PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ApeSwap (Bills v2)

01 February 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 BillHelper 7

1.3.2 CustomTreasury 7

1.3.3 BillNft 7

1.3.4 CustomBill 8

1.3.5 CustomBillRefillable 8

2 Findings 9

2.1 BillHelper 9

2.1.1 Issues & Recommendations 9

2.2 CustomTreasury 10

2.2.1 Privileged Functions 10

2.2.2 Issues & Recommendations 11

2.3 BillNft 14

2.3.1 Privileged Functions 14

2.3.2 Issues & Recommendations 15

2.4 CustomBill 18

2.4.1 Privileged Functions 19

2.4.2 Issues & Recommendations 20

2.5 CustomBillRefillable 24

2.5.1 Privileged Functions 24

2.5.2 Issues & Recommendations 25

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 Bills V2 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 (Jungle Fund)
URL	https://apeswap.finance/
Platform	BNB Smart Chain
Language	Solidity
Preliminary	https://github.com/ApeSwapFinance/apeswap-jungle-fund/tree/c1c144c67cca280359306e0f22918d43a09913fc
Resolution 1	https://github.com/ApeSwapFinance/apeswap-jungle-fund/tree/f99e049515be98ff581c1478c7edf60b4a72d27b
Resolution 2	https://github.com/ApeSwapFinance/apeswap-jungle-fund/pull/20/commits/a90e6eb686e36b39e5304a67aa1694fb529e9fe5

1.2 Contracts Assessed

Name	Contract	Live Code Match
BillHelper	Not in use	N/A
CustomTreasury	0x08D025B4cFf40DcDA6C54771fEc56A9beB1e1F97	MATCH
BillNft	Proxy: 0xB75BE16984A27d3023e1cF744E2587e9Aa8750c9 Implementation: 0x399e72a62f926ca095de35d9b5720d6f43046abe	MATCH
CustomBill	0x1262661d2b42e08129c37f874066b116f5ed8043	MATCH
CustomBillRefillable	0x1262661d2b42e08129c37f874066b116f5ed8043	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	1	1	-
Medium	2	2	-	-
Low	6	4	-	2
Informational	5	4	1	-
Total	**15**	**11**	**2**	**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 BillHelper

No issues found.

1.3.2 CustomTreasury

ID	Severity	Summary	Status
01	HIGH	Any token held by `customTreasury` can be stolen by a malicious governance	PARTIAL
02	LOW	The contract owner could be transferred to a malicious address	ACKNOWLEDGED
03	LOW	The `PendingOwnableUpgradeable` dependency could use OpenZeppelin's `Ownlable2StepUpgradeable` library	ACKNOWLEDGED
04	INFO	Lack of events for `toggleBillContract` and `deposit`	✓ RESOLVED

1.3.3 BillNft

ID	Severity	Summary	Status
05	MEDIUM	`setTokenURI` cannot be called as it was marked as internal	✓ RESOLVED
06	LOW	`supportsInterface` only calls one of its inherited contracts	✓ RESOLVED
07	INFO	Lack of validation	✓ RESOLVED
08	INFO	Unnecessary import	✓ RESOLVED

1.3.4 CustomBill

ID	Severity	Summary	Status
09	HIGH	Deposits do not support tokens that have a fee on transfer	✓ RESOLVED
10	LOW	`CustomBill` cannot be safely inherited after upgrades	✓ RESOLVED
11	LOW	The different tiers ceilings are not checked to be in increasing order	✓ RESOLVED
12	INFO	Typographical errors	PARTIAL

1.3.5 CustomBillRefillable

ID	Severity	Summary	Status
13	MEDIUM	`refillPayoutToken` is vulnerable to reentrancy	✓ RESOLVED
14	LOW	`CustomBillRefillable` does not use upgradeable dependencies	✓ RESOLVED
15	INFO	Gas optimization	✓ RESOLVED

2 Findings

2.1 BillHelper

`BillHelper` is a helper contract that allows users to redeem multiple bills at once.

2.1.1 Issues & Recommendations

No issues found.

2.2 CustomTreasury

`CustomTreasury` is a simple treasury contract that allows the transfer of ERC20 tokens to the `payoutAddress`, the DAO.

It contains a `deposit` function which is only callable by the bill contracts. It transfers `principalToken` to the `payoutAddress` and transfers out `payoutToken` to the bill. The `principalToken` is a bill parameter and can be any token, while the `payoutToken` is always the same.

The `withdraw` function allows the owner (the governance) to withdraw any ERC20 from this contract to an address.

The `toggleBillContract` function enables adding a contract to the list of bills, which then allows this address to call `deposit`.

2.2.1 Privileged Functions

- `deposit [ Bill ]`
- `deposit_FeeInPayout [ Bill ]`
- `withdraw`
- `toggleBillContract`

2.2.2 Issues & Recommendations

Issue #01	**Any token held by `customTreasury` can be stolen by a malicious governance**
Severity	HIGH SEVERITY
Description	A malicious governance could call `withdraw` to steal all the tokens held by the `customTreasury`. The `payoutToken` could be stolen by a malicious address that was previously added to the `billContract` mapping, which allows it to call `deposit` and steal the entire balance of the contract. PoC: 1. Governance adds a malicious contract to the `billContract` mapping using the `toggleBillContract` function. 2. This contract calls `deposit` with 0 `_amountPrincipalToken` and the balance of the contract as `_amountPayoutToken`, effectively emptying the `customTreasury` contract.
Recommendation	Consider designing a strong governance structure where it is unlikely and ideally impossible for the governance to abuse these privileges. It should also be noted that the client, ApeSwap, is a very reputable actor within the DeFi space. They have been around for a significant amount of time and have shown adequate behavior and professionalism over that period. The main risk for this issue is therefore that governance is centralized within a single private key (or single entities have the ability to execute the above actions). This could cause theft or loss in case this private key was stolen, the single entities go rogue or similar situations where the organization overall remains honest but a single party is compromised. Generally speaking, a good governance structure ensures that not a single party can execute any of these critical functionalities. It is therefore ideal to lock all critical functionality behind a reputable multi-signature set up.
Resolution	PARTIALLY RESOLVED ApeSwap uses multiple ApeSwap DAO multi-signature contracts to manage sensitive smart contract ownership. All owners of these multi-signature contracts are managed through hardware wallets.

Issue #02	The contract owner could be transferred to a malicious address
Severity	LOW SEVERITY
Location	L36 `_transferOwnership(_initialOwner);`
Description	The ownership of the contract is transferred directly to the `_initialOwner` without using the pending flow. This could result in a transfer to a malicious address. However, the current flow may be preferred by the client to simplify the deployment process.
Recommendation	Consider using the pending flow if the deployment simplification is not needed.
Resolution	ACKNOWLEDGED This is intended for deployments through a `CustomBillFactory` contract. Funds are not added to the contract without ownership verification first. Once contracts have been funded, they prefer to use `setPendingOwner()`.

Issue #03	The `PendingOwnableUpgradeable` dependency could use OpenZeppelin's `Ownlable2StepUpgradeable` library
Severity	LOW SEVERITY
Description	The ApeSwap's `PendingOwnable` library is a custom contract and they can consider using OpenZeppelin's contract instead. ApeSwap's `PendingOwblableUpgradeable` does not define the gap variable — that could break the upgradeability of the contract in the future if a variable were to be added.
Recommendation	Consider using Open Zeppelin libraries when available. Upgradeable `Ownable2Step` can be found here: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol
Resolution	ACKNOWLEDGED

Issue #04	Lack of events for `toggleBillContract` and `deposit`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. `billContractToggled` has been implemented, but it is not emitted.
Recommendation	Consider adding events for the above functions.
Resolution	RESOLVED

2.3 BillNft

`BillNft` is a NFT contract that follows the ERC721 standard from OpenZeppelin and also EIP5725, Transferable Vesting NFT. This EIP defines transferable vesting NFTs which release underlying tokens over time. Whoever owns these NFTs can at some point claim vested tokens, the contract then forwards the call from the NFT directly to the `CustomBill` contract that implements the vestings. It is minted to users by the `CustomBill` after a successful deposit.

The Bill NFTs can be minted by addresses with the `MINTER_ROLE`, which can be given by admins with the `DEFAULT_ADMIN_ROLE` or `FACTORY_ROLE` roles.

2.3.1 Privileged Functions

- `setBaseURI`
- `lockURI`
- `addMinter [ DEFAULT_ADMIN_ROLE or FACTORY_ROLE ]`
- `mint [ MINTER_ROLE ]`
- `setTokenURI [ URI_SETTER_ROLE ]`
- `claim [ Approved address or owner of the NFT ]`
- `claimMany [ Approved address or owner of the NFT ]`

2.3.2 Issues & Recommendations

Issue #05	**`setTokenURI` cannot be called as it was marked as internal**
Severity	MEDIUM SEVERITY
Location	L172 `function setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual`
Description	`setTokenURI` cannot be called as it has been incorrectly marked as `internal`.
Recommendation	Consider marking `setTokenURI` as `external`. Additionally, `uriLocked` should be checked that it is not set to `true` before allowing the setting of the URI of a specific token.
Resolution	RESOLVED

Issue #06	**supportsInterface only calls one of its inherited contracts**
Severity	LOW SEVERITY

Location	L153-155
	`function supportsInterface(bytes4 interfaceId) public view override(AccessControlEnumerableUpgradeable, ERC721EnumerableUpgradeable) returns (bool) {`
	`  return interfaceId == type(IERC5725).interfaceId || super.supportsInterface(interfaceId);`
	`}`

Description	`supportsInterface` overrides two contracts; however, as `super` is used, only the last inherited contract is overridden, which is `ERC721EnumerableUpgradeable`.

