Findings

Stats

Severity	Bug Count
Critical	2
Medium	2
Low	1
Informational	1
Total	6

Critical Risk

[C-01] Missing access control in `updateFairLaunchProperties()` function

Description:

The `updateFairLaunchProperties()` function allows external callers to update the `fairLaunchProperties` variable, which is used to determine whether the token is available for buying or selling. This function lacks proper access control, enabling any unauthorized user to modify the `fairLaunchProperties` variable

Impact:

Unauthorized users could enable or disable token trading, leading to potential market manipulation or disruption.

Recommendation:

To remediate this issue, it is recommended to implement access control mechanism in `updateFairLaunchProperties()` function. This can be achieved using Solidity's `onlyOwner` modifier from the OpenZeppelin library or a custom access control mechanism.

```
- function updateFairLaunchProperties(FairLaunchProperties memory
_fairLaunchProperties) external returns (bool) {....}
+ function updateFairLaunchProperties(FairLaunchProperties memory
_fairLaunchProperties) external onlyOwner returns (bool) {....}
```

[C-02] `getPrice()` will always revert due to absence of `fallback()` and `receive()` functions

Description:

The `getPrice()` function is designed to calculate the price of `saleToken` by dividing the contract's ETH balance by the balance of `saleToken` held by the contract. However, the function will always revert due to the absence `fallback()` or `receive()` function, and the constructor is also not marked as payable. As a result, the ETH balance of the contract will always be zero, causing the function to revert whenever it is called.

Code Snippet

```
    function getPrice() public view returns (uint256) {
        ...

        if (saleTokenBalance == 0) revert saleTokenBalanceZeroError();
@>      if (ethBalance == 0) revert EthBalanceZeroError();

        ...
    }
```

Impact:

Due to absence of Eth balance in the contract, `getPrice()` will always revert and no user will be able to buy or sell tokens

Recommendation:

To remediate this issue, it is recommended to implement either a `receive` function or a `fallback` function. Additionally, ensure the constructor is marked as payable if ETH is intended to be sent during contract deployment.

Medium Risk

[M-01] Missing blacklist management for `onlyNotBlacklisted()` modifier

Description:

The `buy()` function uses the `onlyNotBlacklisted` modifier to prevent blacklisted addresses from purchasing `saleToken`. However, there is no implementation provided to add or manage blacklisted addresses, even though the modifier checks for a role "BLACKLISTED_ROLE". Without the ability to assign or revoke this role, the blacklist functionality is incomplete and ineffective.

Impact:

This issue can lead to unauthorized users being able to purchase `saleToken`, potentially bypassing security measures intended to prevent certain addresses from participating in the fair launch.

Remediation:

It is recommended to implement functions to manage the blacklist by assigning or revoking the "BLACKLISTED_ROLE".

[M-02] User won't be able to buy token if he has less ETH in his wallet after paying for Token

Severity: Medium

Context: hyacinth_test.sol#L199

Description:

In the `buy()` function, there is a check to ensure that the sender's ETH balance is not less than the ETH amount sent:

```
if (address(msg.sender).balance < ethAmount)
    revert InsufficientETHBalanceError();
```

This check is redundant and inaccurate because it checks the balance of `msg.sender` after the ETH has already been sent to contract. In some cases, user balance might be lower by the amount of `ethAmount`, depends on user balance, but the user has already paid enough to buy token.

Impact:

This can cause unnecessary reverts and prevent users with sufficient ETH from purchasing `saleToken` if their balance becomes lower than `ethAmount` (which is his personal matter) after sending it to the contract.

Remediation:

Remove the redundant balance check to avoid unnecessary reverts:

```
// Remove this check
// if (address(msg.sender).balance < ethAmount) revert
InsufficientETHBalanceError();
```

Low Risk

[L-01] Use `safeTransfer()`/`safeTransferFrom()` instead of `transfer`/`transferFrom`

Description: The `buy()` and `sell()` function uses the `transfer` and `transferFrom` method to transfer `saleToken` from buyer and seller.

Using `transfer`/`transferFrom` can be unsafe because it does not handle all possible scenarios for token transfer failures. Instead, using `safeTransfer`/`safeTransferFrom` from the OpenZeppelin `SafeERC20` library ensures that the transfer will revert if it fails.

Remediation: Use the `safeTransfer`/`safeTransferFrom` method from the OpenZeppelin `SafeERC20` library to handle token transfers safely:

Informational

[I-01] Dead Code

Description:

The contract includes an internal function `_notZeroAddress()` and a modifier `notZeroAddress()` to check if an address is the zero address:

```
function _notZeroAddress(address _address) internal pure {
    if (_address == address(0)) revert ZeroAddressError();
}

modifier notZeroAddress(address _address) {
    _notZeroAddress(_address);
    _;
}
```

However, if these are not utilized anywhere in the contract, they become dead code, which can increase the complexity of the codebase without providing any functionality.

Remediation: Remove the dead code if it is not being used anywhere in the contract.