PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ApeSwap (MasterApeV2)

16 January 2023

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 MasterApeV2 6

1.3.2 RewarderTimer 7

1.3.3 MasterApeAdminV2 7

1.3.4 ContractWhitelist 7

2 Findings 8

2.1 MasterApeV2 8

2.1.1 Privileged Functions 9

2.1.2 Issues & Recommendations 10

2.2 RewarderTimer 19

2.2.1 Privileged Functions 19

2.2.2 Issues & Recommendations 20

2.3 MasterApeAdminV2 25

2.3.1 Privileged Functions 26

2.3.2 Issues & Recommendations 27

2.4 ContractWhitelist 30

2.4.1 Privileged Functions 30

2.4.2 Issues & Recommendations 31

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 ApeSwap's MasterApeV2 contracts on the BNB 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	ApeSwap (MasterApeV2)
URL	https://apeswap.finance/
Platform	BNB Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MasterApeV2`	`0x71354AC3c695dfB1d3f595AfA5D4364e9e06339B`	MATCH
`RewarderTimer`	`Not yet deployed`	UNDEPLOYED
`MasterApeAdminV2`	`0xC8C355f475c4925C362a424b28fF4030A419377c`	MATCH
`ContractWhitelist`	`Dependency`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	3	3	-	-
Low	12	10	-	2
Informational	10	10	-	-
Total	**26**	**24**	**-**	**2**

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 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 MasterApeV2

ID	Severity	Summary	Status
01	MEDIUM	`updatePool` can in certain cases return early without updating the last reward time, granting excessive rewards retroactively if the emission-rate is ever re-enabled	✓ RESOLVED
02	LOW	`updateEmissionRate` appears to order its calls in the wrong order, allowing adjusted emissions to be granted retroactively	✓ RESOLVED
03	LOW	`pendingBanana` will revert if `totalAllocPoint` is zero	✓ RESOLVED
04	LOW	Contract does not fully safeguard against reentrancy	✓ RESOLVED
05	LOW	getPoolInfo emits parameters in the wrong order	✓ RESOLVED
06	LOW	Owner of the dummy token can allocate the BANANA tokens emitted by the V1 contract to themself	ACKNOWLEDGED
07	LOW	`massUpdatePools` might run out of gas eventually and make it impossible to call `updateEmissionRate`	✓ RESOLVED
08	INFO	EOAs can deposit on behalf of a contract and bypass `ContractWhitelist`	✓ RESOLVED
09	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	✓ RESOLVED
10	INFO	Lack of validation	✓ RESOLVED
11	INFO	Typographical errors	✓ RESOLVED
12	INFO	Gas optimizations	✓ RESOLVED
13	INFO	`SafeMath` is unnecessary from Solidity version 0.8 onwards	✓ RESOLVED

1.3.2 RewarderTimer

ID	Severity	Summary	Status
14	HIGH	updatePool uses lpSupply after deposits and withdrawals which allows for reward manipulation	✓ RESOLVED
15	MEDIUM	Governance risk: onReward can revert in multiple ways, preventing users from withdrawing from the MasterApeV2	✓ RESOLVED
16	MEDIUM	An exploiter can prevent pool addition by calling updatePool prematurely	✓ RESOLVED
17	LOW	getSecondsToEmptyRewards appears to be incorrect	✓ RESOLVED
18	LOW	pendingToken will revert if totalAllocPoint is zero	✓ RESOLVED
19	LOW	set grants rewards retroactively	✓ RESOLVED
20	INFO	Typographical errors	✓ RESOLVED

1.3.3 MasterApeAdminV2

ID	Severity	Summary	Status
21	LOW	Unnecessary on-chain operations	ACKNOWLEDGED
22	INFO	Typographical errors	✓ RESOLVED
23	INFO	Gas optimization	✓ RESOLVED

1.3.4 ContractWhitelist

ID	Severity	Summary	Status
24	LOW	Contract lacks on-chain enumerability for whitelisted contracts	✓ RESOLVED
25	LOW	Toggle functions are undesired as they may cause annoyance during retries	✓ RESOLVED
26	INFO	Gas optimization	✓ RESOLVED

2 Findings

2.1 MasterApeV2

`MasterApeV2` is an update to the `MasterApeV1` Masterchef contract. It will serve the same purpose while providing additional functionalities, such distributing additional tokens in addition to BANANA tokens. As `MasterApeV1` is the only address able to mint BANANA tokens, the `MasterApeV2` contract will be pulling tokens from the V1 contract by staking a dummy token into it.

`MasterApeV2` distributes BANANA tokens to users staking tokens on it. These tokens are expected to be Liquidity Provider tokens that are minted when providing liquidity to ApeSwap pools. Once a LP token has been added to the farm list, any user can deposit this token on the contract and receive BANANA tokens over time, at the rate decided by the owner. Rewards tokens are sent at every deposit and withdrawal. The total rewards distributed are limited to 3.3 BANANA per seconds maximum (10 BANANA per block).

Farms are allocated a portion of the total emission of BANANA tokens. These allocation points can be updated at any time to increase, decrease or shut down rewards for a particular farm. The global emission rate can also be updated but is limited to the emission rate of the `MasterApeV1`, which controls the creation of BANANA tokens.

A deposit fee can be set on a farm. On deposit, a portion of the staked tokens are taken and sent to an address defined by the owner of the contract. They will not be taken into account in the tokens staked by the user. The maximum possible deposit fee for any individual pool is set at 10%.

Deposits are limited to individual users (EOAs) but can also be opened to a set of whitelisted contracts.

To cap the supply of BANANA tokens, the team has introduced a hardcap variable that will limit the amount of BANANA tokens minted. Once this limit is reached, all farm emissions will shut down. This hardcap variable can be updated by the owner.

In addition to the BANANA rewards, a Rewarder contract can be linked to a farm to distribute an additional token on top of the BANANA rewards. This Rewarder must comply with a defined interface but can use different reward mechanisms. The `RewarderTimer` that can be found below in the report is an example of a rewarder.

2.1.1 Privileged Functions

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

- `initialize`
- `setFeeAddress`
- `updateEmissionRate`
- `setPendingMasterApeOwner`
- `acceptMasterApeOwnership`
- `updateHardCap`
- `add`
- `set`
- `transferOwnership`
- `renounceOwnership`
- `toggleWhitelist`
- `setContractWhitelist`
- `setBatchContractWhitelist`

2.1.2 Issues & Recommendations

Issue #01	**updatePool can in certain cases return early without updating the last reward time, granting excessive rewards retroactively if the emission-rate is ever re-enabled**
Severity	MEDIUM SEVERITY
Location	Line 343 `if (totalBanana == 0) return;`
Description	The `updatePool` function returns early without updating the last reward time. This can most notably cause severe issues when rewards are re-enabled on a pool. In this case, the new reward rate will be fully granted over the historical period. Issue scenario: 1. Governance disables emissions on all pools (by simply setting the reward rate to zero) 2. Governance waits a year before deciding to re-enable them 3. `totalBanana == 0` over that period and `updatePool` returns early on the re-enable. 4. The new reward rate over the whole year is granted to all stakers
Recommendation	Consider always updating the `lastRewardTime` and whether this early return is necessary. Also consider simplifying `updatePool`.
Resolution	RESOLVED

Issue #02	**updateEmissionRate appears to order its calls in the wrong order, allowing adjusted emissions to be granted retroactively**
Severity	LOW SEVERITY

Location	Line 204-206
	`function updateEmissionRate(uint256 _bananaPerSecond) external onlyOwner {`
	`  _updateEmissionRate(_bananaPerSecond);`
	`  massUpdatePools();`
	`}`

Description	The `massUpdatePools` call occurs after the emissions have been adjusted. Typically, this call is done before to ensure that the previous rate is used up to this point.
Recommendation	Consider explaining why these lines were re-ordered. If this was by accident, consider first updating the pools before updating the emissions.
Resolution	RESOLVED `massUpdatePools` has been moved before `_updateEmissionRate`.

Issue #03	**pendingBanana will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	Within the `pendingBanana` function, a division is made by the `totalAllocPoint` variable at some point. 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 `lastRewardTime` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.timestamp` and `stakeSupply`, like so: `if (block.timestamp > pool.lastRewardTime && stakeSupply != 0 && totalAllocPoint != 0) {`
Resolution	RESOLVED

Issue #04	Contract does not fully safeguard against reentrancy
Severity	LOW SEVERITY
Description	Although the critical functions are all safeguarded from reentrancy, a malicious actor can still re-enter from one of these functions into the rest of the system. Most notably, `updatePool` and `harvestFromMasterApe` can still be re-entered into.
Recommendation	Consider marking these functions as `nonReentrant` and `external`, then moving their bodies to internal functions which are unguarded to still allow calling them from the other guarded functions.
Resolution	RESOLVED

