



N i g h t c l u b F i n a n c e

Audit Report

Table of Contents

1. Audit Summary

1.1 Vulnerability Summary

1.2 Audit Scope

1.3 Source Code

2. Disclaimer

3. Global Overview

4. Vulnerabilities Findings

5. Contract Privileges

5.1 Maximum Fee Limit Check

5.2 Contract Pausability Check

5.3 Max Transaction Amount Check

5.4 Exclude From Fees Check

5.5 Ability to Mint Check

5.6 Ability to Blacklist Check

6. Contract Snapshot

7. Website Review

8. Certificate of Proof

1 | Audit Summary

Project Name	NightClub PEG (BEER) - (https://nightclubfinance.xyz/)
Platform	CORE DAO
Language	Solidity (0.6.0 <0.8.0)
Contract Address	0x0e0E80686120f7AC316d52aB9E75b40a52e0bEA9
Delivery Date	May 13, 2024
Audit Msolodology	Static Analysis, Manual Review
Key Components	NightClub

1.1 | Vulnerability Summary

Vulnerability Summary	Total	🕒 Pending	🕒 Declined	🕒 Acknowledged	🕒 Partially Resolved	🕒 Resolved
🔴 Critical	0	0	0	0	0	0
🟠 Major	0	0	0	0	0	0
🟡 Medium	0	0	0	0	0	0
🟢 Minor	0	0	0	0	0	0
🔵 Informational	0	0	0	0	0	0
🟢 Discussion	0	0	0	0	0	0

1.2 | Audit Scope

External Dependencies

This audit focused on identifying security flaws in code and the design of EscrowDapp Contract. It was conducted on the source code provided by the EscrowDapp team. The following files were made available in the course of the review:

Privileged Functions

The contract contains the following privileged functions that are restricted by role with the modifier. Since the contract is the owner cannot modify the contract configurations and address attributes.

Audit methodology

Dependencies

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.

1.3 | Source Code

```
pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by 'account'.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves 'amount' tokens from the caller's account to 'recipient'.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a (Transfer) event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that 'spender' will be
     * allowed to spend on behalf of 'owner' through (transferFrom). This is
     * zero by default.
     *
     * This value changes when (approve) or (transferFrom) are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets 'amount' as the allowance of 'spender' over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an (Approval) event.
     */
}
```

```
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for [name] and [symbol], initializes [decimals] with
     * a default value of 18.
     *
     * To select a different value for [decimals], use (_setupDecimals).
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) public {
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual returns (string memory) {
        return _symbol;
    }
}
```

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}

/** @dev Creates 'amount' tokens and assigns them to 'account', increasing
 * the total supply.
 *
 * Emits a (Transfer) event with 'from' set to the zero address.
 *
 * Requirements:
 *
 * - 'to' cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}

/**
 * @dev Destroys 'amount' tokens from 'account', reducing the
 * total supply.
 *
 * Emits a (Transfer) event with 'to' set to the zero address.
 *
 * Requirements:
 *
 * - 'account' cannot be the zero address.
 * - 'account' must have at least 'amount' tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}
```

```
contract Operator is Context, Ownable {
    address private _operator;

    event OperatorTransferred(address indexed previousOperator, address indexed newOperator);

    constructor() internal {
        _operator = msg.sender();
        emit OperatorTransferred(address(0), _operator);
    }

    function operator() public view returns (address) {
        return _operator;
    }

    modifier onlyOperator() {
        require(_operator == msg.sender, "operator: caller is not the operator");
        _;
    }

    function idOperator() public view returns (bool) {
        return msg.sender() == _operator;
    }

    function transferOperator(address newOperator) public onlyOwner {
        _transferOperator(newOperator);
    }

    function _transferOperator(address newOperator_) internal {
        require(newOperator_ != address(0), "operator: zero address given for new operator");
        emit OperatorTransferred(address(0), newOperator_);
        _operator = newOperator_;
    }
}

// File contracts/interfaces/IOracle.sol

pragma solidity 0.6.12;

interface IOracle {
    function update() external;

    function consult(address _token, uint256 _amountIn) external view returns (uint144 amountOut);

    function swap(address _token, uint256 _amountIn) external view returns (uint144 amountOut);
}

// File contracts/NightClub.sol

contract NightClub is ERC20Burnable, Operator {
    using SafeMath for uint8;
    using SafeMath for uint256;

    // Initial distribution for the first 24h genesis pools
    uint256 public constant INITIAL_GENESIS_POOL_DISTRIBUTION = 20000 ether;
    // Distribution for LP wallet
    uint256 public constant INITIAL_AIRDROP_WALLET_DISTRIBUTION = 5000 ether;

    // Have the rewards been distributed to the pools
    bool public rewardPoolDistributed = false;
}
```

2 | 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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

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 FusionTech prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 FusionTech to perform a security assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. FusionTech's position is that each company and individual are responsible for their own due diligence and continuous security.

FusionTech'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.

The assessment services provided by FusionTech is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

3 | Global Overview

FusionTech uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

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.

4 | Vulnerabilities Findings

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID SWC-	Description	Status
100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected soler Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT	Passed
SWC-107	Instruction Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected solar balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length	Passed
SWC-134	Arguments Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

Recommendation

Private keys belonging to the employer and/or contract owner should be stored properly. The initial asset allocation procedure should involve consultation with the community.

5 | Contract Privileges

5.1 | Maximum Fee Limit Check

FusionTech tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

ERROR	Description
Code	Centralization: Operator Fee
CEN-01	Manipulation

TYPE OF FEE	DESCRIPTION
Transacted without Tax	0%
Max buy fee	0%
Max sell fee	0%

5.2 | Contract Pausability Check

FusionTech tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	● Owner cannot pause the contract

5.3 | Max Transaction Amount Check

FusionTech tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

PRIVILEGE CHECK	DESCRIPTION
Can owner set max tx amount?	● Owner cannot set max transaction amount

5.4 | Exclude From Fees Check

FusionTech tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	● Owner can exclude from fees

FUNCTION

```
function excludeFromFees(address account, bool excluded) external onlyOwner{
    _isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded); }
```

5.5 | Ability to Mint Check

FusionTech tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non- xed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A msolod to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	● Owner cannot mint new tokens

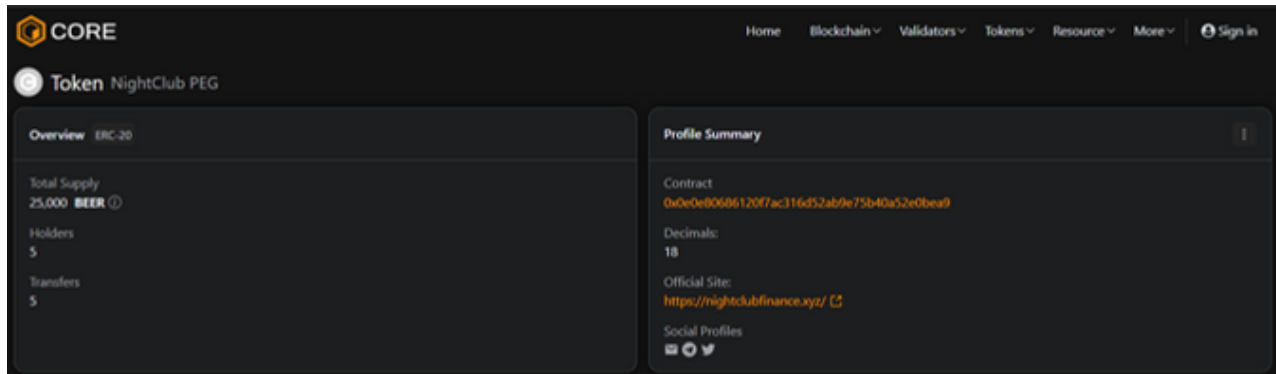
5.6 | Ability to Blacklist Check

FusionTech tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting msolods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This msolod can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	● Owner cannot blacklist an wallet/address

6 | Contract Snapshot

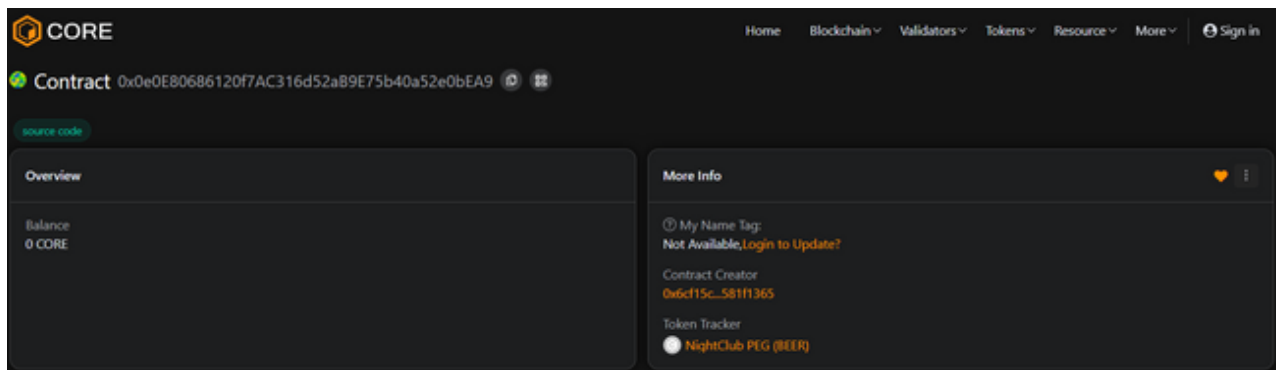


The screenshot shows the CORE Token interface for NightClub PEG. The top navigation bar includes links for Home, Blockchain, Validators, Tokens, Resource, and More, along with a Sign in button. The main header displays the token name 'Token NightClub PEG'. The left sidebar shows an Overview section with the following details:

- Total Supply: 25,000 BEER
- Holders: 5
- Transfers: 5

The right sidebar contains a Profile Summary section with the following information:

- Contract: 0x0e0E80686120f7ac316d52ab9e75b40a52e0bea9
- Decimals: 18
- Official Site: <https://nightclubfinance.xyz/>
- Social Profiles: [Twitter](#), [Telegram](#), [Facebook](#)

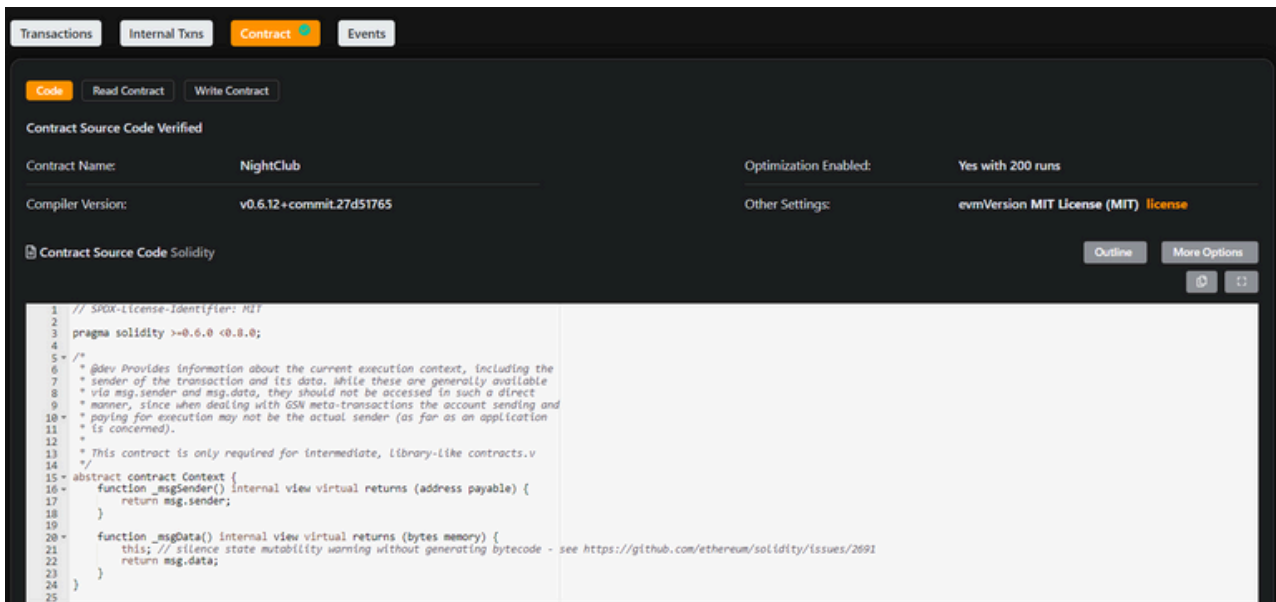


The screenshot shows the CORE Contract interface for NightClub PEG. The top navigation bar is identical to the previous screenshot. The main header displays the contract name 'Contract 0x0e0E80686120f7AC316d52aB9E75b40a52e0bEA9'. The left sidebar shows an Overview section with the following details:

- Balance: 0 CORE

The right sidebar contains a More Info section with the following information:

- My Name Tag: Not Available, [Login to Update?](#)
- Contract Creator: 0x6cf15c581f1365
- Token Tracker: [NightClub PEG \(BEER\)](#)



The screenshot shows the CORE Contract Source Code page for NightClub PEG. The top navigation bar includes links for Transactions, Internal Txns, Contract, and Events. The main header displays the contract name 'Contract NightClub PEG'. The left sidebar shows a Code section with the following details:

- Contract Name: NightClub
- Compiler Version: v0.6.12+commit.27d51765
- Optimization Enabled: Yes with 200 runs
- Other Settings: evmVersion MIT License (MIT) [license](#)

The right sidebar contains a Contract Source Code Solidity section with the following code:

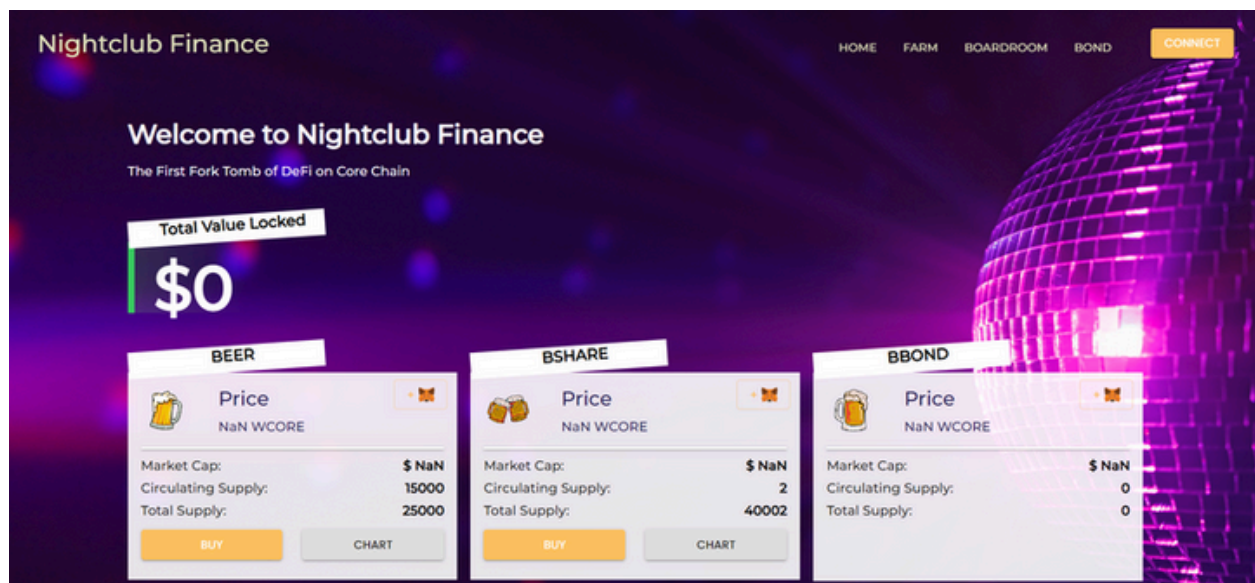
```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity >=0.6.0 <0.8.0;
4
5 /*
6  * @dev Provides information about the current execution context, including the
7  * sender of the transaction and its data. While these are generally available
8  * via msg.sender and msg.data, they should not be accessed in such a direct
9  * manner, since when dealing with GSN meta-transactions the account sending and
10  * paying for execution may not be the actual sender (as far as an application
11  * is concerned).
12  *
13  * This contract is only required for intermediate, library-like contracts.
14  */
15 abstract contract Context {
16     function _msgSender() internal view virtual returns (address payable) {
17         return msg.sender;
18     }
19
20     function _msgData() internal view virtual returns (bytes memory) {
21         this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
22         return msg.data;
23     }
24 }
25

```

7 | Website Review

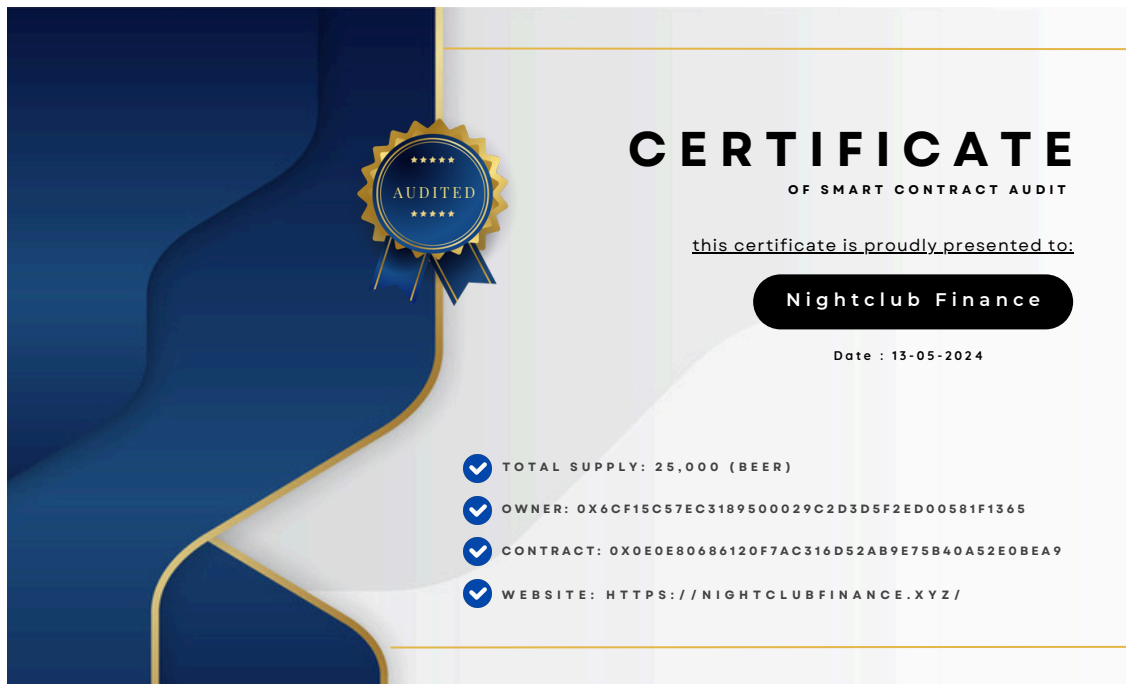
FusionTech checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

8 | Certificate of Proof

● Smart Contract Audited



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

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

Volatile Code

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

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather 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 requirements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexa-decimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

About

Founded in 2022 by leading academics in the field of Computer Science, FusionTech is going to be a leading blockchain security company that serves to verify the security and correctness of smart contracts KYC and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.



<https://fusiontech.network>

<https://twitter.com/fusiontechh>

<https://t.me/fusiontechchat>

<https://github.com/fusiontechofficial>