PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For SpikeSwap

11 January 2022

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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for SpikeSwap on the Binance Smart Chain. 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	SpikeSwap
URL	https://spikeswap.exchange/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`ySpikeToken`	`0x9145BDFeb48e0B5d1B649EED068E80f209889f6d`	MATCH
`MasterChef`	`0x0d6BD191CFEA27e3B6375Fdc12248d9769b5B61C`	MATCH
`Referral`	`0xc9B32f92390548D4A6d6A222eb9fb4E044E9E481`	MATCH
`Timelock`	`0x0A20A8c297D058B34610E15a0B33af35CD8F2afe`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4	4	-	-
Medium	3	3	-	-
Low	9	7	-	2
Informational	9	3	-	6
Total	**25**	**17**	**-**	**8**

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 ySpikeToken

ID	Severity	Summary	Status
01	HIGH	`updateYspikeRouter` could be used to siphon all liquidity fees or turn the token into a honeypot	RESOLVED
02	HIGH	LP tokens from `addLiquidity` are sent to operator	RESOLVED
03	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
04	LOW	BNB dust after adding liquidity will accumulate in the contract and not be retrievable	RESOLVED
05	LOW	Inconsistency between usage of `_msgSender()` and `msg.sender`	RESOLVED
06	INFO	Unnecessary updating of `amount` in `_transfer` to `sendAmount`	RESOLVED
07	INFO	Public functions can be made external	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
08	HIGH	Pools other than the ySpike token pool do not support fee on transfer tokens and can result in exploitation of referral commission minting and loss of user funds	RESOLVED
09	HIGH	`harvestTime` and `startBlockHarvest` can be arbitrarily changed by the owner to result in loss of pending rewards	RESOLVED
10	MEDIUM	Lack of maximum upper limit for deposit fees	RESOLVED
11	MEDIUM	Setting `devAddress` to the zero address results in `updatePool` reverting	RESOLVED
12	MEDIUM	Adding an address that does not implement the IBEP20 interface will cause `massUpdatePools` and `updatePool` to fail	RESOLVED
13	LOW	`feeAddr` can be set to the zero address	RESOLVED
14	LOW	Owner can change `yspikeReferral` to gain commissions on all users	RESOLVED
15	LOW	Duplicated pools may be added to the Masterchef	RESOLVED
16	LOW	Invalid referral contract address can cause referral related functions to revert	RESOLVED
17	LOW	`updateEmissionRate` will run out of gas if there are too many pools	ACKNOWLEDGED
18	INFO	Emissions will not be reduced if `updateEmissionRate` is not called	ACKNOWLEDGED
19	INFO	Usage of finney is not recommended	ACKNOWLEDGED
20	INFO	Rounding issue due to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
21	INFO	Pool uses the contract balance to figure out the total deposits	ACKNOWLEDGED
22	INFO	Lack of event emission in functions that change sensitive state	ACKNOWLEDGED
23	INFO	Public functions can be made external	RESOLVED
24	INFO	Non-modifiable state variables can be declared as immutable	ACKNOWLEDGED

1.3.3 Referral

ID	Severity	Summary	Status
25	LOW	Referral addresses for users without referral records can be arbitrarily set by operator	ACKNOWLEDGED

1.3.4 Timelock

No issues found.

2 Findings

2.1 ySpikeToken

The ySpike token is an ERC-20 token which will be used as the main reward token for the Masterchef. It allows for ySpike 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 ySpike team.

There is a transfer tax on the token that is set at an initial 7.5% and can be set to a maximum of 10%, which is imposed on all transfers except in the following cases:

- Recipient is burn address
- Tax rate is set to 0%

A portion of the transfer tax, set at an initial 30%, will be sent to the burn address if the `burnRate` is activated. The remaining transfer tax amount will be used for adding liquidity.

2.1.1 Token Overview

Address	0x9145BDFeb48e0B5d1B649EED068E80f209889f6d
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	Up to 10%
Pre-mints	20

2.1.2 Privileges

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

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

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

- `updateTransferTaxRate`
- `updateBurnRate`
- `updateMinAmountToLiquify`
- `updateSwapAndLiquifyEnabled`
- `updateYspikeRouter`
- `transferOperator`

2.1.3 Issues & Recommendations

Issue #01	**`updateYspikeRouter` could be used to siphon all liquidity fees or turn the token into a honeypot**
Severity	HIGH SEVERITY
Description	The owner can update the router that generates liquidity to an address or contract of choice. This contract could be a malicious contract that simply keeps the tokens sent to it or, a contract that prevents selling by reverting when called.
Recommendation	Consider allowing this function to be called once. `require(yspikeRouter == address(0), "Router set already");` If this is not possible, consider a significantly longer timelock as the owner so that users can have sufficient time to react to future router changes.
Resolution	RESOLVED The built in liquidity provision feature has been removed.

Issue #02	LP tokens from `addLiquidity` are sent to operator
Severity	HIGH SEVERITY

Location	Line 176~
	`yspikeRouter.addLiquidityETH{value: ethAmount}(`
	`address(this),`
	`tokenAmount,`
	`0, // slippage is unavoidable`
	`0, // slippage is unavoidable`
	`operator(),`
	`block.timestamp`
	`);`

Description	The LP tokens provided from `addLiquidity` are sent to the operator, who will be able to decompose the LPs and obtain the underlying tokens.
Recommendation	Consider setting the "to" address for adding of liquidity to a burn address, ensuring that the received liquidity pool tokens are locked forever.
Resolution	RESOLVED The built in liquidity provision feature has been removed.

Issue #03	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	This function could be used to pre-mint tokens before ownership is transferred to the Masterchef contract. This could have a significant impact on the tokenomics of the project.
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 20 tokens were pre-minted and ownership has been transferred to the Masterchef.

Issue #04	**BNB dust after adding liquidity will accumulate in the contract and not be retrievable**
Severity	LOW SEVERITY
Description	When ySpike/BNB liquidity is added, there could be some BNB which is leftover as dust. Subsequent adding of liquidity does not use this BNB balance. This BNB balance would grow over time and be stuck in the token contract as there is no way to transfer the BNB out from the token contract.
Recommendation	Consider adding an `onlyOperator` access controlled function to allow withdrawing of any BNB balances in the token contract.
Resolution	RESOLVED The built in liquidity provision feature has been removed.

Issue #05	Inconsistency between usage of `_msgSender()` and `msg.sender`
Severity	LOW SEVERITY
Description	There are instances in the code where `msg.sender` is used, and others where `_msgSender()` is used. This inconsistency could cause some issues as `_msgSender()` returns `msg.sender` for regular transactions, but for meta transactions, it can be used to return the end user instead of the relayer
Recommendation	Consider using `_msgSender()` to replace all instances of `msg.sender`.
Resolution	RESOLVED Instances of `msg.sender` have been replaced with `_msgSender()` in the token contract.

Issue #06	Unnecessary updating of `amount` in `_transfer` to `sendAmount`
Severity	INFORMATIONAL

Location	Line 115~
	`super._transfer(sender, BURN_ADDRESS, burnAmount);`
	`super._transfer(sender, address(this), liquidityAmount);`
	`super._transfer(sender, recipient, sendAmount);`
	`amount = sendAmount;`

Description	After all the `super._transfer` calls are done, amount is set to `sendAmount`, but is not used after that.
Recommendation	Remove the `amount = sendAmount` statement as it is unnecessary, to reduce gas costs.
Resolution	RESOLVED The highlighted line has been removed.

Issue #07	Public functions can be made external
Severity	INFORMATIONAL
Description	The following functions can be changed from public to external. - `updateTransferTaxRate` - `updateBurnRate` - `updateMinAmountToLiquify` - `updateSwapAndLiquifyEnabled` - `updateYspikeRouter` - `transferOperator` 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 (https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices)
Recommendation	Change the visibility of the above functions to external.
Resolution	RESOLVED The above functions have been made external.

2.2 MasterChef

The ySpike Masterchef is a fork of Goose Finance's Masterchef with some changes made. A notable feature of forking the latter is the removal of the `migrator` function from Sushiswap, which can be used maliciously to steal tokens staked in the Masterchef. ySpike currently allows the deposit fee to be set up to 100%.

There is a `harvestTime` which is initially set at 28800 seconds, or 8 hours, after the `startBlockHarvest`.

Each account that deposits can also register a referral address. If registered, a referral commission on the referred user's pending harvest of up to 20%, initially set at 3%. These tokens will be minted additionally on top of the Masterchef rewards. The dev address will also receive an additional 10% of minted Masterchef rewards.

Emissions will decrease over time until a minimum of 0.1 ySpike tokens per block. This is done every 9600 blocks, or 8 hours, by 3%, by calling the `updateEmissionRate`. The function can be called by anyone.

2.2.1 Privileges

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

- `add`
- `set`
- `updateHarvestTime`
- `updateStartBlockHarvest`
- `setYspikeReferral`
- `setReferralCommissionRate`
- `transferOwnership`
- `renounceOwnership`

The following functions can be called by the DevAddr:

- `setDevAddr`

The following functions can be called by the FeeAddr:

- `setFeeAddr`

2.2.2 Issues & Recommendations

Issue #08	**Pools other than the ySpike token pool do not support fee on transfer tokens and can result in exploitation of referral commission minting and loss of user funds**
Severity	HIGH SEVERITY
Description	Although there is a specific check that subtracts the `transferTax` from the `_amount` during deposit only for the ySpike token pool, other pools do not perform this check. If there are any other fee on transfer tokens being used, it will result in the draining of the token balance as the Masterchef will credit more of the token since it uses the `_amount` parameter supplied in the deposit function instead of actual received tokens. A malicious actor can target such pools, but depositing and withdrawing large amounts to reduce the contract's token balance. This will result in an inflated `accYspikePerShare` due to a small denominator, and in turn higher pending rewards. The higher pending rewards will also affect the amount of referral commission tokens minted, allowing the malicious actor to obtain large amounts of minted tokens which can then be sold for a profit.
Recommendation	Use the actual value received by the Masterchef instead of the amount specified by the user during the deposit.

```
if (_amount > 0) {
    uint256 balanceBefore = pool.lpToken.balanceOf(address(this));
    pool.lpToken.safeTransferFrom(msg.sender, address(this), _amount);
    _amount = pool.lpToken.balanceOf(address(this)) - balanceBefore;

    if (pool.depositFeeBP > 0) {
        uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
        pool.lpToken.safeTransfer(feeAddress, depositFee);
        user.amount = user.amount.add(_amount).sub(depositFee);
    } else {
        user.amount = user.amount.add(_amount);
    }
}
```

Resolution	RESOLVED
	The code in deposit has been updated to account for fee on transfer tokens.
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.safeTransferFrom(msg.sender, address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)) - balanceBefore;`

Issue #09	**harvestTime and startBlockHarvest can be arbitrarily changed by the owner to result in loss of pending rewards**
Severity	HIGH SEVERITY

Location	Line 310
	`uint256 lastBlockHarvest = startBlockHarvest.add(harvestTime);`
	`if (block.number >= startBlockHarvest && block.number <= lastBlockHarvest) {`

