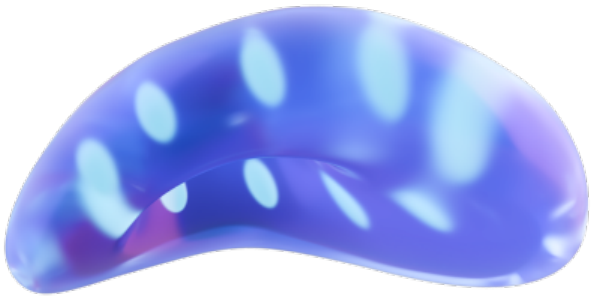
Smart Contract Security Audit

Dohrnii Token

2022-02-09



1. Introduction

Dohrnii aims to solve lack of financial education and lack of tools to support retail investors challenges by building a Financial Empowerment platform that has two main objectives - first, it aims at providing financial literacy to everyone through a gamified education feature; second, it looks to support investors by simplifying the creation of AI alerts, smart notifications and algorithm building.

With Dohrnii, decentralized projects will be able to raise and exchange capital cheap and fast. Users will be able to participate in a secure and compliant environment and to use assets that go way beyond the current ERC20 standard.

As requested by Dohrnii and as part of the vulnerability review and management process, Red4Sec has been asked to perform a security code audit in order to evaluate the security of IDO contract.

2. Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered neither "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project.

3. Scope

The scope of this evaluation includes the following smart contracts:

- **ERC20/DHN.sol**
  b40d5d49d42150d7b3b4e667e0ec2c0c299511d06fc42addb8a3acf5ed2e4337
- **VIP180/DHN.sol**
  a0102c5ff10ef9b22f89800a3886c18d6fd4831bff141af31fe4a4a986888d3d

And the subsequent revision up to the versions:

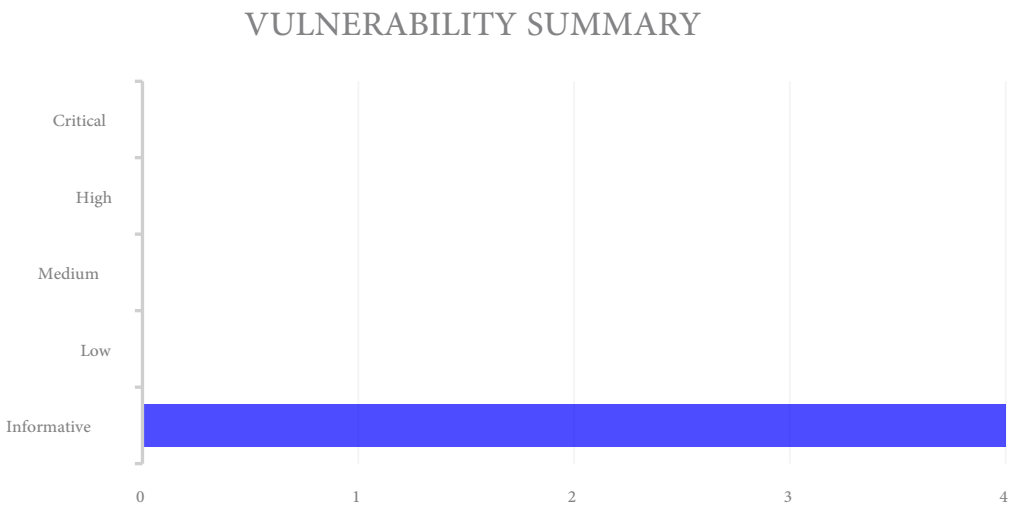
- **ERC20/DHN.sol**
  a833852b0e0a3af0e1ba171989208abe2909dd32f85ce7d92aec594ef1d6558a
- **VIP180/DHN.sol**
  848ba16930cd64ac0d1989b887697cc7e55d337b37448cea4ed29ebaca22c89a

4. Conclusions

To this date, 9th of February 2022, the general conclusion resulting from the conducted audit, is that **Dohrnii smart contracts are secure** and does not present any known vulnerabilities, although Red4Sec has found a few potential improvements, these do not pose any risk by themselves. We have classified such issues as informative only, but they will help Dohrnii to continue to improve the security and quality of its developments.

- The overall impression about the smart contract's security is optimal. However, Red4Sec has given some additional recommendations on how to continue improving and how to apply good practices.
- A few low impact issues were detected and classified only as informative, but they will continue to help Dohrnii improve the security and quality of its developments.
- The Dohrnii project has been developed using outdated versions, and in some cases, versions marked as obsolete, both from the compiler and from third-party libraries. This is an absolute discouraged practice and should be solved prior to the deployment.

Found vulnerabilities have been classified in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST).



Below we have a complete list of the issues detected by Red4Sec, presented and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
Id.	**Vulnerability**	**Risk**	**State**
DRN01	GAS Optimization	Informative	Partially Fixed
DRN02	Outdated Third-Party Libraries	Informative	Assumed
DRN03	Outdated Compiler Version	Informative	Fixed
DRN04	Use of Require statement without message	Informative	Fixed

5. Issues and Recommendations

DRN01 – GAS Optimization (Informative)

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that they execute faster or in order to save GAS.

On Ethereum blockchain, GAS is an execution fee which is used to compensate miners for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

These are some of the requirements that must be met to reduce GAS consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Explicit function visibility.
- Use of proper data types.
- Use hard-coded CONSTANT instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to arithmetical operations and comparisons.

Execution Cost

In order to implement a GAS optimization when making the deployment of the smart contract, to make the code easier to read and to minimize human errors, it is advisable to directly use solidity literals[1] with the full amount of *totalSupply* from the Smart Contract.

In this case, it is recommended to modify said value by `372_000_000 ether`, in this way, we will avoid performing unnecessary operations (cast and multiplication), thus saving GAS.

```
constructor(address wallet) public {
  require(wallet != address(0));
  _mint(wallet, uint256(372000000) * 1 ether);
}
```

[1] https://docs.soliditylang.org/en/latest/types.html#rational-literals

This behaviour has been observed in

- ERC20\DHN.sol:14
- VIP180\DHN.sol:15

Additionally, a possible optimization of GAS in the existing requirements has been identified. It is recommended to shorten the length of the *require* messages so that they do not occupy more than 32 bytes.

Also, in case you need to provide the user with more information, it is recommended to assign an error code and detail it in the official documentation of the project, for example in a section of common errors.

This behaviour has been observed in

- ERC20\ERC20.sol
- ERC20\SafeMath.sol
- VIP180\ERC20.sol
- VIP180\SafeMath.sol

Unnecessary Import

The DHN contract imports the *builtin for vechain*[2] library to access native functions of the blockchain, but apparently the token does not use these functionalities.

As can be seen in the following image, the import of the *Builtin* library is performed, however, it is not necessary since said library only contains interfaces and they are not used throughout the execution flow of the DHN contract.

```
contract DHN is ERC20("Dohrnii", "DHN") {
  using Builtin for DHN;
  /**
  * @param wallet Address of the wallet,
```

This behaviour has been observed in

- VIP180\DHN.sol:9

[2] https://github.com/vechain/thor-builtins

Storage Optimization

The use of the *immutable*[3] keyword is recommended to obtain less expensive executions, by having the same behaviour as a constant. However, by defining its value in the constructor we have a significant save of GAS.

This behaviour has been observed in

- ERC20\ERC20.sol [name, symbol]
- VIP180\ERC20.sol [name, symbol]

DRN02 – Outdated Third-Party Libraries (Informative)

The smart contracts analyzed inherit functionalities from open-zeppelin contracts that have been labelled obsolete and/or outdated, version 3.3; this does not imply a vulnerability by itself, because their logic does not present them, but it does imply that an update is not carried out by third party packages or libraries.

Currently the latest version of OpenZeppelin contracts is 4.4.2 therefore it would be convenient to include it as a reference instead of including the sources, in this way we will keep the development environment updated.

Additionally, these OpenZeppelin contracts are under the MIT license, which requires its license/copyright to be included within the code.

By using the original sources, in case the project resolves any vulnerability or bug in the code, you would obtain this update automatically. Consequently, avoiding inheriting known vulnerabilities.

Recommendations:

- Include third-party codes by package manager.
- Include in Dohrnii project any references/copyright to OpenZeppelin code since it is under MIT license.

[3] https://docs.soliditylang.org/en/v0.6.5/contracts.html#immutable

DRN03 – Outdated Compiler Version (Informative)

Solc frequently launches new versions of the compiler. Using an outdated version of the compiler can be problematic, especially if there are errors that have been made public or known vulnerabilities that affect this version.

We have detected that the audited contract uses the following version of Solidity pragma >=0.6.0 <0.8.0:

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;
```

The 0.6.X version of Solc is affected by different known bugs that have already been fixed in later versions. It is always a good policy to use the most up to date version of the pragma.

References

- https://github.com/ethereum/solidity/blob/develop/Changelog.md

DRN04 – Use of Require statement without reason message (Informative)

Throughout the audit, it was verified that the reason message is not specified in some require methods, in order to give the user more information, which consequently makes it more user friendly.

```
constructor(address wallet) public {
  require(wallet != address(0));
  _mint(wallet, uint256(372000000) * 1 ether);
}
```

This functionality is compatible since version 0.4.22 and the contract's pragma indicates >=0.6.0 <0.8.0, this will result in compatibility with this feature.

This behaviour has been observed in

- ERC20\DHN.sol:13
- VIP180\DHN.sol:14

Invest in Security, invest in your future