



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

For ApeSwap IA0

30 January 2022



[paladinsec.co](https://paladinsec.co)



[info@paladinsec.co](mailto: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 IAO	6
1.3.2 IAOLinearVesting	6
2 Findings	7
2.1 IAO	7
2.1.1 Privileged Roles	8
2.1.2 Issues & Recommendations	9
2.2 IAOLinearVesting	17
2.2.1 Privileged Roles	17
2.2.2 Issues & Recommendations	18

# 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 advocacy 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 IAO contracts 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

<b>Project Name</b>	ApeSwap (IAO)
<b>URL</b>	<a href="https://apeswap.finance/">https://apeswap.finance/</a>
<b>Platform</b>	Binance Smart Chain
<b>Language</b>	Solidity

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
IAO	IAO.sol	
IAOLinearVesting	IAOLinearVesting.sol	
Resolution	<a href="https://github.com/ApeSwapFinance/apeswap-iao/tree/feature/linear-vesting-audit/contracts">https://github.com/ApeSwapFinance/apeswap-iao/tree/feature/linear-vesting-audit/contracts</a>	

## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	2	1	-	1
● Low	3	3	-	-
● Informational	7	6	1	-
Total	12	10	1	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 IAO

ID	Severity	Summary	Status
01	MEDIUM	Gov privilege: Governance can withdraw all deposited tokens and offering tokens at any time which could lead to loss of all users funds	ACKNOWLEDGED
02	MEDIUM	Lack of reentrancy guard on before-after deposit allows for deposit value exploits if the ERC-20 token allows for reentrancy	RESOLVED
03	LOW	Lack of usage of upgradeable dependencies	RESOLVED
04	LOW	Deposited amount could still be zero for fee-on-transfer tokens causing the contract to malfunction	RESOLVED
05	LOW	Best practice: Contract unnecessarily rounds in multiple locations	RESOLVED
06	INFO	setOfferingAmount, setRaisingAmount and userTokenStatus can be made external	PARTIAL
07	INFO	Lack of events for setOfferingAmount, setRaisingAmount and finalWithdraw	RESOLVED
08	INFO	harvest does not adhere to checks-effects-interactions	RESOLVED
09	INFO	Limiting the gas usage of calls can be dangerous across hard forks that affect gas usage	RESOLVED
10	INFO	Typographical errors	RESOLVED

## 1.3.2 IAOLinearVesting

ID	Severity	Summary	Status
11	INFO	harvest lacks checks-effects-interactions for refunded portion	RESOLVED
12	INFO	Lack of validation	RESOLVED

## 2 Findings

---

### 2.1 IAO

The Initial Ape Offering (IAO) contract is a seed funding contract which allows users to provide a specific staking token to a launching project. In return, the user will receive a portion of the project's tokens. Each IAO has a total offering amount which is the amount of project tokens for sale. It also has a total raising amount which is the amount of tokens the project is looking to raise. If this amount is exceeded, users will be refunded by the amount of deposited tokens that exceed the raising amount. In case the raising amount is not reached, users receive offering tokens at the predetermined price.

The contract allows for both the native gas token (eg. BNB) and ERC-20 contributions, although one specific token must be chosen for each IAO.

The contract specifies a specific start block after which users can contribute their tokens and an end block after which the payouts and refunds occur.

Instead of providing users with all offering tokens all at once, users receive these in four vesting periods with each period unlocking a quarter of the tokens. The first period immediately unlocks at the end of the IAO, which means the initially unlocked tokens is 25% with the remainder will be given to users in 3 linearly distributed moments in the future.

The contract supports ERC-20 fee-on-transfer tokens.

## 2.1.1 Privileged Roles



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

- `setOfferingAmount`
- `setRaisingAmount`
- `finalWithdraw`
- `sweepToken`






## 2.1.2 Issues & Recommendations

<b>Issue #01</b>	<b>Gov privilege: Governance can withdraw all deposited tokens and offering tokens at any time which could lead to loss of all users funds</b>
<b>Severity</b>	 MEDIUM SEVERITY
<b>Location</b>	<u>Line 273</u> <code>function finalWithdraw(uint256 _stakeTokenAmount, uint256 _offerAmount)</code>
<b>Description</b>	<p>The IAO contract is upgradeable, which means that the contract admin can add new features to the contract over time. This does also mean that if the admin wants to take the tokens staked in the contract, they can execute an upgrade to do so. The contract also contains a function <code>finalWithdraw</code> which can be called at any time to take out all tokens within the contract.</p> <p>It should be noted that ApeSwap is considered a reputable party within the DeFi space and we expect the outright abuse of this functionality to be highly unlikely. However, as loss or theft of governance keys is not impossible, this risk remains present to some extent.</p>
<b>Recommendation</b>	Consider adding safety features to prevent the theft of tokens in case the admin keys are stolen. This requires both safety features within the <code>finalWithdraw</code> function and at the ownership level of the upgradeable contract (the admin address).
<b>Resolution</b>	 ACKNOWLEDGED
	<p>The client acknowledges that they have high privileges over these contracts, but has explained that they want to make sure that if things go wrong and they need to unlock stuck funds, they should be able to do so.</p> <p>We agree that ApeSwap is one of the few uniquely-positioned projects that has a long standing reputation to do this. However, as theft of keys is still a risk, we have left this issue open.</p>

**Issue #02**

**Lack of reentrancy guard on before-after deposit allows for deposit value exploits if the ERC-20 token allows for reentrancy**

**Severity**

 MEDIUM SEVERITY

**Description**

Within the `deposit` function, a before-after pattern is used to figure out how many ERC-20 tokens were actually received by the contract. This pattern works by checking the contract balance before and after the transfer and setting the user amount to the difference in these two amounts. However, such a pattern should always be accompanied with a reentrancy guard as this pattern can be dangerously exploited in case the depositor can re-enter during the transfer.



This is possible because if a reentrancy were to be permitted, the depositor could make another deposit while the previous before-after is not yet finished. This would cause the second deposit to be counted in the "after" of both the first and second deposit and allows the users to inflate their deposits by multiples. This exploit can then be chained by reentering for example 30 times to make the contract believe you have deposited 30 times the amount you actually have.



**Recommendation**

Consider adding reentrancy guards to both `depositNative` and `deposit`.

**Resolution**

 RESOLVED

Issue #03 Lack of usage of upgradeable dependencies	
Severity	 LOW SEVERITY
Description	The upgradeable contract does not use ReentrancyGuardUpgradeable. This might cause issues if OpenZeppelin decides to adjust the constructor for this contract.
Recommendation	Consider using upgradeable dependencies consistently, as not using these causes the constructors to not be called. Of course the initializers still need to be manually called.
Resolution	 RESOLVED The ReentrancyGuardUpgradeable library was added in both the IAO and IAOLinearVesting contracts.

Issue #04 Deposited amount could still be zero for fee-on-transfer tokens causing the contract to malfunction	
Severity	 LOW SEVERITY
Location	<u>Line 137</u> <code>uint256 finalDepositAmount = getTotalStakeTokenBalance() - pre;</code>
Description	The deposit functions have a non-zero validation check at the start of their function. However, due to fee-on-transfer tokens, the <code>finalDepositAmount</code> could still amount to zero. This would result in the undesired side effect that the same user could be added multiple times to the <code>addressList</code> , something which could affect third-party contracts and frontends.
Recommendation	Consider validating that the <code>finalDepositAmount</code> is greater than zero.
Resolution	 RESOLVED Validation has been added in both the IAO and IAOLinearVesting contracts at lines 157 and 163 respectively:  <code>require(finalDepositAmount &gt; 0, 'final deposit amount is zero');</code>

**Issue #05****Best practice: Contract unnecessarily rounds in multiple locations****Severity** LOW SEVERITY**Location**Line 233

```
uint256 payAmount = (raisingAmount * getUserAllocation(_user)) /  
1e12;
```

**Description**

The contract unnecessarily rounds the refunding amount and bases user allocation on a rounded percentage. This makes it difficult for third-parties to guarantee that there is no way to take advantage of the protocol by abusing the rounding mechanics.

Currently, `getUserAllocation` (the percentage of the sale contributed by the user) rounds down, which causes `getRefundingAmount` to round up. As these two amounts still must balance, there does not seem to be a direct way to abuse this as an exploiter.


However, we believe that by avoiding the `1e12` precision altogether and instead explicitly chaining the operations, the rounding risk and directions can be assessed more carefully.

**Recommendation**

Consider inlining math to avoid having to excessively round operations to `1e12` precision. At this point, it will be easy to validate that all math rounds against the favor of the user, as this is an important protocol property.

**Resolution** RESOLVED

The `getUserAllocation` function has been inlined to avoid the `1e12` rounding.

**Issue #06****setOfferingAmount, setRaisingAmount and userTokenStatus can be made external****Severity** INFORMATIONAL**Description**

Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. 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 marking the above variables as external.

**Resolution** PARTIALLY RESOLVED

This issue has been resolved in the `IAOLinearVesting` contract: `setOfferingAmount` and `setRaisingAmount` have been made external. However, `userTokenStatus` in the `IAO` contract is still public.

**Issue #07****Lack of events for setOfferingAmount, setRaisingAmount and finalWithdraw****Severity** INFORMATIONAL**Description**

Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. 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 adding events for the above functions.

**Resolution** RESOLVED

**Issue #08****harvest does not adhere to checks-effects-interactions****Severity** INFORMATIONAL**Location**Line 169

```
userInfo[msg.sender].claimed[harvestPeriod] = true;
```

**Description**

The harvest function marks the claimed boolean as true after the offeringToken is transferred to the user. This is considered bad practice as it theoretically opens up the protocol to reentrancy and other malicious code execution exploits. Given that this function is already guarded by a reentrancy guard, this issue is simply provided as an informational recommendation to the developer to always try to adhere to checks-effects-interactions.


Without exaggeration, checks-effects-interactions should be considered the most important pattern within Solidity development and we heavily encourage all developers to consistently adhere to it.

**Recommendation**

Consider moving the claimed boolean adjustment above the external call (transfer).

**Resolution** RESOLVED

The checks-effects-interactions pattern has been adhered too.

**Issue #09****Limiting the gas usage of calls can be dangerous across hard forks that affect gas usage****Severity** INFORMATIONAL**Location**Line 297

```
(bool success, ) = _to.call{gas: 23000, value: _amount}("");
```

**Description**

When native gas tokens are transferred to the user, a limited gas allocation is given to prevent reentrancy. However, as Ethereum makes no formal guarantees on the limits of operational gas costs across hard forks, this could cause these calls to start reverting if ever a network upgrade was done.

**Recommendation**

As this is quite unlikely since even WBNB would stop working, this issue has been marked as resolved on the note that the client will carefully check and test whenever they deploy this contract in new situations and environments.

**Resolution** RESOLVED

**Issue #10****Typographical errors****Severity** INFORMATIONAL**Description**

The contract contains typographic errors on the following lines of code.

Line 191

```
// 1e6 = 100%
```

These multipliers should be 1e12, 1e10 and 1e6.

Line 245

```
uint256 offeringTokensVested
```

This should be called offeringTokensVesting as they are not yet vested.

Line 301

```
IERC20(stakeToken).safeTransfer(_to, _amount);
```

stakeToken is unnecessarily cast to IERC20 again. It is already stored as IERC20.

**Recommendation**

Consider fixing the above typographical errors.

**Resolution** RESOLVED



---

## 2.2 IAOLinearVesting

The IAOLinearVesting contract is extremely similar to the IAO contract from the previous section. It extends upon this contract by moving the four cliffs-based vesting method into a continuously linear vesting method.

As all issues from the previous section still apply within this contract, they will not be repeated. It should be noted that the client must carefully resolve them in both sections for these issues to be marked as fully resolved.

An initial cliff of 25% of the total bought tokens amount is still present. This allocation immediately unlocks upon IAO ending while the remaining 75% vests linearly. This is similar to the previous IAO contract where this remainder vested linearly but in 3 distinct installments.



### 2.2.1 Privileged Roles

The following functions can be called by the feeToSetter:

- `setOfferingAmount`
- `setRaisingAmount`
- `finalWithdraw`
- `sweepToken`



## 2.2.2 Issues & Recommendations

<b>Issue #11</b>	<b>harvest lacks checks-effects-interactions for refunded portion</b>
<b>Severity</b>	 INFORMATIONAL
<b>Location</b>	<u>Line 182</u> <code>currentUserInfo.refunded = true;</code>
<b>Description</b>	<p>The harvest function is not written using the checks-effects-interactions pattern which could cause side-effects in case the contract token allows for reentrancy.</p> <p>Specifically, if the refunded parameter would be used in third-party contracts, the exploiter could execute code within these third-party contracts which would still believe that the user was not refunded at this point in time, while in fact they already are.</p>
<b>Recommendation</b>	Consider rewriting the harvest function to adhere to checks-effects-interactions.
<b>Resolution</b>	 RESOLVED <code>currentUserInfo.refunded</code> has been set to true at an earlier stage of the function.

<b>Issue #12</b>	<b>Lack of validation</b>
<b>Severity</b>	 INFORMATIONAL
<b>Location</b>	<u>Line 306</u> offeringTokenVestedHarvest = (offeringTokenVestedAmount * unlockBlocks) / totalVestingBlocks;
<b>Description</b>	The contract contains a section of code which lacks proper validation. This could cause errors in case unexpected inputs are provided. The also contract does not work for zero totalVestingBlocks.
<b>Recommendation</b>	Consider using the following requirement.  <code>require(totalVestingBlocks &gt; 0, "");</code>
<b>Resolution</b>	 RESOLVED  This issue was not resolved as recommended, but validation for <code>_vestingBlockOffset &gt; 0</code> was added in Line 109.





**PALADIN**  
BLOCKCHAIN SECURITY