Recommendation	Consider using the following implementation:
	`function supportsInterface(bytes4 interfaceId) public view override(AccessControlEnumerableUpgradeable, ERC721EnumerableUpgradeable) returns (bool) {`
	`  return interfaceId == type(IERC5725).interfaceId || AccessControlEnumerableUpgradeable.supportsInterface(interfaceId) || ERC721EnumerableUpgradeable.supportsInterface(interfaceId);`
	`}`

Resolution	RESOLVED

Issue #07	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.
Recommendation	Consider validating the `initialize` function.
Resolution	RESOLVED

Issue #08	Unnecessary import
Severity	INFORMATIONAL
Location	L 38 `using StringsUpgradeable for uint256;`
Description	StringsUpgradeable is not used directly in the contract.
Recommendation	Consider removing this import.
Resolution	RESOLVED

2.4 CustomBill

`CustomBill` is inspired by and based on Ohm's `BondDepository` contract.

It allows users to buy the `payoutToken` by providing some `principalToken`. However, contrary to a normal purchase, `payoutToken` is then vested linearly over a configurable vesting period that is a minimum of 36 hours.

The price at which users can buy `payoutToken` can be configured and it automatically adjusts with two separate mechanisms.

First, there is a `totalDebt` mechanism. Whenever a bond is bought, `totalDebt` increases which causes the `debtRatio` (`totalDebt/payoutTokens.totalSupply()`) to increase, which directly causes the price to increase. In other words, whenever bonds are purchased, the price increases. `totalDebt` will then slowly decrease automatically in a non-linear fashion (eg. a percentage of `totalDebt` decays every so often).

Second, there is an adjustment mechanism. The governance can independently configure an adjustment direction, rate, buffer time and target. Every time the buffer time passes, the price will be adjusted in the configured direction at a rate of rate, until target is reached. This adjustment is done indirectly through the "Control Variable", which is the proxy variable controllable by the governance to influence the price. The governance should be extremely careful with managing the `controlVariable`, as an incorrectly set `controlVariable` could cause an undesirable situation where `payoutToken` is given at a significant discount.

The DAO address can change the treasury which receives the bond fees.

The contract contains logic referred to as the "payout subsidy". This is an accounting variable which accumulates the total payout given over time. Whenever `paySubsidy` is called by a subsidy controller, it resets. We have confirmed with the client that they do not plan to use this mechanism and therefore the controller itself

was not included within the scope of this audit. For this specific audit, this variable therefore purely has accounting purposes similar to `totalPayoutGiven`.

The governance can set a minimum price; however, as soon as the actual price reaches above this, the minimum is removed. This minimum is therefore only useful for avoiding awful mistakes during initial configuration.

Additionally, the control variable was scaled up 100x, allowing for more tuning. This should be kept in mind when analyzing the different parameters of the bond.

2.4.1 Privileged Functions

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

- `setPendingOwner`
- `acceptOwnership [ pendingOwner ]`
- `revokeOwnership`
- `setBillTerms`
- `setMaxTotalPayout`
- `setAdjustment`
- `changeTreasury [ DAO ]`
- `redeem [only user, billNft or approved redeemer]`

2.4.2 Issues & Recommendations

Issue #09	Deposits do not support tokens that have a fee on transfer
Severity	HIGH SEVERITY
Description	Within the deposit function, there is no logic that supports transfer-tax token. Therefore, during a deposit as it first send tokens from users to the `customBill` and then transfers that amount to the `customTreasury`, most of the time it would revert because the contract wouldn't hold enough token.
Recommendation	Consider not using transfer tax token, or consider adding logic for transfer-tax tokens if such tokens may be used.
Resolution	RESOLVED A before-after pattern was added for `payoutToken`, however, it was not added for `principalToken`, which means the treasury might receive fewer tokens than expected. However, the client is aware of this.

Issue #10	`CustomBill` cannot be safely inherited after upgrades
Severity	LOW SEVERITY
Description	The `CustomBill` contract is an upgradeable contract; however, as the `__gap` variable is not defined (more info here), it is not safe to upgrade it as the contracts that inherit this one after an upgrade could break, which is, in fact what `CustomBillRefillable` does. This issue is only rated as low because the contracts will be deployed as OpenZeppelin's clone. This standard does not allow for any upgradeability.
Recommendation	Consider adding a `__gap` variable that follows OpenZeppelin's convention to allow for the upgradability of both contracts. During an upgrade, it is important to update the `__gap` variable accordingly, i.e., when adding new storage slots to the contract.
Resolution	RESOLVED

Issue #11	The different tiers ceilings are not checked to be in increasing order
Severity	LOW SEVERITY
Location	L116-121 `for (uint256 i; i < tierCeilings.length; i++) { require(fees[i] <= 1e6, "Invalid fee"); feeTiers.push( FeeTiers({tierCeilings: tierCeilings[i], fees: fees[i]}) ); }`
Description	The `tierCeilings` should be checked to be in increasing order, as an incorrect order would malfunction.
Recommendation	Consider using the following implementation: `uint256 previousCeiling; for (uint256 i; i < tierCeilings.length; i++) { require(fees[i] <= 1e6, "Invalid fee"); require(previousCeiling == 0 || previousCeiling < tierCeilings[i], “only increasing order”); previousCeiling = tierCeilings[i]; feeTiers.push(FeeTiers({tierCeilings: tierCeilings[i], fees: fees[i]})); }`
Resolution	RESOLVED

Issue #12	Typographical errors
Severity	INFORMATIONAL

Description	We have consolidated the typographical errors below.
	
	L19
	`using SafeERC20Upgradeable for IERC20Upgradeable;`
	
	The above import is not used within the contract. Consider removing it.
	
	L77
	`uint256 fees; // in ten-thousandths (i.e. 33300 = 3.33%)`
	
	The fees are in millionths, not ten-thousandths.
	
	L226
	`function changeTreasury(address _treasury) external`
	
	The `_treasury` address should be checked to be non-zero.

Recommendation	Consider fixing the typographical errors.
Resolution	PARTIALLY RESOLVED

2.5 CustomBillRefillable

`CustomBillRefillable` inherits the `CustomBill` contract and all its functionality. It also allows admins with the `REFILL_ROLE` to refill the payout token in the `customTreasury` contract.

The `REFILL_ROLE` can be given by the owner of the contract using the `grantRefillRole` function and removed using the `revokeRefillRole` function.

As the `DEFAULT_ADMIN_ROLE` is not given to anyone, the `grantRole` and `revokeRole` functions can never callable by anyone, not even the owner.

2.5.1 Privileged Functions

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

- `All the customBill privileged functions`
- `grantRefillRole [ owner ]`
- `revokeRefillRole [ owner ]`
- `refillPayoutToken [ REFILL_ROLE ]`

2.5.2 Issues & Recommendations

Issue #13	`refillPayoutToken` is vulnerable to reentrancy
Severity	MEDIUM SEVERITY
Description	The `refillPayout` function uses a before-after pattern that is known to be vulnerable to reentrancy. This issue is only rated as medium because this function is privileged; however, a `nonReentrant` modifier should still be added.
Recommendation	Consider adding a `nonReentrant` check to `refillPayoutToken`.
Resolution	RESOLVED

Issue #14	`CustomBillRefillable` does not use upgradeable dependencies
Severity	LOW SEVERITY
Description	`CustomBillRefillable` may be an upgradeable contract and thus should use upgradeable OpenZeppelin's upgradeable dependencies. This allows the developer to safely update the contract in case `AccessControlEnumerable` is updated in the future. This issue is only rated as low because the contracts will be deployed as OpenZeppelin's clone. This standard does not allow for any upgradeability.
Recommendation	Consider using the `AccessControlEnumerableUpgradeable` contract. This would also resolve the issue with functions collision of `_msgSender()` and `_msgData()`.
Resolution	RESOLVED

Issue #15	Gas optimization
Severity	INFORMATIONAL

Location	L69-71
	`terms.maxTotalPayout += refillAmount;`
	`emit BillRefilled(address(payoutToken), refillAmount);`
	`emit MaxTotalPayoutChanged(terms.maxTotalPayout);`

Description	The `terms.maxTotalPayout` value is in fact read twice, which consumes additional gas.

Recommendation	Consider using the following implementation to save some gas:
	`uint256 maxTotalPayout = terms.maxTotalPayout + refillAmount;`
	`terms.maxTotalPayout = maxTotalPayout;`
	`emit BillRefilled(address(payoutToken), refillAmount);`
	`emit MaxTotalPayoutChanged(maxTotalPayout);outChanged(terms.maxTotalPayout);`

Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY