Quantstamp Security Assessment Certificate

June 2nd 2022 — Quantstamp Verified

Pine

This audit report was prepared by Quantstamp, the leader in blockchain security.

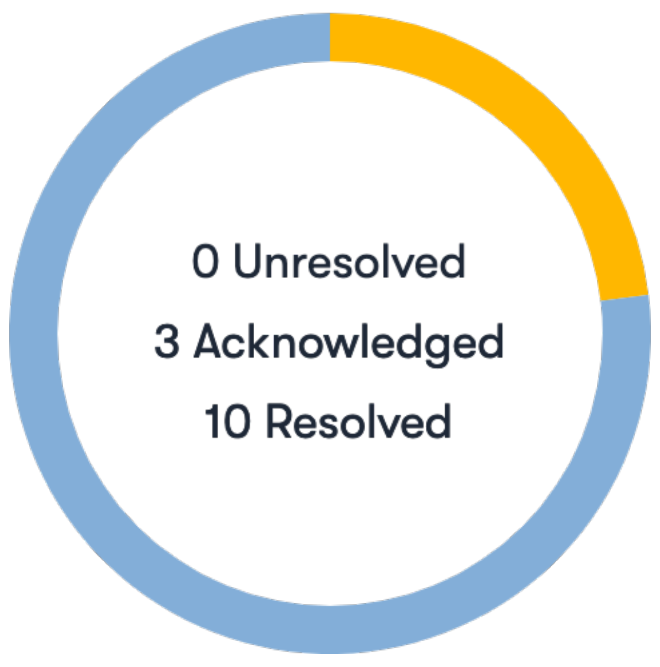
Executive Summary

Type	Lending Pools
Auditors	Souhail Mssassi, Research Engineer Rabib Islam, Research Engineer Roman Rohleder, Research Engineer
Timeline	2022-04-06 through 2022-04-29
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Pine Documentation
Documentation Quality	Low
Test Quality	Undetermined

Source Code

Repository	Commit
lending-borrowing-smart-contracts	fea59e1
lending-borrowing-smart-contracts	f7c110c
lending-borrowing-smart-contracts	9b7f7cf

Total Issues	**13**	(10 Resolved)
High Risk Issues	5	(4 Resolved)
Medium Risk Issues	1	(1 Resolved)
Low Risk Issues	5	(3 Resolved)
Informational Risk Issues	1	(1 Resolved)
Undetermined Risk Issues	1	(1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Initial audit:
Through reviewing the code, we found **13 potential issues** of various levels of severity: 5 high-severity, 1 medium-severity, 5 low-severity, 1 informational-severity and 1 undetermined issues. We recommend addressing all the issues before deploying the code.
After the re-audit:
The pine team has fixed the majority of the issues and the contracts are ready to be deployed.

ID	Description	Severity	Status
QSP-1	Replay Attack On Signature In `Borrow` Function	High	Fixed
QSP-2	Front Run on Signature in The `Borrow` Function	High	Fixed
QSP-3	Fees Can Be Bypassed	High	Fixed
QSP-4	Signature Malleability	High	Fixed
QSP-5	Pool Owner Can Steal NFTs	High	Acknowledged
QSP-6	Usage Of `transfer` Instead Of `safeTransfer`	Medium	Mitigated
QSP-7	Missing Value Verification	Low	Acknowledged
QSP-8	Missing Value Verification Can Lead To A Denial Of Service	Low	Fixed
QSP-9	Missing Address Verification	Low	Fixed
QSP-10	Dead Code	Low	Fixed
QSP-11	Privileged roles and ownership	Low	Acknowledged
QSP-12	Events not emitted on state change	Informational	Mitigated
QSP-13	Missing the pausable modifier in the `flashLoan` function	Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.2

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Replay Attack On Signature In `Borrow` Function

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`

Description: In the `borrow` function, we are providing the parameter `signature`, which is verified in function `VerifySignaturePool02.verify`. The issue here is that nothing prevents from double submitting the signature, and thus making a replay attack possible.

Recommendation: Consider associating for each signature a nonce that it will be incremented, to prevent the replay attack.

Update: Fixed by taking the calling address and a new incrementing nonce into account for the signature. Fixed in commits 427d171, 5e82702 and 1460097.

