PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ApeSwap (Jungle Fund)

22 September 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 5

1.3 Findings Summary 6

1.3.1 CustomBillFixedPrice 7

1.3.2 CustomBillFixedPriceMigration 7

2 Findings 8

2.1 CustomBillFixedPrice 8

2.1.1 Privileged Functions 9

2.1.1 Issues & Recommendations 10

2.2 CustomBillFixedPriceMigration 20

2.2.1 Privileged Functions 20

2.2.2 Issues & Recommendations 21

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 the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for the ApeSwap's Jungle Fund 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
URL	https://www.apeswap.com/
Platform	BNB Smart Chain
Language	Solidity
Preliminary Contracts	https://github.com/ApeSwapFinance/apeswap-jungle-fund/tree/e07ef11c1b261c4dcc0ef5e0af895b2abf5de06a/contracts
Resolution	https://github.com/ApeSwapFinance/apeswap-jungle-fund/tree/2b033609e81767e57dc59ebecce5c50a61cdc53d/contracts/CustomBillFixed

1.2 Contracts Assessed

Name	Contract	Live Code Match
CustomBillFixedPrice	Dependency	MATCH
CustomBillFixedPrice Migration	Proxy 0x302f1120aDA3f40aeD3F6410fad75e8428576a1F 0x872C7beE4c034CEba697de18f794e91F10062E2e 0x8146d7887C2e7ce7AD0c1950E17676c632B5A71A 0xFeFEDF8EB74711dF186630F4870BcA9EEDA78C85 0xa2a328864f8Eca97072471cc8BF08411Da523bc9 0xac1CB77150Dcc3d540F08e808181CCCf0Ea29726 0xDCa0B400394BA6F6CBfc3Ce37B0193D4fa68eDFf 0x3Eb85D568214ae04c2873880Fa715a5C8105dd17 Implementation 0x044869dAeB7261B30247002B3Bf0636770BcC850	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	1	-	-	1
High	0	-	-	-
Medium	2	2	-	-
Low	5	3	-	2
Informational	9	6	-	3
Total	**17**	**11**	**-**	**6**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
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 CustomBillFixedPrice

ID	Severity	Summary	Status
01	GOV	Treasury owner can disable deposits	ACKNOWLEDGED
02	MEDIUM	`_setFeeTiers` deducts the array length in the loop	✓ RESOLVED
03	MEDIUM	Term parameters lack a reasonable limit	✓ RESOLVED
04	LOW	Lack of impact on `accumulatedRewards` if fee is turned on for payouts	ACKNOWLEDGED
05	LOW	`BillCreated` event emits wrong expires parameter	✓ RESOLVED
06	LOW	`approve` might revert for non-ERC20 tokens	✓ RESOLVED
07	LOW	Check for minimum `payOut` received instead of `truePrice`	✓ RESOLVED
08	INFO	Contract is not compatible with tokens that have a fee on transfer as `principalToken`	✓ RESOLVED
09	INFO	Initializer is not disabled	ACKNOWLEDGED
10	INFO	Unnecessary check for payout to be lesser than maxPayout	✓ RESOLVED
11	INFO	Lack of check that the `vestingTerm` state variable is the same as the `vestingTerm` that the user expects	ACKNOWLEDGED
12	INFO	`setMaxTotalPayout` function is unnecessary	✓ RESOLVED
13	INFO	Inconsistency in `setBillTerms` for `PARAMETER.MIN_PRICE`	✓ RESOLVED
14	INFO	Lack of upper length limit for `feeTiers` can cause `currentFee()` to revert	✓ RESOLVED
15	INFO	Unnecessary check for `totalPayoutGiven == 0` within initialize	✓ RESOLVED
16	INFO	`allIssuedBillIds()` will revert if `billIssuedIds` has too many key/value pairs in the set	ACKNOWLEDGED

1.3.2 CustomBillFixedPriceMigration

ID	Severity	Summary	Status
17	LOW	Reward distribution is not guaranteed	ACKNOWLEDGED

2 Findings

2.1 CustomBillFixedPrice

`CustomBillFixedPrice` is a simple bond contract, where users can provide a `principalToken` which mints an NFT to the provided `_depositor` address. The deposited amount will then be used to calculate the total payout value using the fixed bill price in `billPrice`. Depending on the `feeInPayout` configuration, the contract uses one of the following methodologies:

a) The total deposit amount is used to calculate the to received amount of payout token and then applied on the payout value.

b) The fee is applied on the total deposited value and the fees adjusted value is then used to calculate the payout amount.

This NFT then represents a position which allows the owner to claim a `payoutToken` which is vested over time. The vesting calculation is handled in the external `vestingCurve` contract which is out of scope for this audit. During the initialization, any address can be assigned as the `vestingCurve` contract; if no address is assigned, the default `LinearVestingCurve` is deployed and assigned.

Users and approved redeemers can claim their vesting payout tokens for a `BillNFT` at any time, depending on the progress which is determined within the `_payoutsAtTime` function.

The contract exposes several configuration variables such as:

- `vestingTerm`: The general vesting duration
- `minimumPrice`: The starting price, on which the fee is applied

- `maxTotalPayout`: The maximum sum of all aggregated scheduled payouts

Moreover, as previously mentioned, a fee is taken either as principal token or as payout token. This fee is handled with a special fee tier logic where the fee will increase/decrease depending on the current tier. The tier is determined by how many principal tokens have been accounted as deposit, which is represented by the `totalPrincipalBilled` variable. ApeSwap leaves it open to increase or decrease the fees with increasing deposits, however, it should be ensured that, if one configuration was determined, this configuration should be continued.

2.1.1 Privileged Functions

- `setBillTerms`
- `setMaxTotalPayout`
- `setFeeTiers`
- `transferOwnership`
- `setPendingOwner`
- `changeFeeTo (DAO)`

2.1.1 Issues & Recommendations

Issue #01	Treasury owner can disable deposits
Severity	GOVERNANCE

Description	The treasury owner can prevent the Bill contract calling the deposit functions:
	`modifier onlyBillContract() {`
	`  require(billContract[msg.sender], "msg.sender not bill contract");`
	`  _;`
	`}`

Recommendation	Consider keeping the owner of the treasury contract as a multi-signature wallet with KYC-ed participants.
Resolution	ACKNOWLEDGED

Issue #02	`_setFeeTiers` deducts the array length in the loop
Severity	MEDIUM SEVERITY

Description	The following code spot attempts to remove all values in the array:
	`if (feeTiers.length > 0) {`
	`  for (uint256 j; j < feeTiers.length; j++) {`
	`  feeTiers.pop();`
	`  }`
	`}`
	
	However, the loop is executed over the dynamic array length, which will result in not all elements being removed from the array.
	
	While the ApeSwap team implemented a workaround to set the correct elements, this function will still revert when attempting to add fewer elements than previously existed. Therefore, redeployment is necessary in that scenario.
	
	Generally speaking, it is always necessary to ensure that loops are always executed over fixed size arrays. Exceptions prove the rule.

Recommendation	Consider caching the array length beforehand and then looping over the cached variable. Once this is fixed, the regular push approach can then be pursued, instead of setting existing elements.
Resolution	RESOLVED

Issue #03	Term parameters lack a reasonable limit
Severity	MEDIUM SEVERITY
Description	`vestingTerm` and `minimumPrice` can be set to an unreasonably high value. While the `deposit` function has a safeguard for a large minimum price, there is no safeguard for a large `vestingTerm`, which can result in the vesting time being set to hundreds of years.
Recommendation	Consider implementing a reasonable limit for both parameters.
Resolution	RESOLVED

Issue #04	Lack of impact on accumulatedRewards if fee is turned on for payouts
Severity	LOW SEVERITY
Description	The contract has two methodologies for applying the fee: a) Apply it on the deposit amount b) Apply it on the payout If the fee is applied on the deposit amount, this will, with increasing fees, reduce a user's share on the `accumulatedRewardsPerShare` as this variable depends on the deposited values. Therefore, if the fee is applied on the payout token, no matter how high the fee is or if it is increased/decreased, all users will receive the same share of `accumulatedRewardsPerShare` since the deposited amount is not manipulated.
Recommendation	In our opinion, this *might be* an unexpected side-effect from the design pattern. However, there is also the possibility that this is desired. Consider if this is desired, if not, consider adjusting the allocation to `accumulatedRewardsPerShare` also in terms of the fee.
Resolution	ACKNOWLEDGED

Issue #05	**`BillCreated` event emits wrong expires parameter**
Severity	LOW SEVERITY

Location	`emit BillCreated(`
	`  _amount,`
	`  payoutBalanceAdded,`
	`  block.timestamp + terms.vestingTerm,`
	`  billId`
	`);`

Description	This parameter is not correctly set and can confuse third-parties. It should be `vestingStartTimestamp + terms.vestingTerm`.
Recommendation	Consider following the recommendation.
Resolution	RESOLVED

Issue #06	**`approve` might revert for non-ERC20 tokens**
Severity	LOW SEVERITY
Description	The `approve` call expects a return value: `function approve(address spender, uint256 value) external returns (bool);` However, for non-compliant tokens, this function might revert due to a lack of the return value. For this purpose, the `safeIncreaseAllowance` or `forceApprove` should be used.
Recommendation	Consider using `forceApprove` or `safeIncreaseAllowance`.
Resolution	RESOLVED

Issue #07	Check for minimum `payOut` received instead of `truePrice`
Severity	LOW SEVERITY
Description	There is a slippage check against the `truePrice`. However, this should be checked against `payoutBalanceAdded` to ensure that the actual amount of tokens added to a bill is at least the minimum of which the user expects. This can be lower than expected for payout tokens that have a fee on transfer, which `truePrice` does not account for.
Recommendation	Allow the function to take a parameter supplied by the user which acts as a minimum amount check against the `payoutBalanceAdded`.
Resolution	RESOLVED

Issue #08	Contract is not compatible with tokens that have a fee on transfer as `principalToken`
Severity	INFORMATIONAL
Description	While the ApeSwap team is already aware of that issue, it is still necessary for Paladin to include in the report that the `principalToken` must never be a token that has fees on transfer.

Recommendation	Never use tokens with a fee on transfer as `principalToken`. Additionally, consider adding a check with `safeTransferFrom` that the after - before token balance of the token is always equal to the transferred amount.
	`uint256 before = principalToken.balanceOf(address(this));`
	`principalToken.safeTransferFrom(msg.sender, address(this), _amount);`
	`uint256 actualAmount = principalToken.balanceOf(address(this) - before;`
	`require(actualAmount == _amount);`

Resolution	RESOLVED Tokens with a fee on transfer are supported by taking the after - before balance. However, the treasury contract will still receive less tokens in that scenario which can result in side-effects.

Issue #09	Initializer is not disabled
Severity	INFORMATIONAL
Description	To prevent undesired access to the implementation contract, it is considered best practice to disable the initializer during contract deployment.
Recommendation	Consider implementing `_disableInitializer` within the constructor.
Resolution	ACKNOWLEDGED

Issue #10	Unnecessary check for payout to be lesser than `maxPayout`
Severity	INFORMATIONAL

Description	Within `deposit`, there is a check that payout is lower than or equal to `maxPayout`.
	
	L333
	`require(payout <= maxPayout(), "Bill too large");`
	
	However, if we look at the value of `maxPayout`, it is the maximum value of the type `uint`, thus making it an unnecessary check as payout is a `uint256` variable and will always return `true` for the check.
	