Description	In `payOrLockupPendingYspike`, the `lastBlockHarvest`, which determines if a user can harvest, is based on the sum of `startBlockHarvest` and `harvestTime`. Although `startBlockHarvest` and `harvestTime` can be modified by the owner, if it remains unchanged after contract deployment, the `lastBlockHarvest` value will always be the same value of the deployment block height + 28800 blocks. This means that users can only harvest if the current block height is between `startBlockHarvest` and `lastBlockHarvest`. Any other calls to `payOrLockupPendingYspike` at a block height outside that range would result in a loss of pending rewards as `user.rewardDebt` would be updated, but the user would not receive any rewards. The owner could set startBlockHarvest to a very early block height to prevent users from harvesting, and losing their pending rewards.
Recommendation	Confirm the behavior of harvest lockups, if it is to be based on the sum of `startBlockHarvest` or the block timestamp when harvest is attempted. In the case it is the latter, the following change can be done to ensure that users will each have their own lockup based on the last harvest attempt, and not lose their pending rewards.

```
function canHarvest(uint256 _pid, address _user) public view returns (bool) {
    UserInfo storage user = userInfo[_pid][_user];
    return block.number >= user.nextHarvestUntil;
}

function payOrLockupPendingYspike(uint256 _pid) internal {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    if (user.nextHarvestUntil == 0) {
        // if user deposits before start block
        if (block.number <= startBlock) {
            user.nextHarvestUntil = startBlock.add(harvestTime);
        }
        else {
           user.nextHarvestUntil = block.number.add(harvestTime);
        }
    }

    uint256 pending =
user.amount.mul(pool.accYspikePerShare).div(1e12).sub(user.rewardDebt);
    if (canHarvest(_pid, msg.sender)) {
        if (pending > 0 || user.rewardLockedUp > 0) {
            uint256 totalRewards = pending.add(user.rewardLockedUp);

            // reset lockup
            totalLockedUpRewards =
totalLockedUpRewards.sub(user.rewardLockedUp);
            user.rewardLockedUp = 0;
            user.nextHarvestUntil = block.number.add(harvestTime);

            // send rewards
            safeYspikeTransfer(msg.sender, totalRewards);
                payReferralCommission(msg.sender, totalRewards);
        }
    } else if (pending > 0) {
        user.rewardLockedUp = user.rewardLockedUp.add(pending);
        totalLockedUpRewards = totalLockedUpRewards.add(pending);
        emit RewardLockedUp(msg.sender, _pid, pending);
    }
}
```

This change would require an additional `nextHarvestUntil` field to the UserInfo struct, which would need to be set to 0 on `emergencyWithdraw`.

Also, `harvestTime` would require a reasonable maximum limit in the setter function to ensure that the lockup for rewards would be reasonable.

Resolution

RESOLVED

The harvest lockup feature has been removed.

Issue #10	Lack of maximum upper limit for deposit fees
Severity	MEDIUM SEVERITY
Description	The deposit fees for each pool can be modified by the owner, up to 100%. This can lead to unsuspecting users depositing into pools that have actual deposit fees differing from those shown in the web interface, resulting in the complete loss of deposited funds.
Recommendation	Consider changing the maximum limit of deposit fees to something reasonable. A common maximum upper limit by similar projects is somewhere around 4%. This change has to be made in both add and set functions.
Resolution	RESOLVED The maximum limit has been changed in add and set: `require(_depositFeeBP <= 400, "set: invalid deposit fee basis points");`

Issue #11	**Setting devAddress to the zero address results in updatePool reverting**
Severity	MEDIUM SEVERITY
Description	As a portion of rewards are minted to `devAddress` in the `updatePool` function, and minting to the zero address will result in a revert in the token contract, all functions that call `updatePool`, namely `deposit`, `withdraw` and `massUpdatePools`, will revert.
Recommendation	Add a non-zero address check in `setDevAddress`. `require(_devAddress != address(0));`

Resolution	RESOLVED
	The recommended check has been added.
	`function setDevAddress(address _devAddress) external {`
	`  require(msg.sender == devAddress, "setDevAddress: FORBIDDEN");`
	`  require(_devAddress != address(0), "setDevAddress: cannot set dev adddres to zero address");`
	`  devAddress = _devAddress;`
	`}`

Issue #12	**Adding an address that does not implement the IBEP20 interface will cause `massUpdatePools` and `updatePool` to fail**
Severity	MEDIUM SEVERITY
Description	If the owner adds an address that does not have the IBEP20 interface implemented, any functions that call the pool's token address will revert. This would result in `massUpdatePools` being unusable as `updatePool` will be called for the affected pool id and revert. Not being able to call `massUpdatePools` would result in new pool additions affecting existing pool pending rewards as `massUpdatePools` is required to be called each time an addition or change is made to any pool to ensure that existing pools get their fair due rewards before the allocation among pools is changed.
Recommendation	Add a sanity check in the add function when adding a token to a new pool. `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED The recommended sanity check has been added in add.

Issue #13	`feeAddr` can be set to the zero address
Severity	LOW SEVERITY
Description	`setFeeAddress` does not check if the address to be set for `feeAddr` is non-zero. This can result in potential issues with pools with deposit fees and tokens that do not allow transfers to the zero address, resulting in deposit transactions into such pools reverting.
Recommendation	Add a non-zero address check in `setFeeAddress`. `require(_feeAddress != address(0));`

Resolution	RESOLVED
	The recommended check has been added.
	`function setFeeAddress(address _feeAddress) external {`
	`    require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");`
	`    require(_feeAddress != address(0), "setFeeAddress: cannot set fee adddres to zero address");`
	`    feeAddress = _feeAddress;`
	`}`

Issue #14	**Owner can change `yspikeReferral` to gain commissions on all users**
Severity	LOW SEVERITY
Description	In `setYspikeReferral`, there is no check that the new referral contract address implements the required functions. This can result in functions like deposit and `payReferralCommission` reverting.
Recommendation	If there is no requirement to change the referral contract after it is set, the following check can be done to ensure that the referral contract can only be set once, at the start of the `setYspikeReferral` function. `require(yspikeReferral == address(0), “ref already set”);`
Resolution	RESOLVED The recommended check has been added to allow setting of `yspikeReferral` only once. `require(address(yspikeReferral) == address(0), "setYspikeReferral: ySpike referral already set");`

Issue #15	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: duplicate pool");`
	`_;`
	`}`
	Note that within the `add` function, `poolExistence[_lpToken]` needs to be set to true to indicate that the pool exists.
	Alternatively, you could account for this by adding in an `lpSupply` variable under `poolInfo`. This has the benefit of accounting for accurately accounting for deposits in the Masterchef.

Resolution	RESOLVED The modifier has been added to prevent adding of the same token to a pool more than once.

Issue #16	Invalid referral contract address can cause referral related functions to revert
Severity	LOW SEVERITY
Description	In `setYspikeReferral`, there is no check that the new referral contract address implements the required functions. This can result in functions like deposit and `payReferralCommission` reverting.
Recommendation	A sanity check can be added to call an expected function of the referral contract. `_yspikeReferral.getReferrer(address(this));`
Resolution	RESOLVED The sanity check has been added.

Issue #17	`updateEmissionRate` will run out of gas if there are too many pools
Severity	LOW SEVERITY
Description	As `updateEmissionRate` calls `massUpdatePools` to update all the pools, and unlike `add` and `set` which have an option to skip the `massUpdatePools` call, if there are too many pools which result in `massUpdatePools` running out of gas, `updateEmissionRate` might not be able to successfully execute. This will result in the emission not being able to decrease even if it has passed the period making it eligible for decreasing.
Recommendation	Add a bool `_withUpdate` parameter to allow skipping of `massUpdatePools` if necessary.
Resolution	ACKNOWLEDGED

Issue #18	Emissions will not be reduced if `updateEmissionRate` is not called
Severity	INFORMATIONAL
Description	Even after the time for emission reduction has passed, if `updateEmissionRate` is not called, the emissions will remain the same.
Recommendation	The ySpike team has to ensure that `updateEmissionRate` is called in a timely manner so that emissions will be reduced.
Resolution	ACKNOWLEDGED

Issue #19	Usage of finney is not recommended
Severity	INFORMATIONAL
Location	Line 69 `uint256 public constant MINIMUM_EMISSION_RATE = 100 finney;`
Description	The denominations finney and szabo have been removed in Solidity compiler version 0.7.0. While this does not affect the contract which is using a compiler version below 0.7.0, it would be advisable to not use the finney denomination in case this contract would be used with higher compiler versions in the future.
Recommendation	The emission rate can be changed to use other denominations like ether.
Resolution	ACKNOWLEDGED

Issue #20	Rounding issue due to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accYspikePerShare` is based upon the `lpSupply` variable. `pool.accYspikePerShare = pool.accYspikePerShare.add(yspikeReward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value, this will cause precision errors due to rounding. This is observed when in pools that support tokens with large supplies.
Recommendation	Consider increasing precision to `1e18` across the entire contract.
Resolution	ACKNOWLEDGED

Issue #21	Pool uses the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	The total number of tokens in the contract is used to determine the total number of deposits. This can cause dilution of rewards if there is a staking pool with the same token as the reward token.
Recommendation	Consider adding an `lpSupply` field to the `PoolInfo` that keeps track of the total deposits. This should be incremented in deposit by the amount received by the contract, and decremented in `withdraw` and `emergencyWithdraw` by the amount transferred out.
Resolution	ACKNOWLEDGED The ySpike team has mentioned that they will not have any pool that has the same token as the reward token.

Issue #22	Lack of event emission in functions that change sensitive state
Severity	INFORMATIONAL
Description	The following functions change the state of contract but do not emit any events: - `add` - `set` - `setYspikeReferral` - `setReferralCommissionRate`
Recommendation	Emit events for the above functions.
Resolution	ACKNOWLEDGED

Issue #23	Public functions can be made external
Severity	INFORMATIONAL
Description	The following functions can be changed from `public` to `external`: - `add` - `set` - `deposit` - `withdraw` - `emergencyWithdraw` - `setDevAddress` - `setFeeAddress` - `updateEmissionRate` - `updateHarvestTime` - `updateStartBlockHarvest` - `setYspikeReferral` - `setReferralCommissionRate` 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 (https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices)
Recommendation	Change the visibility of the above functions to external.
Resolution	RESOLVED The above functions have been made external.

Issue #24	Non-modifiable state variables can be declared as immutable
Severity	INFORMATIONAL
Description	The following state variables can be set as constant as they are defined inside of the constructor and never modified: - `yspike` - `startBlock`
Recommendation	Declare these state variables as immutable.
Resolution	ACKNOWLEDGED

2.3 Referral

The Referral contract stores and returns referrer information for addresses that deposit in the Masterchef contract. This information is used for the rewarding of referral commission on harvests.

2.3.1 Privileges

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

- `updateOperator`
- `renounceOwnership`
- `transferOwnership`

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

- `recordReferral`

2.3.2 Issues & Recommendations

Issue #25	**Referral addresses for users without referral records can be arbitrarily set by operator**
Severity	LOW SEVERITY
Description	The `recordReferral` function is used by the operator role to record the referrer address of a referred user. As this is a role that can be held by multiple addresses, the owner of Referral can add an address other than MasterChef into the role. This address can then set the referrer of user addresses that have not been set to an arbitrary address, thus earning the commission fees of those users' harvests. This allows stealing up to 20% of the user harvest.
Recommendation	Make the Masterchef the sole operator and renounce ownership afterwards. In this case, there is no way for the operators to arbitrarily update the referrals for users. Alternatively, remove the mapping and only use a single address as the variable for operator.
Resolution	ACKNOWLEDGED The ySpike team has stated that they will renounce the ownership after setting Masterchef as the lone operator. This issue will be marked as resolved once this has been done.

2.4 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. This contract should be the owner of the Masterchef contract to time delay making sensitive changes such as adding a new pool, or changing the allocation for an existing pool.

Parameter	Value	Description
Delay	24 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	6 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.
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.

2.4.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY