Competitive Security Assessment

Nesten

Mar 24th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Nesten
Platform & Language	Solidity
Codebase	• https://github.com/Nesten-Inc/contracts • audit commit - 916a99e96702e3e6d228c8973703a8ae097addc4 • final commit - 1fb9b1debe66a9fe42fd8e907db424ecdf52d5ed
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

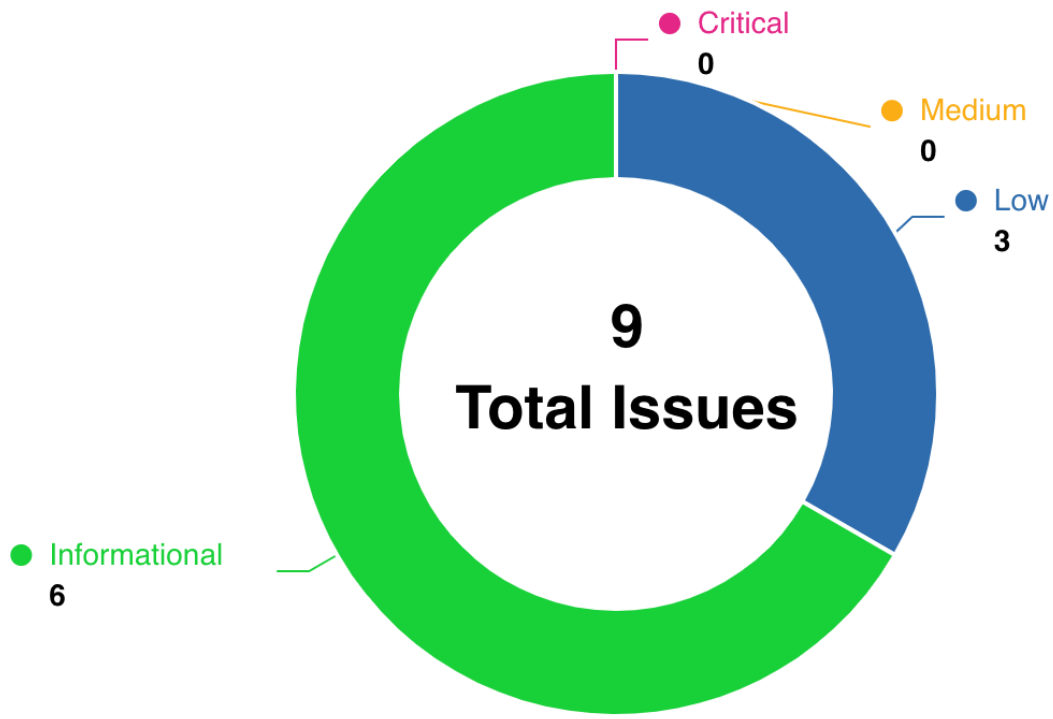
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	3	0	0	2	1	0
Informational	6	0	0	5	0	1

Audit Scope

File	Commit Hash
./Contracts/Lock.sol	916a99e96702e3e6d228c8973703a8ae097addc4
./DWIN.sol	916a99e96702e3e6d228c8973703a8ae097addc4
./Contracts/base.sol	916a99e96702e3e6d228c8973703a8ae097addc4

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
NST-1	Unlocked Pragma Version	Language Specific	Informational	Fixed	rajatbeladiya
NST-2	Attacker can Dos `lockedOf[_account]` to prevent withdraw tokens	Logical	Low	Mitigated	rajatbeladiya
NST-3	Functions not used internally could be marked external	Gas Optimization	Informational	Fixed	rajatbeladiya
NST-4	Gas Optimization - Cache array length outside of loop	Gas Optimization	Informational	Fixed	rajatbeladiya

NST-5	Miss Zero address check for changing state function	Logical	Low	Fixed	rajatbeladiya
NST-6	Missing Event Emission in `Lock` contract `lockedTransfer` function	Code Style	Informational	Fixed	hunya
NST-7	Missing Event Emission in `Lock` contract `withdrawOf` function	Code Style	Informational	Declined	hunya
NST-8	totalLocked() returns wrong amount	Logical	Informational	Fixed	rajatbeladiya
NST-9	unhandled return values of ERC20 transfers	Code Style	Low	Fixed	rajatbeladiya

NST-1: Unlocked Pragma Version

Category	Severity	Code Reference	Status	Contributor
Language Specific	Informational	• code/DWIN.sol#L2 • code/Contracts/Lock.sol#L2 • code/Contracts/base.sol#L2	Fixed	rajatbeladiya

Code

```
2:pragma solidity ^0.8.9;

2:pragma solidity ^0.8.4;

2:pragma solidity ^0.8.4;
```

Description

rajatbeladiya : Nesten solidity files have a pragma solidity version number with ^0.8.4. The caret (^) points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

rajatbeladiya : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contract.

Client Response

Solidity versions have been fixed.

NST-2:Attacker can Dos `lockedOf[_account]` to prevent withdraw tokens

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/Contracts/Lock.sol#L23 • code/Contracts/Lock.sol#L30-L43 • code/Contracts/Lock.sol#L52-L60	Mitigated	rajatbeladiya

Code

```
23:         lockedOf[_to].push(LockedBalance({ amount:_amount, posted:current, until:
(current+_period) }));

30:     function readyToWithdrawOf(address _account)
31:         public
32:         view
33:         returns(uint) {
34:         uint toWithdraw = 0;
35:         uint current = block.timestamp;
36:         for (uint i = 0; i < lockedOf[_account].length; ) {
37:             if (current >= lockedOf[_account][i].until) {
38:                 toWithdraw += lockedOf[_account][i].amount;
39:             }
40:             unchecked { i++; }
41:         }
42:         return toWithdraw;
43:     }

52:         for (uint i = 0; i < lockedOf[_account].length;) {
53:             uint j = lockedOf[_account].length-1-i; // reversed i
54:             if (current >= lockedOf[_account][j].until) {
55:                 toWithdraw += lockedOf[_account][j].amount;
56:                 lockedOf[_account][j] = lockedOf[_account][lockedOf[_account].length-1];   //
swap with the last item
57:                 lockedOf[_account].pop();    // remove last item
58:             }
59:             unchecked { i++; }
60:         }
```

Description

rajatbeladiya : Here protocol have locked transfers mechanism and it is uses `lockedOf[_account]` array for it in Lock.sol

While withdrawing these locked funds, it is looping over the `lockedOf[_account]`.

So when the array of `lockedOf[_account]` is large enough, `withdrawOf()` function always reverts because of the gas limit DoS and users will not be able to withdraw their tokens.

Attack Scenario: `lockedTransfer` lets you deposits `0 _amount token` for `_to`.

Suppose Alice has following token amounts [`10000 releasedAmount`, `1000 notReleasedAmount`]

So, the Attacker can deposit 0 amount multiple times for Alice which will lead to `lockedOf[Alice]` array large enough that Alice will not be able to withdraw his actual 11000 token amounts because of out of gas DoS.

Alice will lose his actual 11000 token amount and it will be stuck to the contract.

Recommendation

rajatbeladiya : Add check `_amount > 0` to `lockedTransfer()` function Bound `lockedOf[_account]` to limit or revise the transfer logic.

Client Response

Disallowing 0 for _amount on lockedTransfer(), gas DoS attack has been mitigated.

NST-3:Functions not used internally could be marked external

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/Contracts/Lock.sol#L30-L33	Fixed	rajatbeladiya

Code

```
30:     function readyToWithdrawOf(address _account)
31:         public
32:         view
33:         returns(uint) {
```

Description

rajatbeladiya : `readyToWithdrawOf()` has public visibility and it is not used in the contract internally. best practice is to mark external which is not used internally.

Recommendation

rajatbeladiya : change public to external for the `readyToWithdrawOf()` function

Client Response

Visibility changed to external.

NST-4:Gas Optimization - Cache array length outside of loop

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/Contracts/Lock.sol#L36 • code/Contracts/Lock.sol#L52	Fixed	rajatbeladiya

Code

```
36:         for (uint i = 0; i < lockedOf[_account].length; ) {

52:         for (uint i = 0; i < lockedOf[_account].length;) {
```

Description

rajatbeladiya : If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Recommendation

rajatbeladiya : Consider below fix in the `readyToWithdrawOf()` and `withdrawOf()` function

```
        uint lockedOfLength = lockedOf[_account].length;
        for (uint i = 0; i < lockedOfLength; ) {
            if (current >= lockedOf[_account][i].until) {
                toWithdraw += lockedOf[_account][i].amount;
            }
            unchecked { i++; }
        }
```

Client Response

Although one is a view, all lengths of the arrays are cached outside of the loop.

NST-5:Miss Zero address check for changing state function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/Contracts/Lock.sol#L19	Fixed	rajatbeladiya

Code

```
19:    function lockedTransfer(address _to, uint _amount, uint _period) external returns(bool) {
```

Description

rajatbeladiya : zero address check is missing for `_to` in `lockedTransfer()` which can lead to lock tokens due to user error.

Recommendation

rajatbeladiya : add zero address check for `_to` in `lockedTransfer()`

Client Response

Now lockedTransfer() reverts the transfer to zero address.

NST-6:Missing Event Emission in `Lock` contract `lockedTransfer` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/Contracts/Lock.sol#L19-L26	Fixed	hunya

Code

```
19:    function lockedTransfer(address _to, uint _amount, uint _period) external returns(bool) {
20:        require(_period > 0, "Period should be greater than 0");
21:        uint current = block.timestamp;
22:        // store lock period
23:        lockedOf[_to].push(LockedBalance({ amount:_amount, posted:current, until:
(current+_period) }));
24:        // transfer token to lock (this contract)
25:        return token.transferFrom(msg.sender, address(this), _amount);
26:    }
```

Description

hunya : function `lockedTransfer` implements a "transfer with lock" logic, and changes state variables. However, it does not emit event to pass the changes out of chain.

Recommendation

hunya : Recommend emitting events, for all the essential state variables that can be changed during runtime.

Consider below fix in the `Lock.lockedTransfer()` function

```
event Lock(address indexed from, address indexed to, uint256 amount, uint256 posted, uint256 until);

function lockedTransfer(address _to, uint _amount, uint _period) external returns(bool) {
    require(_period > 0, "Period should be greater than 0");
    uint current = block.timestamp;
    // store lock period
    lockedOf[_to].push(LockedBalance({ amount:_amount, posted:current, until:(current+_period) }));
    emit Lock(msg.sender, _to, _amount, current, current+_period);
    // transfer token to lock (this contract)
    return token.transferFrom(msg.sender, address(this), _amount);
}
```

Client Response

LockedTransfer event has been added to help tracking.

NST-7:Missing Event Emission in `Lock` contract `withdrawOf` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/Contracts/Lock.sol#L30-L43	Declined	hunya

Code

```
30:     function readyToWithdrawOf(address _account)
31:         public
32:         view
33:         returns(uint) {
34:         uint toWithdraw = 0;
35:         uint current = block.timestamp;
36:         for (uint i = 0; i < lockedOf[_account].length; ) {
37:             if (current >= lockedOf[_account][i].until) {
38:                 toWithdraw += lockedOf[_account][i].amount;
39:             }
40:             unchecked { i++; }
41:         }
42:         return toWithdraw;
43:     }
```

Description

hunya : Function `withdrawOf` implements "withdraw the locked token" logic, and changes state variables. However, it does not emit events to pass the changes out of chain.

Recommendation

hunya : Recommend emitting events, for all the essential state variables that can be changed during runtime. Consider below fix in the `Lock.withdrawOf()` function

```
event Withdrawal(address indexed account, uint256 amount);

function withdrawOf(address _account)
    external
    returns(bool) {
    // count amount and remove withdrawed lock ups. This DOES NOT preserve the order of list.
    uint toWithdraw = 0;
    uint current = block.timestamp;
    // for (uint i = lockedOf[_account].length-1; i >= 0; i--) { // from backward.... gives very
weird error...
    for (uint i = 0; i < lockedOf[_account].length;) {
        uint j = lockedOf[_account].length-1-i; // reversed i
        if (current >= lockedOf[_account][j].until) {
            toWithdraw += lockedOf[_account][j].amount;
            lockedOf[_account][j] = lockedOf[_account][lockedOf[_account].length-1];   // swap with
the last item
            lockedOf[_account].pop();    // remove last item
        }
        unchecked { i++; }
    }
    require (toWithdraw > 0, "There is no token to withdraw.");
    emit Withdrawal(_account, toWithdraw);
    // transfer them.
    return token.transfer(_account, toWithdraw);
}
```

Client Response

We don’t need an extra event to track these state changes. To preserve gas, we will track Transfer events instead.

NST-8:totalLocked() returns wrong amount

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/Contracts/Lock.sol#L75-L80	Fixed	rajatbeladiya

Code

```
75:     function totalLocked()
76:         external
77:         view
78:         returns(uint) {
79:             return token.balanceOf(address(this));
80:     }
```

Description

rajatbeladiya : `totalLocked()` function is used to return the total amount of locked tokens, but it is returning the total balance of the contract. If any one transfer tokens to `Lock.sol` contract directly then calculations will be wrong because actual `totalLocked()` will be different.

Recommendation

rajatbeladiya : Maintain `totalDeposit` state in the contract

Client Response

totalLocked is added to track the total amount locked, instead of returning the balance of the contract.

NST-9:unhandled return values of ERC20 transfers

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	• code/Contracts/Lock.sol#L25 • code/Contracts/Lock.sol#L63	Fixed	rajatbeladiya

Code

```
25:         return token.transferFrom(msg.sender, address(this), _amount);

63:         return token.transfer(_account, toWithdraw);
```

Description

rajatbeladiya : ERC20 implementations are not always consistent. Some implementations of transfer and transferFrom could return `false` on failure instead of reverting. It is safer to wrap such calls into `require()` statements to these failures.

Recommendation

rajatbeladiya : Use require to check the return value and revert on 0/false or use OpenZeppelin's SafeERC20 wrapper functions.

Client Response

Now check if the return value of ERC20 transfer is false.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.