verichains

SECURITY AUDIT OF

THE DEHR SMART CONTRACTS

Public Report

Dec 09, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Dec 09, 2021. We would like to thank the DeHR for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the The DeHR Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About The DeHR Smart Contracts

DeHR is the revolutionary Web 3.0 promotes a direct and safe connection between both Employers and Job Seekers through creating more value, convenience, and credibility for all parties through a disruptive AI technology providing an end-to-end solution from selection to onboarding Talents.

The DeHR smart contracts include a ERC20 contract, a staking contract and a vesting contract.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of The DeHR Smart Contracts. It was conducted on commit 2aa1e3f1bff3c5ae773f32c6d5819cca52628994 from git repository link: *https://gitlab.com/dehr-v1/smart-contract/-/tree/develop*.

There are 3 files in our audit scope. They are DeHR.sol, Staking.sol, Vesting.sol files.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)

- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted on Dec 08, 2021 and a total effort of 6 working days was dedicated to identifying and documenting security issues in the code base of the The DeHR Smart Contracts.

2.2. Contract code

The The DeHR Smart Contracts was written in Solidity language, with the required version to be ^0.8.3. The source code was written based on OpenZeppelin's library.

The provided source codes consist of three contracts which inherit some contracts from OpenZeppelin.

2.2.1. DeHR token contract

The DeHR token is an ERC20 token contract. Besides the default ERC20 functions, the contract uses a BP contract to avoid Whale trading to control the price of tokens in the IDO time.

In addition, the contract has a logic flow that allows Operator mint new token with 2 phase. The totalSupply can be changed by this flow.

Table 2 lists some properties of the audited DeHR token contract (as of the report writing time).

PROPERTY	VALUE
Name	DeHR
Symbol	DHR
Decimals	18
Total Supply	1,000,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

Table 2. The DeHR token contract properties

2.2.2. Staking contract

Staking is a contract that allows investors to stake the DeHR token for profit.

2.2.3. Vestings contract

The DeHR team uses this contract to release the token. The owner can withdraw token follow the phases.

2.3. Findings

During the audit process, the audit team found some vulnerabilities in the given version of The DeHR Smart Contracts.

2.3.1. Vesting.sol - TimeLockedWallet - meaningless locking HIGH

This vesting contract is used to store locking for the owner, limited by time:

```
function withdraw(address _to) external onlyOwner onlyStarted {
        uint256 unlockedAmount;
        uint256 unlockedDueDate;

        if (lastWithdrawTime == 0) {
            // Cliff
            unlockedDueDate = startDate + cliffDurationMonth * MONTH_DURA…
  TION;
            unlockedAmount =
                (totalLockAmount * cliffReleasePercent) /
                BASE_PERCENT;
        } else {
            // Period
            unlockedAmount =
                (totalLockAmount * periodReleasePercent) /
                BASE_PERCENT;
            unlockedDueDate =
                lastWithdrawTime +
                periodDurationMonth *
                MONTH_DURATION;
        }

        require(unlockedDueDate < block.timestamp, "NOT MATURE");

        lastWithdrawTime = block.timestamp;
        dehrToken.transfer(_to, unlockedAmount);

        emit Withdraw(_to, unlockedAmount);
    }
```

Snippet 1. Vesting.sol The code of `withdraw` function

But the owner has another function to withdraw all tokens at any time:

```
function withdrawTokenByOwner(address _to) external onlyOwner {
        uint256 amount = dehrToken.balanceOf(address(this));
        dehrToken.transfer(_to, amount);
    }
```

Snippet 2. Vesting.sol The code of `withdrawTokenByOwner` function

UPDATES

- *2021-12-09*: This issue has been acknowledged and fixed by the DeHR team in commit 3cc905dcf51bf7163e84635be0697a8a3f1c4ce6. The withdrawTokenByOwner function was removed.

2.3.2. Vesting.sol - Wrong calculating time in getWithdrawDate function MEDIUM

The getWithdrawDate function calculates the date that the owner can withdraw your tokens. With current logic calculating, the function skips the cliff time. So the date does not correspond with time in the withdraw function.

RECOMMENDATION

We suggest changing the getWithdrawDate function like the below code:

```
function getWithdrawDate() external view returns (uint256) {
        if (lastWithdrawTime == 0) {
            return startDate + cliffDurationMonth * MONTH_DURATION;
        }
        else{
            return lastWithdrawTime + periodDurationMonth * MONTH_DURATIO…
  N;
        }

    }
```

Snippet 3. Vesting.sol Recommend fixing in `getWithdrawDate` function

UPDATES

- *2021-12-09*: This issue has been acknowledged and fixed by the DeHR team in commit 3cc905dcf51bf7163e84635be0697a8a3f1c4ce6.

2.3.3. Staking.sol - Unsafe using transfer, transferFrom method through IERC20 interface

LOW

There are some functions in the contract that use transfer, transferFrom methods to call functions from the token contract. With the DeHR token contract in the audit scope, it doesn't have any problems but the Staking contract allows changing the token contract. So we can't ensure that the transfer function of another token contract works exactly as expected.

For instance, the transfer function can return false with the function call failure instead of returning true or revert like ERC20 Oppenzepplin. With withdraw logic, the user doesn't receive anything while the acc.amount is set to zero.

```
84  function withdraw() external {
85          Account storage acc = stakeInfo[msg.sender];
86          require(acc.lockTo < block.timestamp, "LOCKED");
87
88          uint256 stakeAmount = acc.amount;
89          uint256 reward = (stakeAmount * rewardPercent) / 100;
90          uint256 totalReturned = stakeAmount + reward;
91
92          // Reset state
93          acc.amount = 0;
94          acc.lockTo = 0;
95
96          dehrToken.transfer(msg.sender, totalReturned);
97          emit Withdraw(msg.sender, stakeAmount, reward, totalReturned);
98      }
```

Snippet 4. Staking.sol Unsafe using `transfer` method in `withdraw` function

There are two functions that are using them. They are stake and withdraw functions.

RECOMMENDATION

We suggest using SafeERC20 library for IERC20 and changing all transfer, transferFrom method using in the contract to safeTransfer, safeTransferFrom which is declared in SafeERC20 library to ensure that there is no issue when transferring tokens.

UPDATES

- *2021-12-09*: This issue has been acknowledged and fixed by the DeHR team. The changeStakeToken function was removed in commit 3cc905dcf51bf7163e84635be0697a8a3f1c4ce6. From now, the contract only uses a token contract.

2.3.4. DeHR.sol - BPContract function INFORMATIVE

Since we do not control the logic of the bpContract, there is no guarantee that bpContract will not contain any security related issues. With the current context, in case the bpContract is compromised, there is not yet a way to exploit the The DeHR Smart Contracts, but we still note that here as a warning for avoiding any related issue in the future.

By the way, if having any issue, the bpContract function can be easily disabled anytime by the contract owner using the setBPEnabled function. In addition, bpContract is only used in a short time in token public sale IDO then the contract owner will disable it forever by the setBPDisableForever function.

APPENDIX

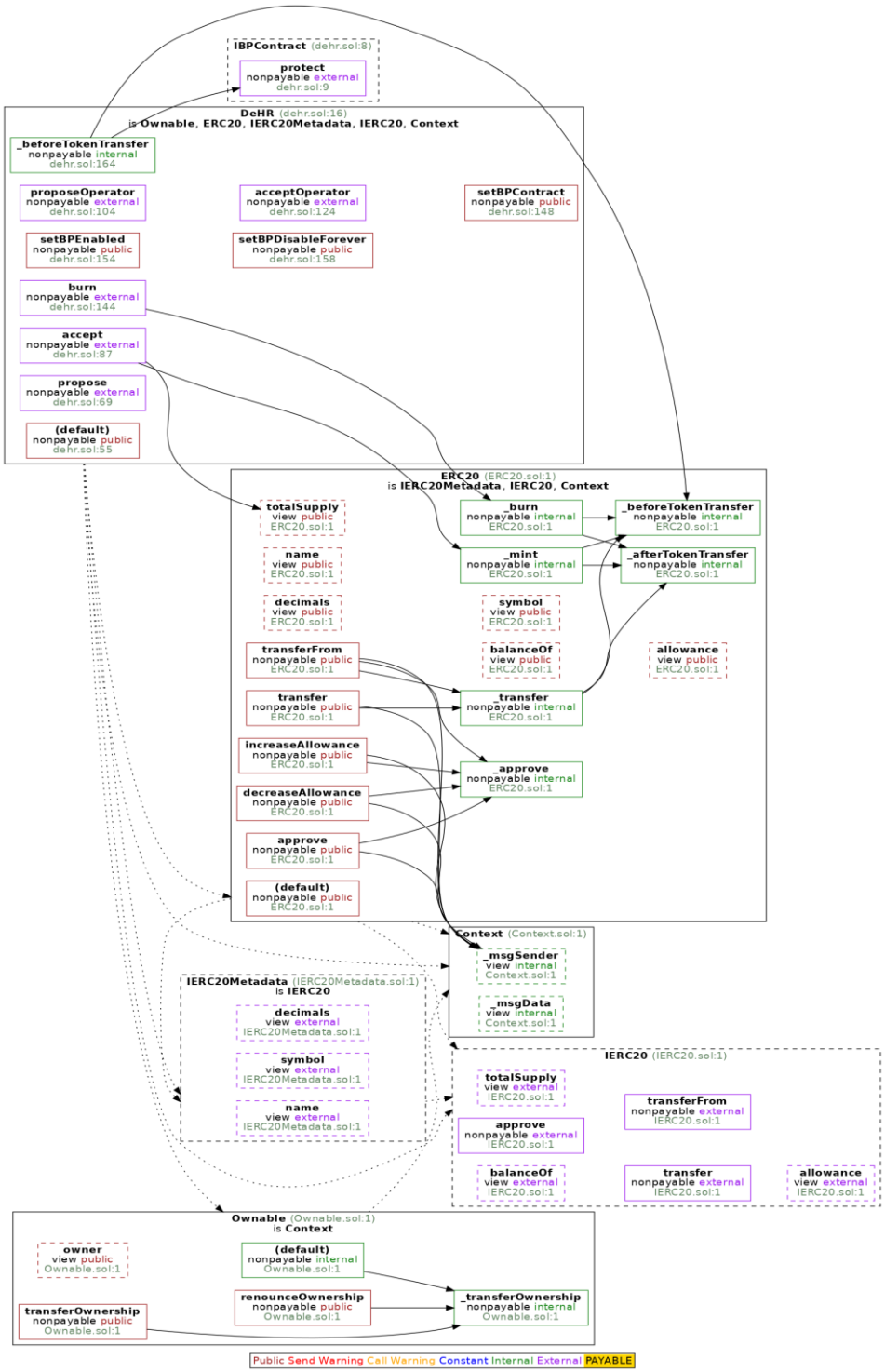

Image 1. DeHR token smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*2021-12-08*	Private Report	Verichains Lab
1.1	*2021-12-09*	Public Report	Verichains Lab

Table 3. Report versions history