PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Ice Colony

19 October 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 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

Project Name	Ice Colony
URL	https://www.icecolony.com/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`IceToken`	`0x6ad1eEdDf1b1019494E6F78377d264BB2518db6F`	MATCH
`MasterChef`	`0xFa0A21fFCd1BB6210160582Cd9E42C7E90668F83`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	1	1	-	-
Low	7	5	-	2
Informational	8	4	1	3
Total	**16**	**10**	**1**	**5**

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	RESOLVED
02	LOW	The anti-whale limit can be set to as low as 0.5% of the total supply	RESOLVED
03	INFO	Governance functionality is broken	ACKNOWLEDGED
04	INFO	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED
05	INFO	Typographical error in the contract	RESOLVED
06	INFO	Masterchef must be excluded from anti-whale	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
07	MEDIUM	Maximum supply cap logic is wrongly defined causing the token to mint either too much or too little potentially causing deposits and withdrawals to revert	RESOLVED
07	LOW	Adding an EOA or a non-token contract as a pool will break `updatePool` and `massUpdatePools`	ACKNOWLEDGED
08	LOW	Duplicated pools may be added to the Masterchef	RESOLVED
09	LOW	Setting `devAddress`, `feeCommissionAddress` or `feeTreasuryAddress` to the zero address will break deposit functionality	RESOLVED
10	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
11	LOW	The `pendingIce` function will revert if `totalAllocPoint` is zero	ACKNOWLEDGED
12	INFO	`ice` can be made immutable	RESOLVED
13	INFO	`BONUS_MULTIPLIER` and `forTreasury` are not actively used	PARTIAL
15	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
16	INFO	Lack of events for `add` and `set`	RESOLVED

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

Address	0x6ad1eEdDf1b1019494E6F78377d264BB2518db6F
Token Supply	65,000 (enforced in the Masterchef)
Decimal Places	18
Transfer Max Size	Current 3% (can be set to as low as 0.5%)
Transfer Min Size	None
Transfer Fees	Up to 15%
Pre-mints	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

Issue #01	`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 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.
Resolution	RESOLVED 25,000 tokens have been pre-minted and ownership of the contract has been transferred to the Masterchef.

Issue #02	The anti-whale limit can be set to as low as 0.5% of the total supply
Severity	LOW SEVERITY
Description	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.
Recommendation	Consider altering the lower limit of setMaxTransfer to at least 1-2% instead.
Resolution	RESOLVED The client has increased the minimum limit to 1%.

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

Issue #04	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	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. 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.
Recommendation	Similar to the broken governance functionality issue, this can just be removed.
Resolution	ACKNOWLEDGED

Issue #05	Typographical error in the contract
Severity	INFORMATIONAL
Location	Line 905 `// 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	RESOLVED

Issue #06	Masterchef must be excluded from anti-whale
Severity	INFORMATIONAL
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	RESOLVED The client has indicated that they will do this during the deployment cycle.

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

Issue #07	**Maximum supply cap logic is wrongly defined causing the token to mint either too much or too little potentially causing deposits and withdrawals to revert**
Severity	MEDIUM SEVERITY

Location	Line 1273
	`uint256 public constant MAX_SUPPLY_CAP = 650000000000000000000;`
	
	Lines 1370-1375
	`function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) { if (IERC20(ice).totalSupply() + totalLockedUpRewards >= MAX_SUPPLY_CAP) { return 0; } return _to.sub(_from).mul(BONUS_MULTIPLIER); }`

Description	The maximum supply limit is currently included in `getMultiplier` which means it will only set the supply to zero once the supply is matched or exceeded. The latter can never happen because the token will revert any attempts that exceed the supply so it will only set the multiplier to zero if the supply has been exactly reached, which is extremely unlikely since there is no specific logic to adjust the reward rate to this. Furthermore, `totalLockedUpRewards` is added to the `totalSupply` in the check which causes double counting since these tokens have already been minted and are thus also accounted for in `totalSupply`. All this will likely cause the Masterchef to malfunction once the total supply is nearly reached, potentially causing deposits and withdrawals to revert if they try to mint amounts that would exceed the total supply. This last part would require people to use `emergencyWithdraw` until the developer sets the allocation points to zero (as `updateEmissionRate` would revert as well). ❗ The current version of `getMultiplier` also uses raw addition instead of SafeMath. As we recommend removing this section, we've not created an explicit issue for this.

Recommendation

Consider using a normal `getMultiplier` function and instead adjusting the reward rate directly in the `updatePool` function.

getMultiplier function

```
function getMultiplier(uint256 _from, uint256 _to) public
view returns (uint256) {
        return _to.sub(_from);
}
updatePool adjustment
uint256 iceReward = multiplier
    .mul(icePerBlock)
    .mul(pool.allocPoint)
    .div(totalAllocPoint);
if(ice.totalSupply() >= MAX_SUPPLY_CAP) {
    iceReward = 0;
}else if
(IERC20(ice).totalSupply().add(iceReward.mul(11).div(10))
>= MAX_SUPPLY_CAP) {
    iceReward =
(MAX_SUPPLY_CAP.sub(IERC20(ice).totalSupply()).mul(10).div(
11));
}
if(iceReward > 0) {
    IERC20(ice).mint(devAddress, iceReward.div(10));
    IERC20(ice).mint(address(this), iceReward);
    pool.accIcePerShare =
pool.accIcePerShare.add(iceReward.mul(1e18).div(lpSupply));
}
pool.lastRewardTime = block.timestamp;
```

Resolution

RESOLVED

The client has implemented the recommendation.

Issue #08	**Adding an EOA or a non-token contract as a pool will break `updatePool` and `massUpdatePools`**
Severity	LOW SEVERITY
Description	`updatePool` will always call `balanceOf(address(this))` on the token of this pool, and will fail if it is not an actual token contract address.
Recommendation	Consider simply adding a test line in the `add` function. If the token does not exist, this will make sure the `add` function fails. `_lpToken.balanceOf(address(this));`
Resolution	ACKNOWLEDGED

Issue #09	Duplicated pools may be added to the Masterchef
Severity	LOW SEVERITY
Description	The add function allows for duplicate pools to be added, which would lead to dilution of emission rewards to stakers.

Recommendation	The addition of a modifier that checks for duplicate pools could help prevent this incident from occurring.
	
	`mapping(IBEP20 => bool) public poolExistence;`
	`modifier nonDuplicated(IBEP20 _lpToken) {`
	`  require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");`
	`  _;`
	`}`
	
	Alternatively, you could account for this by adding in an `lpSupply` variable under `poolInfo`. This has the benefit of accurately accounting for deposits in the Masterchef.

Resolution	RESOLVED Although the client has added an escape mechanism to still be able to add duplicate pools, this should not be done by accident and the mechanism should clearly remind them that this will in fact have a negative effect on the rewards of these pools. We reminded the client that they should avoid abusing their escape mechanism since there is no good use to doing this.

Issue #10	**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	RESOLVED The recommended checks have been implemented.

Issue #11	**`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	RESOLVED

Issue #12	The `pendingIce` function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	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.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	ACKNOWLEDGED

Issue #13	`ice` can be made `immutable`
Severity	INFORMATIONAL
Description	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.
Recommendation	Consider making `ice` explicitly `immutable`.
Resolution	RESOLVED

Issue #14	`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	PARTIALLY RESOLVED `forTreasury` has been removed while the client has opted to leave in `BONUS_MULTIPLIER`.

Issue #15	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	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. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. Each `lpToken.balanceOf(address(this))` query can then be replaced with this lpSupply as well.
Resolution	ACKNOWLEDGED

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	RESOLVED


PALADIN
BLOCKCHAIN SECURITY