PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ApeSwap (Banana Allocator)

02 October 2022

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 BananaAllocator 6

1.3.2 SweeperUpgradeable and Sweeper 6

2 Findings 7

2.1 BananaAllocator 7

2.1.1 Privileged Functions 8

2.1.2 Issues & Recommendations 9

2.2 SweeperUpgradeable 14

2.2.1 Privileged Functions 15

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 Banana Allocator 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 (Banana Allocator)
URL	https://apeswap.finance
Network	BNB Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`BananaAllocator`	`Proxy: 0xa44b8b1fae0dc8004380831ed77be445810ee7e8 Implementation: 0xa3743e40365df159b461777877b66ee1b02e10bb`	MATCH
`SweeperUpgradeable`	`Dependency`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	5	5	-	-
Informational	7	6	1	-
Total	**12**	**11**	**1**	**-**

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 BananaAllocator

ID	Severity	Summary	Status
01	LOW	Lack of validation	RESOLVED
02	LOW	`addBananaRoute` contains a suboptimal if-statement	RESOLVED
03	LOW	Contract lacks `emergencyWithdraw` functionality	RESOLVED
04	INFO	Codebase contains more unprivileged functions than is strictly needed	PARTIAL
05	INFO	Typographical errors	RESOLVED
06	INFO	Gas optimizations	RESOLVED
07	INFO	Lack of event for `changeMinimumBanana`	RESOLVED

1.3.2 SweeperUpgradeable and Sweeper

ID	Severity	Summary	Status
08	LOW	Declaration of uninitialized memory pointers can theoretically lead to side-effects	RESOLVED
09	LOW	`Ownable` is not initialized in the `SweeperUpgradeable` contract	RESOLVED
10	INFO	Typographical errors	RESOLVED
11	INFO	Gas optimizations	RESOLVED
12	INFO	Lack of events for `refuseNativeSweep` and `_lockToken`	RESOLVED

2 Findings

2.1 BananaAllocator

`BananaAllocator` is used to move tokens from one address to another, either on the same blockchain or on a different blockchain. The contract allows the owner to add, remove, and modify "banana routes" which dictate where the tokens should be sent and under what conditions. The contract also has methods for staking and unstaking tokens in the `MasterApe` contract.

`BananaAllocator` extends the `SweeperUpgradeable`, which is described within this audit as well.

2.1.1 Privileged Functions

- `addBananaRoute`
- `removeBananaRoute`
- `stakeFarm`
- `unstakeFarm`
- `sweepTokens`
- `changeMinimumBanana`
- `sweepNFTs`
- `sweepTokensAndNFTs`
- `sweepNative`
- `refuseNativeSweep`
- `lockToken`
- `lockTokens`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Lack of validation
Severity	LOW SEVERITY

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 86
	`function moveBanana(uint256 pid, bool revert_) private {`
	
	The `pid` parameter is not validated to have a configured route. In case chains exist where the BANANA token can be transferred to the zero address, this could result in BANANA tokens being burned when tokens are staked while a route has not been configured yet.
	
	Line 120
	`uint256 actionId,`
	
	It is desirable to make the `actionId` an `enum` — this will automatically validate the value of the parameter and better communicate to auditors what the values mean.
	
	Line 127
	`getBananaRouteFromPid[farmPid] = BananaRoute(toAddress, actionId, chainId, minimumBanana);`
	
	The `farmPid` is not validated to be unique and could overwrite the previous `farmPid`. Consider validating that the `farmPid` has not been already set, or use an enumerable set which would be even cleaner.

Recommendation	Consider validating the function parameters mentioned above.
Resolution	RESOLVED An enumerable set is now used instead of an array.

Issue #02	addBananaRoute contains a suboptimal if-statement
Severity	LOW SEVERITY

Location	Line 125
	`if (chainId == 0 || chainId == currentChainId)`
	`require(chainId != 1, "BananaAllocator: wrong settings");`

Description	This `if` statement is rather strange: the requirement will never occur if the `chainId` is zero for example. The current definition of this requirement seems to be that if the origin chain is Ethereum, the tokens must be bridged out. We wonder if this is desired.
Recommendation	Consider rewriting this `if` statement into a simple requirement which explicitly indicates why the setting is wrong and does not contain unnecessary clauses.
Resolution	RESOLVED

Issue #03	Contract lacks `emergencyWithdraw` functionality
Severity	LOW SEVERITY
Description	In certain instances, withdrawing from a Masterchef is only possible through the `emergencyWithdraw` function, such as when an issue occurs with the reward calculating logic (overflow, division by zero...). The BananaAllocator will presently need to be upgraded in such an instance as there is no functionality to call the `emergencyWithdraw` function.
Recommendation	Consider adding an `onlyOwner` function to `emergencyWithdraw`.
Resolution	RESOLVED

Issue #04	Codebase contains more unprivileged functions than is strictly needed
Severity	INFORMATIONAL
Description	An important best practice within Solidity is to not provide more unprivileged functions than are strictly needed within a codebase. There is no direct benefit for users to be able to call the various move banana functions. It is therefore best practice to put these various functions behind a guarded modifier (eg. `onlyMover`) to reduce the attack vectors for exploiters. No significant exploits were found. The only malicious action that seems executable by unprivileged users is to call `moveBananaPid` while the route has not been configured. Depending on the BANANA implementation, such a call might succeed (specifically if it does not revert on transfers to the zero address).
Recommendation	Consider making all functions privileged. Consider validating that a route exists when a `pid` is provided.
Resolution	PARTIALLY RESOLVED The functions are not privileged, but they validate that the route exists.

Issue #05	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 4
	`import "@openzeppelin/contracts/access/Ownable.sol";`
	
	This import is unused and can be removed.
	
	Line 139
	`delete (getBananaRouteFromPid[pid]);`
	
	This statement contains unnecessary brackets.
	
	Various address variables and parameters can be typed explicitly as `IERC20`, `IMasterApe` and `IAnyswapV4Router`.
	
	The following functions should be marked as external:
	- `bananaRoutesLength`
	- `moveBananaIndex`
	- `moveBananaAll`
	- `moveBananaPid`
	- `moveBananaPids`

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #06	Gas optimizations
Severity	INFORMATIONAL
Description	We have consolidated the sections which can be further optimized for gas usage below. Line 66 `for (uint256 index = 0; index < bananaRoutePids.length; index++) {` Consider caching the length in order to save gas. The `currentChainId` is currently stored and loaded every time it's needed. This is an expensive operation. Consider removing this variable and use `block.chainId` directly instead as it costs close to nothing. This has an additional benefit that if the chain ever gets forked and the chainId changed, it will have the new one.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

Issue #07	Lack of event for `changeMinimumBanana`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	RESOLVED

2.2 SweeperUpgradeable

`SweeperUpgradeable` is used to sweep ERC20 tokens and NFTs from a contract. The contract has a mapping of `lockedTokens` which keeps track of which tokens are locked and cannot be transferred. Once locked, tokens can no longer be unlocked, which means the governance privilege is permanently renounced. The contract also has a boolean value, `allowNativeSweep`, which keeps track of whether native coin sweep is allowed.

The contract has four sweep functions: `sweepTokens`, `sweepNFTs`, `sweepTokensAndNFTs`, and `sweepNative`. The `sweepTokens` function is used to transfer ERC20 tokens to the contract owner. The `sweepNFTs` function is used to transfer NFTs to the contract owner. The `sweepTokensAndNFTs` function is used to transfer both ERC20 tokens and NFTs to the contract owner. The `sweepNative` function is used to transfer native coins to the contract owner. The `refuseNativeSweep` function is used to refuse native coin sweep, after which it cannot be re-enabled.

The `SweeperUpgradeable` is inherited by the `BananaAllocator`. All issues and privileges within this audit section are therefore present within that contract as well.

It should be noted that the contract is upgradeable — this means that the tokens are not really permanently locked and the native token is not really permanently prevented from being sweeped. This is because the proxy admin can at any point upgrade the contract with new functionality that bypasses the lock.

Note that other token types like ERC1155 are presently not swipeable.

2.2.1 Privileged Functions

- `sweepTokens`
- `sweepNFTs`
- `sweepTokensAndNFTs`
- `sweepNative`
- `refuseNativeSweep`
- `lockToken`
- `lockTokens`
- `transferOwnership`
- `renounceOwnership`

Issue #08	Declaration of uninitialized memory pointers can theoretically lead to side-effects
Severity	LOW SEVERITY

Description	The codebase defines empty arrays by simply not initializing them. This is not recommended as it leads to a dangling memory pointer.
	
	`NFT[] memory empty;`
	`IERC20[] memory empty;`
	
	This poses a potential risk which is illustrated in the following code snippet:
	
	`contract Test {`
	`  function test() external view returns (uint256) {`
	`  assembly {`
	`  mstore(msize(), 5);`
	`  }`
	`  uint256[] memory empty;`
	`  return empty.length;`
	`  }`
	`}`
	
	In this code-snippet, the function returns a length of 5. However, if we replace this snippet with an initialized array, it returns a length of zero:
	
	`uint256[] memory empty = new uint256[](0);`
	
	Though we expect this issue to not pose a risk as long as no assembly is used, we are unable to prove this and therefore strongly recommend initializing the arrays.

Recommendation	Consider initializing the empty arrays:
	
	`NFT[] memory empty = new NFT[](0);`
	`…`
	`IERC20[] memory empty = new IERC20[](0);`

Resolution	RESOLVED

Issue #09	**Ownable is not initialized in the `SweeperUpgradeable` contract**
Severity	LOW SEVERITY
Description	The owner is not transferred to the `msg.sender` in the `SweeperUpgradeable` contract. This prevents anyone from calling the privileged function, even the owner. This issue is only raised as Low as it is initialized in the parent contract.
Recommendation	Consider calling `Ownable__init()` in the `initializeSweeper` function. Consider also marking the `initializeSweeper` function as `onlyInitializing`.
Resolution	RESOLVED

Issue #10	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Line 2
	`pragma solidity 0.8.15.0;`
	
	This pragma can probably be simplified to 0.8.15 as subversions are highly uncommon. It is not strictly a problem from the looks of it as it still compiles.
	
	Line 19
	`* @dev Sweep any ERC20 token`
	
	The contract also sweeps NFTs.
	
	SweeperUpgradeable::37
	`function initializeSweeper(address[] memory _lockedTokens, bool _allowNativeSweep) internal`
	
	This function should be annotated with an `onlyInitializing` modifier.
	
	`* @dev Transfers erc20 tokens to owner`
	
	These tokens (alongside with the NFTs in the other functions) are not necessarily transferred to the owner, as the owner can pick any recipient.
	
	`token.transfer(to, balance);`
	
	`safeTransfer` can be used here to make sweeping compliant with tokens that don't return a bool on transfer.
	
	`_to.transfer(balance);`
	
	Replacing this with `_to.call{value: balance}()` makes this section compliant with more recipient contracts.

```
@dev Lock single token so they can't be transferred from the
contract.

* @dev Lock single token so they can't be transferred from
the contract.
```

These two lines reference a singular token, "they" should be replaced with "it".

```
* Once locked it can't be unlocked

* Once locked it can't be unlocked
```

These lines refer to multiple tokens, "it" should be replaced by "they".

The following functions should be marked as external:

- `sweepTokens`
- `sweepNFTs`
- `sweepNative`
- `refuseNativeSweep`
- `lockToken [ only in SweeperUpgradeable ]`

Recommendation Consider fixing the typographical errors.

Resolution RESOLVED

Issue #11	Gas optimizations
Severity	INFORMATIONAL

Description	`lockTokens(_lockedTokens);`
	
	`sweepTokensAndNFTs(tokens, empty, to);`
	
	`sweepTokensAndNFTs(empty, nfts, to);`
	
	`lockToken(tokens[i]); // From Sweeper::126`
	
	The `onlyOwner` modifier is called multiple times within these sections of code, resulting in wasted gas.

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

Issue #12	Lack of events for `refuseNativeSweep` and `_lockToken`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the functions.
Resolution	RESOLVED However, the `RefusedNativeSweep` event is not emitted if `_allowNativeSweep` is set to `false` in the `initializeSweeper` function.


PALADIN
BLOCKCHAIN SECURITY