Issue #05	`getPoolInfo` emits parameters in the wrong order
Severity	LOW SEVERITY
Description	`getPoolInfo` emits its return values in a different order than it advertises.
Recommendation	Consider re-ordering either the function description or the actual contents.
Resolution	RESOLVED

Issue #06	**Owner of the dummy token can allocate the BANANA tokens emitted by the V1 contract to themself**
Severity	LOW SEVERITY
Description	The MasterApeV2 deposits a dummy token into MasterApeV1 in order to pull BANANA tokens for rewards. This dummy token must be deposited on the V2 contract before calling initialize but nothing prevents the owner of the token to mint more and deposit it on the V1 contract on his behalf, taking a part of the rewards for himself.
Recommendation	Make sure to burn ownership of the dummy token or at least keep it on a secured wallet like a multi-signature contract.
Resolution	ACKNOWLEDGED The client acknowledged this issue with the following comment: —— Our approach for the dummy token is to create a bare-bones, owner-less ERC-20 token with a supply of 1e18. - Find the dummy token here: MasterApeV2 Farm Token - 1e18 shows as a supply of 1 on block explorers to send a message that this is a unique token with a unique purpose.

Issue #07	**`massUpdatePools` might run out of gas eventually and make it impossible to call `updateEmissionRate`**
Severity	LOW SEVERITY
Description	`massUpdatePools` loops on every pool added on the contract. The more pools added, the more gas it will require to execute and it could reach the maximum gas limit of the chain, making it impossible to call. As this function is called when updating the emission rate, it could prevent `updateEmissionRate` from being called for good.
Recommendation	Gas limit for a transaction varies between chains. While it is pretty high on BSC, it is much lower on other chains, and extra care should be taken if someone is deploying the contract on these chains.
Resolution	RESOLVED `massUpdatePools` is now optional on `updateEmissionRate`.

Issue #08	**EOAs can deposit on behalf of a contract and bypass `ContractWhitelist`**
Severity	INFORMATIONAL
Description	The deposit functions still allow depositing into smart contracts. This might not be desired by the team as they attempted to prevent deposits by smart contracts.
Recommendation	Consider whether this is desired. If not, consider only allowing whitelisted contracts to call depositTo with a different recipient.
Resolution	RESOLVED The client has indicated that this is fine.

Issue #09	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and `pendingRewards`, `accBananaPerShare` is based upon the `stakeSupply` variable. `accBananaPerShare = accBananaPerShare.add(bananaReward.mul(1e12).div(stakeSupply));` However, if this `stakeSupply` has a severely large value, this will cause precision errors 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. It should be noted that even a precision of 1e18 has been considered small when tokens like PolyDoge were added to masterchefs of our client. In case the client thinks it is probable that such tokens will be added, we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED The precision has been increased to 1e18..

Issue #10	Lack of validation
Severity	INFORMATIONAL

Description	The contract contains functions with parameters which are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase.
	
	Consider validating the following function parameters:
	
	Line 146
	`bananaPerSecond = bananaPerSecond_;`
	
	This parameter should be validated to be lower than the cap.
	
	Line 195
	`function setFeeAddress(address _feeAddress) external {`
	
	This address should be validated to be non-zero, as transfers to it might start failing if it is ever set to the zero address.

Recommendation	Consider validating the function parameters mentioned above.
Resolution	RESOLVED

Issue #11	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 77
	`address public pendingMasterApeV1Owner = address(0);`
	
	This is an unnecessary initialization.
	
	Line 84
	`///  @notice 40 BANANAs per block in MAV1`
	
	It is in fact set to 10 Banana per block.
	
	Line 185
	`pool.stakeToken.safeTransfer(address(msg.sender), amount);`
	
	Line 392
	`pool.stakeToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
	
	The cast to `address` is unnecessary in these locations.
	
	Line 265
	`// TODO: Look into this (probably bad PCS coding and can be fixed)`
	
	A TODO note has been left.
	
	`add` and `set` can be marked as `external`. `_safeBananaTransfer`, `_updateEmissionRate` and `_updateHardCap` can be marked as `private`.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #12	Gas optimizations
Severity	INFORMATIONAL

Description	We have consolidated the sections which can be further optimized for gas usage below.
	
	Line 188
	`pool.rewarder.onReward(_pid, msg.sender, msg.sender, 0, user.amount);`
	
	The final parameter can be zero here to avoid a storage read.
	
	Lines 383-406 and 423-438
	
	In the `depositTo` and `WithdrawTo` functions, `user.amount` is read several times from storage and could be cached.
	
	Lines 391-393
	`uint256 preStakeBalance = pool.stakeToken.balanceOf(address(this));`
	`pool.stakeToken.safeTransferFrom(address(msg.sender), address(this), _amount);`
	`finalDepositAmount = pool.stakeToken.balanceOf(address(this)) - preStakeBalance;`
	
	`pool.stakeToken` is read several times from storage and could be cached.
	
	Line 447
	`if (banana.totalSupply() >= hardCap && masterApe.BONUS_MULTIPLIER() > 0) {`
	
	`totalSupply` is already cached and can be re-used.
	
	Lines 452 and 459
	`banana.transfer(address(0x0000000000000000000000000000000000000dEaD), surplus);`
	
	`address` does not need to be re-cast here.

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

Issue #13	`SafeMath` is unnecessary from Solidity version 0.8 onwards
Severity	INFORMATIONAL
Description	Throughout the contract, SafeMath is used to protect the contracts against integer overflows. However, from Solidity version 0.8, such protections are baked into the standard math operators of Solidity. Therefore, it is no longer necessary to use SafeMath in the current version of the codebase. Using SafeMath in this version will slightly increase gas usage.
Recommendation	Consider removing the SafeMath dependency and reverting to standard math operators throughout all contracts starting from version 0.8. Initializable seems to also be removable as it could be merged into the constructor.
Resolution	RESOLVED

2.2 RewarderTimer

`RewarderTimer` distributes an additional token to farms added to `MasterApeV2`. Each `RewarderTimer` has its token set on contract initialization. Reward tokens need to be sent to the contract.

To initiate rewards for a farm, the farm ID needs to be added to the `RewarderTimer` set, and the rewarder address needs to be set on the `MasterApeV2`. `onReward` will be called on every deposit and withdrawal to send the additional rewards.

Rewards on the `RewarderTimer` are distributed proportional to the time staked, same as the BANANA rewards from `MasterApeV2`. Rewards end after a defined date that can be updated by the owner.

It should be noted that governance can take out the reward token at any point in time via `sweepTokens`.

2.2.1 Privileged Functions

The following functions can be called by the owner and other priviledged roles of the contract:

- `onReward [ MasterApeV2 ]`
- `setBonusEndSeconds`
- `setRewardPerSecond`
- `add`
- `set`
- `sweepTokens`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #14	**`updatePool` uses `lpSupply` after deposits and withdrawals which allows for reward manipulation**
Severity	HIGH SEVERITY
Description	`updatePool` uses the new `lpSupply` instead of the old one. This allows for reward manipulation: 1. Flashloan 1 million tokens and deposit into pool 1 2. `updatePool` generates rewards assuming the `totalSupply` is 1 million tokens. All rewards generated since the last `updatePool` are effectively lost. Or worse: 1. Deposit 100 tokens into the pool 2. Wait 1 year. 3. Withdraw 99 tokens from the pool The pool now generates rewards based on the fact that 1 token has been staked for a year. The user effectively managed to 100x their eligible rewards.
Recommendation	Consider passing the `oldSupply` as a parameter.
Resolution	RESOLVED

Issue #15	**Governance risk: onReward can revert in multiple ways, preventing users from withdrawing from the MasterApeV2**
Severity	MEDIUM SEVERITY

Description	onReward is always called within the MasterApeV2. If this function were to revert, withdrawals on the MasterApe would therefore revert as well until the governance changes the rewarder of the pool.
	
	We have identified the following cases which will cause onReward to revert:
	
	Line 219
	`uint256 reward = time.mul(rewardPerSecond).mul(pool.allocPoint) / totalAllocPoint;`
	
	This multiplication can overflow and revert. This division can revert through division by zero. Consider checking `allocPoint`, consider limiting the `rewardPerSecond` and maximum `allocPoint`.
	
	Line 277
	`rewardToken.safeTransfer(_to, _amount);`
	
	This transfer can revert if there are insufficient tokens. Consider attempting the transfer or much more safely consider using pull-based harvesting where the rewarder needs to be harvested explicitly after a per-user pending rewards balance has incremented.

Recommendation	Consider implementing the recommendations mentioned after each issue.
Resolution	RESOLVED Rewards can be pulled directly without any deposit or withdrawal. The token transfer in onReward is attempted only if enough tokens are available.

Issue #16	**An exploiter can prevent pool addition by calling `updatePool` prematurely**
Severity	MEDIUM SEVERITY
Description	The `add` function is only allowed for pools which do not have a `lastRewardTime` yet. However, anyone can call `updatePool` on a non-added pool to force this pool to have a `lastRewardTime`. Governance will then be unable to add this pool to the pools array.
Recommendation	Consider moving the pools array to an `EnumerableSet` and checking on contains instead. Consider only allowing existing pools to be updated or even making these functions internal. Consider `set` to only be callable on existing pools.
Resolution	RESOLVED Only existing pools can be updated.

Issue #17	**`getSecondsToEmptyRewards` appears to be incorrect**
Severity	LOW SEVERITY
Description	The `getSecondsToEmptyRewards` function presently does not appear to indicate the time it takes to empty the reward balance. `getUnharvestedRewards` seems to also be able to underflow right now due to the fact that the rewarder can over-distribute right now.
Recommendation	Consider removing this function if it is unused. Consider refactoring this function to match the real intention. Consider explaining the real intention to us if you believe it is implemented correctly.
Resolution	RESOLVED The function has been fixed.

Issue #18	pendingToken will revert if totalAllocPoint is zero
Severity	LOW SEVERITY
Description	Within 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 lastRewardTime if the totalAllocPoint variable is greater than zero. This check can simply be added to the existing check that verifies the block.timestamp and stakeSupply, like so: `if (block.timestamp > pool.lastRewardTime && stakeSupply != 0 && totalAllocPoint != 0) {`
Resolution	RESOLVED

Issue #19	set grants rewards retroactively
Severity	LOW SEVERITY
Description	The set function applies the new allocation points retroactively.
Recommendation	Consider prefacing this function with a massUpdatePools call (can be made optional with a _withUpdate parameter).
Resolution	RESOLVED

Issue #20	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 38
	`using SafeMath for uint128;`
	
	This line appears to be unused and can be removed.
	
	Line 43
	`/// \`rewardDebt\` The amount of rewards entitled to the user.`
	
	This is not what `rewardDebt` means.
	
	Line 77
	`// Keep track of number of reward tokens paid to find remaining reward balance`
	
	This comment appears to be copied.
	
	Line 189
	`token.transfer(_to, balance);`
	
	`safeTransfer` should be used here.
	
	`getSecondsToEmptyRewards` and `rewardBalance` can be made external. `_safeTransferRewardInternal` and `_getLpSupply` can be made `private`.
	
	Finally, we do not see the benefit of the initializer. Why not use a constructor instead? What about the ownership of this contract, should it not be set in the initializer?

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

2.3 MasterApeAdminV2

`MasterApeAdminV2` is meant to be the owner of `MasterApeV2`. It is used as a proxy to manage the MasterApe contract, as well as providing helper functions to add farms in batches and keep farm allocations on a fixed percentage of the total allocations.

Farms can be added to a `fixedPercentFarm` set that will keep allocations to a defined percent of the total tokens allocated. When updating a farm, the owner can call `_syncFixedPercentFarms` to adjust the allocations of the fixed percentage farms to match their requested allocation percent.

It splits the functions' permissions in two: a `farmAdmin` that can manage the configuration of the farms, while the owner of the contract still has admin rights.

2.3.1 Privileged Functions

The following functions can be called by various privileged roles of the contract:

- `setPendingMasterApeOwner [owner]`
- `setPendingMasterApeV1Owner [owner]`
- `updateEmissionRate [owner]`
- `updateHardCap [owner]`
- `toggleWhitelist [owner]`
- `setContractWhitelist [owner]`
- `setBatchContractWhitelist [owner]`
- `sweepTokens [owner]`
- `transferOwnership [owner]`
- `renounceOwnership [owner]`
- `transferFarmAdminOwnership [farmAdmin]`
- `syncFixedPercentFarms [farmAdmin]`
- `addMasterApeFarms [farmAdmin]`
- `setMasterApeFarms [farmAdmin]`
- `addFixedPercentFarmAllocation [farmAdmin]`
- `setFixedPercentFarmAllocation [farmAdmin]`

2.3.2 Issues & Recommendations

Issue #21	Unnecessary on-chain operations
Severity	LOW SEVERITY
Description	`MasterApeAdminV2` implements a set of functions aiming to keep some farms with a fixed percentage of allocation. This requires some calculations that could be done off-chain especially since farms can be updated in batches, thus allowing to update all the farms at the same time with no periods of discrepancy where farms are not balanced. In addition, fixed percentages allocations are not enforced in any way, as farms can be updated through `setMasterApeFarms` without calling `_syncFixedPercentFarms`.
Recommendation	Consider doing these calculations off-chain
Resolution	ACKNOWLEDGED

Issue #22	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 55
	`address public pendingMasterApeOwner = address(0);`
	
	Unnecessary initialization.
	
	Line 70
	`uint256 public constant MAX_BONUS_MULTIPLIER = 4;`
	
	This variable seems to be unused.
	
	Line 206
	`/// @notice Add a batch of farms to the MasterApe contract`
	
	This should be `Set` instead of `Add` on `setMasterApeFarms`.
	
	Line 229
	`// Set all pids with no update`
	
	This should be "with no mass update".
	
	Line 345-346
	`poolBananaPerDay = poolBananaPerSecond * 3600 * 24;`
	`poolBananaPerMonth = poolBananaPerDay * 30;`
	
	For better readability, native time variables like "1 days" should be used.

	Line 363
	`function getTotalAllocationPercent() public view returns (uint256) {`
	`  return totalFixedPercentFarmPercentage;`
	`}`
	
	`totalFixedPercentFarmPercentage` is already a public variable.
	
	The `_syncFixedPercentFarms` and `_removeFromArray` functions should be marked as `private`.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #23	**Gas optimization**
Severity	INFORMATIONAL

Location	Line 177~
	`function addMasterApeFarms(`
	`  uint256[] memory _allocPoints,`
	`  IERC20[] memory _stakeTokens,`
	`  uint16[] memory _depositFeesBP,`
	`  IRewarder[] memory _rewarders,`
	`  bool _withMassPoolUpdate,`
	`  bool _syncFixedPercentageFarms`
	`) external onlyFarmAdmin {`

Description	Functions inputs should be stored in `calldata`
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	RESOLVED

2.4 ContractWhitelist

`ContractWhitelist` is a dependency used by ApeSwap to only allow specific contracts to interact with their contracts.

The whitelist is disabled by default, but when enabled, only whitelisted contracts can interact with the contracts which extend this whitelist (note that functions need to be flagged explicitly with `checkEOAorWhitelist` for them to be guarded).

The owner can toggle the whitelist with the `toggleWhitelist` function. Contracts can be added and removed by the owner with either the `setContractWhitelist` or `setBatchContractWhitelist` functions.

Contract developers should note that even though they might be whitelisted today, the ApeSwap team retains the right to revoke the whitelist at a later date.

2.4.1 Privileged Functions

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

- `toggleWhitelist`
- `setContractWhitelist`
- `setBatchContractWhitelist`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #24	Contract lacks on-chain enumerability for whitelisted contracts
Severity	LOW SEVERITY
Description	It is presently extremely difficult for users and the team itself to figure out all the contracts which are whitelisted. To do so, users need to either go over all historical events or go through all owner transactions, neither of which is exactly easy to do.
Recommendation	Consider using an `EnumerableSet` for the whitelist and adding a `length` and `at` function to iterate over the full set.
Resolution	RESOLVED

Issue #25	Toggle functions are undesired as they may cause annoyance during retries
Severity	LOW SEVERITY
Location	Line 50-53
Description	Toggle functions are typically not used within Solidity smart contracts as sometimes transactions stuck in the mempool are re-tried by an impatient user by simply adding another transaction at a later nonce. With a toggle function, the latter transaction would untoggle the first transaction if both get confirmed.
Recommendation	Consider refactoring the function to `setWhitelistEnabled(bool)`.
Resolution	RESOLVED

```
function toggleWhitelist() external virtual onlyOwner {
      whitelistEnabled = !whitelistEnabled;
      emit UpdateWhitelistStatus(whitelistEnabled);
}
```

Issue #26	Gas optimization
Severity	INFORMATIONAL
Location	Line 65 `function setBatchContractWhitelist(address[] memory _addresses, bool[] memory _enabled) external onlyOwner {`
Description	The parameters can be moved to `calldata` to save gas.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY