PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Seasonal Tokens

10 November 2021

paladinsec.co

info@paladinsec.co

Table of Contents

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.

1 Overview

This report has been prepared for Seasonal Tokens on the Ethereum network. 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	Seasonal Tokens
URL	https://seasonaltokens.org/
Platform	Ethereum
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
SeasonalTokenFarm	0xE8adB0111CcB570e366c73eE799242eFfC319404	MATCH
SpringToken	0xf04aF3f4E4929F7CD25A751E6149A3318373d4FE	MATCH
SummerToken	0x4D4f3715050571A447FfFa2Cd4Cf091C7014CA5c	MATCH
AutumnToken	0x4c3bAe16c79c30eEB1004Fb03C878d89695e3a99	MATCH
WinterToken	0xCcbA0b2bc4BAbe4cbFb6bD2f1Edc2A9e86b7845f	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	2	2	-	-
Low	4	2	1	1
Informational	17	6	-	11
Total	**24**	**11**	**1**	**12**

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 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 SeasonalTokenFarm

ID	Severity	Summary	Status
01	HIGH	`receiveApproval` and `receiveSeasonalTokens` do not validate the `msg.sender` causing anyone to be able to pull approved tokens from anyone into the farm	RESOLVED
02	MEDIUM	Contract uses homogeneous liquidity parameter for heterogeneous liquidity NFT tokens	RESOLVED
03	LOW	The `withdraw` function does not adhere to checks-effects-interactions	RESOLVED
04	LOW	`hasDoubledAllocation` cycles after 4 intervals (36 months) have elapsed which is not inline with the contract description	RESOLVED
05	INFO	`nextWithdrawalTime` and `getPayoutSizes` can be made `external`	RESOLVED
06	INFO	The `onERC721Received` use `memory` instead of `calldata` on parameter	RESOLVED
07	INFO	Usage of if-revert pattern compared to require	RESOLVED
08	INFO	Unused `tokenOwner` within the `harvest` functions	RESOLVED
09	INFO	`INonfungiblePositionManager` wrongly defines the ERC-721 `safeTransferFrom` function	RESOLVED
10	INFO	Gas optimization: Using `msg.sender` might be cheaper within `harvest`	RESOLVED

1.3.2 Season Tokens (Spring, Summer, Autumn, Winter)

ID	Severity	Summary	Status
11	MEDIUM	Merged mining is still theoretically possible	RESOLVED
12	LOW	Lack of difficulty adjustment period is prone to miner collusion	RESOLVED
13	INFO	Mint event does not emit exact `epochCount`	ACKNOWLEDGED
14	INFO	Usage of if-revert pattern compared to require	ACKNOWLEDGED
15	INFO	Batching mechanism might lead to adverse incentives if mining power is highly concentrated	ACKNOWLEDGED
16	INFO	`name`, `symbol`, `totalSupply` and `decimals` functions can be made `external`	ACKNOWLEDGED
17	INFO	Gas optimization: Unnecessary `else` clause	ACKNOWLEDGED
18	INFO	The `transferAnyERC20Token` function should use `safeTransferFrom`	ACKNOWLEDGED
19	INFO	Ambiguous reversion reasons for `transfer` and `transferFrom` functions	ACKNOWLEDGED
20	INFO	The `approveAndCall` and `safeApproveAndCall` use `memory` instead of `calldata` on parameter	ACKNOWLEDGED
21	INFO	`totalSupply()` does not reflect `TOTAL_SUPPLY`	ACKNOWLEDGED

1.3.3 TestNftPositionManager

No issues found.

1.3.4 Owned

ID	Severity	Summary	Status
22	LOW	Ownership cannot be renounced	ACKNOWLEDGED
23	INFO	Lack of events for `transferOwnership`	ACKNOWLEDGED
24	INFO	Ambiguous errors	ACKNOWLEDGED

2 Findings

2.1 SeasonalTokenFarm

The SeasonalTokenFarm is a staking contract which allows investors to stake NFT-like liquidity tokens. Although the type of token was not included within the scope of this audit, we assume it to be ERC721 wrapped Uniswap V3 tokens. Each LP is rewarded with the four seasonal tokens. However, the allocation points of the LPs vary over time and are automatically adjusted every 9 months according to a repeating schedule over four 9 month periods. Currently the contract has no notion of governance and runs off donations.

2.1.1 Privileged Roles

None

2.1.2 Issues & Recommendations

Issue #01	**`receiveApproval` and `receiveSeasonalTokens` do not validate the `msg.sender` causing anyone to be able to pull approved tokens from anyone into the farm**
Severity	HIGH SEVERITY

Location	Lines 225-227
	`function receiveApproval(address from, uint256 tokens, address token, bytes calldata data) public override {`
	`    data; // suppress unused variable compiler warnings`
	`    receiveSeasonalTokens(from, token, tokens);`
	
	Lines 230-236
	`function receiveSeasonalTokens(address from, address tokenAddress, uint256 amount) public {`
	`. . .`
	`ERC20Interface(tokenAddress).transferFrom(from, address(this), amount);`

Description	The `receiveApproval` and `receiveSeasonalTokens` functions do not validate the sender of the transaction and allow anyone to determine the wallet that sends the actual tokens using the from parameter. This means a malicious party can set the from parameter to any wallet that has approved ERC-20 tokens to the farm. Non-seasonal-tokens can furthermore be transferred in and are then lost forever. ❗ The function furthermore uses `transferFrom` which does not validate the return value of the transfer.
Recommendation	Consider removing the from parameter completely or adding validation that this can only be called from the Season Tokens. Consider also using `safeTransferFrom` by OpenZeppelin.
Resolution	RESOLVED `receiveSeasonalTokens` is internal while `receiveApproval` can now only be called by the tokens.

Issue #02	Contract uses homogeneous liquidity parameter for heterogeneous liquidity NFT tokens
Severity	MEDIUM SEVERITY
Description	The contract aggregates the 'liquidity' parameter of different liquidity NFTs. As these tokens are non-fungible, we are unsure whether this parameter always denotes the same number of value. Digging slightly deeper into the intentions, it seems like the client wants to use the Uniswap V3 NFT position manager which is out of the scope of this audit. However, it does seem like there is still some tick range permitted which makes us wonder how homogeneous this "liquidity" parameter will be in the value it denotes. An attacker could strategically create LP NFTs which perhaps have a high liquidity parameter but a lower value.
Recommendation	Consider carefully validating which NFT position manager will be used, in case this is the standard Uniswap v3 position manager, consider carefully validating what the "liquidity" parameter means and whether it can be abused or manipulated by third parties. The goal of the farm should be that each NFT might have a different "liquidity" parameter, but it is directly linear with the value of all staked NFTs.
Resolution	RESOLVED The client now requires that the liquidity tokens need to have exactly the same tick range and more importantly need to have the same fee tier.

Issue #03	**The `withdraw` function does not adhere to checks-effects-interactions**
Severity	LOW SEVERITY
Description	The `withdraw` function does not have reentrancy guards and is not written in checks-effects-interactions which can cause the drainage of the rewards and staking token if ever an ERC-777 or similar token is added as the staking token. This is because such tokens allow the sender and recipient to execute arbitrary code during the sending. This issue has been marked as low severity as the seasonal reward tokens that we were provided do not seem to contain reentrancy vectors on the standard transfer functions.
Recommendation	Consider marking the above functions compliant with checks-effects-interactions. If a reentrancy-guard is used, consider validating that all other functions which remain unguarded can be safely reentered into, or consider adding guards to these functions as well.
Resolution	RESOLVED The code has been rewritten to adhere to the checks-effects-interactions pattern.

Issue #04	**`hasDoubledAllocation` cycles after 4 intervals (36 months) have elapsed which is not inline with the contract description**
Severity	LOW SEVERITY
Description	The contract description states: *A few months after each token's halving, the `allocationSize` for the ETH/Token trading pair doubles* Within the code, this behavior is encoded within the `hasDoubledAllocation` method which will double the allocations of the tokens for subintervals every 4 intervals. The Spring token is doubled for the 3 last intervals. The Summer token for the 2 last intervals. The Autumn token for the last interval. The Winter token is never doubled. After 4 intervals are finished, all tokens again receive a single allocation, which means all tokens except Winter are halved in allocation. Since these are allocations, this sort of approximates the behavior the client is looking for but due to the cyclical behavior this is not exact and tokens lose their allocation points again after each cycle. Furthermore, these cycles might not line up with the Spring token cycles, as the farm cycles are based on the farm deployment time.
Recommendation	Consider whether the current cyclical behaviour, which does not exactly match the contract description, is desirable. If not, consider adhering to the contract description exactly and simply doubling allocations each period.
Resolution	RESOLVED The client has indicated that this is in fact the desired behavior and that if the allocation points would be normalized, they exactly resemble what would be expected by the contract description.

Issue #05	**`nextWithdrawalTime` and `getPayoutSizes` can be made external**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract can be marked as external to indicate this behavior to third-party reviewers and to save gas on certain occasions.
Recommendation	Consider marking the above functions as external.
Resolution	RESOLVED

Issue #06	**The `onERC721Received` use `memory` instead of `calldata` on parameter**
Severity	INFORMATIONAL
Description	The `onERC721Received` uses memory for a data parameter instead of calldata. Using `calldata` instead of `memory` can save gas and make the parameter `immutable`.

Recommendation	Consider modifying data parameter from
	`…, bytes memory data)`
	to
	`… , bytes calldata data)`

Resolution	RESOLVED

Issue #07	Usage of if-revert pattern compared to require
Severity	INFORMATIONAL
Description	Throughout the contract, the if-revert pattern is used instead of requiring the inverse logical expression. Using the if-revert pattern is not considered best practice within solidity and might confuse auditors and third-party reviewers that exclusively review solidity codebases.
Recommendation	Consider using `require` wherever possible instead of if-revert patterns.
Resolution	RESOLVED

Issue #08	Unused `tokenOwner` within the `harvest` functions
Severity	INFORMATIONAL
Description	The `harvestSpring`, `harvestSummer`, `harvestAutumn` and `harvestWinter` functions all take a `tokenOwner` parameter which is not used.
Recommendation	Consider removing the `tokenOwner` parameter from the above functions.
Resolution	RESOLVED The `tokenOwner` variable has been removed

Issue #9	**`INonfungiblePositionManager` wrongly defines the ERC-721 `safeTransferFrom` function**
Severity	INFORMATIONAL
Location	Line 30 `function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;`
Description	The `INonfungiblePositionManager` defines `safeTransferFrom` function with a payable modifier which is not a standard in ERC721 `safeTransferFrom`.
Recommendation	Consider removing the payable modifier.
Resolution	RESOLVED

Issue #10	**Gas optimization: Using `msg.sender` might be cheaper within `harvest`**
Severity	INFORMATIONAL
Description	The `harvest` function repeatedly reads the owner from storage. It is however already checked that this owner must be equal to `msg.sender`.
Recommendation	Consider always using `msg.sender` within the `harvest` function to save on storage reading costs. Alternatively, the owner can be cached to memory.
Resolution	RESOLVED

2.2 Season Tokens

The Season tokens are four different tokens that are part of the Seasonal protocol. They are minted through a PoW algorithm similar to how block mining within Bitcoin works. The seasonal tokens each have a different initial era offset and then halve in emissions every 3 years. The initial eras are scheduled in a manner to allow each halving to take 9 months since the previous one, similar to how the seasons would progress.

Rewards are distributed once someone submits a nonce that solves the mining challenge with the mining target. Users need to use off-chain mining software to figure out the correct solution. Once a correct nonce is submitted, the sender receives a reward and a new challenge is generated.

2.2.1 Token Overview

Token Name	SpringToken
Address	0xf04aF3f4E4929F7CD25A751E6149A3318373d4FE
Token Supply	33,112,800
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

Token Name	SummerToken
Address	0x4D4f3715050571A447FfFa2Cd4Cf091C7014CA5c
Token Supply	33,112,800
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

Token Name	AutumnToken
Address	0x4c3bAe16c79c30eEB1004Fb03C878d89695e3a99
Token Supply	33,112,800
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

Token Name	WinterToken
Address	0xCcbA0b2bc4BAbe4cbFb6bD2f1Edc2A9e86b7845f
Token Supply	33,112,800
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.2.2 Privileged Roles

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

- `transferAnyERC20Token`
- `transferOwnership`
- `acceptOwnership`

2.2.3 Issues & Recommendations

Issue #11	Merged mining is still theoretically possible
Severity	MEDIUM SEVERITY
Location	Line 209 `return bytes32(uint256(blockhash(block.number - 1)) ^ _tokensMinted ^ TOKEN_IDENTIFIER);`
Description	Currently the challenge number, which is the mathematical challenge for miners to guess, is based upon a XOR between the previous block hash, the number of tokens minted and the token identifier. However, there might still be overlap with this setup as it could first of all be that people fork season token having an identical `TOKEN_IDENTIFIER` but more importantly `_tokensMinted ^ TOKEN_IDENTIFIER` can result in the same outcome for unique values. For example, if `_tokensMinted` is 2 and `TOKEN_IDENTIFIER` is 1, this would result in the same challenge as `_tokensMinted` being 1 and `TOKEN_IDENTIFIER` 2 if the two tokens are minted on the same block. Although this might be quite an edge case, it might be cleaner to develop a challenge generating method that does not have such side-effects, as this is an indication that the current method is flawed.
Recommendation	Consider using a less flawed challenge method, for example using `address(this)` as the `TOKEN_IDENTIFIER` and hashing the previous challenge number for a pseudorandom new one. We still include the blockhash to prevent potential pre-mining. `return keccak256(abi.encodePacked(challengeNumber, blockhash(block.number - 1)), address(this));`

Resolution

RESOLVED

The client has explained that some overlap is always to be expected with mining algorithms. However, the primary concern which is that the `_tokensMinted` and `TOKEN_IDENTIFIER` could overlap has been addressed by the fact that `_tokensMinted` is in fact always going to be a very large number and not just 1 or 2 since 1 real token is represented as 10^18.

The client has however acknowledged that the recommendation would have been a better implementation as it would protect against blind forks of the contract potentially merge mining. As it's going to still be quite difficult to align the block and `tokensMinted` variables perfectly this issue has been marked as resolved given that the primary concern was in fact not present. Forks should still consider to use the recommendation.

Issue #12	Lack of difficulty adjustment period is prone to miner collusion
Severity	LOW SEVERITY
Description	Currently the difficulty is adjusted after every mint: If the mint was quick, difficulty is increased, if it was slow, difficulty is decreased. Such a mechanism is very prone to miner collusion as there is incentive to wait a while and then mine again repeatedly. This was famously done with Bitcoin Cash using their emergency difficulty adjustment mechanism. ❗ It should be noted that a longer adjustment period might cause lag in the initial periods where the difficulty is still adjusting.
Recommendation	Consider adding a difficulty adjustment period that balances the adjustment lag and the collusion risk. If this is not done, consider setting the response of `getAdjustmentInterval` to zero as there might not be an adjustment interval.
Resolution	RESOLVED The client has added test cases and simulations to the test suite that aim to validate that this behavior is unlikely to occur. In addition, the client has explained that such a scheme is actually used very closely within Ethereum. The client has furthermore walked us through the steps why this issue will not have consequences and why Ethereum was comfortable with a similar design.

Issue #13	Mint event does not emit exact epochCount
Severity	INFORMATIONAL
Location	Lines 134-135 `emit Mint(msg.sender, totalRewardAmount, _scheduledNumberOfRewards(block.timestamp), newChallengeNumber);`
Description	Instead of returning the exact epoch count within the mint event, the mint event returns the ex-ante expected number of "blocks" minted.
Recommendation	Consider keeping track of the number of "blocks" minted and using this for the `epochCount` variable within this event.
Resolution	ACKNOWLEDGED This behavior has been designed like this to save on gas costs.

Issue #14	Usage of if-revert pattern compared to `require`
Severity	INFORMATIONAL
Location	Line 110 `if (uint256(digest) > _miningTarget) revert("Digest is larger than mining target");`
Description	The code that validates that the `mint` must revert if an invalid nonce is provided is written in the traditional flow where the non happy-path is given and the code reverts if it is reached. However, within Solidity it is considered best practice to define the happy-path in a require statement.
Recommendation	Consider adjusting the previous requirement to use a require statement. `require (uint256(digest) <= _miningTarget, "Digest is larger than mining target");`
Resolution	ACKNOWLEDGED

Issue #15	Batching mechanism might lead to adverse incentives if mining power is highly concentrated
Severity	INFORMATIONAL
Description	The contract contains a batching mechanism which allows the miners to receive extra rewards if their solution is sufficiently better than what is required. This is included to allow miners to voluntarily mine for a more difficult solution without the difficulty decreasing as a way to signal that gas fees are too high right now. However, if there are only a few miners, this might create incentive for them to collude and wait for better solutions while they are mining too slowly. Paladin has tried to look for scenarios where this might incentivize incorrect behavior but was not able to find such scenarios ourselves. However, we've decided to still include this as an issue as we were also unable to prove that the current more complex mining mechanism does not cause odd incentives.
Recommendation	Consider carefully simulating whether the batching mechanism has any adverse effects. If so, it might be better to adjust the reward period currently static at 10 minutes using a secondary adjustment loop.
Resolution	ACKNOWLEDGED The client has indicated that the batching mechanism was carefully designed to avoid introducing opportunities for abuse but this method has never been tried before and there could be unforeseen complications in the future. We agree that a mechanism to buffer against fee costs is valuable.

Issue #16	**`name`, `symbol`, `totalSupply` and decimals functions can be made `external`**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract can be marked as external to indicate this behavior to third-party reviewers and to save gas on certain occasions.
Recommendation	Consider marking the aforementioned functions as external.
Resolution	ACKNOWLEDGED

Issue #17	**Gas optimization: Unnecessary `else` clause**
Severity	INFORMATIONAL

Location	Lines 232-236
	`if (_miningTarget < MINIMUM_TARGET)`
	`    _miningTarget = MINIMUM_TARGET;`
	
	`if (_miningTarget > MAXIMUM_TARGET)`
	`    _miningTarget = MAXIMUM_TARGET;`

Description	The `_adjustDifficulty` logic if statements are non-overlapping as long as `MAXIMUM_TARGET >= MINIMUM_TARGET`. The second statement could therefore be optimized using `else if`.
Recommendation	Consider using `else if` for the second statement.
Resolution	ACKNOWLEDGED

Issue #18	**The `transferAnyERC20Token` function should use `safeTransferFrom`**
Severity	INFORMATIONAL
Location	Line 495 `ERC20Interface(tokenAddress).transfer(owner, tokens);`
Description	The `transferAnyERC20Token` function is used by the owner to transfer any ERC20 tokens that were transferred by mistake to the contract. It is recommended as a standard in ERC20 to use the `safeTransferFrom` function instead of `transfer`. ❗ In addition, this function does not emit an event on success. Functions that affect the status of sensitive variables should emit events as notifications.

Recommendation	Consider replacing
	`ERC20Interface(tokenAddress).transfer(owner, tokens);`
	
	with
	`ERC20Interface(tokenAddress).safeTransferFrom(address(this), owner, tokens, "");`
	
	Also consider adding an event for the above function.

Resolution	ACKNOWLEDGED

Issue #19	**Ambiguous reversion reasons for `transfer` and `transferFrom` functions**
Severity	INFORMATIONAL
Description	The `transfer` and `transferFrom` functions have ambiguous revert messages. `require(to != address(0) && to != address(this), "Invalid address");`
Recommendation	Consider adding an explicit reversion message for both functions.
Resolution	ACKNOWLEDGED

Issue #20	**The `approveAndCall` and `safeApproveAndCall` use `memory` instead of `calldata` on parameter**
Severity	INFORMATIONAL
Description	The `approveAndCall` and `safeApproveAndCall` use `memory` on data parameter instead of `calldata`. Using `calldata` instead of `memory` will save gas and make the parameter `immutable`.

Recommendation	Consider modifying data parameter from
	`…, bytes memory data)`
	to
	`… , bytes calldata data)`

Resolution	ACKNOWLEDGED

Issue #21	`totalSupply()` does not reflect `TOTAL_SUPPLY`
Severity	INFORMATIONAL
Description	`TOTAL_SUPPLY` variable is used to keep the maximum supply that a token can have, therefore having `totalSupply()` function returning the current supply that has been mined. It should furthermore be noted that `TOTAL_SUPPLY` is not an exact `MAX_SUPPLY` but approximate, assuming that the token emission rate remains at one token per 10 minutes. There could therefore be less tokens in final circulations if minting drops below this rate due to gas cost for example.
Recommendation	Consider renaming `TOTAL_SUPPLY` into `MAX_SUPPLY` to reflect the actual purpose of this variable.
Resolution	ACKNOWLEDGED

2.3 TestNftPositionManager

The TestNftPositionManager is a simple NFTPosition manager that mimics Uniswap V3 NonfungiblePositionManager. It lets you create pools of token pairs. It has been created only for testing of SeasonalTokenFarm purposes. Since it is clearly not fit for production as anyone can call "createLiquidityToken" to create mock positions, we refrain from pointing out informational issues like the fact that certain functions miss events or can be marked as external.

2.3.1 Issues & Recommendations

No issues found.

2.4 Owned

The Owned dependency contract allows for ownership definition and transfership functions to be included in derivative contracts. It is similar to Ownable but includes the Claimable pattern.

2.4.1 Issues & Recommendations

Issue #22	Ownership cannot be renounced
Severity	LOW SEVERITY
Description	Currently there is no functionality to renounce the ownership which means setting it to the zero address. If the community desires that ownership should be renounced, this might be a difficult thing to do.
Recommendation	Consider adding a `renounceOwnership` method, don't forget to also zero out the `newOwner`.
Resolution	ACKNOWLEDGED The client has indicated that the farm contract doesn't have an owner so this issue only affects the seasonal token contracts, which have already been deployed.

Issue #23	Lack of events for `transferOwnership`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	ACKNOWLEDGED

Issue #24	Ambiguous errors
Severity	INFORMATIONAL
Description	None of the errors within the Owned contract have an error message. This could be confusing for users when these require statements will revert, as there is no error message to explain why this revert happened.
Recommendation	Consider adding error messages.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY