



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Preliminary Report

For Ice Colony

12 October 2021



[paladinsec.co](https://paladinsec.co)



[info@paladinsec.co](mailto:info@paladinsec.co)

# Table of Contents

<b>Disclaimer</b>	<b>3</b>
<b>1 Overview</b>	<b>4</b>
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 IceToken	6
1.3.2 MasterChef	6
<b>2 Findings</b>	<b>7</b>
2.1 IceToken	7
2.1.1 Token Overview	7
2.1.2 Privileged Operations	8
2.1.3 Issues & Recommendations	9
2.2 MasterChef	12
2.2.1 Privileged Operations	12
2.2.2 Issues & Recommendations	13



# 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 advocacy 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 Ice Colony on the Polygon 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

<b>Project Name</b>	Ice Colony
<b>URL</b>	<a href="https://www.icecolony.com/">https://www.icecolony.com/</a>
<b>Platform</b>	Polygon
<b>Language</b>	Solidity

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
IceToken	IceToken.sol	
MasterChef	Masterchef.sol	

## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	0	-	-	-
● Low	7			
● Informational	9			
Total	16	0	0	0

### 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 IceToken

ID	Severity	Summary	Status
01	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	
02	LOW	The anti-whale limit can be set to as low as 0.5% of the total supply	
03	INFO	Governance functionality is broken	
04	INFO	delegateBySig can be frontrun and cause denial of service	
05	INFO	Typographical error in the contract	
06	INFO	Masterchef must be excluded from anti-whale	

## 1.3.2 MasterChef

ID	Severity	Summary	Status
07	LOW	Adding an EOA or a non-token contract as a pool will break updatePool and massUpdatePools	
08	LOW	Duplicated pools may be added to the Masterchef	
09	LOW	Setting devAddress, feeCommissionAddress or feeTreasuryAddress to the zero address will break deposit functionality	
10	LOW	updateEmissionRate has no maximum safeguard	
11	LOW	The pendingIce function will revert if totalAllocPoint is zero	
12	INFO	ice can be made immutable	
13	INFO	BONUS_MULTIPLIER and forTreasury are not actively used	
14	INFO	Deposits and withdrawals can revert when the hard-cap is almost reached	
15	INFO	Pools use the contract balance to figure out the total deposits	
16	INFO	Lack of events for add and set	

## 2 Findings

---

### 2.1 IceToken

The contract allows for Ice tokens to be minted when the `mint` function is called by Owner, whom at the time of deployment would be the deployer. Ownership is generally transferred to the Masterchef via the `transferOwnership` function for emission rewards to be minted and distributed to users staking in the Masterchef.

The `mint` function can be used to pre-mint tokens for various uses including injection of initial liquidity, token presale, airdrops, and others.

There are also transfer taxes that can be set to a maximum value of 15%, and anti-whales limiting transfer sizes up to 3% currently (though this can be set to a minimum of 0.5%). The developer has the ability to whitelist certain addresses to exclude them from being subject to the anti-whale limitations, as well as set certain addresses to have their own unique transfer taxes applied (again, up to 15%).

#### 2.1.1 Token Overview

<b>Address</b>	TBC
<b>Token Supply</b>	50,000 (enforced in the Masterchef)
<b>Decimal Places</b>	18
<b>Transfer Max Size</b>	Current 3% (can be set to as low as 0.5%)
<b>Transfer Min Size</b>	None
<b>Transfer Fees</b>	Up to 15%
<b>Pre-mints</b>	TBC

## 2.1.2 Privileged Operations


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

- `mint`
- `setWhitelist`
- `setWhaleDeactivate`
- `setMaxTransfer`
- `updateTransferTaxRate`
- `setDeveloperAddress`








## 2.1.3 Issues & Recommendations

<b>Issue #01</b>	<b>mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>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.</p> <p>This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the Blockchain.</p>
<b>Recommendation</b>	Consider being forthright if this mint function has been 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.
<b>Resolution</b>	

<b>Issue #02</b>	<b>The anti-whale limit can be set to as low as 0.5% of the total supply</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	The anti-whale limit can be set arbitrarily low up to 0.5% of the total supply. If the governance ever decides to do this this could seriously hamper usage for retail users and hurt the project even more in the long run.
<b>Recommendation</b>	Consider altering the lower limit of setMaxTransfer to at least 1-2% instead.
<b>Resolution</b>	

Issue #03 Governance functionality is broken	
Severity	 INFORMATIONAL
Description	<p>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.</p> <p>It should be noted that this issue is present in pretty much every single farm out there including PancakeSwap and even SushiSwap.</p>
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	

Issue #04 delegateBySig can be frontrun and cause denial of service	
Severity	 INFORMATIONAL
Description	<p>Currently if delegateBySig is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up delegateBySig transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well.</p> <p>This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.</p>
Recommendation	Similar to the broken governance functionality issue, this can just be removed.
Resolution	

Issue #05      Typographical error in the contract	
Severity	<span>INFORMATIONAL</span>
Location	<u>Line 905</u> // default tax is 7% of every transfer
Description	The comment states that default tax is 7%, but it is in fact currently 5%, with the upper limit being 15%.
Recommendation	Consider correcting this typographical error for the convenience of 3rd party reviewers.
Resolution	

Issue #06      Masterchef must be excluded from anti-whale	
Severity	<span>INFORMATIONAL</span>
Description	The Masterchef must be excluded from anti-whale, otherwise harvesting may fail if the pending rewards to be sent exceed the anti-whale limitations.
Recommendation	No resolutions required, and will be marked as Resolved once the client acknowledges this.
Resolution	



---

## 2.2 MasterChef

The IceColony Masterchef is a modified fork of the Panther Masterchef. Just like Panther, rewards can only be harvested after a specific interval (configurable to at most 14 days) has passed. The deposit fees can be split over two addresses: `feeCommissionAddress` and `feeTreasuryAddress`. Finally, deposit fees can be set to at most 5%.


### 2.2.1 Privileged Operations

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


- `add`
- `set`
- `setDevAddress`
- `setFeeCommissionAddress`
- `setFeeTreasuryAddress`
- `updateEmissionRate`




## 2.2.2 Issues & Recommendations

<b>Issue #07</b>	<b>Adding an EOA or a non-token contract as a pool will break updatePool and massUpdatePools</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	updatePool will always call <code>balanceOf(address(this))</code> on the token of this pool, and will fail if it is not an actual token contract address.
<b>Recommendation</b>	<p>Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails.</p> <pre><code>_lpToken.balanceOf(address(this));</code></pre>
<b>Resolution</b>	

<b>Issue #08</b>	<b>Duplicated pools may be added to the Masterchef</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	The add function allows for duplicate pools to be added, which would lead to dilution of emission rewards to stakers.
<b>Recommendation</b>	<p>The addition of a modifier that checks for duplicate pools could help prevent this incident from occurring.</p> <pre><code>mapping(IBE20 =&gt; bool) public poolExistence; modifier nonDuplicated(IBE20 _lpToken) {     require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");     _; }</code></pre> <p>Alternatively, you could account for this by adding in an <code>lpSupply</code> variable under <code>poolInfo</code>. This has the benefit of accurately accounting for deposits in the Masterchef.</p>
<b>Resolution</b>	

**Issue #09****Setting devAddress, feeCommissionAddress or feeTreasuryAddress to the zero address will break deposit functionality****Severity** LOW SEVERITY**Description**


Within most token contracts, minting or transferring tokens to the zero address will revert the transaction. Additionally, deposits will break if either feeCommissionAddress or feeTreasuryAddress are ever set to the zero address. Deposit-based harvests will break as well.

**Recommendation**

To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following:

```
require(_devAddress != address(0), "!nonzero");  
require(_feeCommissionAddress != address(0), "!nonzero");  
require(_feeTreasuryAddress != address(0), "!nonzero");
```

to the relevant configuration functions.

**Resolution****Issue #10****updateEmissionRate has no maximum safeguard****Severity** LOW SEVERITY**Description**


Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.


**Recommendation**


Consider adding a MAX\_EMISSION\_RATE variable and setting it to a reasonable value.

```
require(_icePerBlock <= MAX_EMISSION_RATE, "Too high");
```

**Resolution**

Issue #11	The pendingIce function will revert if totalAllocPoint is zero
Severity	 LOW SEVERITY
Description	<p>In the pendingIce function, at some point a division is made by the totalAllocPoint variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.</p>
Recommendation	<p>Consider only calculating the accumulated rewards since the lastRewardBlock if the totalAllocPoint variable is greater than zero.</p> <p>This check can simply be added to the existing check that verifies the block.number and lpSupply, like so:</p> <pre>if (block.number &gt; pool.lastRewardBlock &amp;&amp; lpSupply != 0 &amp;&amp; totalAllocPoint &gt; 0) {</pre>
Resolution	

Issue #12	ice can be made immutable
Severity	 INFORMATIONAL
Description	<p>Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers.</p>
Recommendation	<p>Consider making ice explicitly immutable.</p>
Resolution	

**Issue #13****BONUS\_MULTIPLIER and forTreasury are not actively used****Severity** INFORMATIONAL**Description**

The constant variable BONUS\_MULTIPLIER does not contain any extra information since it is constant and cannot be changed from 1. The public icePerBlock variable does indicate all information that is necessary to understand the current emission rate.


Additionally, the constant variable forTreasury is not currently being used anywhere either, as calculation in the deposit function is done based on the commission variable instead only.

**Recommendation**

Consider removing the BONUS\_MULTIPLIER and forTreasury variables.

**Resolution**



**Issue #14****Deposits and withdraws can revert when the hard-cap is almost reached****Severity** INFORMATIONAL**Location**Lines 1348-1350

```
if (IERC20(ice).totalSupply() + totalLockedUpRewards >=
MAX_SUPPLY_CAP) {
    return 0;
}
```

**Description**

Currently, the hard-cap of 50,000 tokens is enforced through the reward multiplier as above.

However, this only checks that the limit is not reached before the mint. This could mean that for example if the current supply is 49,999 tokens and there is a request to mint 2 tokens, this request will still pass since the supply is not reached and the final supply will be 50,001 tokens.

**Recommendation**

Consider not minting the tokens in case the supply exceeds the total supply. The least intrusive solution is to change:

```
ice.mint(devAddress, iceReward.div(10));
ice.mint(address(this), iceReward);
```

To

```
if (ice.totalSupply().add(iceReward.mul(11).div(10)) <=
MAX_SUPPLY_CAP) {
    // The whole emission can be mint
    ice.mint(devAddress, iceReward.div(10));
    ice.mint(address(this), iceReward);
} else if (ice.totalSupply() < MAX_SUPPLY_CAP) {
    // The emission can be partially mint
    ice.mint(address(this),
MAX_SUPPLY_CAP.sub(ice.totalSupply()));
}
```

This will only ever mint the total supply at most.

A shorter but more advanced approach could be to simply wrap all mint statements in try/catch structures. Even if the mint fails, the main transaction will still succeed.

**Resolution**

Issue #15 Pools use the contract balance to figure out the total deposits	
Severity	INFORMATIONAL
Description	<p>As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef.</p> <p>More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.</p>
Recommendation	Consider adding an <code>lpSupply</code> variable to the <code>PoolInfo</code> that keeps track of the total deposits. Each <code>lpToken.balanceOf(address(this))</code> query can then be replaced with this <code>lpSupply</code> as well.
Resolution	

Issue #16 Lack of events for add and set	
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	



**PALADIN**  
BLOCKCHAIN SECURITY