External SHIELD

Pyrand

Smart Contract Audit Report

April 19th, 2024

externalshield.com

Table of Content

Executive Summary

Overview

PROJECT DETAIL	
Project Name	Pyrand
Language	Solidity
Scope	https://etherscan.io/address/0x67D8216cfC3CCDdA18CAf204034cEd50d4A7Ff8d
nSLOC	148

Vulnerability Summary

Vulnerability Level	Total	Acknowledged	Fixed	Declined
High	**0**	**0**	**0**	**0**
Medium	**5**	**5**	**0**	**0**
Low	**2**	**2**	**0**	**0**
Informational	**6**	**6**	**0**	**0**

Audit Scope

- **Smart Contracts**
  - PYRXPresale.sol

Vulnerability Severity Classification

- **High** - vulnerabilities where assets face direct threats of theft, loss, or compromise. Exploitation of these vulnerabilities can result in significant harm to the system or assets.

- **Medium** - vulnerabilities imply that assets are not at direct risk, but the function of the protocol or its availability could be impacted with a hypothetical attack path with stated assumptions, but external requirements.

- **Low** - It involve non-critical issues that do not directly impact the security or functionality of the system.

- **Informational** - Informational vulnerabilities focus on aspects such as code style, syntax, providing recommendations for improvement in non-critical areas to enhance code readability, maintainability, or compliance with best practices. These vulnerabilities do not pose any direct risk to the system or assets.

Methodology

Our audit methodology combines industry best practices, security standards, and specialized tools to ensure a comprehensive evaluation of smart contracts. Key aspects of our methodology include:

- **Code Review:** Analyzing the smart contract code line by line to identify vulnerabilities and potential risks.

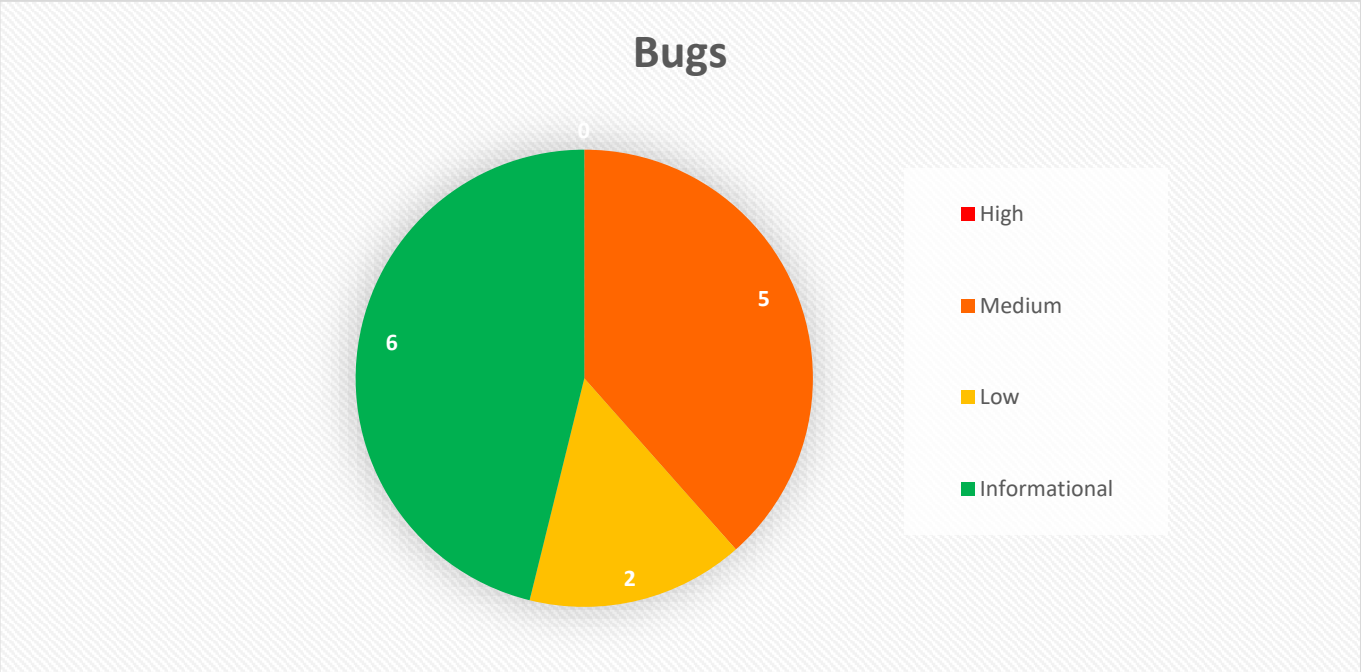
- **Security Testing:** Conducting various tests, including static analysis, dynamic analysis, and manual review, to uncover security weaknesses

- **Functional Testing:** Verifying the smart contract's functionality against intended specifications and business logic.

- **Risk Assessment:** Evaluating risks associated with the smart contract's design, implementation, and deployment.

By following this structured approach and methodology, ExternalShield delivers comprehensive smart contract audits that provide valuable insights and recommendations to enhance security and functionality.

Findings



ID	Name	Severity	Status
M-1	Chainlink's `latestRoundData` might return stale or incorrect results Description	Medium	Acknowledged
M-2	Missing check for `ethDivider` leads to no tokens for users and lost ETH	Medium	Acknowledged
M-3	Premature Withdrawal of Claimable Tokens Before Presale Ends	Medium	Acknowledged
M-4	Token rates can be changed once presale starts	Medium	Acknowledged

ID	Name	Severity	Status
M-5	Centralization Risk for trusted owners	Medium	Acknowledged
L-1	Lack of two step ownership transfer	Low	Acknowledged
L-2	Missing checks for `address(0)` when assigning values to address state variables	Low	Acknowledged
NC-1	`address’es` shouldn't be hard-coded	Informational	Acknowledged
NC-2	Deprecated `SafeMath` library used for Solidity `>= 0.8`	Informational	Acknowledged
NC-3	`public` functions not called by the contract should be declared `external` instead	Informational	Acknowledged
NC-4	Missing events for critical parameter change	Informational	Acknowledged
NC-5	Gas Optimization - `a = a + b` is more gas effective than `a += b` for state variables (excluding arrays and mappings)	Gas	Acknowledged
NC-6	Gas Optimization - Use Custom Errors instead of Revert Strings to save Gas	Gas	Acknowledged

Medium Severity Issues

M-1 : Chainlink's `latestRoundData` might return stale or incorrect results

Description:

On `ChainlinkPriceOracle.sol`, you are using `latestRoundData`, but there is no check if the return value indicates stale data.

`latestRoundData()` is used to fetch the asset price from a Chainlink aggregator, but it's missing additional validations to ensure that the round is complete. If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the Chainlink system) consumers of this contract may continue using outdated stale data / stale prices.

This could lead to stale prices according to the Chainlink documentation: https://docs.chain.link/data-feeds/price-feeds/historical-data

Code Location:

```
692:     function getLatestPrice() public view returns (int) {
    (
        ,
        int price,
        ,
        ,
    ) = priceFeed.latestRoundData();
    // for BNB / USD price is scaled up by 10 ** 8
    return price / 1e8;
}
```

Recommendation:

Add the below check for returned data

```
function getLatestPrice() public view returns (int) {
       (
           uint80 roundID,
           int price,
           uint256 timestamp,
           uint256 udatedAt,
       ) = priceFeed.latestRoundData();
       // for BNB / USD price is scaled up by 10 ** 8
       return price / 1e8;
   }
require(updatedAt >= roundID, "Stale price");
require(timestamp != 0,"Round not complete");
require(answer > 0,"Chainlink answer reporting 0");
```

Status:

Acknowledged

M-2 : Missing check for `ethDivider` leads to no tokens for users and lost ETHs

Description:

The `changeRate()` function allows the contract owner to set the `ethDivider` value. It has check for the `_usdtRate` and _ethT0Usd` that it should be greater than zero, but there is not check for the ethDivider.  If the `ethDivider` is set to `0`, it will lead to users receiving `0` tokens when they try to buy tokens with ETH. This is because the number of tokens a user receives is calculated by multiplying the amount of ETH by the ETH price and then dividing by `ethDivider`.

```
uint256 tokens = weiAmount.mul(uint256(ethPrice)).mul(ethDivider).div(tokenPriceUSD);
```

If `ethDivider` is 0, the result will always be 0 regardless of the amount of ETH sent. Users eth can be lost.

Code Location:

```
781:  uint256 tokens =
      weiAmount.mul(uint256(ethPrice)).mul(ethDivider).div(tokenPriceUSD);
```

```
850: function changeRate(uint256 _usdtRate, uint256 _ethToUsd, uint256 divider) public
onlyOwner {
         require(_usdtRate > 0, "Rate cannot be 0");
         require(_ethToUsd > 0, "Rate cannot be 0");
         tokenPriceUSD = _ethToUsd;
         usdtRate = _usdtRate;
         ethDivider = divider;
     }
```

Recommendation:

add check,

```
`require(ethDivider > 0, “ethDivider cannot be 0”);`
```

Status:

Acknowledged

M-3 : Premature Withdrawal of Claimable Tokens Before Presale Ends

Description:

The withdraw function in the PYRXPresale contract allows the contract owner to withdraw claimable tokens before the presale has officially ended. This functionality can lead to a scenario where the tokens intended for the presale can be depleted by the owner, potentially leaving no tokens for participants to claim. This undermines the integrity of the presale process and can result in loss of trust for the participants.

Code Location:

```
822: function withdraw(uint amount) external onlyOwner {
        require(
            token.balanceOf(address(this)) > 0,
            "There are not enough tokens in contract"
        );
        require(claimableTokens >= amount, "not enough tokens to withdraw");
        token.safeTransfer(msg.sender, amount);
        claimableTokens -= amount;
    }
```

Recommendation:

allow owners to withdraw unsold claimable tokens once presale ended

Status:

Acknowledged

M-4 : Token rates can be changed once presale starts

Description:

The `changeRate()` function allows the contract owner to change the USD to ETH conversion rate (_ethToUsd), the USDT rate (_usdtRate), and the divider (ethDivider) used in token purchase calculations. This rates can be change once the presale has started, allowing for arbitrary adjustments to the token pricing mechanism, potentially disadvantaging users or manipulating the presale outcome.

Code Location:

```
  850: function changeRate(uint256 _usdtRate, uint256 _ethToUsd, uint256 divider) public
onlyOwner {
          require(_usdtRate > 0, "Rate cannot be 0");
          require(_ethToUsd > 0, "Rate cannot be 0");
          tokenPriceUSD = _ethToUsd;
          usdtRate = _usdtRate;
          ethDivider = divider;
      }
```

Recommendation:

Restrict the rates once the presale starts

Status:

Acknowledged

M-5 : Centralization Risk for trusted owners

Description:

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Code Location:

```
717: contract PYRXPresale is Ownable {
```

```
823:     function deposit(uint amount) external onlyOwner {
```

```
830:     function withdraw(uint amount) external onlyOwner {
```

```
853:     function changeWallet(address payable _wallet) external onlyOwner {
```

```
858:     function changeRate(uint256 _usdtRate, uint256 _ethToUsd, uint256 divider)
public onlyOwner {
```

Recommendation:

Implement a decentralized governance mechanism, such as a multisig wallet or a DAO, to replace the single owner. This would distribute the power to make critical changes to the contract among multiple trusted parties, reducing the risk of a single point of failure.

Status:

Acknowledged

Low Severity Issues

L-1 : Lack of two step ownership transfer

Description:

here `Ownable` contract used to transfer ownership of the contract using `transferOwnership()`. Recommend considering implementing a two step process where the owner or admin nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two step procedure on the critical functions.

Code Location:

```
103: function transferOwnership(address newOwner) public virtual onlyOwner {
      require(
          newOwner != address(0),
          "Ownable: new owner is the zero address"
      );
      _transferOwnership(newOwner);
  }
```

Recommendation:

implement two-step ownership transfer like this

```
function setPendingOwner(address _newPendingOwner) external onlyOwner {
    require(_newPendingOwner != address(0));
  pendingOwner = _newPendingOwner;
}
```

```
function acceptOwnership() external {
    require(msg.sender == pendingOwner);
    owner = msg.sender;
    pendingOwner = address(0);
}
```

Status:

Acknowledged

L-2 : Missing checks for `address(0)` when assigning values to address state variables

Description:

The `changeWallet()` function in the `PYRXPresale` contract allows the contract owner to change the wallet address where funds are collected. However, there is no check to prevent the owner from setting the wallet address to `address(0)`. This could lead to loss of funds as any funds sent to `address(0)` are irretrievable.

Code Location:

```
845: function changeWallet(address payable _wallet) external onlyOwner {
        wallet = _wallet;
    }
```

Recommendation:

add check for `address(0)`

Status:

Acknowledged

Informational Issues

NC-1 : `address’es` shouldn't be hard-coded

Description:

It is often better to declare `address’es` as `immutable`, and assign them via constructor arguments. This allows the code to remain the same across deployments on different networks, and avoids recompilation when addresses need to change.

Code Location:

```
681: priceFeed = AggregatorV3Interface(0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419);
```

```
724:IERC20 public immutable token = IERC20(0xE58278e99f0c21c0216396aD8ce1E9Da36EDEFCC);
```

```
727: IERC20 public immutable usdt = IERC20(0xdAC17F958D2ee523a2206206994597C13D831ec7);
```

```
733: address public wallet = payable(0xe6CbDc506D6459395f4659288Bd84A9d1c643BD1);
```

Recommendation:

remove hardcoded addresses and assign them via constructor arguments

Status:

Acknowledged

NC-2 : Deprecated `SafeMath` library used for Solidity `>= 0.8`

Description:

The `SafeMath` library is being used in the PYRXPresale contract. However, this library is deprecated for Solidity versions >= 0.8 because overflow and underflow checks are built into the compiler since version `0.8`

Code Location:

```
711: using SafeMath for uint256;
```

Recommendation:

Remove safeMath library for solidity version >=0.8

Status:

Acknowledged

NC-3 : public functions not called by the contract should be declared external instead

Description:

In the given contract, there are several public functions that are not called internally within the contract. These functions should be declared as `external` to make the contract more gas efficient and to clearly signal that these functions are not intended to be used internally.

Code Location:

```
760:     function startPresale() public onlyOwner{
```

```
765:     function endPresale() public onlyOwner{
```

```
797:     function buyTokensWithUSDT(uint256 amount) public {
```

```
833:     function claimTokens() public {
```

```
850:     function changeRate(uint256 _usdtRate, uint256 _ethToUsd, uint256 divider)
public onlyOwner {
```

```
864:     function getETHPriceInUSD() public view returns (int256) {
```

```
868:     function getDivider() public view returns (uint256) {
```

```
872:     function getTokenUsdPrice() public view returns (uint256) {
```

```
876:     function getUsdtRate() public view returns (uint256) {
```

```
880:     function progressETH() public view returns (uint256) {
```

```
884:     function progressUSDT() public view returns (uint256) {
```

```
888:     function soldTokens() public view returns (uint256) {
```

```
892:     function checkPresaleStatus() public view returns (bool) {
```

```
896:     function checkPresaleEnd() public view returns (bool) {
```

```
900:     function getClaimableTokens() public view returns (uint256) {
```

Recommendation:

change functions visibility from public to external

Status:

Acknowledged

NC-4 : Missing events for critical parameter change

Description:

The current implementation lacks an event emission upon changing the wallet address, Token Price Update, Presale start/end, Ownership trasnfer, Deposit/Withdraw Tokens - which reduces transparency and makes it difficult for external observers (e.g., users, external contracts, or services) to track changes in the contract state.

Code Location:

```
760: function startPresale() public onlyOwner{
        hasPresaleStarted = true;
    }
```

```
765: function endPresale() public onlyOwner{
        hasPresaleEnded = true;
    }
```

```
815: function deposit(uint amount) external onlyOwner {
        require(amount > 0, "Deposit value must be greater than 0");
        token.safeTransferFrom(msg.sender, address(this), amount);
        claimableTokens += amount;
    }
```

```
822: function withdraw(uint amount) external onlyOwner {
        require(
            token.balanceOf(address(this)) > 0,
            "There are not enough tokens in contract"
        );
        require(claimableTokens >= amount, "not enough tokens to withdraw");
        token.safeTransfer(msg.sender, amount);
        claimableTokens -= amount;
    }
```

```
845: function changeWallet(address payable _wallet) external onlyOwner {
        wallet = _wallet;
    }
```

Recommendation:

emit an event whenever critical parameters

```
event WalletChanged(address indexed oldWallet, address indexed newWallet);
```

Status:

Acknowledged

NC-5 : Gas Optimization - `a = a + b` is more gas effective than `a += b` for state variables (excluding arrays and mappings)

Description:

It saves 16 gas per instance. There is 7 instance in the contract. Using `a = a + b` is more gas effective than `a += b` for state variables excluding arrays and mappings.

Code Location:

```
786:    weiRaised += weiAmount;
```

```
788:    Customer[msg.sender].tokensBought += tokens;
789:    tokensSold += tokens;
```

```
805:    usdtRaised += amount;
```

```
807:    Customer[msg.sender].tokensBought += tokens;
808:    tokensSold += tokens;
```

```
818:    claimableTokens += amount;
```

Recommendation:

change it to `weiRaised = weiRaised + weiAmount`

Status:

Acknowledged

NC-6 : Gas Optimization - Use Custom Errors instead of Revert Strings to save Gas

Description:

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

Additionally, custom errors can be used inside and outside of contracts (including interfaces and libraries).

Source:
> Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., revert("Insufficient funds.");), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Consider replacing **all revert strings** with custom errors in the solution, and particularly those that have multiple occurrences:

Code Location:

```
771:         require(hasPresaleStarted,"Presale not started");
772:         require(!hasPresaleEnded, "Presale has ended");
773:         require(msg.value >= 0.001 ether, "cant buy less with than 0.001 ETH");
```

```
779:         require(ethPrice > 0, "Invalid ETH price");
```

```
784:         require(claimableTokens >= tokens, "Not enough tokens availible");
```

```
793:         require(callSuccess, "Call failed");
```

```
798:         require(hasPresaleStarted,"Presale not started");
799:         require(!hasPresaleEnded, "Presale has ended");
800:         require(amount >= 1 * 10 ** 6, "cant buy less with than 1 USDT");
```

```
803:         require(claimableTokens >= tokens, "Not enough tokens availible");
```

```
816:         require(amount > 0, "Deposit value must be greater than 0");
```

```
827:         require(claimableTokens >= amount, "not enough tokens to withdraw");
```

```
834:         require(hasPresaleEnded,"Presale has not ended");
835:         require(Customer[msg.sender].tokensBought > 0, "No tokens to be
claimed");
```

```
851:         require(_usdtRate > 0, "Rate cannot be 0");
852:         require(_ethToUsd > 0, "Rate cannot be 0");
```

Recommendation:

Use custom errors.

Status:

Acknowledged

Disclaimer

This disclaimer is intended to outline the limitations and responsibilities associated with the audit report provided by ExternalShield for the smart contract in question.

- The audit report is based on the information and documentation provided to ExternalShield at the time of the audit. Any changes or updates made to the smart contract after the audit may impact its security and functionality.

- While every effort has been made to conduct a thorough audit, it is important to note that no audit can provide absolute assurance of the smart contract's security or functionality.

- The audit report may not cover vulnerabilities or risks arising from third-party dependencies or interactions with external systems beyond the scope of the audit.

- ExternalShield shall not be held liable for any damages, losses, or liabilities arising from the use of the audit report or its recommendations.

Thank you for entrusting us with the audit of your smart contract.