PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For DracoForce

08 November 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for DracoForce on the Fantom network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	DracoForce
URL	https://dracoforce.com/
Platform	Fantom
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`DrcToken`	`0x8d05B42749428C26613deB12f8989Cb8D1f5c17f`	MATCH
`MasterChef`	`0x3D45191668dC53FFD60ea86F664716F4b320c372`	MATCH
`Timelock`	`0xD4a6440Ba658237B73ABB34c86Fd02CC273E6134`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	1	-	-	1
Low	4	1	-	3
Informational	3	-	-	3
Total	**8**	**1**	**-**	**7**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DRC Token

ID	Severity	Summary	Status
01	MEDIUM	`MaxTransactionSize` can cause transfers from important contracts such as MasterChef to fail	ACKNOWLEDGED
02	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
03	INFO	Governance functionality is broken	ACKNOWLEDGED

1.3.2 MasterChef

ID	Severity	Summary	Status
04	LOW	Rewards are calculated based on block number instead of timestamp	ACKNOWLEDGED
05	LOW	`deposit` and `withdraw` function calls will fail if the pending rewards to be minted causes the total supply to exceed the maximum supply	ACKNOWLEDGED
06	LOW	`pendingDrc` will show inaccurate pending harvests on the dApp frontend if the pending rewards causes `totalSupply` to be exceed `MAXSUPPLYCAP`	ACKNOWLEDGED
07	INFO	Initial reward emission set in constructor can be higher than `MAX_EMISSION_RATE`	ACKNOWLEDGED
08	INFO	State variables initialized in the constructor and never modified can be set to immutable	ACKNOWLEDGED

1.3.3 Timelock

No issues found.

2 Findings

2.1 DrcToken

The DRC token is a simple ERC-20 token which will be used as the main reward token for the Masterchef. It allows for DRC tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the DracoForce team. Users should therefore check that ownership has been transferred to the Masterchef. The token has a maximum supply of 8250.

There is a maximum transfer size which is modifiable by the `maxTransactionSizeOwner`, which is the deployer of the token contract. This size starts off as 0, and can be changed to only be larger than the existing size by the `maxTransactionSizeOwner`. The owner of the contract is not subjected to the size check during transfers. Any other address that attempts to make a transfer larger than the maximum transfer size will fail.

2.1.1 Token Overview

Address	0x8d05B42749428C26613deB12f8989Cb8D1f5c17f
Token Supply	8,250
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	100

2.1.2 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`
- `transferOwnership`
- `renounceOwnership`

The following functions can be called by the `maxTransactionSizeOwner` of the contract:

- `renounceMaxTransactionSizeOwner`
- `setMaxTransactionSize`

2.1.3 Issues & Recommendations

Issue #01	**`MaxTransactionSize` can cause transfers from important contracts such as MasterChef to fail**
Severity	MEDIUM SEVERITY
Description	As rewards minted in the Masterchef are transferred from the Masterchef contract to the address doing the harvesting, if the pending amount is larger than `MaxTransactionSize`, harvests as well as other functionality that do harvests (deposit/withdraw) will revert. As the pending harvest amount will keep increasing with time unless a successful harvest is done, such users will not be able to deposit, withdraw or harvest from the Masterchef, unless they do an `emergencyWithdraw` and forgo the rewards.
Recommendation	Consider adding the Masterchef as a whitelisted sender not subjected to the `MaxTransactionSize` restriction. Once the Masterchef address is set, it should not be modifiable.
Resolution	ACKNOWLEDGED

Issue #02	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED 100 tokens were pre-minted and ownership has been transferred to the Masterchef.

Issue #03	Governance functionality is broken
Severity	INFORMATIONAL
Description	Although there is YAM-related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism. It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.
Recommendation	The broken delegation-related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, used by many of the larger projects.
Resolution	ACKNOWLEDGED

2.2 MasterChef

The DracoForce Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function from Pancakeswap, which has been used maliciously to steal user's tokens. Deposit fees have been limited to at most 4%.

2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `add`
- `set`
- `updateEmissionRate`
- `updateStartBlock`
- `transferOwnership`
- `renounceOwnership`

The following functions can be called by the DevAddr of the contract:

- `setDevAddr`

The following functions can be called by the FeeAddr of the contract:

- `setFeeAddr`

2.2.2 Issues & Recommendations

Issue #04	**Rewards are calculated based on block number instead of timestamp**
Severity	LOW SEVERITY
Description	As rewards are calculated using `block.number` instead of `block.timestamp`, and block intervals are not always consistent on Fantom, it might be possible to accelerate the rewards beyond the expected emission rate by having blocks produced at a faster rate. This is a known issue for EVM-based chains such as Avalanche and Fantom. At the point of this review, it was observed that the average block time was 1 second.
Recommendation	Consider switching from calculation of rewards per block to rewards per second.
Resolution	ACKNOWLEDGED

Issue #05	**deposit and withdraw function calls will fail if the pending rewards to be minted causes the total supply to exceed the maximum supply**
Severity	LOW SEVERITY

Description	In the token contract, there is a require statement that causes the transaction to revert if the amount to be minted causes the totalSupply to exceed the MAXSUPPLY.
	
	DrcToken Line 1001:
	`function _mint(address account, uint256 amount) internal {`
	`  require(account != address(0), "BEP20: mint to the zero address");`
	`  require(_totalSupply.add(amount) <= MAXSUPPLY, "Max supply reached");`
	
	`  _totalSupply = _totalSupply.add(amount);`
	`  _balances[account] = _balances[account].add(amount);`
	`  emit Transfer(address(0), account, amount);`
	`}`
	
	This will cause all updatePool calls to fail once this happens. As deposit and withdraw function calls all include a call to updatePool, such calls will fail. In such a case, users who have already deposited would be required to call emergencyWithdraw to withdraw their funds, but will have to forgo any pending rewards which have not been harvested.
	
	Also, this could cause the MAXSUPPLYCAP to not be minted if updatePool is called when the amount to be minted causes this exceeding.

Recommendation The following change could be done to ensure that if there is an exceed of the `MAXSUPPLYCAP`, only the difference between the `MAXSUPPLYCAP` and `totalSupply` is minted.

```
uint256 drcReward =
multiplier.mul(DrcPerBlock).mul(pool.allocPoint).div(totalAl
locPoint);
uint256 devReward = drcReward.div(10);
uint256 totalRewards =
drc.totalSupply().add(devReward.add(drcReward);

if (totalRewards  <= drc.maxSupply()) {

    //  mint as normal as not at maxSupply
    drc.mint(devaddr, devReward);
    drc.mint(address(this), drcReward);

} else {

    // mint the difference only to MC, update drcReward
    drcReward= drc.maxSupply().sub(drc.totalSupply());
    drc.mint(address(this), drcReward);

}

if (drcReward != 0) {
    // only calculate and update if drcReward is non 0
    pool.accDrcPerShare = pool.accDrcPerShare.add(
        drcReward.mul(1e18).div(pool.lpSupply)
    );
}

pool.lastRewardBlock = block.number;
```

Resolution ACKNOWLEDGED

Issue #06	**pendingDrc will show inaccurate pending harvests on the dApp frontend if the pending rewards causes totalSupply to be exceed MAXSUPPLYCAP**
Severity	LOW SEVERITY
Description	Similarly to updatePool, pendingDrc does not check if the pending rewards will cause the total supply to exceed the MAXSUPPLYCAP. This can cause inaccurate pending harvests to be shown towards the end of token emissions.

Recommendation	Consider factoring in the MAXSUPPLYCAP, and set the pending reward to be the difference between MAXSUPPLYCAP and totalSupply if the pending reward causes totalSupply to be exceed MAXSUPPLYCAP.
	
	`uint256 drcReward = multiplier.mul(DrcPerBlock).mul(pool.allocPoint).div(totalAllocPoint);`
	
	`if (drc.totalSupply().add(drcReward) > drc.maxSupply()) {`
	`  drcReward = drc.maxSupply().sub(drc.totalSupply());`
	`}`
	`accDrcPerShare = accDrcPerShare.add(drcReward.mul(1e18).div(pool.lpSupply));`

Resolution	ACKNOWLEDGED

Issue #07	Initial reward emission set in constructor can be higher than `MAX_EMISSION_RATE`
Severity	INFORMATIONAL
Description	There is no check in the constructor to ensure that `DrcPerBlock` set is lesser than or equals to the `MAX_EMISSION_RATE`.
Recommendation	Consider adding the following check in the constructor: `require(_DrcPerBlock < MAX_EMISSION_RATE);`
Resolution	ACKNOWLEDGED

Issue #08	State variables initialized in the constructor and never modified can be set to `immutable`
Severity	INFORMATIONAL
Description	The following state variable(s) can be set to `immutable` as they are never changed after initialization: • `drc`
Recommendation	Add the `immutable` keyword to the above state variable.
Resolution	ACKNOWLEDGED

2.3 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	8 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	2 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Maximum Delay	30 days	The maximum delay indicates the highest value that the `delay` can be set.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.3.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY