QuillAudits

Audit Report
July, 2022

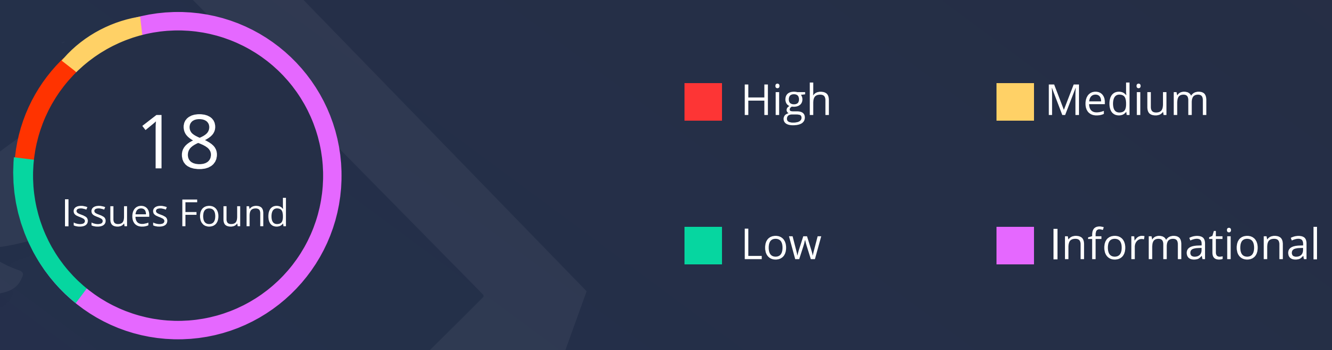
For

Pixel War

Table of Content

Executive Summary

Project Name	Pixel War DAO
Overview	The project contains an ERC20 Token called PXP with 1 billion supply which will be used for Vesting, Staking and exchanging 1155 tokens with the OrderBook. Staking allows users to deposit PXP in exchange for a governance token. Users can stake PXP and earn additional PXP as rewards which is in proportion to their share in the pool. 1155OrderBook lists whitelisted ERC1155 tokens escrowed in the contract, for sale in exchange for PXP.
Timeline	26th May, 2022 to 9th June, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse PixelWar DAO codebase for quality, security, and correctness. *https://github.com/PixelWarOrg/dao-contracts*
Commit ID	0455692954d8bc3b1b0b03eef4f3226a356e5a31
Fixed In	4aa5804e52940f3d4b15530f71321d40c54a3e26



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**2**	**3**	**2**
Partially Resolved Issues	0	0	0	0
Resolved Issues	**3**	0	**1**	**7**

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - PXP

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A.1 Floating Pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to use 0.8.12 version.

Auditor’s Comment

As per discussion with Pixelwar team and as Hardhat does not supporting fixed pragma 0.8.12 so ,we recommend, 0.8.4 - 0.8.7 Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Status

Acknowledged

Informational Issues

A.2 State variables that could be declared immutable

Description

The value 1000000000 * 1e18 should be declared as a constant in the contract instead of hardcoding.

Recommendation

Add the immutable attributes to state variables that never change after contract creation.

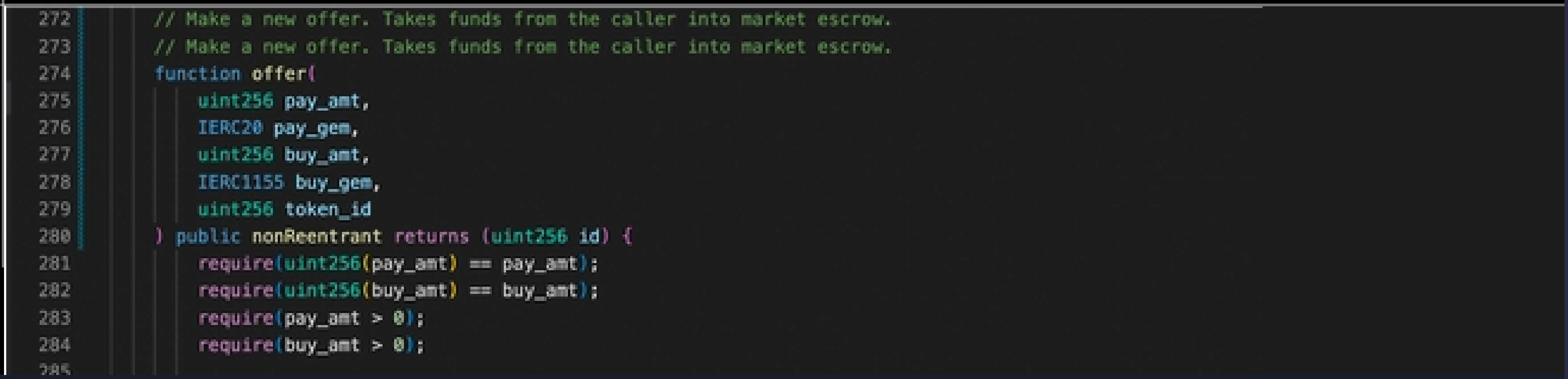
Status

Resolved

B. Contract - OrderBook

High Severity Issues

B.1 Non-whitelisted NFTs and Pay Gem tokens can be used for creating orders

Line	Function - offer()
274	

```
272        // Make a new offer. Takes funds from the caller into market escrow.
273        // Make a new offer. Takes funds from the caller into market escrow.
274        function offer(
275            uint256 pay_amt,
276            IERC20 pay_gem,
277            uint256 buy_amt,
278            IERC1155 buy_gem,
279            uint256 token_id
280        ) public nonReentrant returns (uint256 id) {
281            require(uint256(pay_amt) == pay_amt);
282            require(uint256(buy_amt) == buy_amt);
283            require(pay_amt > 0);
284            require(buy_amt > 0);
285
```

Description

The function offer() is a public function which can be called with any ERC20 and ERC1155 address for pay_gem and buy_gem addresses respectively.

Remediation

It is recommended to restrict the visibility to internal instead of public so that an offer can be created only after verification from make() function

Status

Resolved

Medium Severity Issues

No issues found

Low Severity Issues

B.2 Renounce ownership

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status

Acknowledged

B.3 Add external modifier instead of public

Description

It is recommended to use external access modifier instead of public for the following functions which are not called from the contract:

- bump()
- getOffer()
- make()
- kill()
- take()

Remediation

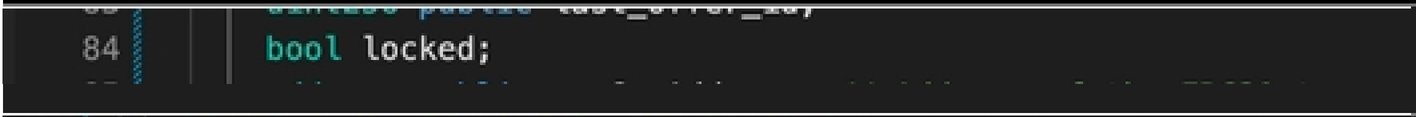
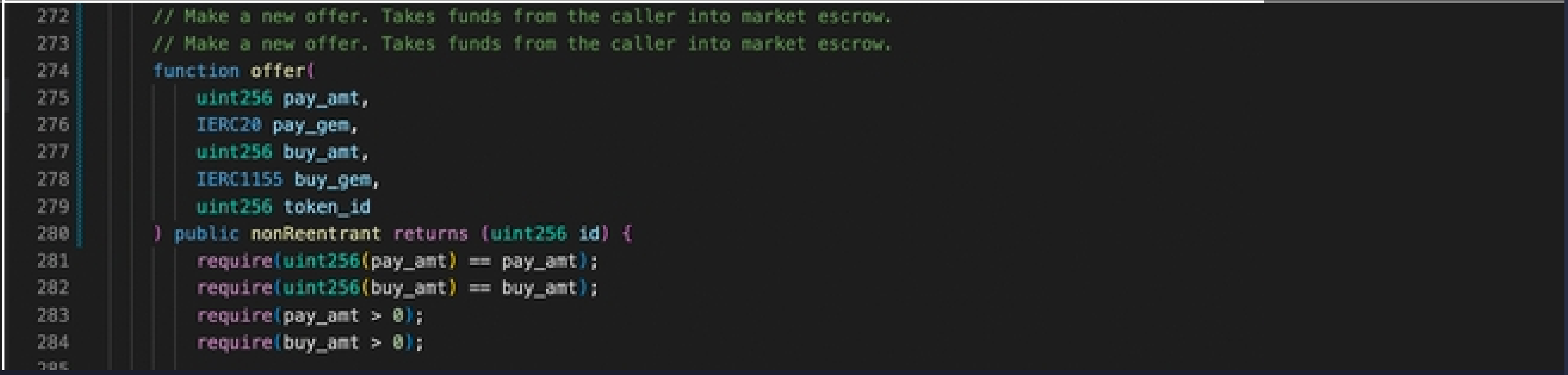
As per the solidity security recommendation, the functions should first update the contract states and then interact with external contracts. Please refer solidity documentation *here*.

Status

Acknowledged

Informational Issues

B.4 Presence of unused code

Line	Function - offer() and make(), variables
84	
274	

```
84      bool locked;
```

```
272     // Make a new offer. Takes funds from the caller into market escrow.
273     // Make a new offer. Takes funds from the caller into market escrow.
274     function offer(
275         uint256 pay_amt,
276         IERC20 pay_gem,
277         uint256 buy_amt,
278         IERC1155 buy_gem,
279         uint256 token_id
280     ) public nonReentrant returns (uint256 id) {
281         require(uint256(pay_amt) == pay_amt);
282         require(uint256(buy_amt) == buy_amt);
283         require(pay_amt > 0);
284         require(buy_amt > 0);
```

Description

The program contains code that is not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact on functionality or correctness.

Remediation

We recommend removing the unused code.

- The variable locked is not used anywhere in the contract.
- The variable pay_gem can be removed from params as its value is already present in storage.

Auditor’s Comment for Variable pay_gem

As you have removed it from make() and used storage variable instead for calling offer().
But we can recommend you to remove it from offer() as well along with the check:
require(address(pay_gem) == payGemAddress, "Wrong pay_gem specified");
This will save some gas here.

Status

Resolved

B.5 Missing Error messages

Description

The require statements miss error messages which are used to describe the reason for revert.

Recommendation

It is recommended to add messages in require statement to make the debugging process easier.

Status

Resolved

B.6 General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Auditor’s Comment

There are still some indentation issues (2 spaces instead of 4 spaces)

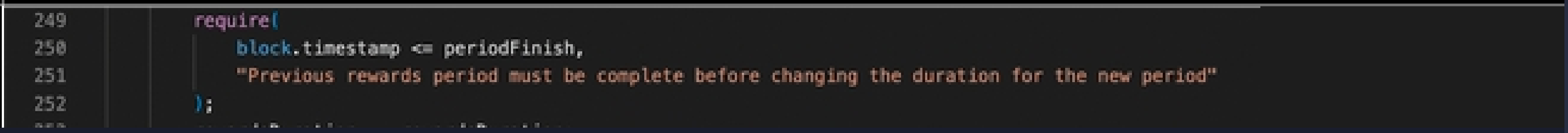
Status

Resolved

C. Contract - Staking

High Severity Issues

C.1 Wrong check for periodFinish

Line	Function - setRewardsDuration()
250	

```
249         require(
250             block.timestamp <= periodFinish,
251             "Previous rewards period must be complete before changing the duration for the new period"
252         );
```

Description

The function setRewardsDuration() checks if the periodFinish is greater than the current time but message says the previous reward period must finish before new period.

Remediation

It is recommended to update the check (block.timestamp <= periodFinish) to: block.timestamp > periodFinish

Status

Resolved

Medium Severity Issues

C.2 Centralization Risk

Description

The function revoke() allows the contract owner to remove all the funds collected in staking contract. This poses a risk for the token holders where their funds can be moved by the contract owner at any time.

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions: with reasonable latency for community awareness on privileged operations; Multisig with community-voted 3rd-party independent co-signers; DAO or Governance module increasing transparency and community involvement;

Status

Acknowledged

Low Severity Issues

C.3 Add external modifier instead of public

Description

It is recommended to use external access modifier instead of public for the following functions which are not called from the contract:

- getRewardForDuration()
- revoke()

Remediation

As per the solidity security recommendation, the functions should first update the contract states and then interact with external contracts.
Please refer solidity documentation *here*.

Status

Resolved

Informational Issues

C.4 General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

Auditor’s Comment: There are still some indentation issues (2 spaces instead of 4 spaces

D. Contract - Mission

High Severity Issues

No issues found

Medium Severity Issues

D.1 Centralization Risk

Description

The function adminEmergencyWithdraw() allows the contract owner to remove all the funds collected in staking contract. This poses a risk for the token holders where their funds can be moved by the contract owner at any time.

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions: with reasonable latency for community awareness on privileged operations; Multisig with community-voted 3rd-party independent co-signers; DAO or Governance module increasing transparency and community involvement;

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

D.2 Presence of unused code

Description

The program contains code that is not essential for execution, i.e. makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact on functionality or correctness.

Remediation

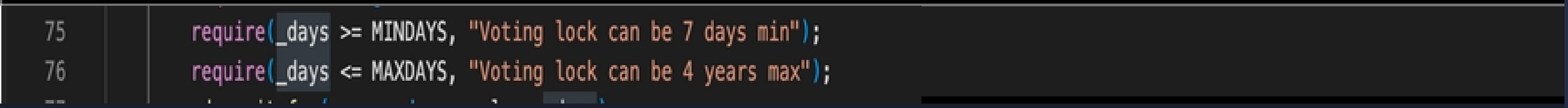
We recommend removing the unused variable burn

Status

Fixed

D.3 Misleading Error messages

Line	Function - create_lock() & increase_unlock_time()

75	
	`75 require(_days >= MINDAYS, "Voting lock can be 7 days min");`
	`76 require(_days <= MAXDAYS, "Voting lock can be 4 years max");`

Description

The following require statements have misleading error messages. The min time and max time is declared different in the contract.

Remediation

It is recommended to update the messages in require statements.

Status

Resolved

D.4 State Variable Default Visibility

Description

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

- IERC721 heroToken
- IERC20 PXP

Remediation

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Ref: *https://swcregistry.io/docs/SWC-108*

Status

Resolved

D.5 General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

Auditor’s Comment: There are still some indentation issues (2 spaces instead of 4 spaces)

E. Contract - Vesting

High Severity Issues

E.1 Contain fallback function

Description

The contract contains receive() and fallback() to receive ethers but the purpose can be resolved with receive() function only. Since the fallback functions is not only called for plain ether transfers (without data) but also when no other function matches. If the contract is used incorrectly, functions that do not exist are called.

Remediation

It is recommended to remove the fallback() function and add proper checks in the receive() function to protect the contract and allow only known sources to send the ETH to the contract.

Status

Fixed

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

E.2 Missing Error messages

Description

The contract does not contain error messages for a lot of require statements.

Remediation

It is recommended to add the messages in require statements to improve readability and make it user-friendly.

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- ✓ Should be able to deploy and mint the initial token supply.
- ✓ Should be able to create make and take orders for gems.
- ✓ Should be able to transfer ERC-20 and ERC-1155 tokens to and from orderbook.
- ✓ Should be able to buy the tokens from orderbook.
- ✓ Should revert if order is cancelled
- ✓ Should be able stake and unstake the tokens.
- ✓ Should allow users to receive rewards at the end of stake
- ✓ Should be able to withdraw & emergencyWithdraw tokens with penalty
- ✓ Should allow owner to vest
- ✓ Should allow the owner to set the needed details
- ✓ Should allow the beneficiary to collect the vested funds according to the time

Automated Tests

No major issues were found after using Slither and Mythril. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the PixelWar. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the PixelWar Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the PixelWarTeam put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

Pixel War

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com