PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For IceSlush Finance

06 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 IceSlush Finance on the Avalanche 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	IceSlush Finance
URL	https://iceslush.finance/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Token`	`0x28bf9cb0a8c0a6fb01680daaae9d6ae76234491b`	MATCH
`MasterChefV2`	`0x5BFfE9747303A0144e240951f7460A61d0343467`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	2	2	-	-
Low	4	4	-	-
Informational	10	7	-	3
Total	**17**	**14**	**-**	**3**

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 Token

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	INFO	Operator modifier and functions, Uniswap interfaces, and other unused functions can be removed	ACKNOWLEDGED

1.3.2 MasterChef

ID	Severity	Summary	Status
03	HIGH	Withdrawal fees not charged in `emergencyWithdraw`	RESOLVED
04	MEDIUM	Rewards per block may result in unintended side effects	RESOLVED
05	MEDIUM	Setting `devaddr` and `feeAddress` to the zero address will break most functionality	RESOLVED
06	LOW	Adding EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`	RESOLVED
07	LOW	`updateEmissionRate` has no maximum safeguard	RESOLVED
08	LOW	The `pendingToken` function will revert if `totalAllocPoint` is zero	RESOLVED
09	INFO	`BONUS_MULTIPLIER` is redundant	RESOLVED
10	INFO	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
11	INFO	`updatePool` does not use SafeERC20	RESOLVED
12	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
13	INFO	`dev` function can be renamed	RESOLVED
14	INFO	`startBlock` can be set to a time in the past	RESOLVED
15	INFO	`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setFeeAddress` and `updateEmissionRate` functions can be made `external`	RESOLVED
16	INFO	Lack of events for `add`, `set` and `setStartBlock`	RESOLVED
17	INFO	There is a difference in the withdraw fee basis points between `add` and `set` functions	RESOLVED

2 Findings

2.1 Token

The contract allows for tokens to be minted when the `mint` function is called by the Owner, who at the time of deployment would be the deployer. However, 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.

2.1.1 Token Overview

Address	0x28bf9cb0a8c0a6fb01680daaae9d6ae76234491b
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	6,000

2.1.2 Privileged Roles

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

- `mint`
- `transferOperator`

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 tokens for dumping.
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 6,000 tokens have been pre-minted to 0xac664711285643DE163C593CE378A7129ca2dde9, and token ownership has been transferred to the Masterchef.

Issue #02	Operator modifier and functions, Uniswap interfaces, and other unused functions can be removed
Severity	INFORMATIONAL
Description	The `onlyOperator` and `transferOperator` functions do not seem to serve any purpose in the token contract, and can therefore be removed to reduce the length of the contract for third-party reviewers. This can be extended to the Uniswap-related interfaces (IUniswapV2Router02, IUniswapV2Router01, IUniswapV2Pair, IUniswapV2Factory) since they too do not seem to be used in any capacity in the token contract. Finally, the `safe32` and `getChainID` functions can be removed as they too are not used.
Recommendation	Consider removing these to reduce the length of the contract.
Resolution	ACKNOWLEDGED

2.2 MasterChef

The IceSlush 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. We commend IceSlush Finance on their decision to fork a relatively safer version of the Masterchef.

Withdrawal fees have an upper limit of 4% (which is also payable on emergency withdrawals), and transfer tax tokens have been properly accounted for using the before-after method put forth by Uniswap. Finally, token emissions have an upper limit of 1 token per block.

2.2.1 Privileged Roles

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

- `add`
- `set`
- `dev`
- `setFeeAddress`
- `updateEmissionRate`
- `setStartBlock`

2.2.2 Issues & Recommendations

Issue #03	Withdrawal fees not charged in `emergencyWithdraw`
Severity	HIGH SEVERITY
Location	Lines 228-242

```
function emergencyWithdraw(uint256 _pid) public nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 fee = 0;
    uint256 amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    if (pool.withdrawFeeBP > 0) {
        fee = user.amount.mul(pool.withdrawFeeBP).div(10000);
        amount = amount.sub(fee);
        pool.lpToken.safeTransfer(feeAddress, fee);
    }
    pool.lpToken.safeTransfer(address(msg.sender), amount);
    emit EmergencyWithdraw(msg.sender, _pid, amount);
}
```

Description	The withdrawal fee calculation in the `emergencyWithdraw` function is calculated on `user.amount`, which in previous lines had already been zeroed out. As such, when users call `emergencyWithdraw`, there are no withdrawal fees payable, and the user is thus able to receive their full deposited amount back. This can be gamed by users who would first harvest their pending rewards then call `emergencyWithdraw` to get all their funds back.
Recommendation	The withdrawal fees should instead be calculated on `amount` rather than `user.amount`, like so: `fee = amount.mul(pool.withdrawFeeBP).div(10000);`
Resolution	RESOLVED

Issue #04	Rewards per block may result in unintended side effects
Severity	MEDIUM SEVERITY
Description	The Masterchef calculates rewards per block which is an anti-pattern on the Avalanche network as it may pose unintended side effects with token emissions and APR displays given that Avalanche has variable block times. The issue is detailed here. Emissions may thus be erratic and potentially gamed, per this tweet by Yield Yak.
Recommendation	Consider replacing all reward calculations from `block.number` to be based on `block.timestamp`.
Resolution	RESOLVED

Issue #05	Setting `devaddr` and `feeAddress` to the zero address will break most functionality
Severity	MEDIUM SEVERITY
Description	Within the token contract, minting or transferring tokens to the zero address will revert the transaction. Withdrawals and emergency withdrawals will break if the `feeAddress` is ever set to the zero address. Harvesting will fail as well.

Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like:
	`require(devaddr != address(0), "nonzero!");`
	`require(feeAddress != address(0), "nonzero!");`
	to the configuration functions.

Resolution	RESOLVED

Issue #06	**Adding EOA or 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 to do so if it is not a token contract.
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	RESOLVED

Issue #07	**`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. Although there is the `maxEmissionRate` variable, it is not currently used in the function.
Recommendation	Consider using the `maxEmissionRate` variable to place an upper limit on the setting of emission rates, like so: `require(_tokenPerBlock <= maxEmissionRate,"Too high");`
Resolution	RESOLVED There is now an upper limit of 1 token per second.

Issue #08	The pendingToken function will revert if `totalAllocPoint` is zero
Severity	LOW SEVERITY
Description	In the `pendingToken` 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	RESOLVED The client has opted to use `if (lpSupply == 0 || totalAllocPoint == 0) return 0;`

Issue #09	`BONUS_MULTIPLIER` is redundant
Severity	INFORMATIONAL
Description	The constant variable `BONUS_MULTIPLIER` does not look to be used in the Masterchef contract and can thus be removed.
Recommendation	Consider removing `BONUS_MULTIPLIER` and associated comments.
Resolution	RESOLVED

Issue #10	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.
Resolution	ACKNOWLEDGED

Issue #11	`updatePool` does not use SafeERC20
Severity	INFORMATIONAL
Location	Line 176 `token.transfer(0x000000000000000000000000000000000000dEaD, burnAmount);`
Description	It is considered best practice to use OpenZeppelin's SafeERC20 for any token transfer operations. Alternatively, tokens can just be minted to the burn address for the same effect.
Recommendation	Consider using `safeTransfer` instead of `transfer` in the function.
Resolution	RESOLVED

Issue #12	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `accTokenPerShare` is based on the `lpSupply` variable. `pool.accTokenPerShare = pool.accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value, precision errors may occur due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to `1e18` across the entire contract.
Resolution	ACKNOWLEDGED

Issue #13	**dev function can be renamed**
Severity	INFORMATIONAL
Description	Users may be confused on what the `dev` function does.
Recommendation	Consider renaming to `setDevAddress`.
Resolution	RESOLVED

Issue #14	`startBlock` can be set to a time in the past
Severity	INFORMATIONAL
Description	The `setStartBlock` function can be called to set `startBlock` and `lastRewardBlock` to a point in the past, which may have unintended side effects.
Recommendation	Consider revamping the `setStartBlock` function to ensure that the `startBlock` can only be moved to a future period, like so: `function setStartBlock(uint256 _startBlock) external onlyOwner { require(block.number < startBlock, "It's too late to postpone mining. It has already started"); require(block.number < _startBlock, "cannot set start block in the past"); uint256 length = poolInfo.length; for (uint256 pid = 0; pid < length; ++pid) { PoolInfo storage pool = poolInfo[pid]; pool.lastRewardBlock = _startBlock; } startBlock = _startBlock; emit UpdateStartBlock(startBlock);`
Resolution	RESOLVED

Issue #15	`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `dev`, `setFeeAddress` and `updateEmissionRate` functions can be made `external`
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making these functions `external`.
Resolution	RESOLVED

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

Issue #17	**There is a difference in the withdraw fee basis points between add and set functions**
Severity	INFORMATIONAL
Description	The add function allows deposit fees to be set to a maximum value of 4%, while set only allows changes to the deposit fees to a maximum value of 3%.
Recommendation	Consider aligning the two functions to avoid setting at an unintended value because of the restriction.
Resolution	RESOLVED Both add and set have a 4% withdrawal fee limit.


PALADIN
BLOCKCHAIN SECURITY