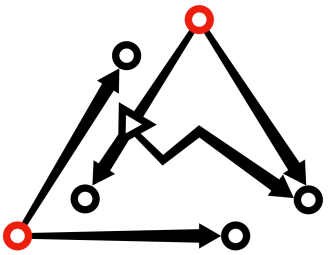

Least Authority

PRIVACY MATTERS

Pawn Smart Contracts

Security Audit Report

Pawn Finance

Updated Final Audit Report: 27 August 2021

Table of Contents

Overview

Background

Pawn Finance has requested that Least Authority perform a security audit of their Pawn Smart Contracts, a peer-to-peer lending and borrowing protocol on the Ethereum blockchain.

Project Dates

- **June 16 - June 29**: Code review *(Completed)*
- **July 2**: Delivery of Initial Audit Report *(Completed)*
- **July 21 - 22:** Verification *(Completed)*
- **July 23:** Delivery of Final Audit Report *(Completed)*
- **August 27:** Delivery of Updated Final Audit Report to include Issue C *(Completed)*

Review Team

- Gabrielle Hibbert, Security Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer
- May-Lee Sia, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Pawn Smart Contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Pawn Smart Contracts: https://github.com/Non-fungible-Technologies/pawn-contracts

Specifically, we examined the Git revisions for our initial review:

`270108f9c921168574a8f8a0134b405aefb228a8`

For the verification, we examined the Git revision:

`4561311ad7bdf988cde7b8d5fc542792a82372b1`

For the verification of Issue C, we examined the Git revision:

`4b00f6a2fb718ca883411f6db6f8c26a9b1c59cf`

For the review, this repository was cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/pawnfi-contracts/tree/in-scope

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- README.md: https://github.com/Non-fungible-Technologies/pawn-contracts/blob/main/README.md
- STATES.md: https://github.com/Non-fungible-Technologies/pawn-contracts/blob/main/doc/STATES.md
- PAWNFI-P2PLoanArchitecture-V1.pdf shared with Least Authority via Telegram on 18 June 2021
- 21-06-25 - Pawn.fi Terms & Conditions - DRAFT.pdf shared with Least Authority via Telegram on 25 June 2021

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Denial of Service (DoS) and other security exploits that would impact the smart contracts intended use or disrupt execution;
- Vulnerabilities in the smart contracts code;
- Protection against malicious attacks and other ways to exploit smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Pawn Finance is a protocol that enables peer-to-peer lending and borrowing on the Ethereum blockchain. Users of the system can act as borrowers or lenders by which borrowers are able to collateralize ERC-20, ERC-721, or ERC-1155, and draw an ERC-20 loan. Users acting as lenders can provide liquidity to the protocol and earn interest proportional to the term of the loan. The protocol collects fees on loan origination. In the case of a borrower defaulting on the terms of a loan, the lender may claim the borrower's collateral.

The critical functions of the protocol, such as loan origination, transfer of funds to borrowers and lenders, and transfer of claimed collateral is performed in `LoanCore.sol`. Users interact with this module through role based access controllers, which limit the use of functions to specific, permissioned roles.

System Design

Our team performed a broad and comprehensive review of the system design and implementation. We investigated potential security vulnerabilities and found that security has been strongly considered in the system design. OpenZeppelin security features are implemented correctly, including correct domain separation in ERC-712 signatures, as well as correct implementations of counters and modern safe math. SafeERC20 is used to remove return value bug concerns. In addition, functions in the codebase are implemented with correct access modifiers for permissioned roles.

In examining `ERC721Permit.sol`, we found that mitigations have been implemented for potential vulnerabilities introduced by EIP-712, including zombie fund attacks. We also found that the Pawn smart contracts use access controllers to interact with `LoanCore.sol`, enabling atomic token transfers and mitigating against the possibility of front-running issues.

We explored the possibility of potential edge-case re-entrance attacks and found that, if a corrupt ERC-20 token is used as principal, the `transfer` method could contain a `call back` to `LoanCore.sol`. This `call back` will create many loans under the same principal, since the balance of the principal is not updated in `LoanCore.sol` until after that call, creating a possibility for a re-entrance attack. Although we did not identify any scenario by which this issue would result in a feasible attack, we recommend an alternative method to record the balance of principal held in `LoanCore.sol` that mitigates the risk of edge-case vulnerabilities (Issue A).

Centralized Administrator Account

Although we did not identify issues that are considered to be security critical, we found that there are centralized features that users should be made aware of. For example, `LoanCore.sol` performs the key functions of the protocol and these functions are restricted by roles with specific permissions. Furthermore, `DEFAULT_ADMIN_ROLE` sets protocol fees and also has the ability to pause the core functionality of the protocol. We found no documentation on the rationale and use cases for these roles and suggest that users be made aware of these features in order to better protect their assets (Suggestion 1).

Off-Chain Communication

In initializing a new loan, the Pawn Finance protocol requires that a borrower or lender propose loan terms to which the counter party agrees. However, reaching agreement on the loan terms requires the borrower and lender to communicate off-chain in order to continue the origination process.

While investigating this off-chain interaction, our team identified a potential opportunity for a replay attack. Given that the loan terms agreement is created off-chain, there is a possibility that the party broadcasting the signed loan terms uses a previous loan terms agreement, one that the counterparty did not intend to be used. We suggest that this off-chain interaction be clarified in the documentation so that a more comprehensive security evaluation of this interaction can be performed. Furthermore, we recommend that users be explicitly informed of the possibility of reusing loan terms agreements (Issue B).

We commend the Pawn Finance team for prioritizing security considerations in their system design and implementation. We encourage any known security flaws to be resolved or mitigated using accepted security methods. Identifying security vulnerabilities and addressing them as early as possible in the design, development, and deployment process facilitates efficiency, reduces costs, and builds user trust, thus enabling broader adoption.

Code Quality

The Pawn Finance smart contract code is well written and organized, and adheres to widely accepted standards and security best practices. Additionally, the implementation demonstrates that the Pawn Finance team has minimized computational complexity and gas costs, which benefits user security and reduces transaction costs. We suggest removing the redundant `require` statement and exploring opportunities to further reduce complexity to the extent possible in order to minimize the attack surface (Suggestion 2).

Tests

Our team found a commendable level of test coverage for success and failure cases, which helps to identify potential edge cases, and helps protect against errors and bugs, which may lead to vulnerabilities or exploits. The Pawn Finance smart contracts test suite was a valuable aid in investigating and determining if the implementation behaves as intended.

Documentation

Our team found the supporting documentation provided by the Pawn Finance team to be sufficient and accurate in describing each of the components of the system and the interactions between those components. In addition, we noted adherence to NatSpec guidelines for best practices on Solidity code comments, further facilitating understanding the intended behavior of each of the components. However, we recommend that the documentation provided by the Pawn Finance team be incorporated into the in-scope repository, in order to provide easier access and better guidance on the intended behavior of the system (Suggestion 3).

In addition, while most of the smart contracts have sufficient documentation for every function, some are more comprehensively documented than others (e.g. `docs/interfaces/IPromissoryNote.md` provides less comprehensive documentation than some of the other smart contracts). We recommend documenting all smart contracts consistently, including functions whose implementations appear obvious, in order to reduce the potential for errors or confusion about intended functionality (Suggestion 3).

Scope and Dependencies

The scope for the security audit was sufficient and included all security critical components. In addition, we found that well audited and maintained third-party dependencies have been utilized and we did not identify any potential security vulnerabilities resulting from their use.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Balance Recording Forces Check-Effects Anti-Pattern	Resolved
Issue B: Minimized Replay Attack	Unresolved
Issue C: Loan Term Countdown Can Begin Before Loan is Funded [Known Issue]	Resolved
Suggestion 1: Make Users Aware of Centralization	Unresolved
Suggestion 2: Remove Unnecessary Require Statement	Resolved
Suggestion 3: Improve Documentation	Unresolved

Issue A: Balance Recording Forces Check-Effects Anti-Pattern

Location

`contracts/LoanCore.sol#L125`

Synopsis

The method chosen to record the balances held by the `LoanCore.sol` smart contract is to check the smart contract's balance after a transfer has occurred, forcing the smart contract to update the state of a balance after a call to the ERC-20 token smart contract has been made. This results in a possibility for a corrupt ERC-20 token to be used to re-enter the loan creation process. As good practice, the check-effects-interactions pattern should be used at all times.

Impact

Using a corrupt ERC-20 token to re-enter the loan creation process allows many invalid loans to be stored on the smart contract originating from the same loan terms agreement. However, the impact of this is likely negligible and would only cost the attacker.

Feasibility

Given that we could not determine an incentive for this attack, we consider this attack to be unlikely.

Remediation

We recommend using the loan state data, `data.terms.principal`, to update and record the balances. This will allow the `LoanCore.sol` smart contract to prevent an ERC-20 transfer from re-entering on line 110, since the received amount would now be updated.

Status

The Pawn Finance team refactored the code in `LoanCore.sol` to follow a stronger checks-effects-interactions pattern by moving the token transfer until after the token balance is updated (now on Line 119).

Upon review of the pull request where the above changes were made, the Pawn Finance team noted an issue with Fee on Transfer tokens and other tokens with nonstandard handling of transfers, causing the potential to miscalculate their internal balance. The Pawn Finance team decided to replace the previous way of updating balances, by pushing them into a mapping, with the use of `transferFrom` and `safeTransfer` patterns. We examined these changes and found no issues.

Verification

Resolved.

Issue B: Minimized Replay Attack

Location

`contracts/OriginationController.sol#L33`

Synopsis

In order to initialize a loan, there is an off-chain agreement that is made between lender and borrower, in which the loan terms are set. This loan terms agreement is then used to create a loan. This agreement is signed by either the borrower or the lender off-chain. It is required that the counterparty of the agreement then broadcast the transaction on-chain, effectively signing the agreement themselves. There is no nonce in the agreement signed, meaning that if multiple iterations of the agreement are signed by one party, the other may broadcast an older version of the agreement.

Impact

If a loan terms agreement is iterated off-chain, an older agreement might be used by a counterparty causing a loan to be started that a counterparty did not intend to be started, resulting in one of the parties carrying suboptimal loan terms.

Feasibility

The feasibility of this attack could not be assessed from the scope of this audit. The case of multiple iterations of loan terms agreements may be limited in possibility.

Mitigation

The Pawn Finance team has responded to this issue noting that this case should not be possible due to application code. However, we assert that application code can be bypassed on-chain, and a bug or incorrect usage of the protocol may result in an issue. The Pawn Finance team also responded that there is no conceivable incentive to perform a replay attack unless a mistake is made during the iteration of a loan terms agreement. Our team is not able to confirm this assertion.

The decentralized approach to solving this issue involves using a nonce on agreements and requiring a delay in the start of a loan for a counterparty to challenge an old loan agreement being used. This is not a trivial design and we do not suggest that this be implemented.

We recommend that the application warn users when they sign a loan terms agreement that this action is irreversible and that no further iterations of the agreement should be made. If a loan agreement must be updated off-chain, it should be known that there is no guarantee that the older signed agreement will not be used to start a loan. This extends to any point in the future.

Status

The Pawn Finance team responded that they intend to address this issue in the future. As a result, this issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue C: Loan Term Countdown Can Begin Before Loan is Funded [Known Issue]

Location

`contracts/OriginationController.sol#L33`

Synopsis

The Pawn Finance team identified an issue where, in `OriginationController.sol`, `intializeLoan` uses an absolute due date to initialize a loan when calling `createLoan` and `startLoan` from `LoanCore.sol`. However, upon loan creation, the repayment term will begin counting down towards an absolute due date without taking into account whether or not a loan has been funded.

Impact

In a worst-case scenario where a lender has negotiated a short repayment term, a borrower could default on a loan without having received the intended funds.

Technical Details

A loan requires a repayment term to be originated, which can be represented using a due date. In contracts/OriginationController.sol#L64-L65:

```
uint256 loanId = ILoanCore(loanCore).createLoan(loanTerms);

ILoanCore(loanCore).startLoan(lender, borrower, loanId);
```

The call data `loanTerms` passes an absolute `dueDate` to `createLoan`, which is then passed to `startLoan`. However, this due date is negotiated off-chain before the loan actually starts. It can be any date, so long as it is anytime later than the `block.timestamp` and thus not expired. In a scenario where a loan is originated on-chain unexpectedly later than the off-chain negotiation of terms, a borrower would have a reduced time frame to repay a loan.

Mitigation

Pawn Finance proposed a mitigation of using a due date that is relative to the `block.timestamp` the `startLoan` transaction was mined, which gives the borrower a repayment period that is consistent with the expectations of the loan terms negotiated off-chain.

Status

The Pawn Finance team have implemented a change using `block.timestamp + terms.relDueDate` to represent the `dueDate`, when calculating if the repayment period has passed and a loan has defaulted.

Verification

Resolved.

Suggestions

Suggestion 1: Make Users Aware of Centralization

Location

contracts/LoanCore.sol#L45

Synopsis

An administrator role `DEFAULT_ADMIN_ROLE` is set during the construction of the `LoanCore.sol` smart contract that has centralized powers that could adversely affect all users of the Pawn Finance protocol. The centralized `DEFAULT_ADMIN_ROLE` is able to pause the functions `createLoan`, `startLoan`, and `claim`, which perform the core functionality of the system.

Using a pause mechanism is considered a safety feature in the event that a bug is identified, the administrator may pause the smart contract to correct the issue before users are affected. Given the prevalence of Miner Extractable Value (MEV) bots in the Ethereum mining ecosystem at this time, this assumption is questionable.

Furthermore, the risk that the pause feature is used in a malicious way by the authorized entity cannot be discounted. Therefore, the emergency stop implemented by this pattern should only be triggered as a last resort under specific circumstances and with a minimum threshold of consensus.

Mitigation

We recommend creating documentation that outlines the specific cases in which the centralized administrator role account is used, and providing supporting documentation describing the security

benefits and trade-offs of the administrator role. We also suggest making users aware of these trade-offs so that they may make informed security decisions.

Status

The Pawn Finance team responded they intend to address this suggestion in the future. As a result, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 2: Remove Unnecessary `Require` Statement

Location

`contracts/LoanCore.sol#L82`

Synopsis

The `require` statement in `LoanCore.sol` (Line 82) checks if the collateral from the borrower has not been created or already been burned by checking if the NFT ID is assigned to `address(0)`. This check burns gas and includes an error string, which adds to the cost of deployment since strings are an expensive datatype to deploy on the Ethereum blockchain. However, the `startLoan` function performs its own check on the address in `LoanCore.sol` (Line 106) when the collateral is transferred. Therefore, the check is effectively performed twice.

We investigated the security implications of omitting the `require` statement in `LoanCore.sol` (Line 82) and found the collateral check performed in the function `startLoan` to be sufficient.

Mitigation

We recommend removing the `require` statement in `LoanCore.sol` (Line 82) to save on gas incurred from the check and the deployment of the error string associated with it.

Status

The Pawn Finance team removed the `require` statement in `LoanCore.sol` (Lines 85 - 88).

However, the check performed by the `startLoan` function in `LoanCore.sol` (Line 106) when the collateral is transferred was deleted in a later commit (see Issue A). The same functionality is found in code added during the same commit that calls `transferFrom` on `collateralToken` and, as a result, we consider this suggestion resolved.

Verification

Resolved.

Suggestion 3: Improve Documentation

Location

`pawnfi-contracts#readme`

`docs/interfaces/IPromissoryNote.md`

Synopsis

Organize Documentation

The supplemental documentation provided by the Pawn Finance team is not currently available in the project repository and is only available in externally stored pdf documents.

Consistently Document Functions

Most of the smart contracts have sufficiently documented every function, however, some are more comprehensively documented than others (e.g. `docs/interfaces/IPromissoryNote.md` provides less comprehensive documentation than some of the other smart contracts).

Mitigation

Organize Documentation

We recommend that all supplementary documentation provided by the Pawn Finance team be incorporated into the in-scope repository, in order to provide easier access and better guidance on the intended behavior of the system.

Consistently Document Functions

We recommend documenting all smart contracts consistently, including functions whose implementations appear obvious, in order to reduce the potential for errors or confusion about intended functionality.

Status

The Pawn Finance team responded they intend to address this suggestion in the future. As a result, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.