QuillAudits

AUDIT REPORT

May , 2025

For

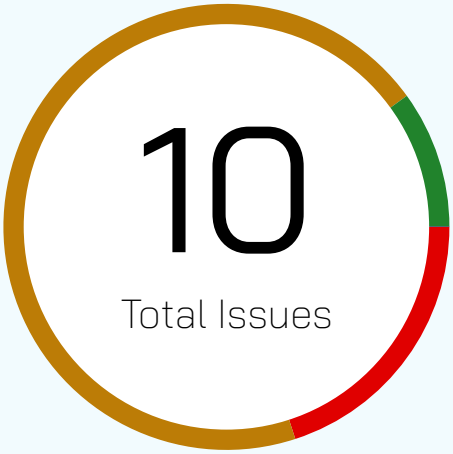
Table of Content

Executive Summary

Project name	PiXL
Project URL	https://pixl.property/
Overview	The platform allows for fractional ownership of properties through ERC1155 tokens with two main property types: BridgingFacility and OnPlan. PropertyLaunchpad manages property creation, share distribution, and dividends. Property.sol handles the ERC1155 tokens with transfer restrictions. PropertyMarketplace facilitates secondary market trading of property shares. The system incorporates staking mechanisms for rewards, a bridge for cross-chain operations, and meta-transaction support for gasless transactions. Security features include pausability, reentrancy protection, and multi-signature requirements for critical operations.
Audit Scope	The scope of this Audit was to analyze the PIXL Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/KameKazeee/DREAMS-Vishal/tree/main/Contracts
Contracts in Scope	Contracts/contracts/PropertyMarketplace.sol Contracts/contracts/EIP712Base.sol Contracts/contracts/USDT.sol Contracts/contracts/bridge.sol Contracts/contracts/NativeMetaTransaction.sol Contracts/contracts/Staking.sol Contracts/contracts/ContextMixin.sol Contracts/contracts/property.sol Contracts/contracts/PropertyLaunchpad.sol Token.sol Vesting.sol

Branch	main
Commit Hash	3d9ac4f3aa18811f32d67f2a5413cab7db818027
Language	Solidity
Blockchain	Plume,EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	23th April 2025 - 6th May 2025
Updated Code Received	7th May 2025
Review 2	7th May 2025
Fixed In	90a1138778b8235a5a8e03df55d33c763a9814e6

Number of Issues per Severity



High	2 (20.00%)
Medium	7 (70.00%)
Low	1 (10.00%)
Informational	0 (0.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	**6**	**1**	0
Acknowledged	0	**1**	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC 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 analyzed 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

A 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 analyzed, 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 behavior 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, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

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 Severity Issues

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 Severity Issues

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

Informational Issues

These are four 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

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

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.

High Severity Issues

Unauthorized Transfer of Seller's Shares via Launchpad Contract

Resolved

Path

src/property*

Function

PropertyLaunchpad.safeTransferFromNFT()
Property.sol::safeTransferFrom()
PropertyMarketplace.buyProperty()

Description

The PropertyLaunchpad.safeTransferFromNFT() function acts as a proxy to call Property.safeTransferFrom() for NFT share transfers. However, it lacks any access control or validation logic.
This is problematic because sellers are required to pre-approve the PropertyLaunchpad contract as a trusted operator to list their shares for sale. This makes the PropertyLaunchpad an authorized actor in Property.sol's onlyAuth modifier, thus allowing anyone to use safeTransferFromNFT() to transfer any seller's shares without going through the PropertyMarketplace and without paying.
In other words, any attacker can bypass the legitimate buyProperty() process and steal shares directly from sellers who've granted allowance to the launchpad.

Misalignment Between Declared APY and Actual Interest Accrual Results in Overpayment

Resolved

Path

src/Staking.sol

Function

unStake

Description

The team intends to offer an 8% APY with a planned lock period of 60 days. However, the current implementation applies the full 8% APY as if it were a 60-day yield instead of calculating the proportional APR for that period. This means users are effectively receiving 8% returns every 60 days, which leads to a much higher annual yield and results in the protocol overpaying users significantly.

Recommendation

To align with the intended 8% APY, the reward calculation should be adjusted to only pay the proportional interest for 60 days, not the full yearly rate. This can be done by calculating the actual APR for the 60-day lock period based on the 8% annual rate.

Medium Severity Issues

Lack of Listing Limits Allows Spam and Duplicate Property Sales

Resolved

Path

src/PropertyMarketplace.sol

Function

sellProperty

Description

The sellProperty function allows users to list their property for sale multiple times without any restriction. Since there's no check to prevent duplicate or excessive listings for the same property NFT by the same user, a malicious actor can repeatedly call this function to create numerous listings. This could spam the system, clutter the marketplace with redundant or fake sales, and potentially degrade the user experience or lead to confusion in property availability.

Recommendation

Introduce restrictions to limit the number of active listings per user per property ID

Potential for Signature Malleability

Resolved

Path

src/*

Function

Verify*()

Description

The contract utilizes Ethereum's ecrecover function to verify that a given message was signed by the holder of the private keys of a specified address. This approach, while standard, is susceptible to ECDSA signature malleability issues. This is because Ethereum signatures (comprised of r, s, v components) are not strictly unique, and it is theoretically possible to manipulate a signature to produce a valid variant without access to the corresponding private key. This could potentially be exploited under certain conditions to bypass signature verifications if additional safeguards are not in place.

Recommendation

It is recommended to use a more comprehensive signature verification library, such as OpenZeppelin's ECDSA library.

Compatibility Issue with Fee On Transfer Tokens in Bridge Contract

Resolved

Path

src/Bridge.sol

Function

*

Description

The Bridge contract is incompatible with fee-on-transfer (FOT) tokens because it assumes that the amount specified for transfer is the same as the amount received. FOT tokens deduct a fee during transfer, meaning the contract may receive less than the specified amount. This breaks internal accounting and can lead to incorrect balances, failed assumptions, or loss of funds across deposits and withdrawals.

Recommendation

To ensure correctness, the contract should not rely on the specified amount but instead calculate and account for the actual amount received.

Pausing Mechanism Implemented but Ineffectively Utilized

Resolved

Path

src/utilityToken/PIXL.sol

Function

*

Description

The contract includes a pausing mechanism using OpenZeppelin's Pausable module, indicating the team's intent to control or halt token activity during emergencies. However, the current implementation fails to utilize it properly. No part of the contract checks the paused state, meaning pausing the contract has no effect on token transfers or minting. As a result, the mechanism is effectively non-functional despite being present.

Recommendation

To properly enforce the pause functionality, the contract should integrate whenNotPaused

Missing gap to Avoid Storage Collisions

Resolved

Path

src/property*

Function

*

Description

A few contracts are intended to be an upgradeable smart contract, but do not have a __gap variable. In upgradeable contracts, it's crucial to include a _gap to ensure that any additional storage variables added in future contract upgrades do not collide with existing storage variables. This is especially important when inheriting from multiple upgradeable contracts.

Recommendation

Include a _gap as the last storage variable to upgradable contracts to reserve space for future storage variables and prevent storage collisions.

Centralization Risk Due to Owner-Only Withdrawals

Acknowledged

Path

src/*

Function

*

Description

A few contracts include withdrawNative and withdrawTokens functions that allow the contract owner to withdraw all native and ERC20 token funds, respectively. While this is commonly implemented for administrative or emergency access, it introduces a significant centralization risk. If the owner's private key is compromised or if the owner acts maliciously, they can unilaterally drain all assets from the contract, potentially resulting in a total loss of user funds.

Recommendation

To mitigate this risk, it is recommended to replace the single-owner control with a multisignature (multisig) wallet that requires multiple trusted parties to authorize withdrawals. Additionally, implementing a timelock mechanism on such sensitive operations can provide a buffer period for stakeholders to respond in case of unexpected or malicious actions.

Unchecked transfers return value.

Resolved

Description

ERC20 token transfers are performed using the transfer and transferFrom methods. These methods return a boolean value indicating the success of the operation, as per the ERC20 standard. However, the contract does not check these return values, which can lead to scenarios where token transfers fail.

Recommendation

Use SafeERC20 library from OpenZeppelin, which handles these inconsistencies and ensures compatibility.

Low Severity Issues

Missing Events on Important State Changes

Resolved

Path

*

Function

*

Description

Without events, users and blockchain-monitoring systems will not be able to easily detect suspicious behavior.
Several critical functions are not triggering events, which will make it difficult to check the behavior of the contracts once they have been deployed.

Recommendation

Ideally, the critical operations should trigger events.

Automated Tests

No major issues were found. 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 PiXL . We performed our audit according to the procedure described above.

Issues of High, Medium and low severity were found. PiXL team resolved almost all and acknowledged one

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May , 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com