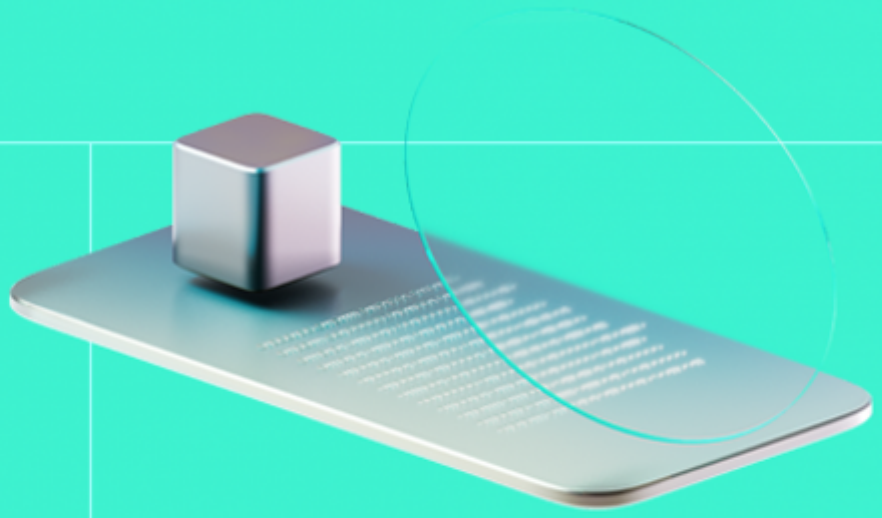
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Dropnest

Date: 04/07/2024

We express our gratitude to the Dropnest team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The `DropnestStaking` protocol is a smart contract system for EVM chains. It is designed to manage deposits for the Dropnest protocol and record the deposits via emitting events into the blockchain. Deposited tokens are then sent to external protocols for farming.

Document

Name	Smart Contract Code Review and Security Analysis Report for Dropnest
Audited By	David Camps Novi, Viktor Lavrenenko
Approved By	Przemyslaw Swiatowiec
Website	https://www.dropnest.xyz/
Changelog	27/06/2024 - Preliminary Report; 04/07/2024 - Final Report
Platform	Ethereum, Base, Optimism, Arbitrum, Linea, Blast, Polygon
Language	Solidity
Tags	Staking, Farming
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Nest-Layer/dropnest-protocol-contracts/
Commit	60be644381df46440497434ab1900258b17e05fc

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	0
Low	1

Vulnerability	Status
F-2024-4044 - Use of transfer() to Send Native Assets may Revert	Fixed

Documentation quality

- Functional requirements are limited.
  - Basic system description is provided.
  - System roles are explained.
- Technical description is complete.
  - NatSpec is provided.
  - Run instructions are present.

Code quality

- The development environment is configured.
- Inefficient gas model: F-2024-4045, F-2024-4050.
- Best practices are followed.

Test coverage

Code coverage of the project is **94.4%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is extensively tested.

Table of Contents

System Overview

The `DropnestStaking` protocol is a smart contract system for EVM chains. It is designed to manage deposits for the Dropnest protocol and record the deposits via emitting events into the blockchain. Deposited tokens are then sent to external protocols for farming.

- **Staking**: Users can stake their ETH or predetermined ERC20 tokens on a specific protocol. The staked tokens are transferred to the farmer address of the protocol.
- **Multiple Staking**: Users can also stake their ETH or ERC20 tokens on multiple protocols at once. The total amount of tokens staked should be equal to the sum of the individual amounts staked on each protocol.

Privileged roles

- The owner of the contract can
  - Add new protocols or update the farmer address for existing ones.
  - Pause and unpause the contract.
  - Set the status of a protocol from active to inactive, and vice-versa.

Risks

- The audited contract `DropnestStaking` is highly centralized, introducing single points of failure and control. This centralization can lead to vulnerabilities in decision-making and operational processes, making the system more susceptible to targeted attacks or manipulation. The centralization is expressed by the reliance on the protocol owners for:
  - Tracking deposits from users into the protocol, and subsequent calculation and management of staking rewards and users' funds, off-chain.
  - Managing interactions with the farming protocols used by the system, off-chain.
  - Pausing and unpausing the contract.
  - Adding and removing supported tokens.
  - Managing the farming protocols' statuses and addresses.
- The digital contract architecture relies on administrative keys for critical operations. Centralized control over these keys presents a significant security risk, as compromise or misuse can lead to unauthorized actions or loss of funds. Such operations are:
  - Pause and unpause the contract.
  - Add and remove supported tokens.
  - Manage the farming protocols' statuses and addresses.
- The implemented `farmAddresses` and `supportedTokens` logic highly depends on external contracts not covered by the audit. This reliance introduces risks if these external contracts are compromised or contain vulnerabilities, affecting the audited project's integrity. More details about such contracts and their usage can be found in the project `README`.
- The `DropnestStaking` contract contains the staking functionality of the system, through which users' deposits are tracked using the events Deposited and ERC20Deposited. The amounts are tracked off-chain, where additional calculations take place in order to complete the staking functionality (balance tracking, rewards, etc.).
- Although the team does not currently plan to support fee-on-transfer tokens, if such kind of tokens (or any other kind of token were the input amount and received amount could be different) were introduced in the system, the off-chain part of the protocol should be adapted. Otherwise, there could be mismatches between the tracked amount emitted by the events and the actual transferred amount, resulting in wrong calculations and balance tracking in the system. The project `README` provides further insights on this topic, where the development team provided technical explanations and how they are resilient to these kind of situations.

Findings

Vulnerability Details

F-2024-4044 - Use of transfer() to Send Native Assets may Revert - Low

Description:

The contract uses built-in `transfer()` function for transferring native tokens.

The `transfer()` function was commonly used in earlier versions of Solidity for its simplicity and automatic reentrancy protection. However, it was identified as potentially problematic due to its fixed gas limit of 2300.

The usage of `transfer()` function can lead to unintended function call revert when the receiving contract's `receive()` or `fallback()` functions require more than 2300 Gas for processing.

```
function _stake(uint256 protocolId, uint256 protocolAmount) private nonZeroAmoun
    address to = farmAddresses[protocolId];
    if (to == address(0)) {
        revert DropnestStaking_ProtocolDoesNotExist();
    }
    if (!protocolStatus[protocolId]) {
        revert DropnestStaking_ProtocolInactive(protocolId);
    }
    emit Deposited(protocolId, msg.sender, to, protocolAmount);
    payable(to).transfer(protocolAmount);
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to use built-in `call()` function instead of `transfer()` to transfer native assets. This method does not impose a gas limit, it provides greater flexibility and compatibility with contracts having more complex business logic upon receiving the native tokens. When working with then `call()` function ensure that its execution is successful by checking the returned boolean value. It is also recommended to fallow the Check-Effects-Interactions (CEI) pattern in every case to prevent reentrancy issues.

Resolution: Fixed in commit ID **84b8746**: the project updated to built-in `call()` function instead of `transfer()` to transfer native assets.

Observation Details

F-2024-4045 - Missing Check Results in Waste of Gas - Info

Description: The function `removeSupportedToken()` does not check whether the token to remove was already added into the protocol list. Therefore, the whole list of tokens may be iterated to fetch the token, and waste Gas unnecessarily, if the token is not in the `tokenList` array.

```
function removeSupportedToken(address token) external onlyOwner {
    supportedTokens[token] = false;
    uint256 length = tokenList.length;
    for (uint256 i = 0; i < length; i++) {
        if (tokenList[i] == token) {
            tokenList[i] = tokenList[length - 1];
            tokenList.pop();
            break;
        }
    }
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Recommendations

Remediation: Consider checking if the token to remove was added into `supportedTokens`.

Resolution: Fixed in commit ID **84b8746**: the modifier `allowedToken` was added into the `removeSupportedTokens()` method.

F-2024-4048 - Single-Step Ownership Transfer Introduces Risks of Losing Ownership - Info

Description:

The project's contract `DropnestStaking` inherits OpenZeppelin's `Ownable` functionality to handle the transfer of the contract's ownership. However, the pattern used by `Ownable` is not considered secure enough.

Ownable's ownership can mistakenly be transferred via `transferOwnership()` to an address that cannot handle it.

```
abstract contract Ownable is Context {
    address private _owner;

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        if (newOwner == address(0)) {
            revert OwnableInvalidOwner(address(0));
        }
        _transferOwnership(newOwner);
    }
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Recommendations

Remediation: Consider using `Ownable2Step` instead of `Ownable` from OpenZeppelin Contracts to enhance the security of your contract ownership

management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Resolution: The finding was resolved in commit ID **84b8746:** the `DropnestStaking` contract was modified and now using `Ownable2Step.sol` instead of `Ownable.sol`.

F-2024-4049 - Minimum Staking Amount is not Compliant with Documentation - Info

Description:

The project's documentation found in ./README.md defines a minimum deposit amount required for staking:

> **Staking**: Users can stake their ETH or predetermined ERC20 tokens on a specific protocol. The staked tokens are transferred to the farmer address of the protocol. The amount of tokens to be staked should be equal to or more than the minimum deposit amount.

However, no minimum deposit amount check is introduced in staking methods.

```
function stakeERC20(uint256 protocolId, address token, uint256 amount) external v
    _stakeERC20(protocolId, token, amount);
}

function _stake(uint256 protocolId, uint256 protocolAmount) private nonZeroAmount
    address to = farmAddresses[protocolId];
    if (to == address(0)) {
        revert DropnestStaking_ProtocolDoesNotExist();
    }
    if (!protocolStatus[protocolId]) {
        revert DropnestStaking_ProtocolInactive(protocolId);
    }
    emit Deposited(protocolId, msg.sender, to, protocolAmount);
    payable(to).transfer(protocolAmount);//@audit-ok use call instead of transfe
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Recommendations

Remediation:

It is recommended to either introduce the aforementioned checks in the code or to update the documentation so that the code is compliant with the system requirements.

Resolution: Fixed in commit ID **84b8746**: the README was updated, removing the aforementioned minimum staking requirement.

F-2024-4050 - Zero Address and Duplicated Entries can be Added into tokenList and protocols Arrays - Info

Description:

The Zero address (`0x0`) can be added into the `tokenList` in both the `constructor()` and `addSupportedToken()` methods.

Additionally, there is no check to enforce that the same address is not added twice in `tokenList` or `protocols` arrays.

```
constructor(address[] memory _supportedTokens, string[] memory _protocols, addres
    if (_protocols.length != _addresses.length) {
        revert DropnestStaking_ArraysLengthMismatch();
    }
    for (uint256 i = 0; i < _protocols.length; i++) {
        _addProtocol(_protocols[i], _addresses[i]);
    }
    for (uint256 i = 0; i < _supportedTokens.length; i++) {
        supportedTokens[_supportedTokens[i]] = true;
        tokenList.push(_supportedTokens[i]);
    }
}

function addSupportedToken(address token) external onlyOwner {
    if (supportedTokens[token]) {
        revert DropnestStaking_TokenAlreadySupported(token);
    }
    supportedTokens[token] = true;
    tokenList.push(token);
}

function _addProtocol(string memory protocolName, address to) private {
    if (to == address(0)) {
        revert DropnestStaking_ZeroAddressProvided();
    }
    protocolCounter++;
    protocols.push(protocolName);
    farmAddresses[protocolCounter] = to;
    protocolStatus[protocolCounter] = true;
    emit ProtocolAdded(protocolCounter, protocolName, to);
}
```

As a consequence, an extra amount of Gas will be wasted in order to iterate through larger arrays in both `removeSupportedTokens()` and `addOrUpdateProtocol()` methods.

```
function removeSupportedToken(address token) external onlyOwner {
    supportedTokens[token] = false;
    uint256 length = tokenList.length;
    for (uint256 i = 0; i < length; i++) {
        if (tokenList[i] == token) {
            tokenList[i] = tokenList[length - 1];
            tokenList.pop();
            break;
        }
    }
}

function addOrUpdateProtocol(string memory protocolName, address farmerAddress) {
    if (farmerAddress == address(0)) {
        revert DropnestStaking_ZeroAddressProvided();
    }
    uint256 length = protocols.length;
    for (uint256 i = 0; i < length; i++) {
        if (keccak256(abi.encodePacked(protocols[i])) == keccak256(abi.encodePacl
            uint256 id = i + 1;
            farmAddresses[id] = farmerAddress;
            emit ProtocolUpdated(id, protocolName, farmerAddress);
            return;
        }
    }
    _addProtocol(protocolName, farmerAddress);
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Mitigated

Recommendations

Remediation: Consider adding a check to prevent from adding duplicates and the zero address in the reported functions.

Resolution: Partially fixed in commit ID **84b8746**:

- Zero address checks were added into `addSupportedToken()` to avoid including `addreess(0)` into the `tokenList`.
- The `constructor()` was updated and now calls `addSupportedToken()` in order to avoid duplicated entries and the zero address.

- No measures were taken to avoid protocol duplicates. The development team accepted the finding and the risks arising from it.

F-2024-4051 - Missing Events for Key Updates - Info

Description:

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying the on-chain contract state for such changes is not considered practical for most users/usages.

The following functions do not emit any events: `addSupportedToken()`, `removeSupportedToken()`.

The absence of events in these functions means that there is no on-chain traceability or transparency when the data is updated.

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Recommendations

Remediation:

To enhance transparency and traceability, it is recommended to emit events in key functions. This will allow users and external services to monitor and react to changes. Ensure that every critical action, especially those modifying contract states or handling funds, emits an event.

Resolution:

Fixed in commit ID **84b8746**: events were added to the reported functions.

F-2024-4057 - Redundant Token Transfer Results in Waste of Gas - Info

Description:

The function `_stakeERC20` performs two token transfers: the first one to transfer tokens from the `msg.sender` to the main contract, and the second to transfer the same tokens to the `farmAddress`.

Since there is no reason to perform the token transfer from the sender to the `farmAddress` in a two-step process, the result of this design pattern is only a waste of gas. Additionally, if working with fee-on-transfer tokens, it would result in paying twice the fee.

```
function _stakeERC20(uint256 protocolId, address tokenAddress, uint256 amount) p
    if (tokenAddress == address(0)) {
        revert DropnestStaking_ZeroAddressProvided();
    }

    if (!protocolStatus[protocolId]) {
        revert DropnestStaking_ProtocolInactive(protocolId);
    }
    address to = farmAddresses[protocolId];
    if (to == address(0)) {
        revert DropnestStaking_ProtocolDoesNotExist();
    }
    IERC20(tokenAddress).safeTransferFrom(msg.sender, address(this), amount);
    IERC20(tokenAddress).safeTransfer(to, amount);
    emit ERC20Deposited(protocolId, tokenAddress, msg.sender, to, amount);
}
```

Assets:

- DropnestStaking.sol [https://github.com/Nest-Layer/dropnest-protocol-contracts/]

Status: Fixed

Recommendations

Remediation: Perform a single transfer from the `msg.sender` to the `farmAddress`.

Resolution: Fixed in commit ID **84b8746**: the code was updated to perform a single token transfer.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

[hknio/severity-formula]

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/Nest-Layer/dropnest-protocol-contracts/
Commit	60be644381df46440497434ab1900258b17e05fc
Remediation commit	84b87464abb914394f1099916841864189107472
Whitepaper	N/A
Requirements	./README
Technical Requirements	./README

Contracts in Scope
./src/DropnestStaking.sol