	L576~
	`function maxPayout() public pure returns (uint256) {`
	`  return type(uint).max;`
	`}`

Recommendation	Consider removing this check.
Resolution	RESOLVED

Issue #11	**Lack of check that the `vestingTerm` state variable is the same as the `vestingTerm` that the user expects**
Severity	INFORMATIONAL
Description	There can be a mismatch between what is displayed on the frontend and in the contract state as the owner of the contract is able to call `setBillTerms` to change the `vestingTerm` anytime. By supplying a `vestingTerm` parameter and checking it against the state variable, this ensures that the `vestingTerm` is the same between the frontend and contract.
Recommendation	Consider adding `vestingTerm` as a user supplied parameter and ensuring it has the same value as the state variable.
Resolution	ACKNOWLEDGED

Issue #12	**`setMaxTotalPayout` function is unnecessary**
Severity	INFORMATIONAL
Description	`setBillTerms` already allows `terms.maxTotalPayout` to be set, thus having the same functionality as `setMaxTotalPayout` when the parameter is `PARAMETER.MAX_TOTAL_PAYOUT`.
Recommendation	Consider removing the `setMaxTotalPayout` function.
Resolution	RESOLVED

Issue #13	Inconsistency in `setBillTerms` for `PARAMETER.MIN_PRICE`
Severity	INFORMATIONAL

Location	L197~
	`require(`
	`  _billTerms.minimumPrice > 0,`
	`  "Minimum price required"`
	`);`

Description	Within `initialize`, there is a check to ensure that `minimumPrice` is greater than 0. However, this check is not within `setBillTerms`, allowing `minimumPrice` to be set to a value that was not allowed in initialize.
Recommendation	Add a similar check within `setBillTerms` to ensure consistency in the allowed values for `minimumPrice`.
Resolution	RESOLVED

Issue #14	Lack of upper length limit for `feeTiers` can cause `currentFee()` to revert
Severity	INFORMATIONAL
Description	There is no upper bound for the number of `feeTiers` that can be set in the `feeTiers` array. If the length of `feeTiers` is too long, it can cause `currentFee()` to revert.
Recommendation	Consider restricting the length of `feeTiers` to a reasonable length.
Resolution	RESOLVED

Issue #15	**Unnecessary check for `totalPayoutGiven == 0` within `initialize`**
Severity	INFORMATIONAL
Description	The default value when the contract is deployed has it as 0 since it is an undeclared state variable and thus defaults to 0.
Recommendation	Consider removing the unnecessary check.
Resolution	RESOLVED

Issue #16	**`allIssuedBillIds()` will revert if `billIssuedIds` has too many key/value pairs in the set**
Severity	INFORMATIONAL
Description	As stated in the comments for `allIssuedBillIds()`, it can revert if the `EnumerableSet` is too large.
Recommendation	Consider using pagination to return specified portions of the `EnumerableSet`.
Resolution	ACKNOWLEDGED

2.2 CustomBillFixedPriceMigration

`CustomBillFixedPriceMigration` is an auxiliary contract for the `CustomBillFixedPrice` contract. It is responsible for calculating the payout for a `tokenID` at a specific timestamp, potentially adding bonus rewards on top of the regular bill payout. For this purpose, the `_payoutsAtTime` function is overridden.

The bonus reward logic is only applied once the deposits are closed and `totalRewards` are being divided by the `totalPrincipalBilled` amount. This will ensure that no manipulation is possible and all users will receive their fair share of reward depending on their deposited amount.

The regular `deposit` function is overridden and deposits can only be executed if the `depositsClosed` variable is set to `false`, which is the case after deployment until the contract owner sets it to true.

2.2.1 Privileged Functions

- `disableDeposits`

2.2.2 Issues & Recommendations

Issue #17	Reward distribution is not guaranteed
Severity	LOW SEVERITY
Description	The contract allows the distribution of bonus rewards. However, it is not ensured that these rewards are in fact as a balance within the contract. Therefore, an attempt to claim might revert due to insufficient funds.
Recommendation	Consider transferring the `totalRewards` amount during the initialization in. Also, the after-before difference should be the amount used for `totalRewards` to account for tokens with a fee on transfer.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY