Nsure

Security Assessment

February 22th, 2021

For :
Nsure

By :

Daniel Tan @ CertiK
daniel.tan@certik.org

Wythe Li @ CertiK
weizhi.li@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Nsure
Description	Nsure is an open insurance platform for Open Finance.
Platform	Ethereum; Solidity; Yul
Codebase	Private GitHub Repository
Commit	230324802b30eccce9f0d49360607e77b13caaa8 fecbd927195a68626b3e1b6d7e2ce96dd8094f74
Zipfile	224c16c6cf86f772fdab2902db8fd4af553ca86c8b1151e4c2ce35a0b8b6e146

Audit Summary

Delivery Date	Feb. 22th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 4, 2021 - Feb. 14, 2021

Vulnerability Summary

Total Issues	17
Total Critical	0
Total Major	2
Total Minor	5
Total Informational	10

Executive Summary

This report has been prepared for **Nsure** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

File in Scope

ID	Contract	SHA-256 Checksum
BY	Buy.sol	fbaa345d1653ca0cd6e78fe832c398bd426d76439a26e5cc8d7fe202ca1ac95a
CC	CapitalConverter.sol	3ce8e711598a4c5d06afb38c54409a1fb32b783e94cb18aec899eeb16b49326a
CS	CapitalStake.sol	13dfa12563ffa5b2f869f2af7d2aaa1517dd346696473a8412a7288e82cbf938
CP	ClaimPurchaseMint.sol	b6b73daecfeddc1e0ae96a133674b14fc2425ff6184a98191194bd5f18510113
LF	LockFunds.sol	bb2931e24540d1c8958129f4ac81e3d74a15d06721982ea74978e8247e8bfad1
PD	Product.sol	d952d816d8803bdf04169c45f86be0ea8703d099b6abd690cd0f19ee60ecd06f
SP	Surplus.sol	c47afda31df37ba7d473097ce1c27b39d4083e12f3e677765b0c02fdeba61302
TS	Treasury.sol	37fd8127e905a4d7667b73f3922bce816df947fd5bea9359f6b6d121f923e493
IC	ICover.sol	861fdfbaf1f8b6fe56d2dccb91d0b603d0c60497d1d3961b235e21bec97eb029
IM	IMerkleDistributor.sol	2dea76db9770f3e9e3b888a668b7b672b90410aca9964b8336ac11c9605456a2
IN	INsure.sol	ab7d005ebb4782f83bd2faff8f72ace75f1f81c372e90216eb13b8e25e476e7d
IW	IWETH.sol	0d461ecceaf082f9c0e6d0522599b1253874809ea93ce6486e3c400699c2557a

Findings

ID	Title	Type	Severity	Resolved
BY-01	Function State Mutability	Coding Style	Informational	✓
BY-02	Proper Usage of "public" and "external" type	Coding Style	Informational	✓
BY-03	Misleading Error Message	Logical Issue	Minor	✓
BY-04	Bring the require forward	Gas Optimization	Informational	✓
BY-05	Missing Check on Existing Currency When Adding Currency	Logical	Informational	✓
BY-06	Checking the Sum of each portion	Logical	Informational	✓
BY-07	Status Value	Logical Issue	Minor	✓
CC-01	Missing Check for Zero Address	Logical Issue	Informational	✓
CC-02	Administrator Capability	Optimization	Major	ⓘ
CC-03	Solidity Version	Coding Style	Informational	✓
CS-01	Function State Mutability	Coding Style	Informational	✓
CS-02	`Checks-effects-pattern` Not Used	Implementation	Minor	✓
CS-03	Missing Return Vallue Check for Transfer Function	Logical Issue	Informational	ⓘ
CS-04	Missing Index Checking	Logic Issue	Informational	✓
CS-05	Compiler Errors	Compile Error	Minor	✓
CP-01	Missing Update of `lastRewardBlock`	Logical Issue	Major	✓
TS-01	Missing Emit Events	Optimization	Minor	✓

BY-01: Function State Mutability

Type	Severity	Location
Optimization	Informational	Buy.sol, CapitalConverter.sol , ClaimPurchaseMint.sol

Description:

If variables are constants, better to define as constants. Constant state variables should be declared constant to save gas.

Buy.sol, ClaimPurchaseMint.sol:

```
        string public version = "1";
```

CapitalConverter.sol:

```
        address public ETHEREUM = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
```

Recommendation:

Consider changing the codes like below:

Buy.sol, ClaimPurchaseMint.sol:

```
        string public constant version = "1";
```

CapitalConverter.sol:

```
        address public constant ETHEREUM = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

BY-02: Proper Usage of "public" and "external" type

Type	Severity	Location
Optimization	Informational	Buy.sol, CapitalConverter.sol, CapitalStake.sol, ClaimPurchaseMint.sol, LockFunds.sol, Product.sol, Surplus.sol, Treasury.sol

Description:

"Public" functions that are never called by the contract could be declared "external" . When the inputs are arrays, "external" functions are more efficient than "public" functions.

Examples:

Functions `addDivCurrency()` , `delDivCurrency()` , `getDivCurrencyLength()` in contract `Buy` ;

Function `convert` in contract `CapitalConverter` ;

Functions `add()` , `set()` , `deposit()` , `unstake()` , `withdraw()` in contract `CapitalStake` ;

Functions `totalSupply()` , `balanceOf()` in contract `ClaimPurchaseMint` ;

Functions `totalSupply()` , `balanceOf()` , `getDivCurrencyLength()` , `addDivCurrency()` in contract `LockFunds` ;

Functions `getLength()` , `getProduct()` , `addProduct()` , `deleteProduct()` , `updateStatus()` in contract `Product` ;

Function `myBalanceOf()` in contract `Surplus` ;
Surplus.sol: myBalanceOf();

Function `myBalanceOf()` in contract `Treasury` .

Recommendation:

Consider declaring the above functions as `external` .

Example:

```
function addDivCurrency(address currency) external onlyOwner {
    divCurrencies.push(currency);
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

BY-03: Misleading Error Message

Type	Severity	Location
Logical Issue	Minor	Buy.sol, CapitalStake.sol, CapitalConverter.sol

Description:

Lack of precision in the error messages.

```
Buy.sol
require(_product.getStatus(_productId) == 0, "disable");
require(divCurrencies[currency] != address(0) && currency < divCurrencies.length, "no
currency");

CapitalConverter.sol
require(_amount <= maxConvert, "too much");
require(balanceOf(_msgSender()) >= _value && _value > 0, "CapitalConverter: _value is not
good");

CapitalStake.sol
require(user.amount >= _amount, "unstake: not good");
```

Recommendation:

Consider giving proper messages.

Alleviation:

The development team heeded our advice and resolved this issue in commit
6da44d95ff4f273fde637a759f842c98c3264880

BY-04: Bring the `require` forward

Type	Severity	Location
Gas Optimization	Informational	Buy.sol, LockFunds.sol

Description:

Function `buyInsurance()` of the contract `Buy` and `claim()` of `LockFunds` need optimization that the `require` :

```
require(block.timestamp <= deadline, "signature expired");
```

could be brought forward to the beginning to save gas.

Recommendation:

Consider bring it forward to the beginning of the function.

Alleviation:

The development team heeded our advice and resolved this issue in commit
6da44d95ff4f273fde637a759f842c98c3264880

BY-05: Missing Check on Existing Currency When Adding Currency

Type	Severity	Location
Logical	Informational	Buy.sol

Description:

There is no check on the existing currency to avoid adding a same currency in the function `addDivCurrency` of the contract `Buy.sol` .

Recommendation:

Consider checking the existing currency before adding a new currency.

Alleviation:

The development team heeded our advice and resolved this issue in commit
a24af43cdb91b21855bb66dad4a444b3906c300d

BY-06: Checking the Sum of each portion.

Type	Severity	Location
Logical Issue	Minor	Buy.sol

Description:

There are functions `setSurplusRate` and `setStakeRate` update the parameters, `surplueRate` and `stakeRate` separately.

Recommendation:

Consider checking the sum of the parameters `setSurplusRate` , `setStakeRate` , and `treasuryRate` to be one.

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

BY-07: Status Value

Type	Severity	Location
Logical Issue	Informational	Buy.sol

Description:

Zero is used as a valid product status in the contract `Buy.sol`.
Example:

```
require(_product.getStatus(_productId) == 0, "disable");
```

Recommendation:

You will also get a zero value once there is an invalid `_productId` is passed into the function since the default value of variable is zero.
Consider using a non-zero value as a valid status.
Example:

```
require(_product.getStatus(_productId) == 1, "disable");
```

Alleviation:

The development team heeded our advice and resolved this issue in commit
6da44d95ff4f273fde637a759f842c98c3264880

CC-01: Missing Check for Zero Address

Type	Severity	Location
Logical Issue	Informational	CapitalConverter.sol, CapitalStake.sol, ClaimPurchaseMint.sol, LockFunds.sol

Description:

Missing zero address for parameter `_nsure` in the constructors of contract `CapitalConverter.sol` , `CapitalStake.sol` , `ClaimPurchaseMint.sol` , and `LockFunds.sol` .

Missing zero address checking for the parameter `_lpToken` in the function `add()` of the contract `CapitalStake.sol` .

Missing zero address checking for the parameter `_signer` in the function `setSigner()` of the contract `ClaimPurchaseMint.sol` .

Missing zero address checking for the parameters `_signer` , `_operator` , and `_currency` in the functions `setSigner()` , `setOperator()` , and `addDivCurrency()` of the contract `LockFunds.sol` respectively.

Recommendation:

Consider adding zero address checkings on addresses like below

```
    require(_token != address(0), "_token is zero");
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

CC-02: Administrator Capability

Type	Severity	Location
Optimization	Major	CapitalConverter.sol, Surplus.sol, Treasury.sol

Description:

Function `payouts()` in contracts *CapitalConverter.sol*, *Surplus.sol*, and *Treasury.sol* are only callable by the operator, and able to transfer tokens to dedicated addresses.

```
    function payouts(address payable _to, uint256 _amount, address token) external
onlyOperator {
        if (token != ETHEREUM) {
            IERC20(token).safeTransfer(_to, _amount);
        } else {
            _to.transfer(_amount);
        }

        emit ePayouts(_to, _amount);
    }
```

Function `setOperator()` has the ability to change the operator of the contracts, exists in the contracts: *Treasury.sol*, *Surplus.sol*, *Product.sol*, *LockFunds.sol*, *CapitalConverter.sol*.

Recommendation:

The advantage of `payouts()` function in the protocol is that the operator can pay for the claim and rescue the assets in this contract after all users migrated. It is also worthy of note the downside of `payouts()` function, where the treasury in this contract can be migrated to any addresses.

To improve the trustworthiness of the project, any plan to call this `payouts()` method is better to move to the execution queue of Timelock, and emitting event for the sensitive action `setOperator()` .

Alleviation:

(Nsure Response) Payout is used to pay for successful claims, In our V1 it will be excuted in a centralized way

CC-03: Solidity Version

Type	Severity	Location
Coding Style	Informational	CapitalConverter.sol, IWETH.sol

Description:

There are two contract `CapitalConverter.sol` and `IWETH` do not specify the version of Solidity.

Recommendation:

Consider specifying the Solidity version for every contract.

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

CS-01: Function State Mutability

Type	Severity	Location
Optimization	Informational	CapitalStake.sol

Description:

There is a function `getMultiplier()` in the contract `CapitalStake.sol` does not read or modify state and declared as view.

Recommendation:

Consider declaring function `getMultiplier()` as pure.

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

CS-02: `Checks-effects-pattern` Not Used

Type	Severity	Location
Implementation	Minor	CapitalStake.sol

Description:

There are state variables are changed after transfers are done in the functions `deposit()` , `withdraw()` , and `unstake()` of the contract `CapitalStake.sol` . This may lead to reentrancy issue.

Recommendation:

It is recommended to follow checks-effects-interactions pattern. It shields public functions from re-entrancy attacks, refer to: https://docs.soliditylang.org/en/v0.8.0/security-considerations.html#re-entrancy

Alleviation:

The development team heeded our advice and resolved this issue in commit 54b487dd7d74a28fe1dd075472c96829680fdc57

CS-03: Missing Return Vallue Check for Transfer

Type	Severity	Location
Logical Issue	Minor	CapitalStake.sol, ClaimPurchaseMint.sol, LockFunds.sol

Description:

Missing return vallue check for transfers.
Examples:

CapitalStake.sol: `nsure.transfer()` ;

ClaimPurchaseMint.sol: `Nsure.mint()` ;

LockFunds.sol: `Nsure.transferFrom()` .

Recommendation:

Consider adding checkings for the returning value of the above calls.
Example:

```
      require(nsure.transfer(), "Failed to do the nsure.transfer()");
```

Alleviation:

(Nsure Response) for transfer and transferFrom functions, no need to return anything, for it will automatically return exemption or return true.。require is not a good fit for mint function，but will add new logic which will handle the return value.

CS-04: Missing Index Checking

Type	Severity	Location
Optimization	Informational	CapitalStake.sol

Description:

Access array data but don't do the early index checking for array `poolInfo` and `userInfo` in the contract `CapitalStake.sol` .
Don't do the early index checking for array `_products` in the contract `Product.sol` .

Recommendation:

Consider checking the index validation before access the data.
Example:

```
function withdraw(uint256 _pid) public whenNotPaused {
  require(_pid < poolInfo.length && _pid < userInfo.length, "invalid _pid");
  PoolInfo storage pool = poolInfo[_pid];
  UserInfo storage user = userInfo[_pid][msg.sender];
  ...
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

CS-05: Compile Errors

Type	Severity	Location
Compile Error	Minor	CapitalStake.sol, LockFunds.sol

Description:

1. Member "balanceOf" not found or not visible after argument-dependent lookup in contract `INsure` .

Example:

```
uint256 nsureBal = nsure.balanceOf(address(this));
```

2. Data location must be "calldata" for parameter in external function, but "memory" was given in the function `burnOuts` of the contract `LockFunds`

Example:

```
function burnOuts(address[] memory _burnUsers, uint256[] memory _amounts)  external
onlyOperator
```

Recommendation:

Consider Adding the definition of the function `balanceOf` in the contract `INsure` .

Consider declaring the parameter as "calldata" for parameter in external function of the contract `LockFunds` .

Alleviation:

The development team heeded our advice and resolved this issue in commit
6da44d95ff4f273fde637a759f842c98c3264880

CP-01: Missing Update of `lastRewardBlock`

Type	Severity	Location
Logical Issue	Major	ClaimPurchaseMint.sol

Description:

Function `mintPurchaseNsure` is missing update of `lastRewardBlock`

```
        function mintPurchaseNsure() internal {
            if (block.number <= lastRewardBlock) {
                return;
            }

            uint256 nsureReward = nsurePerBlock.mul(block.number.sub(lastRewardBlock));
            Nsure.mint(address(this), nsureReward);
        }
```

Recommendation:

Consider adding update of `lastRewardBlock` :

```
        function mintPurchaseNsure() internal {
            if (block.number <= lastRewardBlock) {
                return;
            }

            uint256 nsureReward = nsurePerBlock.mul(block.number.sub(lastRewardBlock));
            Nsure.mint(address(this), nsureReward);
            lastRewardBlock = block.number;
        }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit
6da44d95ff4f273fde637a759f842c98c3264880

TS-01: Missing Emit Events

Type	Severity	Location
Optimization	Informational	Treasury.sol, Surplus.sol, Product.sol, LockFunds.sol, CapitalConverter.sol

Description:

There serveral sensitive actions without emitting events.

Examples:

Function `setOperator()` in contract `CapitalConverter` ;

Function `updateBlockReward()` , `updateWithdrawPending()` , `add()` , `set()` in contract `CapitalStake` ;

Functions `setClaimDuration()` , `setSigner()` , `setDeadlineDuration()` , `updateBlockReward()` in contract `ClaimPurchaseMint` ;

Functions `setOperator()` , `addDivCurrency()` , `setDeadlineDuration()` , `setClaimDuration()` in contract `LockFunds` ;

Function `setOperator()` in contract `Product` ;

Function `setOperator()` in contract `Surplus` ;

Function `setOperator()` in contract `Treasury` .

Recommendation:

Consider emitting events for the above sensitive actions.

Example:

```
function setOperator(address _operator) external onlyOwner {
    operator = _operator;
    emit SetOperator(_operator);
}
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 6da44d95ff4f273fde637a759f842c98c3264880

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

❗ : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

❗✓ : Issue partially resolved. Not all instances of an issue was resolved.