

Security Assessment

DRAGON CROWN

Verified On Feb 20th, 2024

 @Vital-Block

 @VB_Audit

 info@vitalblock.org

 www.vitalblock.org



PREPARED FOR:

DRAGON CROWN



TABLE OF CONTENTS

TABLE OF CONTENTS	3
DOCUMENT PROPERTIES	4
ABOUT VBS	5
SCOPE OF WORK	6
AUDIT METHODOLOGY	7
AUDIT CHECKLIST	9
EXECUTIVE SUMMARY	10
CENTRALIZED PRIVILEGES	11
RISK CATEGORIES	12
AUDIT SCOPE	13
AUTOMATED ANALYSIS	14
KEY FINDINGS	19
MANUAL REVIEW	20
VULNERABILITY SCAN	28
REPOSITORY	29
INHERITANCE GRAPH	30
PROJECT BASIC KNOWLEDGE	31
AUDIT RESULT	32
REFERENCES	37



INTRODUCTION

Auditing Firm	 VITAL BLOCK SECURITY
Client Firm	 DRAGON CROWN
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	DCRN.sol Factory.sol MarginStakingManager.sol Masterchef.sol Router.sol
Language	Solidity
Centralization	Active ownership
Website	https://dragoncrown.org/
Network Chain	 Arbitrum On
Telegram Ann	https://t.me/DRAGONCROWN_ANNOUNCEMENTS
Twitter	https://twitter.com/DragonCrownSwap
Doc	https://dragoncrown.org/DragonCrownDocs.pdf
Prelim Report Date	February 18 th , 2023
Final Report Date	February 20 th 23 2023

 i Verify the authenticity of this report on our GitHub Repo: <https://www.github.com/vital-block>



Document Properties


Client	DRAGON CROWN
Title	Smart Contract Audit Report
Target	DRAGON CROWN
Audit Version	1.0
Author	Akhmetshin Marat
Auditors	Akhmetshin Marat, James BK, Benny Matin
Reviewed by	Dima Meru
Approved by	Prince Mitchell
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	February 20 th , 2024	James BK	Final Released
1.0-AP	February 20 TH , 2024	Benny Matin	Release Candidate

Contact

For more information about this document and its contents, please contact Vital Block Security Inc.

Name	Akhmetshin Marat
Phone 	+44 7944 248057
Email	info@vitalblock.org



In the following, we show the specific pull request and the commit hash value used in this audit.

- [DCRNTOKEN](#) (DR78740)
- [DCRN#code](#) (DTVP78)

About Vital Block Security

Vital Block Security provides professional, thorough, fast, and easy-to-understand smart contract security audit. We do in-depth and penetrative static, manual, automated, and intelligent analysis of the smart contract. Some of our automated scans include tools like ConsenSys MythX, Mythril, Slither, Surya. We can audit custom smart contracts, Solidity, DApps, Rust, NFTs, etc (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/vital_block), Twitter (http://twitter.com/Vb_Audit), or Email (info@vitalblock.org).

Table 1.2: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

Methodology (1)

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [4]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

SCOPE OF WORK

Vital Block was consulted by **DRAGON CROWN** to conduct the smart contract audit of its Rust source code. The audit scope of work is strictly limited to mentioned .SOL file only:

O.DCRN.Sol
O.FACTORY.Sol
O.MARGINSTAKINGMANAGER.Sol
O.MASTERCHEF.Sol
O.ROUTER.Sol

 External contracts and/or interfaces dependencies are not checked due to being out of scope.

Verify audited contract code Repo.

Public Contract Link

[O.DCRN.Sol](#)

[O.FACTORY.Sol](#)

[O.MARGINSTAKINGMANAGER.Sol](#)

[O.MASTERCHEF.Sol](#)

[O.ROUTER.Sol](#)

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of Vital Block Security auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
 - Remix IDE Developer Tool
 - Open Zeppelin Code Analyzer
 - SWC Vulnerabilities Registry
 - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges.

We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	<ul style="list-style-type: none">○ Token Supply Manipulation○ Access Control and Authorization○ Assets Manipulation○ Ownership Control○ Liquidity Access○ Stop and Pause Trading○ Ownable Library Verification
----------------------	---

Common Contract Vulnerabilities

- **Integer Overflow**
- **Lack of Arbitrary limits**
- **Incorrect Inheritance Order**
- **Typographical Errors**
- **Requirement Violation**
- **Gas Optimization**
- **Coding Style Violations**
- **Re-entrancy**
- **Third-Party Dependencies**
- **Potential Sandwich Attacks**
- **Irrelevant Codes**
- **Divide before multiply**
- **Conformance to Solidity Naming Guides**
- **Compiler Specific Warnings**
- **Language Specific Warnings**

REPORT

- **The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.**
- **The client's development team reviews the report and makes amendments to the codes.**
- **The auditing team provides the final comprehensive report with open and unresolved issues.**

PUBLISH

- **The client may use the audit report internally or disclose it publicly.**


 **It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.**








Table 1.0 The Full Audit Checklist

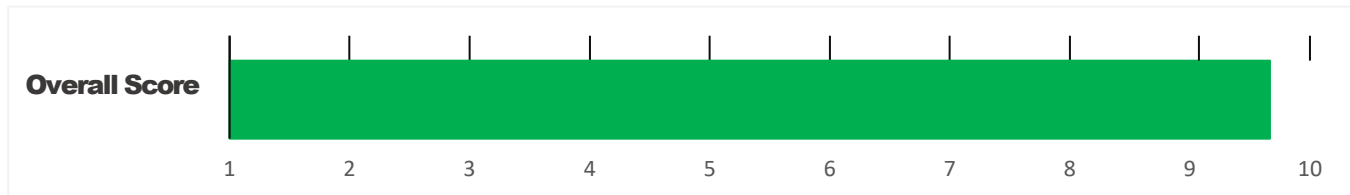
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

EXECUTIVE SUMMARY

Vital Block Security has performed the automated and manual analysis of the **DRAGON CROWN** Sol code. The code was reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical ! 	Major " 	Medium # 	Minor \$ 	Unknown % 
Open	0	0	0	2	0
Informational	0	0	2	0	1
Acknowledge	0	0	0	0	0
Noteworthy OnlyOwner Privileges	Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router				

DRAGON CROWN Smart contract has achieved the following score: **92.5**



i Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

i Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.



CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- **Privileged roles can be granted the power to `pause()` the contract in case of an external attack.**
- **Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.**

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.






- **The client can lower centralization-related risks by implementing below mentioned practices:**
- **Privileged role's private key must be carefully secured to avoid any potential hack.**
- **Privileged role should be shared by multi-signature (multi-sig) wallets.**
- **Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.**
- **Renouncing the contract ownership, and privileged roles.**
- **Remove functions with elevated centralization risk.**

 Understand the project's initial asset distribution. Assets in the liquidity pair should be locked.

Assets outside the liquidity pair should be locked with a release schedule.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical ! 	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major " 	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium # 	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor \$ 	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown % 	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.



Key Findings

Overall, these contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), 2 medium-severity vulnerabilities, 3 low-severity vulnerabilities, and 2 informational recommendations.

Table 2.1: Key **DRAGON CROWN** Audit Findings

ID	Severity	Title	Category	Status
DNY-001	Low	In updateForMinter , the following equation is used inside an unchecked block	Coding Practice	Fixed
DNY-002	Low	In updateFormapping , the following equation is used inside an unchecked block	Business Logic	Fixed
DNY-003	Low	In updateForAmount, Relevant Function Snippet	Coding Practice	Fixed
DNY-004	Informational	In updateForOwner, Relevant Function Snippet	Coding Practice	Fixed
DNY-005	Informational	In Suggested Constant/Immutable Usages For Gas Efficiency	Coding Practice	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to page 10 for details.






AUDIT SCOPE

DRAGON CROWN

ID	Repo	Comment	File	SHM321 Checksum
DBY	contracts/yieldlend/gov	cC512486	DCRN.sol	6788099YIRHVS853PKFMGHEF44309200KDHFCBUGIJN
DBI	contracts/yieldlend/gov	cC512486	DCRN.sol	347520JHDB7549H22H3BVDIOETYUHF009JBIKBDI33BJ4
DBW	contracts/yieldlend/gov	cC512486	DCRN.sol	1988Y73HUGFDINN353840NFMTEJER73649RGFIMDIDH
DBG	contracts/yieldlend/gov	cC512486	DCRN.sol	4438648TEOHBF6378309EHROECNEPOEJDNTE8EYEU3
DBL	contracts/yieldlend/gov	cC512486	DCRN.sol	66390028765RVNKDBYFTGW553T2KOEHIUJJJIE
DBA	contracts/yieldlend/gov	cC512486	Factory.sol	09825539BDYG543DVNKOMIKEBYRJUFHHFHFIE333222
DBJ	contracts/yieldlend/gov	cC512486	Factory.sol	8654RJVT3DWI865YK26437903JJDGGDHGWY6E
DBE	contracts/yieldlend/gov	cC512486	Factory.sol	7763888636TGYGFFTFHBETT66TFTCTVYBHYT
DBP	Contracts/yieldlend/gov	cC512486	Factory.sol	88530486494YRHFEICBGEIEGWTWYUWUHEJEHEIE33U3
DBM	contracts/yieldlend/gov	cC512486	Factory.sol	1209873KHJLKJNFJHGE98763990029774BCUHHUU239
DBV	contracts/yieldlend/gov/interfaces	cC512486	MaginStakingManager.sol	23456UGFYUHE98756EFHJHE7654ESDFGHGERTYUJ3897
DBQ	contracts/yieldlend/gov/interfaces	cC512486	MaginStakingManager.sol	37889UHBIONE07TYRDFGVBN5678939IJWSFVDYUHDCL
DBS	contracts/yieldlend/gov/interfaces	cC512486	MaginStakingManager.sol	678903098TFHJKFCPOIUGFGHJKE9865ERGBEIVBHE8767
DBR	contracts/yieldlend/gov/interfaces	cC512480	MaginStakingManager.sol	98765SDFGBNFCOI56789UIYHGGHEJDIUYTRDCVBN3459
DCD	contracts/yieldlend/gov/interfaces	cC512481	Masterchef.sol	3348y9808hgtrusvnm43100ejfojgfnut8496230hb574he
DHU	contracts/yieldlend/gov/interfaces	cC512481	Masterchef.sol	9864byf5f379eig28ffre64085jv1613251guhkdme87
DGG	contracts/yieldlend/gov/interfaces	cC512481	Router.sol	7ej2d8jg765tjfiowg538ij74dwftv6478ij3gs820
DTR	contracts/yieldlend/gov/interfaces	cC512481	Router.sol	864fr46de438hdguw903rfdcb246dbuhb2917enk



AUTOMATED ANALYSIS

Symbol	Definition
	Function modifies state
	Function is payable
	Function is internal
	Function is private
	Function is important

```

| **DRAGON CROWN** | Interface | |||
| L | totalSupply | External ! | ! | NO ! |
| L | decimals | External ! | ! | NO ! |
| L | symbol | External ! | ! | NO ! |
| L | name | External ! | ! | NO ! |
| L | getOwner | External ! | | NO ! |
| L | balanceOf | External ! | ! | NO ! |
| L | transfer | External ! | " ! ! | NO ! |
| L | allowance | External ! | ! | NO ! |
| L | approve | External ! | " ! ! | NO ! |
| L | transferFrom | External ! | " | NO ! |
|||||
| **IFactoryV2** | Interface | |||
| L | getPair | External ! | | NO ! |
| L | createPair | External ! | " | NO ! |
|||||
| **IV2Pair** | Interface | |||
| L | factory | External ! | | NO ! |
| L | getReserves | External ! | | NO ! |
| L | sync | External ! | " | NO ! |

```

|||||

| ****IRouter01**** | Interface | |||

| L | factory | External ! | |NO!|

| L | ETH | External ! | |NO!|

| L | addLiquidityETH | External ! | # |NO!|

| L | addLiquidity | External ! | " |NO!|

| L | swapExactETHForTokens | External ! | # |NO!|

| L | getAmountsOut | External ! | |NO!|

| L | getAmountsIn | External ! | |NO!|

|||||

| ****IRouter02**** | Interface | IRouter01 |||

| L | swapExactTokensForETHSupportingFeeOnTransferTokens | External ! | " |NO!|

| L | swapExactETHForTokensSupportingFeeOnTransferTokens | External ! | # |NO!|

| L | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ! | " ! |NO!|

| L | swapExactTokensForTokens | External ! | " |NO!|

|||||

| ****Protections**** | Interface | |||

| L | checkUser | External ! | " ! |NO!|

| L | setLaunch | External ! | " ! |NO!|

| L | setLpPair | External ! | " ! |NO!|

| L | **DCRN** | External ! | " |NO!|

| L | removeSniper | External ! | " |NO!|

|||||

| ****Cashier**** | Interface | |||

| L | setRewardsProperties | External ! | " |NO!|

| L | tally | External ! | " |NO!|

| L | load | External ! | # |NO!|

| L | cashout | External ! | " |NO!|

| L | giveMeWelfarePlease | External ! | " |NO!|

| L | getTotalDistributed | External ! | |NO!|

| L | getUserInfo | External ! | |NO!|

| L | getUserRealizedRewards | External ! | |NO!|









```

| L | getPendingRewards | External ! | | NO ! |
| L | initialize | External ! | " | NO ! |
| L | getCurrentReward | External ! | | NO ! |
|||||
| **SOL** | Implementation | SafeMath |||
| L | <Constructor> | Public ! | # | NO ! |
| L | transferOwner | External ! | " | onlyOwner |
| L | renounceOwnership | External ! | " | NO ! |
| L | setOperator | Public ! | " | NO ! |
| L | renounceOriginalDeployer | External ! | " | NO ! |
| L | <Receive Ether> | External ! | # | NO ! |
| L | totalSupply | External ! | | NO ! |
| L | decimals | External ! | | NO ! |
| L | symbol | External ! | | NO ! |
| L | name | External ! | | NO ! |
| L | getOwner | External ! | ! | NO ! |
| L | balanceOf | Public ! | ! | NO ! |
| L | allowance | External ! | ! | NO ! |
| L | approve | External ! | " ! | NO ! |
| L | _approve | Internal $ | " | |
| L | approveContractContingency | Public ! | " ! | onlyOwner |
| L | transfer | External ! | " | NO ! |
| L | transferFrom | External ! | " | NO ! |
| L | setNewRouter | External ! | " | onlyOwner |
| L | setLpPair | External ! | " | onlyOwner |
| L | setInitializers | External ! | " | onlyOwner |
| L | isExcludedFromFees | External ! | | NO ! |
| L | isExcludedFromDividends | External ! | | NO ! |
| L | isExcludedFromProtection | External ! | | NO ! |
| L | setDividendExcluded | Public ! | " | onlyOwner |
| L | setExcludedFromFees | Public ! | " | onlyOwner |

```

OPTIMIZATIONS | DRAGON CROWN

ID	Title	Category	Status
DTV	Logarithm Refinement Optimization	Gas Optimization	Acknowledged 
DOP	Checks Can Be Performed Earlier	Gas Optimization	Acknowledged 
DDP	Unnecessary Use Of SafeMath	Gas Optimization	Acknowledged 
DWY	Struct Optimization	Gas Optimization	Acknowledged 
DGT	Unused State Variable	Gas Optimization	Acknowledged 

General Detectors



DoS with Failed Call

This contract uses external calls that may fail, resulting in loss of functionality.



Attention
Required



Misuse of Boolean Constant

The usage of specific true/false values in this contract may lead to errors.



Attention
Required



Numeric Notation Best Practices

The numeric notation used in this contract is unconventional, possibly worsening the reading/debugging experience



Attention
Required

- | | |
|--|--|
| ✓ No compiler version inconsistencies found | ✓ No tautologies or contradictions found |
| ✓ No unchecked call responses found | ✓ No faulty true/false values found |
| ✓ No vulnerable self-destruct functions found | ✓ No inaccurate divisions found |
| ✓ No assertion vulnerabilities found | ✓ No redundant constructor calls found |
| ✓ No old solidity code found | ✓ No vulnerable transfers found |
| ✓ No external delegated calls found | ✓ No vulnerable return values found |
| ✓ No external call dependency found | ✓ No uninitialized local variables found |
| ✓ No vulnerable authentication calls found | ✓ No default function responses found |
| ✓ No invalid character typos found | ✓ No missing arithmetic events found |
| ✓ No RTL characters found | ✓ No missing access control events found |
| ✓ No dead code found | ✓ No redundant true/false comparisons found |
| ✓ No risky data allocation found | ✓ No state variables vulnerable through function calls found |
| ✓ No uninitialized state variables found | ✓ No buggy low-level calls found |
| ✓ No uninitialized storage variables found | ✓ No expensive loops found |
| ✓ No vulnerable initialization functions found | ✓ No bad numeric notation practices found |
| ✓ No risky data handling found | ✓ No missing constant declarations found |
| ✓ No number accuracy bug found | ✓ No missing external function declarations found |
| ✓ No out-of-range number vulnerability found | ✓ No vulnerable payable functions found |
| ✓ No map data deletion vulnerabilities found | ✓ No vulnerable message values found |



DOY-01 Key Findings

Category	Severity ●	Location	Status
Status Mathematical Operations	Low	DCRN.sol Contracts – 218-227	Informational

Description

In **updateForMinter**, the following equation is used inside an unchecked block

```
function _mint(address account, uint256 amount) internal virtual {  
    require(account != address(0), "BEP20: mint to the zero address");  
  
    _totalSupply = _totalSupply + amount;  
    _balances[account] = _balances[account] + amount;  
    emit Transfer(address(0), account, amount);  
}
```

Minter can not issue more **DRAGON CROWN** Token indefinitely.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the **DCRN** contract.

Thus, this enables the approval of a token account for confidential transfers, even if it is associated with a different mint. Ideally, token accounts should only be allowed to hold tokens from the specific mint they are associated with. By not checking the mint consistency, the function effectively approves arbitrary token accounts for confidential transfers. Such unauthorized token mixing may have security and financial implications, as it could result in loss of value or assets for users who rely on the token system's integrity.

Recommendation

Incorporate the following verification within process approve account to confirm that the token account's associated mint aligns with the mint for which the confidential transfer approval is sought.



DNY-02 Key Findings

Category	Severity ●	Target	Status
Business Logic	Medium	Contract/DCRN.sol 276-285	Fixed

Description

In `updateFormmapping`, the following equation is used inside an unchecked block

```
contract BEP20 is Context, IBEP20, Ownable {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string public name;
    string public symbol;
    uint8 public decimals;

    uint8 private _setupDecimals = 18;
```

Description

The function `Mapping ()` does not have the override specifier. It should be noted that since (a function that overrides only a single interface function does not require the override specifier (see doc). However, all other instances of this in the code base contain the override specifier.

Recommendation

We recommend either checking for overflow in this case, or ensuring that the `PairsIn` is close enough it will never cause an overflow

DNY-03 Key Findings

Category	Severity ●	Target	Status
Inconsistency	Informational	DCRN.sol 335-341	Acknowledge

Description

In **updateForAmount**, Relevant Function Snippet

```
function transfer(  
    address recipient,  
    uint256 amount  
) public virtual override returns (bool) {  
    _transfer(_msgSender(), recipient, amount);  
    return true;  
}
```

Description

The function **amount0()** does not have the override specifier. It should be noted that since amount0 > a function That overrides only a single interface function does not require the override specifier. However, all other instances of this in the codebase contain the override specifier

Recommendation

We recommend adding the override specifier to **amount()** or removing the override specifier from all other functions this applies to for consistency.

DNY-04 Key Findings

Category	Severity ●	Target	Status
Coding Practices	low	contracts/Factory.sol 241-244	Informational

Description

In `updateForOwner`, Relevant Function Snippet

```
function _approve(address owner, address spender, uint value) private {  
    allowance[owner][spender] = value;  
    emit Approval(owner, spender, value);  
}
```

Description

For Ownership efficiency, the DRAGON CROWN Team is engineered with the reserve cache mechanism, which necessitates the common steps to be followed when operating with the reserve Ownership data in different scenarios, including the tax generation, update, and eventual persistence.

Recommendation

Revise the above functions to following a consistent approach to use the reserve cache mechanism.

DST-05 Key Findings

Category	Severity ●	Location	Status
Coding Practices	Low	Masterchef.sol 1150-1155	Informational

Description

In Suggested Constant/Immutable Usages For Gas Efficiency

```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```

Description

An immutable state variable can only be assigned during contract creation, but will remain constant throughout the life-time of a deployed contract. The main benefit of declaring a state as immutable is that reading the state is significantly cheaper than reading from regular storage, since it is not stored in storage anymore. Instead, an immutable state will be directly inserted into the runtime code.

-- Update reward variables for all pools. Be careful of gas spending!

Recommendation

Revisit the state variable definition and make extensive use of constant gas/ immutable states.

Vulnerability Scan

REENTRANCY

✓ No reentrancy risk found

Severity Major

Confidence Parameter Certain

✗ **Mintable**: More amount of the **DRAGON CROWN** token can **NOT** be minted by a private wallet or contract. (This is Essentially normal for most contracts)

```
contract BEP20 is Context, IBEP20, Ownable {
    using SafeMath for uint256;
    using Address for address;

    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

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

Vulnerability Description

Scanning Line:



Repository:

<https://bscscan.com/address/0x61Efb60A075479D96Ad3D1C54dd87D71AfFF980f>

Audited Files

O.DCRN.Sol
O.FACTORY.Sol
O.MARGINSTAKINGMANAGER.Sol
O.MASTERCHEF.Sol
O.ROUTER.Sol

Contract Creator Address

Not Established

Deployed Contracts:

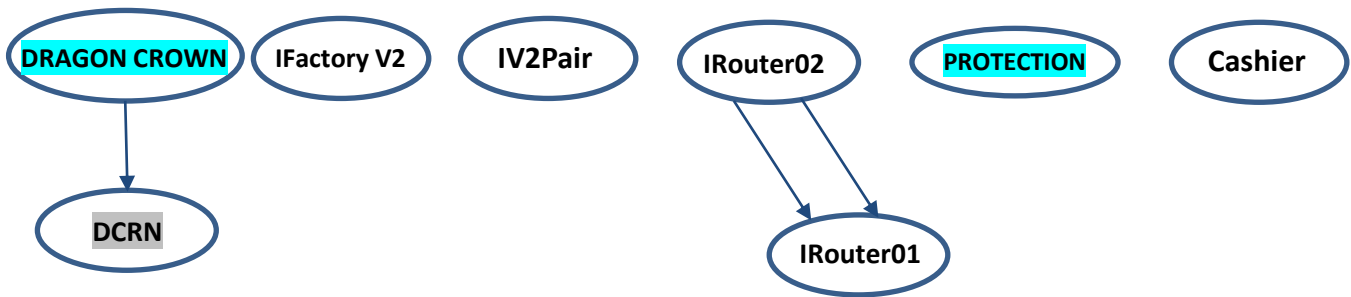
Not Deployed

Creator TXH Contracts:

Not Refillable



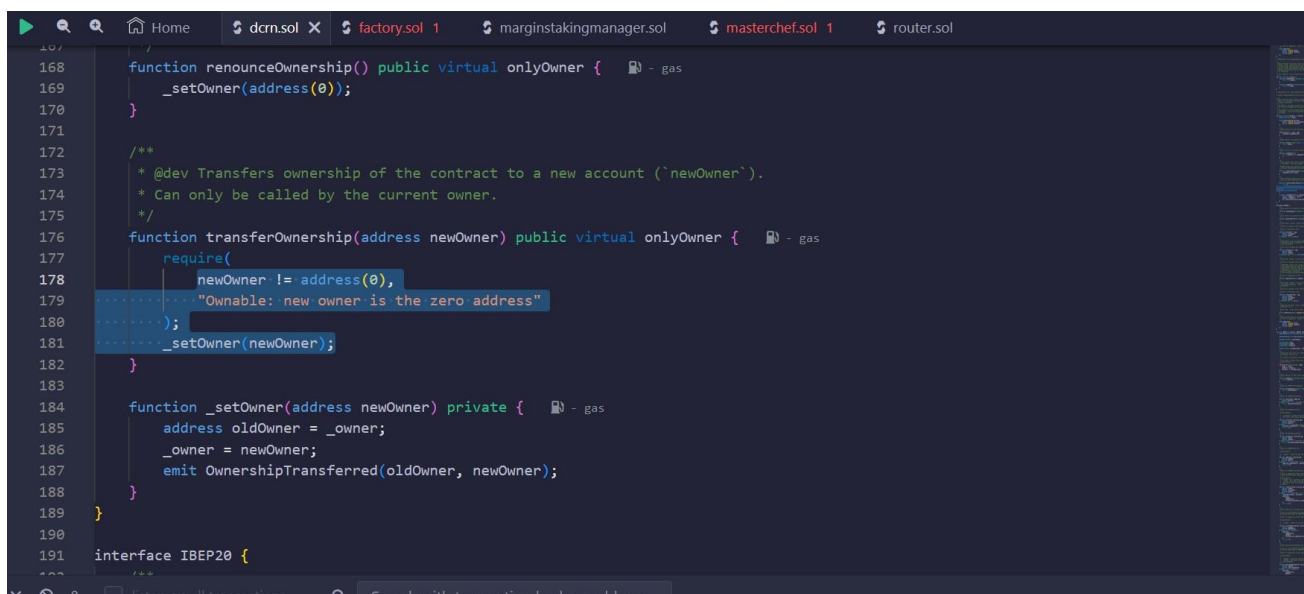
INHERITANCE GRAPH



Identifier	Definition	Severity
CEN-12	Centralization privileges of DRAGON CROWN	Medium 🟡

Vulnerability 0 : No important security issue detected.

Threat level: Low



```

168 function renounceOwnership() public virtual onlyOwner {
169     _setOwner(address(0));
170 }
171
172 /**
173  * @dev Transfers ownership of the contract to a new account (`newOwner`).
174  * Can only be called by the current owner.
175  */
176 function transferOwnership(address newOwner) public virtual onlyOwner {
177     require(
178         newOwner != address(0),
179         "Ownable: new owner is the zero address"
180     );
181     _setOwner(newOwner);
182 }
183
184 function _setOwner(address newOwner) private {
185     address oldOwner = _owner;
186     _owner = newOwner;
187     emit OwnershipTransferred(oldOwner, newOwner);
188 }
189
190
191 interface IBEP20 {

```

ISSUES CHECKING STATUS

Issue Description		Checking Status
1.	Compiler errors.	PASSED
2.	Race Conditions and reentrancy. Cross-Function Race Conditions.	PASSED
3.	Possible Delay In Data Delivery.	PASSED
4.	Oracle calls.	PASSED
5.	Front Running.	PASSED
6.	Sol Dependency.	PASSED
7.	Integer Overflow And Underflow.	PASSED
8.	DoS with Revert.	PASSED
9.	Dos With Block Gas Limit.	PASSED
10.	Methods execution permissions.	PASSED
11.	Economy Model of the contract.	PASSED
12.	The Impact Of Exchange Rate On the solidity Logic.	PASSED
13.	Private use data leaks.	PASSED
14.	Malicious Event log.	PASSED
15.	Scoping and Declarations.	PASSED
16.	Uninitialized storage pointers.	PASSED
17.	Arithmetic accuracy.	PASSED
18.	Design Logic.	PASSED
19.	Cross-Function race Conditions	PASSED
20.	Save Upon solidity contract Implementation and Usage.	PASSED
21.	Fallback Function Security	PASSED



MANUAL REVIEW

Dragon Crown (DCRN) pioneers the convergence of decentralized finance (DeFi) and play-to-earn (P2E) gaming, aiming to establish a groundbreaking ecosystem driven by enhanced token utility and immersive user experiences. Dragon Crown merges two integral components: Dragon Crown Swap and Dragon Crown War.

TOKEN NAME: DRAGON CROWN

Ticker: DCRN

DECIMALS: 18

Total Supply: 10000000

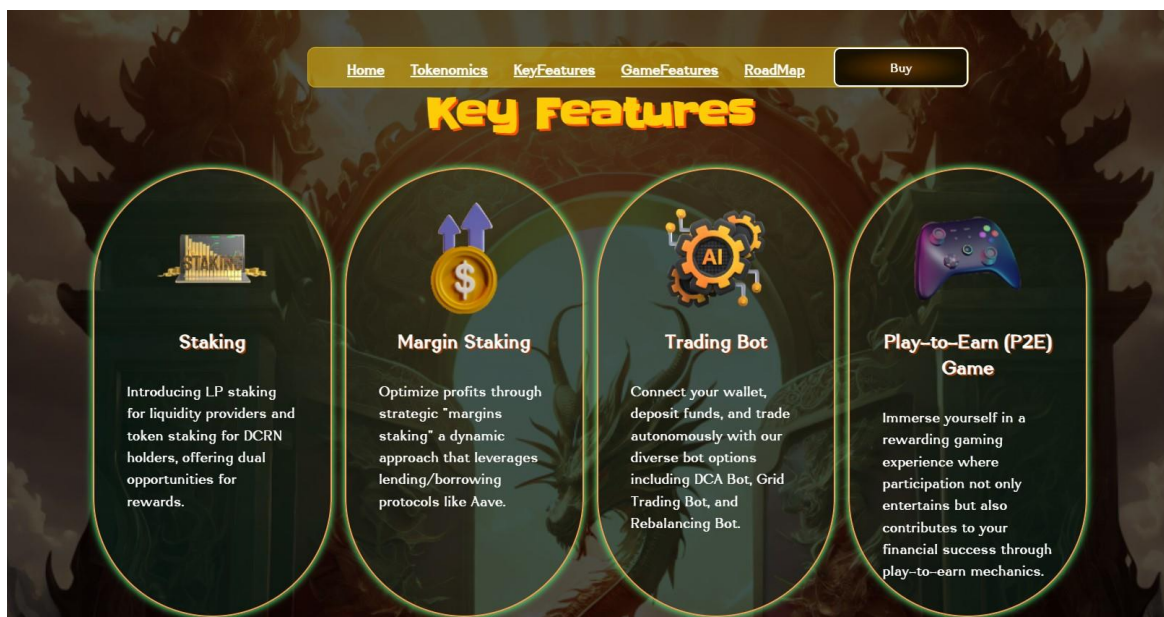
Block Chain: Arbitrum On

Standard: ERC404

Tax System: 3% sell fee / 2% buy fee:



The DRAGON CROWN Platform Is Launching On The ARB Network



The screenshot displays the 'Key Features' section of the Dragon Crown website. The navigation bar includes links for Home, Tokenomics, KeyFeatures, GameFeatures, RoadMap, and a Buy button. The main content area is titled 'Key Features' in large yellow text. Below this, there are four circular icons representing different features: Staking (a building), Margin Staking (a coin with an upward arrow), Trading Bot (a gear with 'AI'), and Play-to-Earn (P2E) Game (a game controller). Each icon is accompanied by a title and a brief description.

Feature	Description
Staking	Introducing LP staking for liquidity providers and token staking for DCRN holders, offering dual opportunities for rewards.
Margin Staking	Optimize profits through strategic "margins staking" a dynamic approach that leverages lending/borrowing protocols like Aave.
Trading Bot	Connect your wallet, deposit funds, and trade autonomously with our diverse bot options including DCA Bot, Grid Trading Bot, and Rebalancing Bot.
Play-to-Earn (P2E) Game	Immerse yourself in a rewarding gaming experience where participation not only entertains but also contributes to your financial success through play-to-earn mechanics.

Identifier	Definition	Severity
CEN-02	Initial asset distribution	Minor 

All of the initially minted assets are sent to the contract deployer when deploying the contract. This is Normal for most deployer and/or contract owner .

```
contract DCRNToken is BEP20 {
    // Addresses excluded from fees
    mapping(address => bool) public isExcludedFromFee;

    mapping(address => bool) public automatedMarketMakerPairs;

    // tax Fee Wallet address
    address public feeWallet;

    uint256 public buyFee = 2;
    uint256 public sellFee = 3;
```

RECOMMENDATION

Project stakeholders should be consulted during the initial asset distribution process.

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully.

Please refer to PAGE-7 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

The DRAGON CROWN project team understands the centralization risk. Some functions are provided privileged access to ensure a good runtime behavior in the project



References

- 1 MITRE. CWE-1041: Use of Redundant Code. <https://cwe.mitre.org/data/definitions/1041.html>.
- 2 MITRE. CWE-1099: Inconsistent Naming Conventions for Identifiers. <https://cwe.mitre.org/data/definitions/1099.html>.
- 3 MITRE. CWE-561: Dead Code. <https://cwe.mitre.org/data/definitions/561.html>.
- 4 MITRE. CWE-563: Assignment to Variable without Use. <https://cwe.mitre.org/data/definitions/563.html>.
- 5 MITRE. CWE-663: Use of a Non-reentrant Function in a Concurrent Context. <https://cwe.mitre.org/data/definitions/663.html>.
- 6 MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. <https://cwe.mitre.org/data/definitions/837.html>.
- 7 MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. <https://cwe.mitre.org/data/definitions/841.html>.
- 8 MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- 9 MITRE. CWE CATEGORY: Business Logic Errors. <https://cwe.mitre.org/data/definitions/840.html>.
- 10 MITRE. CWE CATEGORY: Concurrency. <https://cwe.mitre.org/data/definitions/557.html>.
- 11 MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- 12 OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.



Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor 

Smart contract is interacting with third party protocols e.g., Pancakeswap router, cashier contract, protections contract. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect and validate third party dependencies regularly, and mitigate severe impacts whenever necessary.



DISCLAIMERS

Vital Block Security provides the easy-to-understand audit of Solidity, Move and Raw source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. 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. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way



to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, VITAL BLOCK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, VITAL BLOCK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, VITAL BLOCK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT’S OR ANY OTHER INDIVIDUAL’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. Vital Block does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.



LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than Vital Block. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites and social accounts owners. You agree that Vital block Security is not responsible for the content or operation of such websites and social accounts and that Vital Block shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.



ABOUT VITAL BLOCK

Vital Block provides intelligent blockchain Security Solutions. We provide solidity and Raw Code Review, testing, and auditing services. We have Partnered with 15+ Crypto Launchpads, audited 50+ smart contracts, and analyzed 200,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Aptos, Oasis, etc.

Vital Block is Dedicated to Making Defi & Web3 A Safer Place. We are Powered by Security engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 5 core members, and 4+ casual contributors.

Website: <https://Vitalblock.org>

Email: info@vitalblock.org

GitHub: <https://github.com/vital-block>

Telegram (Engineering): https://t.me/vital_block

Telegram (Onboarding): https://t.me/vitalblock_cmo





vital-block



info@vitalblock.org



www.Vitalblock.org



Vital Block Dedicated to securing Public and Private Blockchain Ecosystem