QuillAudits

Audit Report
March, 2022

For

PAWNSPACE

Contents

Scope of the Audit

The scope of this audit was to analyze and document the PawnSpace Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

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, Theo.

Issue Categories

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.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	0
Closed	1	1	1	3

Introduction

From **March 03, 2022 to March 25, 2022** - QuillAudits Team performed a security audit for PawnSpace smart contracts.
The code for the audit was taken from following the official link:

https://github.com/Project-Cre8/PawnSpace-SmartContract/commit/1ec29b4ca8d84f46589597d37e9ba77ea00a7a9a

V	Date	Commit ID	Date
1	3rd March	65c697fdaadfa6c3cc0031768431a14a4eebe43f	contracts/*
2	25th March	9d135ec8872305659ce813a7f1f5a419a2be4d45	contracts/*

Issues Found – Code Review / Manual Testing

A. PawnSpace contracts

High severity issues

1. autoAccept function transfers requestToken to wrong destination

Line	Function - constructor()

192-213	function autoAccept(uint256 orderId, uint256 newOfferId, address lender) external override {
	require(msg.sender == offerSpace, "PawnSpace: AutoAccept can only be called from OfferSpace");
	orders[orderId].acceptedBlockTimestamp = block.timestamp;
	orders[orderId].acceptedOfferId = newOfferId;
	orders[orderId].borrower = ownerOf(orderId);
	
	address feeTo = IPawnFactory(factory).feeTo();
	uint256 fee = feeTo != address(0) ? orders[orderId].autoAcceptAmount.mul(IPawnFactory(factory).fee()).div(1000) : 0;
	
	// transfer
	IERC20(orders[orderId].requestToken).transferFrom(
	lender,
	msg.sender,
	orders[orderId].autoAcceptAmount.sub(fee)
	);
	if (fee > 0) {
	IERC20(orders[orderId].requestToken).transferFrom(lender, feeTo, fee);
	}
	
	emit AutoAccept(lender, orderId, newOfferId);
	}

Description
The autoAccept function transfers requestToken to the offerSpace (msg.sender) contract instead to the borrower.

Remediation
Transfer the requestToken to the borrower

Status: Fixed

Medium severity issues

2. Missing zero address validation

Line	Function - constructor()

89 - 92	function setFeeTo(address _feeTo) external override {
	require(msg.sender == feeToSetter, 'PawnFactory: FORBIDDEN');
	feeTo = _feeTo;
	}
	And also other instances.

Description

Functions set important address state variables and don't check for zero address check, this can lead to loss of funds at worst and incorrect deployment parameters at best.

Remediation

Declare listed variables as immutable.

Status: Fixed

Low severity issues

3. Insufficient events

Line	Code/Function

89 - 111	function setFeeTo(address _feeTo) external override {
	require(msg.sender == feeToSetter, 'PawnFactory: FORBIDDEN');
	feeTo = _feeTo;
	}
	
	function setFeeToSetter(address _feeToSetter) external override {
	require(msg.sender == feeToSetter, 'PawnFactory: FORBIDDEN');
	feeToSetter = _feeToSetter;
	}
	
	function setFee(uint256 _fee) public {
	require(msg.sender == feeToSetter, 'PawnFactory: FORBIDDEN');
	require(_fee <= 1000, 'PawnFactory: Cannot be greater than 1000');
	fee = _fee;
	}

Line	Code/Function

194 - 201	function setLoanToActive(uint256 offerId, address loanOwner) external override {
	require(initialized, "SYSTEM NOT INITIALIZED");
	require(msg.sender == pawnSpace, "OfferSpace: FORBIDDEN");
	offers[offerId].loanStartTimestamp = block.timestamp;
	offers[offerId].expiredTimestamp = block.timestamp.add(offers[offerId].period);
	offers[offerId].active = true;
	_mint(loanOwner, offerId);
	}

283 - 288	function modifyAutoAccept(uint256 orderId, uint256 amount, uint256 interest) external override {
	require(ownerOf(orderId) == msg.sender, "PawnSpace: only owner can modify");
	require(orders[orderId].acceptedBlockTimestamp == 0, "PawnSpace: cannot edit active order");
	orders[orderId].autoAcceptAmount = amount;
	orders[orderId].autoAcceptInterest = interest;
	}

Description
startDate can be set to any arbitrary value such as 0 or any date of the past.

Remediation
It is advised to add sufficient require checks for the same.

Status: Fixed

Informational Issues

4. State variables that can be immutable

Line	Code/Function
17	address public override USDCAddr;

Description
Variables can be marked as immutable to save gas.

Remediation
Declare listed variables as immutable.

Status: Fixed

5. Pawnspace.sol, Order function:

Description
nftToken, which is an external contract, raises a security threat ,since we don't know the token standard.

Status: Fixed

6. Redundant check of amount value in the offer function

Line	Code/Function

98 - 139	function offer(
	uint256 orderId,
	uint256 amount,
	uint256 interest,
	uint256 period
	) external override returns (uint256 offerId, address _offeror) {
	require(initialized, "SYSTEM NOT INITIALIZED");
	require(IPawnSpace(pawnSpace).exists(orderId), 'OfferSpace: nonexistent order');
	require(IERC721(pawnSpace).ownerOf(orderId) != msg.sender, 'OfferSpace: orderor cannot offer');
	
	require(IPawnSpace(pawnSpace).getAcceptedBlockTimestamp(orderId) == 0, 'OfferSpace: cannot offer to accepted order');
	require(amount > 1000, "OfferSpace: Amount too small");
	
	// we can allow any amount, below or over autoAcceptAmount. This is because of interest.
	
	require(IERC20(IPawnSpace(pawnSpace).getRequestToken(orderId)).balanceOf(msg.sender) >= amount, 'PawnSpace: not enough balance');
	require(
	
	IERC20(IPawnSpace(pawnSpace).getRequestToken(orderId)).allowance(msg.sender, pawnSpace) >= amount,
	'OfferSpace: no allowance to pawnspace'
	);
	require(amount > 1000, "OfferSpace: invalid amount");
	
	[...]

Description

amount value is being checked twice in the offer function which can be optimised.

Remediation

Declare listed variables as immutable.

Status: Fixed

Automated Tests

Slither

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity

Closing Summary

Numerous issues were discovered during the initial audit. In the End all the Issues have been Fixed by the Auditee.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Pawnspace 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 Pawnspace Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

PAWNSPACE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com