PXP MarketPlace

Smart Contract Security Audit

Prepared by ShellBoxes

April 1st, 2022 - April 8th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Pirate X Pirate
Version	1.0
Classification	Public

Scope

The PXP MarketPlace Contract in the PXP MarketPlace Repository

Files	SHA256 Hash
PXPMarketplace.sol	cb06ab11f620e1801fea725de1247cecbc20da66db1050e9ea7d97d802c6a024

Re-Audit Files

Files	SHA256 Hash
PXPMarketplace.sol	2cc256c49d16152b910f888f8e37d323fca15b830b63e8c7f9e8eca0814a9d2c

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Pirate X Pirate engaged ShellBoxes to conduct a security assessment on the PXP Market-Place beginning on April 1st, 2022 and ending April 8th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Pirate X Pirate

Pirate X Pirate is a blockchain-based NFT adventure game with a turn-based dice combat system. It is built to be a sustainable platform with long-term updates planned. Pirate X Pirate is a world where you are rewarded with in-game money by adventuring across the high seas.Recruit your crew, form your fleet, then harvest resources or test your skills fighting against other pirates to earn.

Issuer	Pirate X Pirate
Website	`https://piratexpirate.io`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the PXP MarketPlace implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 high-severity, 2 medium-severity, 4 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Missing Verification In The Transfer Calls	HIGH	Fixed
Fees Can Be Bypassed	MEDIUM	Fixed
Race Condition	MEDIUM	Fixed
For Loop Over Dynamic Array	LOW	Acknowledged
Missing Value Verification	LOW	Fixed
Missing Address Verification	LOW	Fixed
Floating Pragma	LOW	Fixed
The Seller Does Not Get The Price Stored In The Offe	INFORMATIONAL	Fixed

3 Finding Details

A PXPMarketplace.sol

A.1 Missing Verification In The Transfer Calls [HIGH]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction has passed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks in effect, the transaction would always succeed, even if the token transfer did not.

Code:

Listing 1: PXPMarketplace.sol

```
205 function purchaseItem(uint256 itemId) external {
206     uint256 currentPrice = getCurrentPrice(itemId);
207     require(TOKEN_ADDRESS != address(0), "Token invalid");
208     IERC20 token = IERC20(TOKEN_ADDRESS);
209     require(itemId > 0 && itemId <= itemCount, "Not exists");
210     require(
211         token.balanceOf(msg.sender) >= currentPrice,
212         "Insufficient fund"
213     );
214     Item storage item = _items[itemId];
215     require(item.sold == false, "Sold");
216     require(item.cancelled == false, "Cancelled");
217     uint256 fee = currentPrice.mul(_feePercent).div(10000);
218     uint256 total = currentPrice.sub(fee);
219     token.transferFrom(msg.sender, item.seller, total);
```

```
220     token.transferFrom(msg.sender, _feeWallet, fee);
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require statement to verify that the transfer has passed successfully.

Status - Fixed

The PXP team has solved the issue by using the safeERC20 implementation to perform transfers.

A.2 Fees Can Be Bypassed [MEDIUM]

Description:

At every purchase in the contract, a percentage is taken from the price as fees, then the contract sends these fees to the fee wallet. In the case where the currentPrice variable is lower than 10000/ _feePercent, the fee variable will be equal to 0 due to the type conversion

Code:

Listing 2: PXPMarketplace.sol

```
205 function purchaseItem(uint256 itemId) external {
206     uint256 currentPrice = getCurrentPrice(itemId);
207     require(TOKEN_ADDRESS != address(0), "Token invalid");
208     IERC20 token = IERC20(TOKEN_ADDRESS);
209     require(itemId > 0 && itemId <= itemCount, "Not exists");
210     require(
211         token.balanceOf(msg.sender) >= currentPrice,
```

```
212         "Insufficient fund"
213     );
214     Item storage item = _items[itemId];
215     require(item.sold == false, "Sold");
216     require(item.cancelled == false, "Cancelled");
217     uint256 fee = currentPrice.mul(_feePercent).div(10000);
218     uint256 total = currentPrice.sub(fee);
219     token.transferFrom(msg.sender, item.seller, total);
220     token.transferFrom(msg.sender, _feeWallet, fee);
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

It is recommended to add require statement to make sure the currentPrice is higher than the 10000/_feePercent

Status - Fixed

The PXP team has solved the issue by requiring the price to be higher than hundredPercent/_feeWithDecimals.

A.3 Race Condition [MEDIUM]

Description:

In the contract, the user can purchase any item by calling the purchase function. The admin has the ability to change the fee percentage, if the user calls the purchase function to buy an item then the admin changes the fee percentage, there is a possibility that the owner's transaction gets mined first, that will make the user's transaction execute with the new value of the fee percentage which will cause the seller to pay an unexpected amount of fees.

Code:

Listing 3: PXPMarketplace.sol

```
97  function setFeePercent(uint256 _percent) external onlyRole(ADMIN_ROLE) {
98      _feePercent = _percent;
99  }
```

Listing 4: PXPMarketplace.sol

```
205 function purchaseItem(uint256 itemId) external {
206     uint256 currentPrice = getCurrentPrice(itemId);
207     require(TOKEN_ADDRESS != address(0), "Token invalid");
208     IERC20 token = IERC20(TOKEN_ADDRESS);
209     require(itemId > 0 && itemId <= itemCount, "Not exists");
210     require(
211         token.balanceOf(msg.sender) >= currentPrice,
212         "Insufficient fund"
213     );
214     Item storage item = _items[itemId];
215     require(item.sold == false, "Sold");
216     require(item.cancelled == false, "Cancelled");
217     uint256 fee = currentPrice.mul(_feePercent).div(10000);
218     uint256 total = currentPrice.sub(fee);
219     token.transferFrom(msg.sender, item.seller, total);
220     token.transferFrom(msg.sender, _feeWallet, fee);
```

Risk Level:

Likelihood – 1

Impact - 4

Recommendation:

It's recommended to add the fee percentage as an argument in the purchase function, then use a require statement to make sure the argument value equals to the fee percentage stored in the contract.

Status - Fixed

The PXP team has solved the issue by adding the fee to the arguments and verifying at every purchaseItem call.

A.4 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 5: PXPMarketplace.sol

```
105 function writeItemAddresses(address[] memory _addresses, bool _listable)
106     external
107     onlyRole(ADMIN_ROLE)
108 {
109     for (uint256 a = 0; a < _addresses.length; a++) {
110         _listableAddresses[_addresses[a]] = _listable;
111     }
112 }
```

Listing 6: PXPMarketplace.sol

```
135 function getAllListed() external view returns (Item[] memory items) {
136     uint256 count = 0;
137     for (uint256 i = 1; i <= itemCount; i++) {
138         Item memory item = _items[i];
139         if (item.sold == false && item.cancelled == false) {
140             count++;
```

```
141             }
142         }
143         items = new Item[](count);
144         uint256 index = 0;
145         for (uint256 i = 1; i <= itemCount; i++) {
146             Item memory item = _items[i];
147             if (item.sold == false && item.cancelled == false) {
148                 items[index] = item;
149                 index++;
150             }
151         }
152         return items;
153     }
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Acknowledged

The PXP team has acknowledged the risk.

A.5 Missing Value Verification [LOW]

Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. In the setFeePercent func-

tion, the contract should verify if _percent is less than 100%.

Code:

Listing 7: PXPMarketplace.sol

```
97 function setFeePercent(uint256 _percent) external onlyRole(ADMIN_ROLE) {
98     _feePercent = _percent;
99 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

Status - Fixed

The PXP team has solved the issue by requiring the new fee value to be lower than 10000.

A.6 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 8: PXPMarketplace.sol

```
90  function setFeeWallet(address payable _address)
91      external
92      onlyRole(ADMIN_ROLE)
93  {
94      _feeWallet = _address;
95  }
```

Listing 9: PXPMarketplace.sol

```
101 function setTokenAddress(address _address) external onlyRole(ADMIN_ROLE)
        ↪ {
102     TOKEN_ADDRESS = _address;
103 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The PXP team has solved the issue by requiring the addresses provided in the arguments to be different from the address(0).

A.7 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.4. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Lock-

ing the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 10: PXPMarketplace.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.4;
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The PXP team has solved the issue by locking the pragma version to 0.8.6.

A.8 The Seller Does Not Get The Price Stored In The Offe [INFORMATIONAL]

Description:

In the contract, the user list an item specifying the start price and the last price, if another user purchases this item the seller will not get the price specified in the listing as the fees will be cut from this amount.

Code:

Listing 11: PXPMarketplace (Line 205)

```
function purchaseItem(uint256 itemId) external {
    uint256 currentPrice = getCurrentPrice(itemId);
    require(TOKEN_ADDRESS != address(0), "Token invalid");
    IERC20 token = IERC20(TOKEN_ADDRESS);
    require(itemId > 0 && itemId <= itemCount, "Not exists");
    require(
        token.balanceOf(msg.sender) >= currentPrice,
        "Insufficient fund"
    );
    Item storage item = _items[itemId];
    require(item.sold == false, "Sold");
    require(item.cancelled == false, "Cancelled");
    uint256 fee = currentPrice.mul(_feePercent).div(10000);
    uint256 total = currentPrice.sub(fee);
    token.transferFrom(msg.sender, item.seller, total);
    token.transferFrom(msg.sender, _feeWallet, fee);
```

Risk Level:

Likelihood – 2

Impact - 2

Recommendation:

It is recommended to notify the users with this behavior, so they will not get unexpected amounts after the purchase.

Status - Fixed

4 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a
    ↪ input-controlled function id
        - (success,returndata) = target.delegatecall(data) (node_modules/
            ↪ @openzeppelin/contracts-upgradeable/proxy/ERC1967/
            ↪ ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #controlled-delegatecall

PXPMarketplace.purchaseItem(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #205-234) ignores return value by token.transferFrom(msg.sender,
    ↪ item.seller,total) (pxp_sol/PXPMarketplace.sol#219)
PXPMarketplace.purchaseItem(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #205-234) ignores return value by token.transferFrom(msg.sender,
    ↪ _feeWallet,fee) (pxp_sol/PXPMarketplace.sol#220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer

PXPMarketplace.getAllListed() (pxp_sol/PXPMarketplace.sol#135-153) uses
    ↪ a dangerous strict equality:
```

```
        - item.sold == false && item.cancelled == false (pxp_sol/
            ↪ PXPMarketplace.sol#139)
PXPMarketplace.getAllListed() (pxp_sol/PXPMarketplace.sol#135-153) uses
    ↪ a dangerous strict equality:
        - item_scope_1.sold == false && item_scope_1.cancelled == false (
            ↪ pxp_sol/PXPMarketplace.sol#147)
PXPMarketplace.getCurrentPrice(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #126-133) uses a dangerous strict equality:
        - item.duration == 0 (pxp_sol/PXPMarketplace.sol#128)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dangerous-strict-equalities

Reentrancy in PXPMarketplace.purchaseItem(uint256) (pxp_sol/
    ↪ PXPMarketplace.sol#205-234):
        External calls:
        - token.transferFrom(msg.sender,item.seller,total) (pxp_sol/
            ↪ PXPMarketplace.sol#219)
        - token.transferFrom(msg.sender,_feeWallet,fee) (pxp_sol/
            ↪ PXPMarketplace.sol#220)
        State variables written after the call(s):
        - item.sold = true (pxp_sol/PXPMarketplace.sol#222)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot
    ↪  (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#98) is a local variable never
    ↪ initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by
```

```
    ↪ IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID() (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return

PXPMarketplace.setFeeWallet(address)._address (pxp_sol/PXPMarketplace.
    ↪ sol#90) lacks a zero-check on :
              - _feeWallet = _address (pxp_sol/PXPMarketplace.sol#94)
PXPMarketplace.setTokenAddress(address)._address (pxp_sol/PXPMarketplace
    ↪ .sol#101) lacks a zero-check on :
              - TOKEN_ADDRESS = _address (pxp_sol/PXPMarketplace.sol
                 ↪ #102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,
    ↪ bool).slot (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98)' in
    ↪ ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,
    ↪ bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/
    ↪ ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used
    ↪ before declaration: require(bool,string)(slot ==
    ↪ _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #pre-declaration-usage-of-local-variables

Reentrancy in PXPMarketplace.listItem(address,uint256,uint256,uint256,
    ↪ uint256) (pxp_sol/PXPMarketplace.sol#155-192):
        External calls:
        - nft.transferFrom(msg.sender,address(this),_tokenId) (pxp_sol/
            ↪ PXPMarketplace.sol#167)
```

```
        State variables written after the call(s):
        - _items[itemCount] = Item(itemCount,_itemAddress,_tokenId,
            ↪ _startPrice,_lastPrice,address(msg.sender),block.timestamp
            ↪ ,_duration,false,false) (pxp_sol/PXPMarketplace.sol
            ↪ #170-181)
        - itemCount ++ (pxp_sol/PXPMarketplace.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2

Reentrancy in PXPMarketplace.cancelListItem(uint256) (pxp_sol/
    ↪ PXPMarketplace.sol#194-203):
        External calls:
        - nft.transferFrom(address(this),msg.sender,item.tokenId) (
            ↪ pxp_sol/PXPMarketplace.sol#200)
        Event emitted after the call(s):
        - Cancelled(item.itemId,item.tokenId,msg.sender) (pxp_sol/
            ↪ PXPMarketplace.sol#202)
Reentrancy in PXPMarketplace.listItem(address,uint256,uint256,uint256,
    ↪ uint256) (pxp_sol/PXPMarketplace.sol#155-192):
        External calls:
        - nft.transferFrom(msg.sender,address(this),_tokenId) (pxp_sol/
            ↪ PXPMarketplace.sol#167)
        Event emitted after the call(s):
        - Listed(itemCount,_itemAddress,_tokenId,_startPrice,_lastPrice,
            ↪ _duration,msg.sender) (pxp_sol/PXPMarketplace.sol#183-191)
Reentrancy in PXPMarketplace.purchaseItem(uint256) (pxp_sol/
    ↪ PXPMarketplace.sol#205-234):
        External calls:
        - token.transferFrom(msg.sender,item.seller,total) (pxp_sol/
            ↪ PXPMarketplace.sol#219)
        - token.transferFrom(msg.sender,_feeWallet,fee) (pxp_sol/
            ↪ PXPMarketplace.sol#220)
        - nft.transferFrom(address(this),msg.sender,item.tokenId) (
            ↪ pxp_sol/PXPMarketplace.sol#225)
```

```
        Event emitted after the call(s):
        - Purchased(item.itemId,msg.sender,item.tokenId,item.itemAddress,
            ↪ currentPrice) (pxp_sol/PXPMarketplace.sol#227-233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3

PXPMarketplace.getCurrentPrice(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #126-133) uses timestamp for comparisons
        Dangerous comparisons:
        - item.duration == 0 (pxp_sol/PXPMarketplace.sol#128)
        - item.duration <= pass (pxp_sol/PXPMarketplace.sol#130)
PXPMarketplace.getAllListed() (pxp_sol/PXPMarketplace.sol#135-153) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - item.sold == false && item.cancelled == false (pxp_sol/
            ↪ PXPMarketplace.sol#139)
        - item_scope_1.sold == false && item_scope_1.cancelled == false (
            ↪ pxp_sol/PXPMarketplace.sol#147)
PXPMarketplace.purchaseItem(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #205-234) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(token.balanceOf(msg.sender) >=
            ↪ currentPrice,Insufficient fund) (pxp_sol/PXPMarketplace.
            ↪ sol#210-213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/AddressUpgradeable.sol#186-189)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
```

```
    ↪ sol#52-57) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/StorageSlotUpgradeable.sol#54-56)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
    ↪ sol#62-67) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
    ↪ sol#72-77) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
    ↪ sol#82-87) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/StorageSlotUpgradeable.sol#84-86)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.
            ↪ sol#213-216)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/
    ↪ utils/cryptography/ECDSA.sol#57-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/
            ↪ cryptography/ECDSA.sol#65-69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage

PXPMarketplace.getAllListed() (pxp_sol/PXPMarketplace.sol#135-153)
    ↪ compares to a boolean constant:
        -item.sold == false && item.cancelled == false (pxp_sol/
            ↪ PXPMarketplace.sol#139)
```

```
PXPMarketplace.getAllListed() (pxp_sol/PXPMarketplace.sol#135-153)
    ↪ compares to a boolean constant:
        -item_scope_1.sold == false && item_scope_1.cancelled == false (
            ↪ pxp_sol/PXPMarketplace.sol#147)
PXPMarketplace.purchaseItem(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #205-234) compares to a boolean constant:
        -require(bool,string)(item.sold == false,Sold) (pxp_sol/
            ↪ PXPMarketplace.sol#215)
PXPMarketplace.purchaseItem(uint256) (pxp_sol/PXPMarketplace.sol
    ↪ #205-234) compares to a boolean constant:
        -require(bool,string)(item.cancelled == false,Cancelled) (pxp_sol
            ↪ /PXPMarketplace.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #boolean-equality

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.4']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access
            ↪ /AccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access
            ↪ /IAccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ interfaces/draft-IERC1822Upgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
            ↪ ERC1967/ERC1967UpgradeUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
            ↪ beacon/IBeaconUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
            ↪ utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
            ↪ utils/UUPSUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ security/ReentrancyGuardUpgradeable.sol#4)
```

```
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ StorageSlotUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/interfaces/
          ↪ IERC1271.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
          ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
          ↪ IERC721.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol
          ↪ #4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol
          ↪ #4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography
          ↪ /ECDSA.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography
          ↪ /SignatureChecker.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/
          ↪ introspection/IERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
          ↪ SafeMath.sol#4)
        - ^0.8.4 (pxp_sol/PXPMarketplace.sol#2)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
       ↪ #different-pragma-directives-are-used
```

```
AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/access/
    ↪ AccessControlUpgradeable.sol#54-55) is never used and should be
    ↪ removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/access/
    ↪ AccessControlUpgradeable.sol#220-224) is never used and should be
    ↪  removed
AccessControlUpgradeable._setupRole(bytes32,address) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/access/
    ↪ AccessControlUpgradeable.sol#211-213) is never used and should be
    ↪  removed
Address.functionCall(address,bytes) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#85-87) is never used and should be
    ↪ removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#95-101) is never used and should be
    ↪ removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#114-120) is never used
    ↪ and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (
    ↪ node_modules/@openzeppelin/contracts/utils/Address.sol#128-139)
    ↪ is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#174-176) is never used and should be
    ↪ removed
Address.functionDelegateCall(address,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#184-193) is never used
    ↪ and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#147-149) is never used and should be
    ↪ removed
```

```
Address.functionStaticCall(address,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#157-166) is never used
    ↪ and should be removed
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/
    ↪ Address.sol#36-42) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts
    ↪ /utils/Address.sol#60-65) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/Address.sol#201-221) is never used and should be
    ↪ removed
AddressUpgradeable.functionCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#114-120) is never used and should be
    ↪ removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#128-139) is never used and should be
    ↪ removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#157-166) is never used and should be
    ↪ removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #60-65) is never used and should be removed
```

```
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is
    ↪ never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
    ↪ #21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and
    ↪  should be removed
ECDSA._throwError(ECDSA.RecoverError) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#23-35) is never used and
    ↪ should be removed
ECDSA.recover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils
    ↪ /cryptography/ECDSA.sol#90-94) is never used and should be
    ↪ removed
ECDSA.recover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#118-126) is never used and
    ↪  should be removed
ECDSA.recover(bytes32,uint8,bytes32,bytes32) (node_modules/@openzeppelin
    ↪ /contracts/utils/cryptography/ECDSA.sol#169-178) is never used
    ↪ and should be removed
ECDSA.toEthSignedMessageHash(bytes) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#202-204) is never used and
    ↪  should be removed
ECDSA.toEthSignedMessageHash(bytes32) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#188-192) is never used and
    ↪  should be removed
ECDSA.toTypedDataHash(bytes32,bytes32) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#215-217) is never used and
    ↪  should be removed
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/
    ↪ utils/cryptography/ECDSA.sol#57-74) is never used and should be
    ↪ removed
```

```
ECDSA.tryRecover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/
    ↪ contracts/utils/cryptography/ECDSA.sol#103-111) is never used and
    ↪  should be removed
ECDSA.tryRecover(bytes32,uint8,bytes32,bytes32) (node_modules/
    ↪ @openzeppelin/contracts/utils/cryptography/ECDSA.sol#134-163) is
    ↪ never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is
    ↪ never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol
    ↪ #27-28) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#21-22) is never used and should be
    ↪ removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#24-25) is never used and should be
    ↪ removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#139-142) is never used and should
    ↪ be removed
ERC1967UpgradeUpgradeable._getAdmin() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol
    ↪ #122-124) is never used and should be removed
ERC1967UpgradeUpgradeable._getBeacon() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol
    ↪ #158-160) is never used and should be removed
ERC1967UpgradeUpgradeable._setAdmin(address) (node_modules/@openzeppelin
    ↪ /contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.
    ↪ sol#129-132) is never used and should be removed
```

```
ERC1967UpgradeUpgradeable._setBeacon(address) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#165-172) is never used and should
    ↪ be removed
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
    ↪ ERC1967UpgradeUpgradeable.sol#180-190) is never used and should
    ↪ be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/utils/Initializable.sol#131-137) is
    ↪ never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/security/
    ↪ ReentrancyGuardUpgradeable.sol#40-42) is never used and should be
    ↪  removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/security/
    ↪ ReentrancyGuardUpgradeable.sol#44-46) is never used and should be
    ↪  removed
SafeMath.add(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#93-95) is never used and should be
    ↪ removed
SafeMath.div(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#191-200) is never used and
    ↪ should be removed
SafeMath.mod(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#151-153) is never used and should be
    ↪ removed
SafeMath.mod(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#217-226) is never used and
    ↪ should be removed
SafeMath.sub(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#168-177) is never used and
    ↪ should be removed
```

```
SafeMath.tryAdd(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#22-28) is never used and should be
    ↪ removed
SafeMath.tryDiv(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#64-69) is never used and should be
    ↪ removed
SafeMath.tryMod(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#76-81) is never used and should be
    ↪ removed
SafeMath.tryMul(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#47-57) is never used and should be
    ↪ removed
SafeMath.trySub(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#35-40) is never used and should be
    ↪ removed
SignatureChecker.isValidSignatureNow(address,bytes32,bytes) (
    ↪ node_modules/@openzeppelin/contracts/utils/cryptography/
    ↪ SignatureChecker.sol#25-41) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
    ↪ sol#72-77) is never used and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.
    ↪ sol#82-87) is never used and should be removed
Strings.toHexString(address) (node_modules/@openzeppelin/contracts/utils
    ↪ /Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils
    ↪ /Strings.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts/utils/Strings.sol#57-67) is never used and should be
    ↪ removed
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/
    ↪ Strings.sol#16-36) is never used and should be removed
```

```
StringsUpgradeable.toHexString(address) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/StringsUpgradeable.sol#72-74) is
    ↪ never used and should be removed
StringsUpgradeable.toHexString(uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/StringsUpgradeable.sol#41-52) is
    ↪ never used and should be removed
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/StringsUpgradeable.sol#16-36) is
    ↪ never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.
    ↪ sol#26-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old
    ↪ versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ security/ReentrancyGuardUpgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/AddressUpgradeable.sol#4) allows old versions
```

```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ utils/StorageSlotUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/interfaces/
   ↪ IERC1271.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/
   ↪ IERC721.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address
   ↪ .sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings
   ↪ .sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
   ↪ cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
   ↪ cryptography/SignatureChecker.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/
   ↪ introspection/IERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
   ↪ SafeMath.sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #incorrect-versions-of-solidity

UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/
   ↪ proxy/utils/UUPSUpgradeable.sol#107) is never used in
```

```
    ↪ PXPMarketplace (pxp_sol/PXPMarketplace.sol#17-236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
pxp_sol/PXPMarketplace.sol analyzed (24 contracts with 75 detectors),
    ↪ 124 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of PXP MarketPlace contract and discovered several issues of varying severity. Pirate X Pirate team addressed 7 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Pirate X Pirate Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

For a Contract Audit, contact us at contact@shellboxes.com