QSP-2 Front Run on Signature in The `Borrow` Function

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`

Description: In the `borrow` function, we are inserting the parameter `signature` which is verified in the `VerifySignaturePool02.verify`, the issue here is that nothing prevent from viewing the signature in the mempools and re-use them.

Recommendation: In the signature consider adding also the address of the caller; this way even if someone has extracted the signature, it will be unusable.

Update: Fixed by taking the calling address into account for the signature. Fixed in commit cb5e245.

QSP-3 Fees Can Be Bypassed

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`

Description: At every `flashLoan` executed in the contract, a percentage is taken from the amount as `amountFee`. In the case where the `_amount*lenderFeeBips` variable is lower than 10000000000, the fee variable will be equal to 0 due to the type conversion. The same issue exists for the `protocolFee` in the `borrow` function(L229)

Recommendation:

- It is recommended to add a require statement to make sure that `_amount*lenderFeeBips` is higher than the 10000000000.
- It is recommended to add a require statement to make sure that `x[4] * protocolFeeBips` is higher than the 10000.

Update: Fixed by implementing the recommended `require()` statements.

QSP-4 Signature Malleability

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/VerifySignaturePool02.sol`

Related Issue(s): SWC-https://swcregistry.io/docs/SWC-117

Description: The given implementation of signature verification in `VerifySignaturePool02.recoverSigner()` using `ecrecover` directly is prone to signature malleability.

Recommendation: Consider using a secure wrapper like OpenZeppelins ECDSA utility library, which performs additional security checks on the signature parameters.

Update: Fixed by making use of OpenZeppelin's ECDSA library, as suggested. Fixed in commit 12bb678.

QSP-5 Pool Owner Can Steal NFTs

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `contracts/ControlPlane01.sol`

Description: Whenever `PineLendingLibrary.nftHasLoan(...)` returns false for an NFT (which simply requires `loanTerms.returnedWei <= loanTerms.borrowedWei`), the owner of a pool (`ERC721LendingPool`) can call `ControlPlane01.withdrawNFT(...)` and thereby transfer the NFT to their wallet.

Recommendation: If this is intended, the team should document this behavior.

Update: The team has acknowledged the risk, and they updated their documentation https://docs.pine.loans/documentations/core/controlplane01#scenario to document the intended behaviour.

QSP-6 Usage Of `transfer` Instead Of `safeTransfer`

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `ERC721LendingPool02.sol`, `ERC1155LendingPool02.sol`, `Router01.sol`, `PineFinancing01.sol`, `LoanRollover01.sol`, `ControlPlane01.sol`

Description: The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. 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 didn't.

```
* `contracts/ERC721LendingPool02.sol` (L143);
* `contracts/ERC721LendingPool02.sol` (L262);
* `contracts/ERC721LendingPool02.sol` (L233);
* .`contracts/ERC721LendingPool02.sol` (L238);
* `contracts/ERC721LendingPool02.sol` (L296);
* `contracts/ERC721LendingPool02.sol` (L307);
* `contracts/ERC721LendingPool02.sol` (L356);
* `contracts/ERC721LendingPool02.sol` (L386);
* `contracts/ERC1155LendingPool02.sol` (L144);
* `contracts/ERC1155LendingPool02.sol` (L235);
* `contracts/ERC1155LendingPool02.sol` (L240);
* `contracts/ERC1155LendingPool02.sol` (L301);
* `contracts/ERC1155LendingPool02.sol` (L312);
* `contracts/ERC1155LendingPool02.sol` (L363);
* `contracts/ERC1155LendingPool02.sol` (L393);
* `contracts/Router01.sol` (L54);
* `contracts/Router01.sol` (L56);
* `contracts/PineFinancing01.sol` (L92);
* `contracts/LoanRollover01.sol` (L54);
* `contracts/ControlPlane01.sol` (L54);
```

Recommendation: Use the `safeTransfer` function from the SafeERC20 implementation, or put the transfer call inside an assert or require verifying that it returned true.

Update: The following lines remains using the require wrapper.

```
L144 of ERC1155LendingPool02.sol
L235 of ERC1155LendingPool02.sol
L240 of ERC1155LendingPool02.sol
L301 of ERC1155LendingPool02.sol
L363 of ERC1155LendingPool02.sol
```

QSP-7 Missing Value Verification

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/ControlPlane01.sol`

Description:

- In the `setFee` function, the contract should verify if `feeBps` is less than 100%.
- In the `setDurationParam` function, the value `ppm` is not verified.

Recommendation:

- Consider verifying `feeBps` to be less than 100%.
- Consider adding a limit for the `ppm` variable.

Update: Partially fixed in commit 341708e, by checking the parameter of `setFee()` (`ControlPlane01.sol`) to be within 100%, as suggested. The parameter in function `setDurationParam()`.

QSP-8 Missing Value Verification Can Lead To A Denial Of Service

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`

Description: In the `setBlockLoanLimit` function located in the `ERC721LendingPool02` contract, the `blockLoanLimit` can be changed, the issue here is that if this is updated to the value 0, the `borrow` function will always revert due to the verification done in the `updateBlockLoanAmount` (L101).

Recommendation: It is recommended to make sure the `blockLoanLimit` surpasses 0.

Update: Fixed in commit 3f265e7, by checking parameter `bll` to be greater than zero, as suggested.

QSP-9 Missing Address Verification

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`, `contracts/PineFinancing01.sol`

Description: Certain functions lack a safety check on the address. The address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

- `contracts/ERC721LendingPool02.sol` (L70,71,72,73);
- `contracts/PineFinancing01.sol` (L15);

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

Update: Fixed in commit bd1fb94, by adding corresponding `require()` statements, as suggested.

- contracts/ERC721LendingPool02.sol (L70,71,72,73): Resolved
- contracts/PineFinancing01.sol (L15): Resolved

QSP-10 Dead Code

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/ControlPlane01.sol`, `contracts/PineFinancing01.sol`, `contracts/Router01.sol`

Description: In the `ControlPlane01` contract, we have a code comment of the function of the function `callLoan`. Further occurrences of commented out code are listed below.

- `contracts/ControlPlane01.sol` (L58-60);
- `contracts/PineFinancing01.sol` (L32);
- `contracts/Router01.sol` (L19-32);

Recommendation: If the commented code is not used, consider removing it before going to mainnet.

Update: Fixed in commit 1d6283e, by removing corresponding comments, as suggested.

- contracts/ControlPlane01.sol (L58-60): Resolved
- contracts/PineFinancing01.sol (L32): Resolved
- contracts/Router01.sol (L19-32): Resolved

QSP-11 Privileged roles and ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/CloneFactory02.sol`, `contracts/Migrations.sol`, `contracts/ERC721LendingPool02.sol`, `contracts/ERC1155LendingPool02.sol`, `contracts/Router01.sol`, `contracts/PineFinancing01.sol`, `contracts/PineWallet01.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The `CloneFactory02.sol` contract contains the following privileged roles:

- `owner`, as initialized during the constructor:
  - Renounce his role and thereby disable all following listed actions, by calling `renounceOwnership()`.
  - Assign an arbitrary address as new `owner`, by calling `transferOwnership()`.
  - Add/Remove arbitrary addresses of the clonable targets list, by calling `toggleWhitelistedTarget()`.

The `Migrations.sol` contract contains the following privileged roles:

- `owner`, as initialized during deployment:
  - Set/Unset `last_completed_migration`, by calling `setCompleted()`.

Contracts `ERC721LendingPool02.sol` and `ERC1155LendingPool02.sol` contain the following privileged roles:

- `owner`, as initialized during deployment to `msg.sender`:
  - Renounce his role and thereby disable all followingly listed actions, by calling `renounceOwnership()`.
  - Assign an arbitrary address as new `owner`, by calling `transferOwnership()`.
  - Set an arbitrary high loan limit per block, by calling `setBlockLoanLimit()`.
  - Set arbitrary pool parameters, by calling `setDurationParam()`.
  - Pausing/Unpausing the contract (impacting the callability of `repay()` and `borrow()`), by calling `pause()`/`unpause()`.
  - Withdraw an arbitrary amount of ETH to itself, by calling `withdraw()`.
  - Withdraw an arbitrary amount of a given ERC20 address to itself, by calling `withdrawERC20()`.

The `Router01.sol` contract contains the following privileged roles:

- `owner`, as initialized during deployment to `msg.sender`:
  - Renounce his role and thereby disable all followingly listed actions, by calling `renounceOwnership()`.
  - Assign an arbitrary address as new `owner`, by calling `transferOwnership()`.
  - Approve an arbitrary amount to an arbitrary address for a given currency to this and all inheriting contracts, by calling `approvePool()`.

The `PineFinancing01.sol` contract contains the following privileged roles:

- `owner`, as initialized during deployment to `msg.sender`:
  - Renounce his role and thereby disable all followingly listed actions, by calling `renounceOwnership()`.
  - Assign an arbitrary address as new `owner`, by calling `transferOwnership()`.
  - Approve/Disapprove arbitrary address/function selector pairs (`approvedMethods[]`) to be called via `buyERC721()`, by calling `setWhitelistedMethods()`.
  - Withdraw an arbitrary amount of ETH from the contract to its address, by calling `withdraw()`.
  - Withdraw an arbitrary amount of any ERC20 from the contract to its address, by calling `withdrawERC20()`.
  - Transfer an arbitrary ERC721 from the contract to its address, by calling `withdrawERC721()`.

The ControlPlane01.sol contract contains the following privileged roles:

- owner, as initialized during deployment to msg.sender:
  - Renounce his role and thereby disable all following listed actions, by calling renounceOwnership().
  - Assign an arbitrary address as new owner, by calling transferOwnership().
  - Add/Remove arbitrary addresses from the whitelisted intermediaries mapping (whitelistedIntermediaries[]), by calling toggleWhitelistedIntermediaries().
  - Set an arbitrary whitelisted factory address (whitelistedFactory), by calling setWhitelistedFactory().
  - Set an arbitrary high fee up to type(uint32).max, 4294967295 or 42.949.672,95 % (feeBps), by calling setFee().
  - Withdraw an arbitrary ETH amount from the contract to its address, by calling withdraw().
  - Withdraw an arbitrary amount of any ERC20 from the contract to its address, by calling withdrawERC20().
  - Liquidate any liquidable NFT to itself, by calling liquidateNFT().
  - Withdraw any NFT with an outstanding loan to itself, by calling withdrawNFT().

The PineWallet01.sol contract contains the following privileged roles:

- owner, as initialized during deployment to msg.sender:
  - Renounce his role and thereby disable all following listed actions, by calling renounceOwnership().
  - Assign an arbitrary address as new owner, by calling transferOwnership().
  - Call an arbitrary function of an address not in the collateralizedCollections[] mapping with the context of the contract, by calling call().
  - Withdraw an arbitrary ETH amount from the contract to its address, by calling withdraw().
  - Is the address that is checked against as the only valid signer in isValidSignature().

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

Update: The team has acknowledged this issue, stating that We acknowldged that all found issues are intended behaviour in this QSP. We will improve documentation.

QSP-12 Events not emitted on state change

Severity: *Informational*

Status: Mitigated

File(s) affected: contracts/CloneFactory02.sol, contracts/ERC721LendingPool02.sol, contracts/ERC1155LendingPool02.sol, contracts/PineFinancing01.sol, contracts/ControlPlane01.sol, contracts/PineWallet01.sol

Description: An event should always be emitted when a state change is performed in order to facilitate smart contract monitoring by other systems which want to integrate with the smart contract.
This is not the case for the functions and the correspondingly modified state variables:

1. CloneFactory02.toggleWhitelistedTarget(), after changing targets[].
2. ERC721LendingPool02.setBlockLoanLimit(), after changing blockLoanLimit.
3. ERC1155LendingPool02.setBlockLoanLimit(), after changing blockLoanLimit.
4. ERC721LendingPool02.setDurationParam(), after changing durationSeconds_poolParam[].
5. ERC1155LendingPool02.setDurationParam(), after changing durationSeconds_poolParam[].
6. ERC721LendingPool02.updateBlockLoanAmount(), after changing blockLoanAmount[].
7. ERC1155LendingPool02.updateBlockLoanAmount(), after changing blockLoanAmount[].
8. PineFinancing01.setWhitelistedMethods(), after changing approvedMethods[].
9. ControlPlane01.toggleWhitelistedIntermediaries(), after changing whitelistedIntermediaries[].
10. ControlPlane01.setWhitelistedFactory(), after changing whitelistedFactory.
11. ControlPlane01.setFee(), after changing feeBps.
12. PineWallet01.depositCollateral(), after changing collateralizedCollections[].

Recommendation: Emit an event in the aforementioned functions.

Update: Partially fixed in commit afa8c09, by emitting events in ERC721LendingPool02.setDurationParam() and ERC1155LendingPool02.setDurationParam(). All other occurences have not been fixed.

1. **Unresolved**
2. **Unresolved**
3. **Unresolved**
4. Resolved
5. Resolved
6. **Unresolved**
7. **Unresolved**
8. **Unresolved**
9. **Unresolved**
10. **Unresolved**
11. **Unresolved**

QSP-13 Missing the pausable modifier in the `flashLoan` function

Severity: Undetermined

Status: Fixed

File(s) affected: `contracts/ERC721LendingPool02.sol`

Description: In the `flashloan` function, the modifier `pausable` is missing, and thus if anything happens to the contract, the function `flashloan` will be working and unstoppable.

Recommendation: Clarify if this is intended behavior and if not make sure to add the `pausable` modifier.

Update: Fixed in commit 07fa774, by adding the `whenNotPaused` modifier to said functions.

Automated Analyses

Slither

The majority of the issues alerted by slither are false positive.

Code Documentation

1. Unresolved `TODO` items in code:
   1. L64 and L83 of `ControlPlane01.sol`: `TODO: check unhealthy` (Unresolved)
2. Typo on L12 of `PineWallet01.sol`: `evnets` → `events`. (Unresolved)

Adherence to Best Practices

1. It's better to use `create2` instead of `create` in the `createClone` function located in the `CloneFactory02.sol` contract. (Fixed)
2. To facilitate logging, it is recommended to index address parameters within events. Therefore, the `indexed` keyword should be added to the (other) address parameters in
   1. `CloneFactory02.Cloned()`, (Fixed)
3. Use explicit types/type widths (i.e. use `uint256` over `uint`). Affected lines: (Fixed)
   1. `PineFinancing01.sol`: L24.
   2. `Router01.sol`: L15 and L59.
   3. `WETH.sol`: L27, L28, L29, L30, L42, L53, L59, L65 and L80.
   4. `PineWallet01.sol`: L47 and L53.
   5. `PineLendingLibrary.sol`: L17.
   6. `ControlPlane01.sol`: L15, L29, L34, L35 and L36.
   7. `VerifySignaturePool02.sol`: L35, L36, L37, L79, L80 and L81.
4. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly (Fixed):
   1. `10000000000` on L133 and L341 of `ERC721LendingPool02.sol`.
   2. `10000000000` on L134 and L348 of `ERC1155LendingPool02.sol`.
   3. `10000` on L220 and L229 of `ERC721LendingPool02.sol`.
   4. `10000` on L222 and L231 of `ERC1155LendingPool02.sol`.
   5. `10000000000` on L23 of `PineLendingLibrary.sol`.
5. To ensure correct behavior, function return values should always be checked. Accordingly, consider adding checks for the following cases: (Fixed)
   1. L46 of `Router01.sol`
   2. L48 of `LoanRollover01.sol`
6. To improve readability and code quality, consider using meaningful variable names instead of short abbreviations. In this regard, consider renaming variables `a`, `b`, `c`, `e`, `f`, `g` and `h` in `ControlPlane01.liquidateNFT()` and `ControlPlane01.withdrawNFT()` with the corresponding `LoanTerms` field names. (Acknowledged)
7. For improved readability and code quality, it is advised to remove duplicate or unused code. In this regard, consider the following cases: (Fixed)
   1. Function `PineLendingLibrary.isUnHealthyLoan()` always returns an unused second return value of `0`. Consider removing it.
   2. Contracts `PineWallet01.sol` and `VerifySignaturePool02.sol` share the same function `recoverSigner()` and `splitSignature()`. Consider dropping the ones in `PineWallet01.sol` and instead import and re-use those in `VerifySignaturePool02.sol`.
   3. Variable `_allowList` on L34 of `PineWallet01.sol` is unused and should therefore be removed.
8. Variable `_baseURIextended` in `PineWallet01.sol` is only written to once with and with a constant string. Consider therefore declaring it a constant and initialized state variable instead. (Fixed)
9. Contracts `ERC721LendingPool02.sol` and `ERC1155LendingPool02.sol` share many identical and many near identical functions. To improve readability and maintainability, consider unifying those into a shared library. (Acknowledged)
10. There are multiple functions with no internal calls that are declared public, whereas declaring them external may result in gas savings. (Fixed)

- `onERC721Received`
- `initialize`
- `setBlockLoanLimit`
- `setDurationParam`
- `pause`
- `unpause`

Test Results

Test Suite Results

truffle test

```
√ should approve the contract for spending NFTs
√  should mint NFTs
√  should not borrow more than what it could
√  should not supply fake valuation
√  should borrow some money and the NFT is located in the contract
√  should borrow some money and the NFT is located in the contract (nftid 2)
√  should return part of the money (eating into principal)
√  should rollover
√  should return part of the money (doesn't eat into principal)
√  admin should withdraw nfts
√  admin should not withdraw nfts on lien
√  should not let people steal NFTs
√  should return all of the money
√  admin should withdraw proceeds
√  should not be liquidated when not expired
√  should not be liquidated by non-pool-owner when expired
√  should be liquidated by pool-owner when expired
√  should withdraw airdropped NFTs
√  should collateralize NFTs
√  should not operate collateralized NFTs
√  should  withdraw assets not on lien
```

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

`cd02eea6f13cbdc20809e8c4f3ec067544b7613119281f32e645a3c594f56b41 ./contracts/CloneFactory02.sol`

`43909573183c88287e1d47e06854a21404032a9aeffb7ca4e6097fffb28c7f8e ./contracts/ControlPlane01.sol`

`6b4f85a358000e24a4e8e8cbc825a18ec22c19b4843dc7895830bfd5d767990a ./contracts/ERC1155LendingPool02.sol`

`62d72c5ca4132178fddd986e0b289cf1924f6086b3e5123fe04aad2ab249a3c4 ./contracts/ERC721LendingPool02.sol`

`12bc031207b72c860d555d7b5b75ae296e5ef276d6fe9870b5cd9da9ea090315 ./contracts/LoanRollover01.sol`

`2e72e6c3288c7101a1035314e0af381f8be7ae13d6cfc5c0b75e4275643660f7 ./contracts/PineFinancing01.sol`

`caeb022107087e3f0ad43bbc693cc18aacf0f25dc9a2d6c4a7809867df039a04 ./contracts/PineLendingLibrary.sol`

`d3d0e998443126d5c1a1dc9d7ca56c32f8da7975b358e0e3c768540299a7af1c ./contracts/PineWallet01.sol`

`781c33ebed10ba21f3405f17e56464fc8190c101b3df093e3a874602071e6cd1 ./contracts/Router01.sol`

`66a60bf91f90384b79ae66000124ef507a6985b7c38787b6c5b626255c109e73 ./contracts/VerifySignaturePool02.sol`

`9a3d3c83efc2f89ba8c0fb40fe3e6b0d1a917a1e3e3faefcd7c2fce8b801a308 ./contracts/interfaces/ICloneFactory02.sol`

`d87c4b6d0c849e98c65f872ebc960f9e0cff24e70ec0f2aba31b94ac07c21ebe ./contracts/interfaces/IControlPlane01.sol`

`32d071d5d57deb257798f4b4a68ace7ae5db44e74302b3383f7e58460d346cc3 ./contracts/interfaces/IFlashLoanReceiver.sol`

`7ee61e2137732d34054c1fad7c973b8491eee3a3d7838b361790ea7f592ec5db ./contracts/interfaces/WETH.sol`

Tests

`a8d67cd1870a4d34b0449405790bc8d9a415f7a27969940e6c1437298c1a575e ./test/erc721lendingpool02.js`

`26ae5f2c89351b60c5cc6480e6c815acfc8b29b3f0b8844307bf96150a05754f ./test/pinewallet.js`

Changelog

- 2022-05-02 - Initial report
- 2022-05-17 - Final report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.