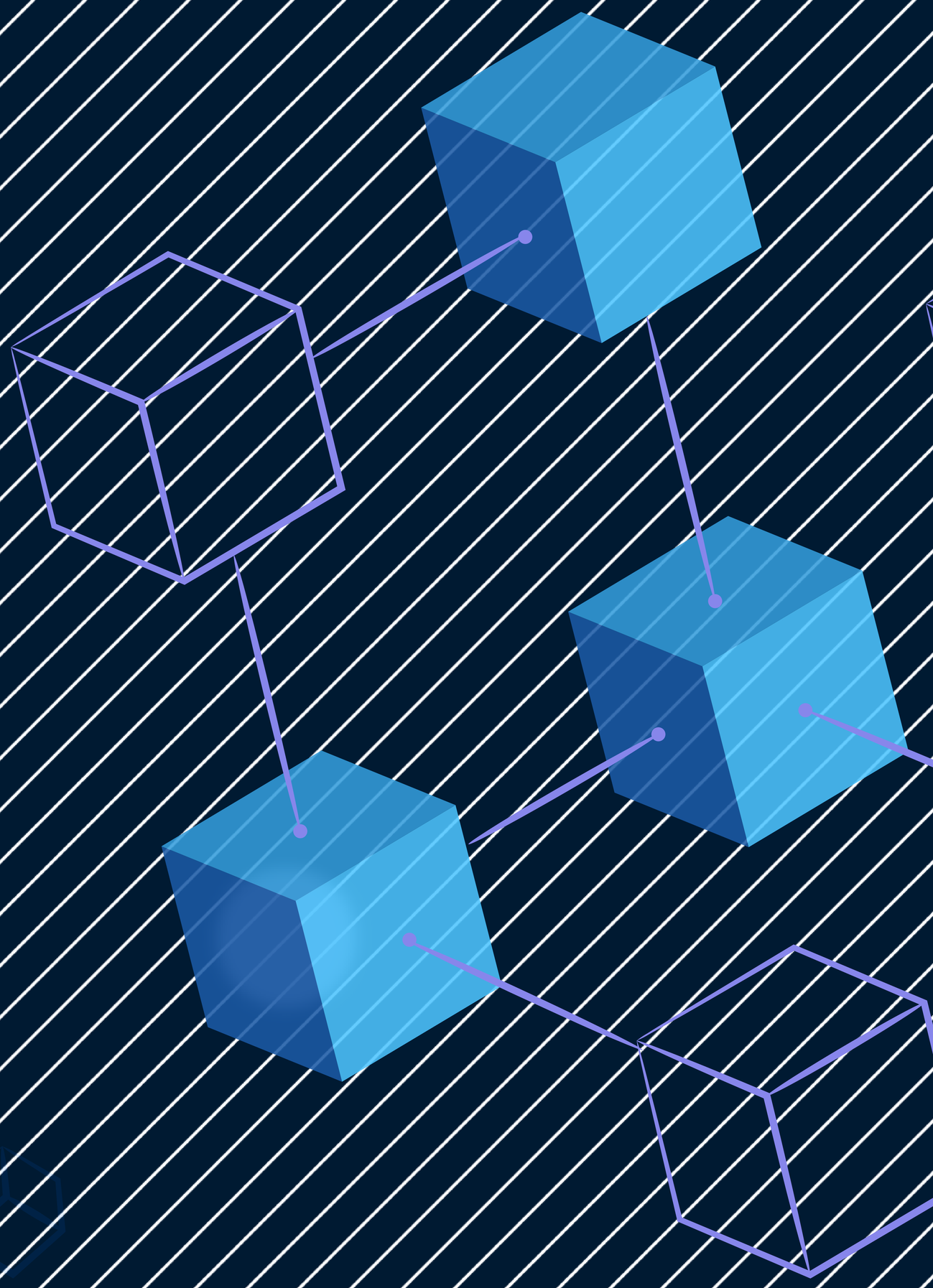
QuillAudits

Audit Report
January, 2022



For

Contents

Contents

Contents

Contents

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Elemon smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	2	11	17	0
Closed	3	0	3	6

Introduction

During the period of **November 4, 2021, to December 15, 2021** - QuillAudits Team performed a security audit for **Elemon** smart contracts.

The code for the audit was taken from the following official repo of Elemon: https://github.com/elemongame/contracts

Note	Date	Commit hash
Version 1	November	c760bf174c09d2b19bb07652c8e2d532b3b4b494
Version 2	December	21f17395ec22bd3758433d462f1cd024f95636f2

Issues Found

A. Contract – ElemonIDO

High severity issues

A.1 Usage Of transfer Instead Of safeTransfer

```
Line 49:
        require(_whiteLists[_msgSender()], "You are not in whitelist");
        require(_userBoughts[_msgSender()] == 0, "You have registerd before");

        _busdToken.transferFrom(_msgSender(), _idoRecepientAddress, PAID_BUSD);
        _userBoughts[_msgSender()] = ELMON_ALLOCATION;
        _totalBought += ELMON_ALLOCATION;
```

```
Line 79:
        require(tokenQuantity > 0, "Token quantity is not enough to claim");
        _elmonToken.transfer(_msgSender(), tokenQuantity);

        emit Claimed(_msgSender(), tokenQuantity);
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Closed

Fixed

The Elemon team has fixed the issue by adding require statements to ensure that the transactions were executed successfully.

Medium severity issues

A.2 For Loop Over Dynamic Array

```
Line 69:
for (uint256 index = startIndex;index < _claimableBlocks.length;
            index++) {
            uint256 claimBlock = _claimableBlocks[index];
            if (block.number >= claimBlock) {
                tokenQuantity +=
                  (userBought * _claimablePercents[claimBlock]) /100;
                _claimCounts[_msgSender()]++;
            } else { break; }
        }
```

```
Line 134:
function setClaimablePercents(uint256[] memory blocks,
        uint256[] memory percents) external onlyOwner {
            require(blocks.length > 0, "Empty input");
            require(blocks.length == percents.length, "Empty input");
            for (uint256 index = 0; index < blocks.length; index++) {
                _claimablePercents[blocks[index]] = percents[index];
            }
        }
```

```
Line 142:
function addToWhiteList(address[] memory accounts) external onlyOwner {
        require(accounts.length > 0, "Invalid input");
        for (uint256 index = 0; index < accounts.length; index++) {
            _whiteLists[accounts[index]] = true;
        }
    }
```

```
Line 149:
function removeFromWhiteList(address[] memory accounts) external
onlyOwner {
        require(accounts.length > 0, "Invalid input");
        for (uint256 index = 0; index < accounts.length; index++) {
            _whiteLists[accounts[index]] = false;
        }
    }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

Low severity issues

A.3 Missing address verification

```
Line 30:
constructor(address busdAddress, address elmonAddress,
        address idoRecepientAddress, uint256 startBlock, uint256 endBlock) {
        _busdToken = IERC20(busdAddress);
        _elmonToken = IERC20(elmonAddress);
        _idoRecepientAddress = idoRecepientAddress;
        _startBlock = startBlock;
   _endBlock = endBlock;
        //THIS PROPERTIES WILL BE SET WHEN DEPLOYING CONTRACT
        //_claimableBlocks = [];
        //_claimablePercents[] = 50;
        //_claimablePercents[] = 25;
        //_claimablePercents[] = 25;
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed
The Elemon team has fixed the issue by adding require conditions to verify the addresses coming from the arguments.

A.4 Missing value verification

```
Line 30:
constructor(address busdAddress, address elmonAddress,
        address idoRecepientAddress, uint256 startBlock, uint256 endBlock) {
        _busdToken = IERC20(busdAddress);
        _elmonToken = IERC20(elmonAddress);
        _idoRecepientAddress = idoRecepientAddress;
        _startBlock = startBlock;
  _endBlock = endBlock;

        //THIS PROPERTIES WILL BE SET WHEN DEPLOYING CONTRACT
        //_claimableBlocks = [];
        //_claimablePercents[] = 50;
        //_claimablePercents[] = 25;
        //_claimablePercents[] = 25;
    }
```

```
Line 134:
function setClaimablePercents(uint256[] memory blocks,
            uint256[] memory percents) external onlyOwner {
            require(blocks.length > 0, "Empty input");
            require(blocks.length == percents.length, "Empty input");
            for (uint256 index = 0; index < blocks.length; index++) {
                  _claimablePercents[blocks[index]] = percents[index];
            }
        }
```

Description
Certain functions lack a safety check in the value, the startBlock and endBlock arguments should be higher than block.number, also the percent array’s values should sum to 100, otherwise, the contract's logic may get hurt if an invalid value has been set.

Remediation
It’s recommended to add required statements to verify that the values that are provided from the arguments are valid.

Status: Closed

Fixed
The Elemon team has fixed the issue by inserting the values statically.

A.5 Renounce Ownership

```
Line 8:
contract ElemonIDO is Ownable, ReentrancyGuard {
```

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

B. Contract – ElemonInfo

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

B.1 Renounce Ownership

```
Line 6:
contract ElemonInfo is Ownable {
```

Description
Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation
It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

C. Contract – ElemonMarketplace

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

C.1 Renounce Ownership

```
Line 11:
contract ElemonMarketplace is Ownable, ReentrancyGuard, IERC721Receiver {
```

Description
Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation
It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

D. Contract – ElemonNFT

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

D.1 Centralization Risks

Description

The role owner has the authority to :
- burn the tokens of users

The role operator has the authority to :
- mint tokens for any users

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:
- Time-lock with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

Status: Closed

Informational issues

D.2 Emit events after critical operations

Remediation
Add and emit event for critical operations like the setOperator() function which would be a best practice for off chain monitoring.

Status: Closed

D.3 Renaming Operators

Remediation
It is advised to rename _operators to some other name such as _minters as its naming clashes with _operatorApprovals variable and the operator parameter in functions setApprovalForAll() and isApprovedForAll(). This can be confusing, leading to reduced code readability and maintainability.

Status: Closed

D.4 Public Visibility in TotalSupply

Remediation
It is advised to specify the visibility type for the _totalsupply state variable(suggested visibility type is public). Currently it defaults to private visibility.

Status: Closed

D.5 Owner left as Operators

Remediation
It is advised to remove the previous owner as operator(i.e. minter) once the ownership is transferred via the setContractOwner() function. Because it is possible for the previous owner to still remain as the minter which might be unexpected.

Status: Closed

D.6 Two Step Verification for the Owner

Remediation
For the setContractOwner, it is advised as a best practice to follow a two step two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible.

Status: Closed

D.7 baseURI immutable

Remediation
The _baseURI is set as https://cryptofight.io/nft/ permanently. It is advised to take a note of it as it cannot be changed later.

Status: Closed

E. Contract – ElemonStakingInitializer

High severity issues

E.1 Usage Of transfer Instead Of safeTransfer

```
Line 138:
if (user.allocation > 0) {
            uint256 pending = (_totalStaked *
               ((user.allocation * accTokenPerShare) /
                  PRECISION_FACTOR - user.rewardDebt)) / _totalAllocation;
            if (pending > 0) {
               rewardToken.transfer(address(_msgSender()),pending);                    }
      }
if (_amount > 0) {
            user.stakedAmount += _amount;
            _totalAllocation -= user.allocation;
            user.allocation = user.stakedAmount +
               _getBonusAmount(user.stakedAmount, user.boostPercent);
            stakedToken.transferFrom(address(_msgSender()),address(this),
               _amount);
            _totalAllocation += user.allocation;
            _totalStaked += _amount;
         }
```

```
Line 207:
if (_amount > 0) {
            user.stakedAmount -= _amount;
            _totalAllocation -= user.allocation;
            user.allocation =
               user.stakedAmount +
               _getBonusAmount(user.stakedAmount, user.boostPercent);
            stakedToken.transfer(address(_msgSender()),_amount);
            _totalAllocation += user.allocation;
            _totalStaked -= _amount;
         }
 if (pending > 0) {
            rewardToken.transfer(address(_msgSender()), pending);
         }
```

```
Line 238:
if (amountToTransfer > 0) {
            stakedToken.transfer(address(_msgSender()), amountToTransfer);
         }
```

```
Line 251:
function emergencyRewardWithdraw(uint256 _amount) external onlyOwner {
            rewardToken.transfer(address(_msgSender()), _amount);
}
```

```
Line 259:
IERC20(_tokenAddress).transfer(address(_msgSender()), _tokenAmount);
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks in effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Closed

Fixed

The Elemon team has fixed the issue by adding require statements to ensure that the transactions were executed successfully.

Medium severity issues

E.2 Race Condition

```
Line 339:
    function _updatePool() internal {
        if (block.number <= lastRewardBlock) {
            return;
        }
        if (_totalAllocation == 0) {
            lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = _getMultiplier(lastRewardBlock, block.number);
        uint256 earnedTokenReward = multiplier * rewardPerBlock;
        accTokenPerShare = accTokenPerShare +
            (earnedTokenReward * PRECISION_FACTOR) /
            _totalAllocation;
        lastRewardBlock = block.number;
    }
```

```
Line 295:
    function updateRewardPerBlock(uint256 _rewardPerBlock) external onlyOwner {
        require(block.number < startBlock, "Pool has started");
        rewardPerBlock = _rewardPerBlock;
        emit NewRewardPerBlock(_rewardPerBlock);
    }
```

Description
The rewardPerBlock has a setter. If a user calls a function that invokes the_update_pool function and the owner then changes the rewardPerBlock, the owner's transaction might get validated before the user's transaction. In that scenario the user will use the new value of the rewardBlock without being aware of that.

Remediation
Add the rewardPerBlock as an argument in the functions that uses the update pool then add a require condition which verifies that the rewardPerBlock provided in the arguments is the same as the one stored in the smart contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

Low severity issues

E.3 Missing address verification

```
Line 96:
constructor(address elemonNftAddress, address elemonInfoAddress) {
        SMART_CHEF_FACTORY = _msgSender();
        _elemonNFT = IERC721(elemonNftAddress);
        _elemonInfo = IElemonInfo(elemonInfoAddress);
        _boostBonusPercents[1] = 2000;
        _boostBonusPercents[2] = 2500;
        _boostBonusPercents[3] = 3000;
        _boostBonusPercents[4] = 3500;
        _boostBonusPercents[5] = 4000;
    }
```

```
Line 107:
function initialize(IERC20Metadata _stakedToken,
    IERC20Metadata _rewardToken, uint256 _rewardPerBlock,
    uint256 _startBlock, uint256 _bonusEndBlock,
    uint256 _poolLimitPerUser, address _admin
    ) external {
        require(!isInitialized, "Already initialized");
        require(_msgSender() == SMART_CHEF_FACTORY, "Not factory");
```

Description

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation

It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

E.4 Missing value verification

```
Line 91:
function initialize(IERC20Metadata _stakedToken,
        IERC20Metadata _rewardToken, uint256 _rewardPerBlock,
        uint256 _startBlock, uint256 _bonusEndBlock,
        uint256 _poolLimitPerUser, address _admin
    ) external {
        require(!isInitialized, "Already initialized");
        require(_msgSender() == SMART_CHEF_FACTORY, "Not factory");
```

```
Line 375:
function setBoostBonusPercents(uint256 level, uint256 percent) external onlyOwner{
    require(level > 0 && percent > 0, "Zero input");
    _boostBonusPercents[level] = percent;
}
```

```
Line 375:
function setVestingRewardPercent(uint256 percent) external onlyOwner{
    _vestingRewardPercent = percent;
}
```

Description

Certain functions lack a safety check in the value, the startBlock and _bonusEndBlock arguments should be higher than block.number. The percent should be also less tha the max value . Otherwise, the contract's logic may get hurt if an invalid value has been set.

Remediation

It’s recommended to add required statements to verify that the values that are provided from the arguments are valid.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

E.5 Renounce Ownership

```
Line 13:
contract ElemonStakingInitializer is Ownable, ReentrancyGuard, IERC721Receiver {
```

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

E.6 Usage Of block.timestamp

```
Line 179:
if (_amount > 0) {
    user.stakedAmount += _amount;
    _totalAllocation -= user.allocation;
    user.allocation = user.stakedAmount + _getBonusAmount(user.stakedAmount,
user.boostPercent);
    require(stakedToken.transferFrom(address(_msgSender()), address(this), _amount),
"Can not transfer staked token");
    _totalAllocation += user.allocation;
    _totalStaked += _amount;
}

user.rewardDebt = user.allocation * accTokenPerShare / PRECISION_FACTOR;
user.lastStakingTime = block.timestamp;
emit Deposit(_msgSender(), _amount);
}
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

E.7 For Loop Over Dynamic Array

```
uint256 rewardTotal = 0;
        for(uint256 index = 0; index < count; index++){
            if(vestingRewards[index].unlockedTime > 0 &&
vestingRewards[index].unlockedTime <= block.timestamp){
                rewardTotal += vestingRewards[index].unlockedQuantity;
                delete vestingRewards[index];
            }
        }
```

Description
When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

F. Contract – ElemonSummon

High severity issues

F.1 Usage Of transfer Instead Of safeTransfer

```
Line 206:
        if (!_isBoughts[affiliateAddress]) {
            IERC20(_paymentTokenAddress).transferFrom(
                _msgSender(),
                _recepientTokenAddress,
                price
            );
        } else {
            uint256 affiliateQuantity = (price * _affiliatePercent) /
                1000 / 100;
            IERC20(_paymentTokenAddress).transferFrom(_msgSender(),
                affiliateAddress, affiliateQuantity);
            IERC20(_paymentTokenAddress).transferFrom(_msgSender(),
                _recepientTokenAddress, price - affiliateQuantity);
```

```
Line 238:
function withdrawToken(address tokenAddress, address recepient,
            uint256 value) public onlyOwner {
            IERC20(tokenAddress).transfer(recepient, value);
        }
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Closed

Fixed

The Elemon team has fixed the issue by adding require statements to ensure that the transactions were executed successfully.

Medium severity issues

F.2 Race Condition

```
Line 206:
  if (!_isBoughts[affiliateAddress]) {
            IERC20(_paymentTokenAddress).transferFrom(_msgSender(),
                _recepientTokenAddress, price);
        } else {
            uint256 affiliateQuantity = (price * _affiliatePercent)/ 1000 /100;
            IERC20(_paymentTokenAddress).transferFrom(_msgSender(),
                affiliateAddress, affiliateQuantity);
            IERC20(_paymentTokenAddress).transferFrom(
                _msgSender(),_recepientTokenAddress,
                price - affiliateQuantity);
        }
```

```
Line 217:
//Request chainlink VRF
        require(LINK.balanceOf(address(this)) >= s_fee,
            "Not enough LINK to pay fee");
        bytes32 requestId = requestRandomness(s_keyHash, s_fee);
        _requestInfos[requestId] = RequestInfo({
            tokenId: tokenId,
            level: level
        });
```

Description
The _affiliatePercent and s_fee variables have setters. If the user calls a function that uses one of these variables, then the owner changes these values. In that case the owner's transaction might get validated first, and the user will use the new values without knowing about it.

Remediation
Add the the _affiliatePercent and s_fee as an argument in the functions that uses these variables then add a require condition which verifies that the provided in the arguments are the same as the one stored in the smart contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

F.3 For Loop Over Dynamic Array

```
Line 113:
function setRarityAbilities(uint256 level, int256[] memory rarities,
            uint256[] memory abilities) external onlyOwner {
            require(level > 0, "Invalid parameters");
            require(rarities.length > 0, "Rarities is invalid");
            require(rarities.length == abilities.length,
            "Rarities or abilities parameter is invalid");
            for (uint256 index = 0; index < rarities.length; index++) {
                  uint256 ability = abilities[index];
                  require(ability > 999, "ability should be greater than 999");
                  _rarityAbilities[level][rarities[index]] = ability;
            }
      }
```

```
Line 242:
function fulfillRandomness(bytes32 requestId, uint256 randomness)
      internal override {
            RequestInfo storage requestInfo = _requestInfos[requestId];
            require(requestInfo.tokenId > 0, "Request is invalid");
            //Get rarity
            _processValue = 0;
            for (uint256 index = 0; index < _rarities.length; index++) {
                  _processValue += _rarityAbilities[requestInfo.level][
                  _rarities[index]];
            }
            uint256 rarityNumber = (randomness % _processValue) + 1;
            _processValue = 0;
            uint256 rarity = 0;
            for (uint256 index = 0; index < _rarities.length; index++) {
                  _processValue += _rarityAbilities[requestInfo.level][
                  _rarities[index]];
            if (rarityNumber <= _processValue) {
                  rarity = _rarities[index];
                  break;
            }
      }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

F.4 fulfillRandomness revert

```
function fulfillRandomness(bytes32 requestId, uint256 randomness) internal override {
        RequestInfo storage requestInfo = _requestInfos[requestId];
        require(requestInfo.tokenId > 0, "Request is invalid");
        //Get rarity
        _processValue = 0;
        for(uint256 index = 0; index < _rarities.length; index++){
            _processValue += _rarityAbilities[requestInfo.level][_rarities[index]];
        }
        uint256 rarityNumber = randomness % _processValue + 1;
        _processValue = 0;
        uint256 rarity = 0;
        for(uint256 index = 0; index < _rarities.length; index++){
            _processValue += _rarityAbilities[requestInfo.level][_rarities[index]];
            if(rarityNumber <= _processValue){
                rarity = _rarities[index];
                break;
            }
        }
```

Description
According to Chainlink documentation on security considerations, fulfillRandomness must not revert. But it is possible for the current fulfillRandomness function to revert(It is advised to rewrite the entire fulfillRandomness to resolve this). If your fulfillRandomness implementation reverts, the VRF service will not attempt to call it a second time.

Remediation
Make sure your contract logic does not revert. Consider simply storing the randomness and taking more complex follow-on actions in separate contract calls made by you or your users.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

Low severity issues

F.5 Usage Of block.timestamp

```
Line 225:
        _isBoughts[_msgSender()] = true;

        emit Purchased(_msgSender(), tokenId, level, block.timestamp);
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

F.6 Missing address verification

```
Line 60:
constructor(address paymentTokenAddress, address recepientTokenAddress,
        address elemonInfoAddress, address elemonNFTAddress,
        uint256 affiliatePercent, address vrfCoordinator,
        address link, bytes32 keyHash, uint256 fee
        ) VRFConsumerBase(vrfCoordinator, link) {
            s_keyHash = keyHash;
            s_fee = fee;

        _paymentTokenAddress = paymentTokenAddress;
        _recepientTokenAddress = recepientTokenAddress;
        _elemonInfo = IElemonInfo(elemonInfoAddress);
        _elemonNFT = IElemonNFT(elemonNFTAddress);
        _affiliatePercent = affiliatePercent;
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

F.7 Missing value verification

```
Line 180:
function setAffiliatePercent(uint256 percent) external onlyOwner {
        _affiliatePercent = percent;
        }
```

```
Line 194:
function setLevelPrice(uint256 level, uint256 price) external onlyOwner {
            require(level > 0, "Level should be greater than 0");
            _levelPrices[level] = price;
            emit LevelPriceSetted(level, price);
        }
```

Description
Certain functions lack a safety check in the value, the startBlock and _bonusEndBlock arguments should be higher than block.number. Otherwise, the contract's logic may get hurt if an invalid value has been set.

Remediation
It's recommended to add require statements to verify that the values that are provided from the arguments are valid.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

Informational issues

Remediation

- As a Chainlink VRF security best practice, it is recommended to mint the nft at the end of fulfillRandomness function, after the random values have been assigned as required.
- Add and emit event for critical operations like the setPaymentTokenAddress(), setRecepientTokenAddress(), setAffiliatePercent(), setElemonInfo(), setElemonNFT(), setKeyHash() and setFee() functions which would be a best practice for offchain monitoring.

G. Contract – ElemonToken

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

G.1 Approve Race

```
Line 6:
contract ElemonToken is ERC20 {
        constructor() ERC20("Elemon Token", "ELMON",
          200000000000000000000000000){}
}
```

Description

The standard ERC20 implementation contains a widely-known racing condition in it approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation

Use increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

G.2 Renounce Ownership

```
Line 9:
    contract ERC20 is Ownable, IERC20, IERC20Metadata {
```

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

H. Contract – MultiSigWallet

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

I. Contract – TokenDistributor

High severity issues

I.1 Usage Of transfer Instead Of safeTransfer

```
function withdrawToken(address tokenAddress, address recepient) public onlyOwner{
        IERC20 token = IERC20(tokenAddress); // [+] Missing address verification
        token.transfer(recepient, token.balanceOf(address(this))); /    }
;
```

```
for(uint256 index = 0; index < addresses.length; index++){
payable(addresses[index]).transfer(amount);         }
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

Medium severity issues

I.2 Missing address verification

```
IERC20 token = IERC20(tokenAddress);
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

I.3 For Loop Over Dynamic Array

```
for(uint256 index = 0; index < addresses.length; index++){
payable(addresses[index]).transfer(amount);        }
```

Description
When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

J. Contract – MysteryBoxStaking

High severity issues

No issues were found.

Medium severity issues

J.1 Missing address verification

```
_mysteryBoxNft = mysteryBoxNft;
```

```
rewardToken = _rewardToken;
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

J.2 Race Condition

```
uint256 multiplier = _getMultiplier(lastRewardBlock, block.number);
uint256 earnedTokenReward = multiplier * rewardPerBlock;
```

Description

The rewardPerBlock has a setter. If a user calls a function that invokes the_update_pool function and the owner then changes the rewardPerBlock, the owner's transaction might get validated before the user's transaction. In that scenario the user will use the new value of the rewardBlock without being aware of that.

Remediation

Add the rewardPerBlock as an argument in the functions that use the update pool then add a require condition which verifies that the rewardPerBlock provided in the arguments is the same as the one stored in the smart contract.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

Low severity issues

J.3 Usage Of block.timestamp

```
user.lastStakingTime = block.timestamp;
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

K. Contract – MysteryBoxShopV2

High severity issues

K.1 Usage Of transfer Instead Of safeTransfer

```
function withdrawToken(address tokenAddress, address recepient) public onlyOwner{
        IERC20 token = IERC20(tokenAddress);
        token.transfer(recepient, token.balanceOf(address(this)))
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

Medium severity issues

K.2 Race Condition

```
uint256 burnQuantity = _price / 2;
```

Description
The _price has a setter. If a user calls a function that invokes setSaleInfo function and the owner then changes the _price , the owner's transaction might get validated before the user's transaction. In that scenario the user will use the new value of the _price without being aware of that.

Remediation
Add the _price as an argument in the functions that uses the paw24nao2fl then add a require condition which verifies that the _price provided in the arguments is the same as the one stored in the smart contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

Low severity issues

K.3 Usage Of block.timestamp

```
require(_startTime <= block.timestamp, "Box sale has not started");
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged
The Elemon team has acknowledged the risk.

L. Contract – MysteryBoxMarketplace

High severity issues

No issues were found.

Medium severity issues

L.1 Missing value verification

```
function setCoolDownDuration(uint256 value) external onlyOwner{
_coolDownDuration = value;
    }
```

Description

Certain functions lack a safety check in the value, the _coolDownDuration argument should be higher than 0.

Remediation

It’s recommended to add required statements to verify that the values that are provided from the arguments are valid.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

Low severity issues

L.2 Usage Of block.timestamp

```
_tokenTimes[tokenId] = block.timestamp;
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

L.3 Renounce Ownership

```
contract MysteryBoxMarketplace is Initializable, OwnableUpgradeable,
PausableUpgradeable, ReentrancyGuardUpgradeable, IERC721Receiver{
```

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status: Acknowledged

Acknowledged

The Elemon team has acknowledged the risk.

TestCases for Functional Testing

Some of the test cases which were part of the functional testing are listed below:

ElemonInfo.sol

- Should be able to setInfo for token **PASS**
- Only Operator can call setInfo **PASS**
- setOperator can only be called by Owner **PASS**

ElemonNFT.sol

- NFT can be burned by only the owner **PASS**
- NFT can only be minted by the operator **PASS**
- setOperator can only be called by Owner **PASS**
- NFT can be transferred/transferred from to other address **PASS**

ElemonToken.sol

- ERC20 transfer should work as expected **PASS**
- Owner should have totalSupply at deployment **PASS**

MultiSigWallet.sol

- Owners should be able to submitTransaction **PASS**
- Owners should be able to confirmTransaction **PASS**
- Owners should be able to executeTransaction **PASS**
- Owners should be able to revokeConfirmation **PASS**
- Only initially set owners should be able to act as owners **PASS**

ElemonMarketplace.sol

- Users should be able to createSellOrders **PASS**
- Users should be able to cancelSellOrder **PASS**
- Users should be able to purchase NFTs **PASS**
- Orders should delete once purchase is complete **PASS**
- Owner can change the fee percent successfully. **PASS**
- Owners can change NFT and Token Addresses. **PASS**
- Correct Fee should be deducted on purchase **PASS**

MysteryBoxMarketplace.sol

- Users can create and sell their order **PASS**
- Users can cancel Order **PASS**
- Users can Purchase **PASS**
- The Owner can update the cool down duration **PASS**

MysteryBoxShopV2.sol

- The owner can update the saleInfo **PASS**
- The user can’t exceed the limited box **PASS**
- The Owner can update the number of boxes per user **PASS**

MysteryBoxStaking.sol

- The user can deposit Tokens **PASS**
- The user can withdraw his tokens **PASS**
- The Owner can recover wrong tokens **PASS**
- The Owner can update reward per block **PASS**

Automated Tests

Slither

```
ElemonIDO.register() (ElemonIDO.sol#54-71) ignores return value by _busdToken.tr
ansferFrom(_msgSender(),_idoRecepientAddress,PAID_BUSD) (ElemonIDO.sol#66)
ElemonIDO.claim() (ElemonIDO.sol#73-110) ignores return value by _elmonToken.tra
nsfer(_msgSender(),tokenQuantity) (ElemonIDO.sol#107)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uncheck
ed-transfer

Reentrancy in ElemonIDO.register() (ElemonIDO.sol#54-71):
        External calls:
        - _busdToken.transferFrom(_msgSender(),_idoRecepientAddress,PAID_BUSD) (
ElemonIDO.sol#66)
        State variables written after the call(s):
        - _userBoughts[_msgSender()] = ELMON_ALLOCATION (ElemonIDO.sol#67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentra
ncy-vulnerabilities-1

ElemonIDO.setIdoBlocks(uint256,uint256) (ElemonIDO.sol#155-169) should emit an e
vent for:
        - _startBlock = startBlock (ElemonIDO.sol#167)
        - _endBlock = endBlock (ElemonIDO.sol#168)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing
-events-arithmetic

ElemonIDO.constructor(address,address,address,uint256,uint256).idoRecepientAddre
ss (ElemonIDO.sol#36) lacks a zero-check on :
                - _idoRecepientAddress = idoRecepientAddress (ElemonIDO.sol#43)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing
-zero-address-validation

Reentrancy in ElemonIDO.register() (ElemonIDO.sol#54-71):
        External calls:
        - _busdToken.transferFrom(_msgSender(),_idoRecepientAddress,PAID_BUSD) (
ElemonIDO.sol#66)
        State variables written after the call(s):
        - _totalBought += ELMON_ALLOCATION (ElemonIDO.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentra
ncy-vulnerabilities-2

Reentrancy in ElemonIDO.claim() (ElemonIDO.sol#73-110):
        External calls:
        - _elmonToken.transfer(_msgSender(),tokenQuantity) (ElemonIDO.sol#107)
        Event emitted after the call(s):
        - Claimed(_msgSender(),tokenQuantity) (ElemonIDO.sol#109)
Reentrancy in ElemonIDO.register() (ElemonIDO.sol#54-71):
        External calls:
        - _busdToken.transferFrom(_msgSender(),_idoRecepientAddress,PAID_BUSD) (
ElemonIDO.sol#66)
        Event emitted after the call(s):
        - Registered(_msgSender()) (ElemonIDO.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentra
ncy-vulnerabilities-3

Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Context._now() (utils/Context.sol#19-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-co
de

Pragma version0.8.9 (ElemonIDO.sol#2) necessitates a version too recent to be tr
usted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC20.sol#3) necessitates a version too recent
to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to b
```

```
Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Context._now() (utils/Context.sol#19-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (ElemonInfo.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ElemonInfo._operators (ElemonInfo.sol#27) is not in mixedCase
Variable ElemonInfo._tokenInfos (ElemonInfo.sol#33) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (utils/Context.sol#15)" inContext (utils/Context.sol#5-23)
Redundant expression "this (utils/Context.sol#20)" inContext (utils/Context.sol#5-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

setOperator(address,bool) should be declared external:
        - ElemonInfo.setOperator(address,bool) (ElemonInfo.sol#64-66)
owner() should be declared external:
        - Ownable.owner() (utils/Ownable.sol#26-28)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (utils/Ownable.sol#45-48)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (utils/Ownable.sol#54-56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
ElemonInfo.sol analyzed (3 contracts with 75 detectors), 14 result(s) found
```

```
ElemonMarketplace.withdrawToken(address,address) (ElemonMarketplace.sol#222-228) ignores return value by token.transfer(recepient,token.balanceOf(address(this))) (ElemonMarketplace.sol#227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Reentrancy in ElemonMarketplace.cancelSellOrder(uint256) (ElemonMarketplace.sol#76-96):
        External calls:
        - elemonContract.safeTransferFrom(address(this),_msgSender(),tokenId) (ElemonMarketplace.sol#88)
        State variables written after the call(s):
        - _tokenOwners[tokenId] = address(0) (ElemonMarketplace.sol#90)
Reentrancy in ElemonMarketplace.createSellOrder(uint256,uint256) (ElemonMarketplace.sol#45-70):
        External calls:
        - elemonContract.safeTransferFrom(_msgSender(),address(this),tokenId) (ElemonMarketplace.sol#62)
        State variables written after the call(s):
        - _tokenOwners[tokenId] = _msgSender() (ElemonMarketplace.sol#64)
Reentrancy in ElemonMarketplace.purchase(uint256,uint256,uint256) (ElemonMarketplace.sol#139-193):
        External calls:
        - require(bool)(elemonTokenContract.transferFrom(_msgSender(),address(this),tokenPrice)) (ElemonMarketplace.sol#154-160)
        - require(bool,string)(elemonTokenContract.transfer(owner(),feeAmount),Fail to fee to contract owner) (ElemonMarketplace.sol#165-168)
        - require(bool,string)(elemonTokenContract.transfer(tokenOwner,tokenPrice - feeAmount),Fail to token to owner) (ElemonMarketplace.sol#171-177)
        - IERC721(_elemonNftAddress).transferFrom(address(this),_msgSender(),tokenId) (ElemonMarketplace.sol#181-185)
        State variables written after the call(s):
        - _tokenOwners[tokenId] = address(0) (ElemonMarketplace.sol#187)
        - _tokenPrices[tokenId] = 0 (ElemonMarketplace.sol#188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ElemonMarketplace.setFeePercent(uint256) (ElemonMarketplace.sol#214-217) should emit an event for:
        - _feePercent = feePercent (ElemonMarketplace.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in ElemonMarketplace.cancelSellOrder(uint256) (ElemonMarketplace.sol#76-96):
        External calls:
        - elemonContract.safeTransferFrom(address(this),_msgSender(),tokenId) (ElemonMarketplace.sol#88)
        State variables written after the call(s):
        - _tokenPrices[tokenId] = 0 (ElemonMarketplace.sol#91)
Reentrancy in ElemonMarketplace.createSellOrder(uint256,uint256) (ElemonMarketplace.sol#45-70):
        External calls:
        - elemonContract.safeTransferFrom(_msgSender(),address(this),tokenId) (ElemonMarketplace.sol#62)
        State variables written after the call(s):
        - _tokenPrices[tokenId] = price (ElemonMarketplace.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in ElemonMarketplace.cancelSellOrder(uint256) (ElemonMarketplace.sol#76-96):
        External calls:
        - elemonContract.safeTransferFrom(address(this),_msgSender(),tokenId) (ElemonMarketplace.sol#88)
        Event emitted after the call(s):
        - SellingOrderCanceled(tokenId,_now()) (ElemonMarketplace.sol#93)
Reentrancy in ElemonMarketplace.createSellOrder(uint256,uint256) (ElemonMarketplace.sol#45-70):
        External calls:
        - elemonContract.safeTransferFrom(_msgSender(),address(this),tokenId) (ElemonMarketplace.sol#62)
        Event emitted after the call(s):
        - NewSellOrderCreated(_msgSender(),tokenId,price,_now()) (ElemonMarketplace.sol#67)
Reentrancy in ElemonMarketplace.purchase(uint256,uint256,uint256) (ElemonMarketplace.sol#139-193):
        External calls:
        - require(bool)(elemonTokenContract.transferFrom(_msgSender(),address(this),tokenPrice)) (ElemonMarketplace.sol#154-160)
        - require(bool,string)(elemonTokenContract.transfer(owner(),feeAmount),Fail to fee to contract owner) (ElemonMarketplace.sol#165-168)
        - require(bool,string)(elemonTokenContract.transfer(tokenOwner,tokenPrice - feeAmount),Fail to token to owner) (ElemonMarketplace.sol#171-177)
        - IERC721(_elemonNftAddress).transferFrom(address(this),_msgSender(),tokenId) (ElemonMarketplace.sol#181-185)
        Event emitted after the call(s):
        - Purchased(_msgSender(),tokenOwner,tokenId,tokenPrice,_now()) (ElemonMarketplace.sol#190)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version0.8.9 (ElemonMarketplace.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/ReentrancyGuard.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ElemonMarketplace._elemonTokenAddress (ElemonMarketplace.sol#20) is not in mixedCase
Variable ElemonMarketplace._elemonNftAddress (ElemonMarketplace.sol#21) is not in mixedCase
Variable ElemonMarketplace._tokenPrices (ElemonMarketplace.sol#27) is not in mixedCase
Variable ElemonMarketplace._tokenOwners (ElemonMarketplace.sol#30) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (utils/Context.sol#15)" inContext (utils/Context.sol#5-23)
Redundant expression "this (utils/Context.sol#20)" inContext (utils/Context.sol#5-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

withdrawToken(address,address) should be declared external:
        - ElemonMarketplace.withdrawToken(address,address) (ElemonMarketplace.sol#222-228)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (utils/Ownable.sol#45-48)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (utils/Ownable.sol#54-56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
ElemonMarketplace.sol analyzed (8 contracts with 75 detectors), 29 result(s) found
```

```
ERC721._checkOnERC721Received(address,address,uint256,bytes) (ElemonNFT.sol#781-812) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (ElemonNFT.sol#788-808)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (ElemonNFT.sol#795)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (ElemonNFT.sol#781-812) potentially used
C721Receiver(to).onERC721Received.selector (ElemonNFT.sol#796)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (ElemonNFT.sol#797)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (ElemonNFT.sol#781-812) potentially used
 == 0 (ElemonNFT.sol#798)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (ElemonNFT.sol#797)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (ElemonNFT.sol#781-812) potentially used
6,uint256)(32 + reason,mload(uint256)(reason)) (ElemonNFT.sol#805)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Address.isContract(address) (ElemonNFT.sol#28-39) uses assembly
        - INLINE ASM (ElemonNFT.sol#35-37)
Address._verifyCallResult(bool,bytes,string) (ElemonNFT.sol#233-254) uses assembly
        - INLINE ASM (ElemonNFT.sol#246-249)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (ElemonNFT.sol#781-812) uses assembly
        - INLINE ASM (ElemonNFT.sol#804-806)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._verifyCallResult(bool,bytes,string) (ElemonNFT.sol#233-254) is never used and should be removed
Address.functionCall(address,bytes) (ElemonNFT.sol#89-94) is never used and should be removed
Address.functionCall(address,bytes,string) (ElemonNFT.sol#102-108) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (ElemonNFT.sol#121-133) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (ElemonNFT.sol#141-158) is never used and should be removed
Address.functionDelegateCall(address,bytes) (ElemonNFT.sol#203-213) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (ElemonNFT.sol#221-231) is never used and should be removed
Address.functionStaticCall(address,bytes) (ElemonNFT.sol#166-177) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (ElemonNFT.sol#185-195) is never used and should be removed
Address.sendValue(address,uint256) (ElemonNFT.sol#57-69) is never used and should be removed
Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Context._now() (utils/Context.sol#19-22) is never used and should be removed
Strings.toHexString(uint256) (ElemonNFT.sol#288-299) is never used and should be removed
Strings.toHexString(uint256,uint256) (ElemonNFT.sol#304-318) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (ERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (ElemonNFT.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721Metadata.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (ElemonNFT.sol#57-69):
        - (success) = recipient.call{value: amount}() (ElemonNFT.sol#64)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (ElemonNFT.sol#141-158):
        - (success,returndata) = target.call{value: value}(data) (ElemonNFT.sol#154-156)
Low level call in Address.functionStaticCall(address,bytes,string) (ElemonNFT.sol#185-195):
        - (success,returndata) = target.staticcall(data) (ElemonNFT.sol#193)
Low level call in Address.functionDelegateCall(address,bytes,string) (ElemonNFT.sol#221-231):
        - (success,returndata) = target.delegatecall(data) (ElemonNFT.sol#229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Constant Strings.alphabet (ElemonNFT.sol#258) is not in UPPER_CASE_WITH_UNDERSCORES
Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (ElemonNFT.sol#568) is not in mixedCase
Variable ERC721._totalSupply (ElemonNFT.sol#330) is not in mixedCase
Variable ElemonNFT._operators (ElemonNFT.sol#844) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (utils/Context.sol#15)" inContext (utils/Context.sol#5-23)
```

```
ElemonShop.withdrawToken(address) (ElemonShop.sol#122-125) ignores return value by token.transfer(owner(),token.balanceOf(address(this))) (ElemonShop.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Reentrancy in ElemonShop.purchase(uint256,uint256) (ElemonShop.sol#98-117):
        External calls:
        - _elemonNft.safeTransferFrom(address(this),_msgSender(),tokenId) (ElemonShop.sol#112)
        State variables written after the call(s):
        - _isSold[tokenId] = true (ElemonShop.sol#113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Reentrancy in ElemonShop.purchase(uint256,uint256) (ElemonShop.sol#98-117):
        External calls:
        - _elemonNft.safeTransferFrom(address(this),_msgSender(),tokenId) (ElemonShop.sol#112)
        Event emitted after the call(s):
        - Purchased(_msgSender(),tokenId,price,_now()) (ElemonShop.sol#115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (ElemonShop.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC721Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IElemonNFT.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/ReentrancyGuard.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Runnable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ElemonShop._elemonNft (ElemonShop.sol#13) is not in mixedCase
Variable ElemonShop._isSold (ElemonShop.sol#16) is not in mixedCase
Variable ElemonShop._starPrices (ElemonShop.sol#19) is not in mixedCase
Variable ElemonShop._tokenStars (ElemonShop.sol#22) is not in mixedCase
Variable Runnable._isRunning (utils/Runnable.sol#18) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (utils/Context.sol#15)" inContext (utils/Context.sol#5-23)
Redundant expression "this (utils/Context.sol#20)" inContext (utils/Context.sol#5-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

setElemonNft(address) should be declared external:
        - ElemonShop.setElemonNft(address) (ElemonShop.sol#29-32)
setStarPrice(uint256,uint256) should be declared external:
        - ElemonShop.setStarPrice(uint256,uint256) (ElemonShop.sol#38-42)
setStarPriceMultiple(uint256[],uint256[]) should be declared external:
        - ElemonShop.setStarPriceMultiple(uint256[],uint256[]) (ElemonShop.sol#48-61)
setTokenStar(uint256,uint256) should be declared external:
        - ElemonShop.setTokenStar(uint256,uint256) (ElemonShop.sol#67-70)
setTokenStarMultiple(uint256[],uint256[]) should be declared external:
        - ElemonShop.setTokenStarMultiple(uint256[],uint256[]) (ElemonShop.sol#76-88)
purchase(uint256,uint256) should be declared external:
        - ElemonShop.purchase(uint256,uint256) (ElemonShop.sol#98-117)
withdrawToken(address) should be declared external:
        - ElemonShop.withdrawToken(address) (ElemonShop.sol#122-125)
setNftContractOwner(address) should be declared external:
        - ElemonShop.setNftContractOwner(address) (ElemonShop.sol#127-129)
withdrawBnb() should be declared external:
        - ElemonShop.withdrawBnb() (ElemonShop.sol#134-137)
renounceOwnership() should be declared external:
```

```
ERC20.allowance(address,address).owner (ERC20.sol#67) shadows:
        - Ownable.owner() (utils/Ownable.sol#26-28) (function)
ERC20._approve(address,address,uint256).owner (ERC20.sol#174) shadows:
        - Ownable.owner() (utils/Ownable.sol#26-28) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Context._msgData() (utils/Context.sol#14-17) is never used and should be removed
Context._now() (utils/Context.sol#19-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (ElemonToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (interfaces/IERC20Metadata.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (utils/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Redundant expression "this (utils/Context.sol#15)" inContext (utils/Context.sol#5-23)
Redundant expression "this (utils/Context.sol#20)" inContext (utils/Context.sol#5-23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable ERC20._totalSupply (ERC20.sol#15) is too similar to ERC20.constructor(string,string,uint256).totalSupply_ (ERC20.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

ElemonToken.constructor() (ElemonToken.sol#7-9) uses literals with too many digits:
        - ERC20(Elemon Token,ELMON,200000000000000000000000000) (ElemonToken.sol#8)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

name() should be declared external:
        - ERC20.name() (ERC20.sol#31-33)
symbol() should be declared external:
        - ERC20.symbol() (ERC20.sol#35-37)
decimals() should be declared external:
        - ERC20.decimals() (ERC20.sol#39-41)
totalSupply() should be declared external:
        - ERC20.totalSupply() (ERC20.sol#43-45)
balanceOf(address) should be declared external:
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. Many issues were discovered during the entire audit; some of them were fixed by the Elemon Team.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Elemon** Contracts. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Elemon** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
January, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com