol#75)
        - require(bool,string)(vestTime_ > block.timestamp,vesting start must be in the future) (contracts/GovernanceTokenV2.sol#77)
TimeLockToken._beforeTokenTransfer(address,address,uint256) (contracts/GovernanceTokenV2.sol#100-109) uses timestamp for comparisons
        Dangerous comparisons:
        - from != address(0x0) && amount > maxTokens (contracts/GovernanceTokenV2.sol#106)
TimeLockToken.calcMaxTransferrable(address) (contracts/GovernanceTokenV2.sol#114-132) uses timestamp for comparisons
        Dangerous comparisons:
        - vestTime[who] > block.timestamp || cliffTime[who] > block.timestamp (contracts/GovernanceTokenV2.sol#123)
        - timelockedTokens[who] < maxTokens (contracts/GovernanceTokenV2.sol#128)
TimeLockToken.balanceLocked(address) (contracts/GovernanceTokenV2.sol#136-155) uses timestamp for comparisons
        Dangerous comparisons:
        - vestTime[who] > block.timestamp || cliffTime[who] > block.timestamp (contracts/GovernanceTokenV2.sol#146)
        - maxTokens >= timelockedTokens[who] (contracts/GovernanceTokenV2.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.3']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.3 (contracts/GovernanceTokenV2.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC20._beforeTokenTransfer(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#331-335) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#275-290) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.3 (contracts/GovernanceTokenV2.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-116)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#121-123)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#132-135)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#150-164)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#178-181)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#197-205)
newTimeLock(uint256,uint256,uint256,uint256) should be declared external:
        - TimeLockToken.newTimeLock(uint256,uint256,uint256,uint256) (contracts/GovernanceTokenV2.sol#70-85)
balanceUnlocked(address) should be declared external:
        - TimeLockToken.balanceUnlocked(address) (contracts/GovernanceTokenV2.sol#158-160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

AUTOMATED TESTING

DisbursementCliff.sol

```
DisbursementCliff.withdraw(address,uint256) (contracts/DisbursementCliff.sol#67-77) ignores return value by token.transfer(_to,_value) (contracts/DisbursementCliff.sol#76)
DisbursementCliff.walletWithdraw() (contracts/DisbursementCliff.sol#80-87) ignores return value by token.transfer(wallet,balance) (contracts/DisbursementCliff.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

DisbursementCliff.withdraw(address,uint256) (contracts/DisbursementCliff.sol#67-77) should emit an event for:
        - withdrawnTokens += _value (contracts/DisbursementCliff.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

DisbursementCliff.constructor(address,address,uint256,uint256,uint256,Token) (contracts/DisbursementCliff.sol#45-62) uses timestamp for comparisons
        Dangerous comparisons:
        - startDate == 0 (contracts/DisbursementCliff.sol#56)
        - cliffDate < startDate (contracts/DisbursementCliff.sol#59)
DisbursementCliff.withdraw(address,uint256) (contracts/DisbursementCliff.sol#67-77) uses timestamp for comparisons
        Dangerous comparisons:
        - _value > maxTokens (contracts/DisbursementCliff.sol#72)
DisbursementCliff.calcMaxWithdraw() (contracts/DisbursementCliff.sol#91-101) uses timestamp for comparisons
        Dangerous comparisons:
        - withdrawnTokens >= maxTokens || startDate > now || cliffDate > now (contracts/DisbursementCliff.sol#97)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Pragma version^0.6.0 (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#3) allows old versions
Pragma version^0.6.0 (contracts/DisbursementCliff.sol#1) allows old versions
solc-0.6.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter DisbursementCliff.withdraw(address,uint256)._to (contracts/DisbursementCliff.sol#67) is not in mixedCase
Parameter DisbursementCliff.withdraw(address,uint256)._value (contracts/DisbursementCliff.sol#67) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

transfer(address,uint256) should be declared external:
        - Token.transfer(address,uint256) (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#16)
transferFrom(address,address,uint256) should be declared external:
        - Token.transferFrom(address,address,uint256) (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#17)
approve(address,uint256) should be declared external:
        - Token.approve(address,uint256) (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#18)
balanceOf(address) should be declared external:
        - Token.balanceOf(address) (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#19)
allowance(address,address) should be declared external:
        - Token.allowance(address,address) (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#20)
totalSupply() should be declared external:
        - Token.totalSupply() (node_modules/@gnosis.pm/util-contracts/contracts/Token.sol#21)
withdraw(address,uint256) should be declared external:
        - DisbursementCliff.withdraw(address,uint256) (contracts/DisbursementCliff.sol#67-77)
walletWithdraw() should be declared external:
        - DisbursementCliff.walletWithdraw() (contracts/DisbursementCliff.sol#80-87)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

one-way-swap.sol

```
OneWaySwap.swap(uint256) (contracts/onewayswap.sol#32-38) ignores return value by oldToken.transferFrom(msg.sender,burnAddress,amount) (contracts/onewayswap.sol#36)
OneWaySwap.swap(uint256) (contracts/onewayswap.sol#32-38) ignores return value by newToken.transfer(msg.sender,amount) (contracts/onewayswap.sol#37)
OneWaySwap.burn(uint256,string) (contracts/onewayswap.sol#40-46) ignores return value by oldToken.transferFrom(msg.sender,burnAddress,amount) (contracts/onewayswap.sol#44)
OneWaySwap.walletWithdraw(ERC20,uint256,address) (contracts/onewayswap.sol#66-71) ignores return value by token.transfer(destination,amount) (contracts/onewayswap.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

OneWaySwap.constructor(ERC20,ERC20,address).burnAddress_ (contracts/onewayswap.sol#24) lacks a zero-check on :
                - burnAddress = burnAddress_ (contracts/onewayswap.sol#28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in OneWaySwap.burn(uint256,string) (contracts/onewayswap.sol#40-46):
        External calls:
        - oldToken.transferFrom(msg.sender,burnAddress,amount) (contracts/onewayswap.sol#44)
        Event emitted after the call(s):
        - Burned(msg.sender,amount,why) (contracts/onewayswap.sol#45)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.9']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.9 (contracts/onewayswap.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#275-290) is never used and should be removed
ERC20._mint(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#252-262) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.9 (contracts/onewayswap.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-56)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#62-65)
name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-116)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#121-123)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#132-135)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#150-164)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#178-181)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#197-205)
swap(uint256) should be declared external:
        - OneWaySwap.swap(uint256) (contracts/onewayswap.sol#32-38)
burn(uint256,string) should be declared external:
        - OneWaySwap.burn(uint256,string) (contracts/onewayswap.sol#40-46)
walletWithdraw(ERC20,uint256,address) should be declared external:
        - OneWaySwap.walletWithdraw(ERC20,uint256,address) (contracts/onewayswap.sol#66-71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- No major issues were found by Slither.

ERC20 checks:

GovernanceTokenV2.sol

```
# Check ERC20

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
```

AUTOMATED TESTING

```
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


# Check TimeLockToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] ERC20 has increaseAllowance(address,uint256)
        [✓] TimeLockToken has increaseAllowance(address,uint256)
```

- All the Slither ERC20 checks were passed successfully.

THANK YOU FOR CHOOSING

HALBORN