// Private Smart Contract Security Assessment 05.06.2024 - 05.27.2024

v1.2 PWN

HALBORN

v1.2 - PWN

Prepared by: HALBORN

Last Updated 05/29/2024

Date of Engagement by: May 6th, 2024 - May 27th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	1	3

TABLE OF CONTENTS

Introduction

The PWN team engaged Halborn to conduct a security assessment on their smart contracts beginning on May 6th, 2024, and ending on May 27th, 2024. The security assessment was scoped to the smart contracts provided in the PWNFinance/pwn _contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

Assessment Summary

Halborn was provided 2 weeks and 3 days for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.
The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some non-critical issues, that were acknowledged and accepted by the PWN team.

Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (slither).
- Testnet deployment (Foundry).

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/with the **remediation commit IDs.**
- Changes that occur outside of the scope of PRs/Commits.

1. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

1.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

1.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

1.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2. SCOPE

FILES AND REPOSITORY

(a) Repository: pwn_contracts

(b) Assessed Commit ID: 4b28847

(c) Items in scope:

- src/loan/token/PWNLOAN.sol
- src/loan/terms/simple/loan/PWNSimpleLoan.sol
- src/loan/terms/simple/proposal/PWNSimpleLoanProposal.sol
- src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol
- src/loan/terms/simple/proposal/PWNSimpleLoanFungibleProposal.sol
- src/loan/terms/simple/proposal/PWNSimpleLoanSimpleProposal.sol
- src/loan/terms/simple/proposal/PWNSimpleLoanListProposal.sol
- src/loan/lib/PWNFeeCalculator.sol
- src/loan/lib/PWNSignatureChecker.sol
- src/loan/vault/PWNVault.sol
- src/loan/vault/Permit.sol
- src/nonce/PWNRevokedNonce.sol
- src/config/PWNConfig.sol
- src/PWNErrors.sol
- src/hub/PWNHub.sol
- src/hub/PWNHubAccessControl.sol
- src/hub/PWNHubTags.sol
- src/interfaces/IPWNLoanMetadataProvider.sol
- src/interfaces/IERC5646.sol
- src/interfaces/IPWNDeployer.sol

Out-of-Scope:

Out-of-Scope: New features/implementations after the remediation commit IDs.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNSAFE ERC721 _MINT FUNCTION	Low	RISK ACCEPTED
EVENTS ARE MISSING `INDEXED` ATTRIBUTE	Informational	ACKNOWLEDGED
MISSING INPUT VALIDATION WHEN ASSIGNING ADDRESSES TO STATE VARIABLES	Informational	ACKNOWLEDGED
OUTDATED COMPILER VERSION	Informational	ACKNOWLEDGED

4. FINDINGS & TECH DETAILS

4.1 UNSAFE ERC721 _MINT FUNCTION

// LOW

Description

In the current scope, the loans are represented by `ERC-721` NFTs, controlled by the `PWNLOAN.sol` contract. Specifically, the access control mechanism of the `mint` function is ruled by the `onlyActiveLoan` modifier, effectively blocking unauthorized parties to call the `mint` method, and inherently preventing against reentrancy attacks, once it's callable only from trusted (tagged) addresses.
`- src/loan/token/PWNLOAN.sol`

```
69      function mint(address owner) external onlyActiveLoan returns
70  (uint256 loanId) {
71          loanId = ++lastLoanId;
72          loanContract[loanId] = msg.sender;
73          _mint(owner, loanId);
74          emit LOANMinted(loanId, msg.sender, owner);
        }
```

Although the current implementation has the proper access control mechanism, and therefore it is less likely to be the target of a reentrancy attack, the `_mint` method can be considered unsafe, as it allows `non-ERC721` receivers to receive the minted `NFT`, effectively blocking the recipient to properly interact with the Loan NFT.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to use `_safeMint` method, effectively preventing ERC-721's to be minted by mistake to incompatible addresses. This recommendation considers the external `mint` function has the proper access control mechanism with the `tag` system, and therefore reentrancy would only be an issue in case of abuse of trust by authorized parties.

Remediation Plan

RISK ACCEPTED: The **PWN team** has accepted the risk related to this finding.

4.2 EVENTS ARE MISSING `INDEXED` ATTRIBUTE

// INFORMATIONAL

Description

Indexed event fields make the data more quickly accessible to off-chain tools that parse events, and adds them to a special data structure known as “topics” instead of the data part of the log. A topic can only hold a single word (32 bytes) so if you use a reference type for an indexed argument, the Keccak-256 hash of the value is stored as a topic instead.
Each event can use up to three indexed fields. If there are fewer than three fields, all of the fields can be indexed. It is important to note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed fields per event (three indexed fields).
This is specially recommended when gas usage is not particularly of concern for the emission of the events in question, and the benefits of querying those fields in an easier and straight-forward manner surpasses the downsides of gas usage increase.

- src/loan/terms/simple/loan/PWNSimpleLoan.sol

```
        event LOANExtended(uint256 indexed loanId, uint40
originalDefaultTimestamp, uint40 extendedDefaultTimestamp);
```

- src/loan/terms/simple/loan/PWNSimpleLoan.sol

```
        event ExtensionProposalMade(bytes32 indexed extensionHash,
address indexed proposer,  ExtensionProposal proposal);
```

- src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol

```
        event ProposalMade(bytes32 indexed proposalHash, address indexed
proposer, Proposal proposal);
```

- src/loan/terms/simple/proposal/PWNSimpleLoanFungibleProposal.sol

```
        event ProposalMade(bytes32 indexed proposalHash, address indexed
proposer, Proposal proposal);
```

- src/loan/terms/simple/proposal/PWNSimpleLoanListProposal.sol

```
        event ProposalMade(bytes32 indexed proposalHash, address indexed
proposer, Proposal proposal);
```

- src/loan/terms/simple/proposal/PWNSimpleLoanSimpleProposal.sol

```
        event ProposalMade(bytes32 indexed proposalHash, address indexed
proposer, Proposal proposal);
```

- src/loan/vault/PWNVault.sol

```
        event VaultPull(MultiToken.Asset asset, address indexed origin);
```

- src/loan/vault/PWNVault.sol

```
        event VaultPush(MultiToken.Asset asset, address indexed
beneficiary);
```

- src/loan/vault/PWNVault.sol

```
        event VaultPushFrom(MultiToken.Asset asset, address indexed
origin, address indexed beneficiary);
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to add the `indexed` keyword when declaring events, considering the following example:

```
    event Indexed(
        address indexed from,
        bytes32 indexed id,
        uint indexed value
    );
```

Remediation Plan

ACKNOWLEDGED: The **PWN team** has acknowledged this finding.

References

https://docs.soliditylang.org/en/v0.8.26

4.3 MISSING INPUT VALIDATION WHEN ASSIGNING ADDRESSES TO STATE VARIABLES

// INFORMATIONAL

Description

In Solidity, it is essential to ensure the integrity and security of your smart contract by validating inputs before assigning them to state variables. One common oversight is the lack of input validation when assigning addresses to state variables, which can potentially lead to unintended consequences or security vulnerabilities.

By implementing proper input validation for addresses, you can prevent the assignment of invalid or malicious addresses to state variables, thereby reducing the risk of unexpected behavior in your smart contract.

– src/config/PWNConfig.sol

```
            _loanMetadataUri[loanContract] = metadataUri;
```

– src/config/PWNConfig.sol

```
            _loanMetadataUri[address(0)] = metadataUri;
```

– src/config/PWNConfig.sol

```
            _poolAdapterRegistry[pool] = adapter;
```

– src/hub/PWNHub.sol

```
            tags[_address][tag] = _hasTag;
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

Implement proper input validation for addresses assigned to state variables by using `require` or `if` statements.

Remediation Plan

ACKNOWLEDGED: The **PWN team** has acknowledged this finding.

4.4 OUTDATED COMPILER VERSION

// INFORMATIONAL

Description

It was identified an outdated compiler version across multiple contracts in-scope.
As from the official Solidity documentation, it is recommended to use the latest released version of Solidity.
It was identified that the contracts in the set under analysis are using solc version `0.8.16`, hence, outdated, considering the current Solidity compiler (solc) version is `0.8.26`.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Update to the most recent version of Solidity (0.8.26), by changing the pragma as follows:

```
pragma solidity 0.8.26;
```

Attention:
Solc compiler `version 0.8.20` switches the default target EVM version to Shanghai. The generated bytecode will include **PUSH0** opcodes. The recently released Solc compiler `version 0.8.25` switches the default target EVM version to Cancun, so it is also important to note that it also adds-up new opcodes such as **TSTORE**, **TLOAD** and **MCOPY**.
Be sure to select the appropriate EVM version in case you intend to deploy on a chain apart from mainnet like L2 chains that may not support **PUSH0**, **TSTORE**, **TLOAD** and/or **MCOPY**, otherwise deployment of your contracts will fail.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this finding.

5. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

```
INFO:Detectors:
Reentrancy in PWNSimpleLoan.createLOAN(PWNSimpleLoan.ProposalSpec,PWNSimpleLoan.LenderSpec,PWNSimpleLoan.CallerSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#371-460):
        External calls:
        - revokedNonce.revokeNonce(msg.sender,callerSpec.nonce) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#384)
          (proposalHash,loanTerms) = PWNSimpleLoanProposal(proposalSpec.proposalContract).acceptProposal({acceptor:msg.sender,refinancingLoanId:callerSpec.refinancingLoanId,proposalData:proposalSpec.proposalData,proposalInclusionProof:proposalSpec.proposalInclusionProof,signature:proposalSpec.signature}) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#394-401)
        - loanId = _createLoan({proposalHash:proposalHash,proposalContract:proposalSpec.proposalContract,loanTerms:loanTerms,lenderSpec:lenderSpec,extra:extra}) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#428-434)
          - loanId = loanToken.mint(loanTerms.lender) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#528)
        State variables written after the call(s):
        - loanId = _createLoan({proposalHash:proposalHash,proposalContract:proposalSpec.proposalContract,loanTerms:loanTerms,lenderSpec:lenderSpec,extra:extra}) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#428-434)
          - loan.status = 2 (src/loan/terms/simple/loan/PWNSimpleLoan.sol#532)
          - loan.creditAddress = loanTerms.credit.assetAddress (src/loan/terms/simple/loan/PWNSimpleLoan.sol#533)
          - loan.originalSourceOfFunds = lenderSpec.sourceOfFunds (src/loan/terms/simple/loan/PWNSimpleLoan.sol#534)
          - loan.startTimestamp = uint40(block.timestamp) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#535)
          - loan.defaultTimestamp = uint40(block.timestamp) + loanTerms.duration (src/loan/terms/simple/loan/PWNSimpleLoan.sol#536)
          - loan.borrower = loanTerms.borrower (src/loan/terms/simple/loan/PWNSimpleLoan.sol#537)
          - loan.originalLender = loanTerms.lender (src/loan/terms/simple/loan/PWNSimpleLoan.sol#538)
          - loan.accruingInterestDailyRate = SafeCast.toUint40(Math.mulDiv(loanTerms.accruingInterestAPR,APR_TO_DAILY_INTEREST_NUMERATOR,APR_TO_DAILY_INTEREST_DENOMINATOR)) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#539-541)
          - loan.fixedInterestAmount = loanTerms.fixedInterestAmount (src/loan/terms/simple/loan/PWNSimpleLoan.sol#542)
          - loan.principalAmount = loanTerms.credit.amount (src/loan/terms/simple/loan/PWNSimpleLoan.sol#543)
          - loan.collateral = loanTerms.collateral (src/loan/terms/simple/loan/PWNSimpleLoan.sol#544)
        PWNSimpleLoan.LOANs (src/loan/terms/simple/loan/PWNSimpleLoan.sol#164) can be used in cross function reentrancies:
        - PWNSimpleLoan._checkRefinanceLoanTerms(uint256,PWNSimpleLoan.Terms) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#485-511)
        - PWNSimpleLoan._createLoan(bytes32,address,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#520-554)
        - PWNSimpleLoan._deleteLoan(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#941-944)
        - PWNSimpleLoan._getLOANStatus(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#1143-1146)
        - PWNSimpleLoan._loanRepaymentAmount(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#803-808)
        - PWNSimpleLoan._settleLoanClaim(uint256,address,bool) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#920-935)
        - PWNSimpleLoan._settleLoanRefinance(uint256,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#601-669)
        - PWNSimpleLoan._updateRepaidLoan(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#763-777)
        - PWNSimpleLoan.claimLOAN(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#837-856)
        - PWNSimpleLoan.createLOAN(PWNSimpleLoan.ProposalSpec,PWNSimpleLoan.LenderSpec,PWNSimpleLoan.CallerSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#371-460)
        - PWNSimpleLoan.extendLOAN(PWNSimpleLoan.ExtensionProposal,bytes,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#974-1069)
```

```
INFO:Detectors:
Reentrancy in PWNSimpleLoan._createLoan(bytes32,address,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#520-554):
        External calls:
        - loanId = loanToken.mint(loanTerms.lender) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#528)
        Event emitted after the call(s):
        - LOANCreated({loanId:loanId,proposalHash:proposalHash,proposalContract:proposalContract,terms:loanTerms,lenderSpec:lenderSpec,extra:extra}) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#546-553)
Reentrancy in PWNVault._pull(MultiToken.Asset,address) (src/loan/vault/PWNVault.sol#77-84):
        External calls:
        - asset.transferAssetFrom(origin,address(this)) (src/loan/vault/PWNVault.sol#80)
        Event emitted after the call(s):
        - VaultPull(asset,origin) (src/loan/vault/PWNVault.sol#83)
Reentrancy in PWNVault._push(MultiToken.Asset,address) (src/loan/vault/PWNVault.sol#92-99):
        External calls:
        - asset.safeTransferAssetFrom(address(this),beneficiary) (src/loan/vault/PWNVault.sol#95)
        Event emitted after the call(s):
        - VaultPush(asset,beneficiary) (src/loan/vault/PWNVault.sol#98)
Reentrancy in PWNVault._pushFrom(MultiToken.Asset,address,address) (src/loan/vault/PWNVault.sol#108-115):
        External calls:
        - asset.safeTransferAssetFrom(origin,beneficiary) (src/loan/vault/PWNVault.sol#111)
        Event emitted after the call(s):
        - VaultPushFrom(asset,origin,beneficiary) (src/loan/vault/PWNVault.sol#114)
Reentrancy in PWNSimpleLoan._settleLoanClaim(uint256,address,bool) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#920-935):
        External calls:
        - _deleteLoan(loanId) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#929)
          - loanToken.burn(loanId) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#942)
        Event emitted after the call(s):
        - LOANClaimed({loanId:loanId,defaulted:defaulted}) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#931)
Reentrancy in PWNSimpleLoan._settleLoanClaim(uint256,address,bool) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#920-935):
        External calls:
        - _deleteLoan(loanId) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#929)
          - loanToken.burn(loanId) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#942)
        - _push(asset,loanOwner) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#934)
          - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#122)
```

```
INFO:Detectors:
PWNConfig.initialize(address,uint16,address)._owner (src/config/PWNConfig.sol#115) shadows:
        - Ownable._owner (lib/openzeppelin-contracts/contracts/access/Ownable.sol#21) (state variable)
IPWNDeployer.deployAndTransferOwnership(bytes32,address,bytes).owner (src/interfaces/IPWNDeployer.sol#31) shadows:
        - IPWNDeployer.owner() (src/interfaces/IPWNDeployer.sol#14) (function)
PWNLOAN.constructor(address).hub (src/loan/token/PWNLOAN.sol#54) shadows:
        - PWNHubAccessControl.hub (src/hub/PWNHubAccessControl.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in PWNSimpleLoan._createLoan(bytes32,address,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#520-554):
        External calls:
        - loanId = loanToken.mint(loanTerms.lender) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#528)
        State variables written after the call(s):
        - loan.status = 2 (src/loan/terms/simple/loan/PWNSimpleLoan.sol#532)
        - loan.creditAddress = loanTerms.credit.assetAddress (src/loan/terms/simple/loan/PWNSimpleLoan.sol#533)
        - loan.originalSourceOfFunds = lenderSpec.sourceOfFunds (src/loan/terms/simple/loan/PWNSimpleLoan.sol#534)
        - loan.startTimestamp = uint40(block.timestamp) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#535)
        - loan.defaultTimestamp = uint40(block.timestamp) + loanTerms.duration (src/loan/terms/simple/loan/PWNSimpleLoan.sol#536)
        - loan.borrower = loanTerms.borrower (src/loan/terms/simple/loan/PWNSimpleLoan.sol#537)
        - loan.originalLender = loanTerms.lender (src/loan/terms/simple/loan/PWNSimpleLoan.sol#538)
        - loan.accruingInterestDailyRate = SafeCast.toUint40(Math.mulDiv(loanTerms.accruingInterestAPR,APR_TO_DAILY_INTEREST_NUMERATOR,APR_TO_DAILY_INTEREST_DENOMINATOR)) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#539-541)
        - loan.fixedInterestAmount = loanTerms.fixedInterestAmount (src/loan/terms/simple/loan/PWNSimpleLoan.sol#542)
        - loan.principalAmount = loanTerms.credit.amount (src/loan/terms/simple/loan/PWNSimpleLoan.sol#543)
        - loan.collateral = loanTerms.collateral (src/loan/terms/simple/loan/PWNSimpleLoan.sol#544)
Reentrancy in PWNSimpleLoan._deleteLoan(uint256) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#941-944):
        External calls:
        - loanToken.burn(loanId) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#942)
        State variables written after the call(s):
        - delete LOANs[loanId] (src/loan/terms/simple/loan/PWNSimpleLoan.sol#943)
References: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in PWNSimpleLoan._createLoan(bytes32,address,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec,bytes) (src/loan/terms/simple/loan/PWNSimpleLoan.sol#520-554):
        External calls:
```

All issues identified by Slither were proved to be false positives or have been added to the issue list in this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.