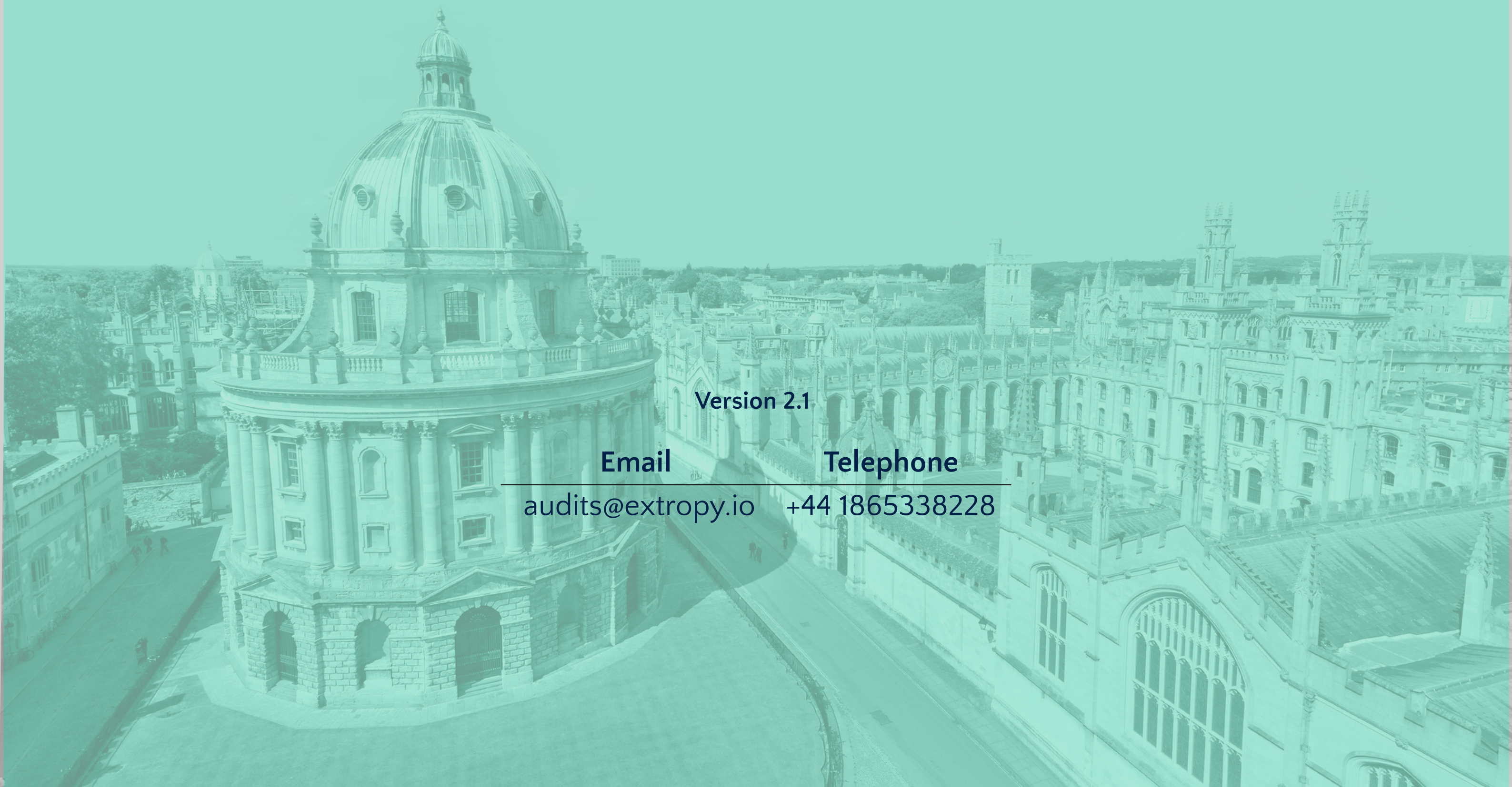
PWN Finance
Security Review Report

June 7, 2024



Version 2.1

Email	Telephone
audits@extropy.io	+44 1865338228

Data Classification	Client Confidential
Client Name	PWN Finance
Document Title	PWN Finance Security Review Report
Document Version	2.1
Author	Extropy Audit Team

Contents

1 | Executive Summary

Extropy was contracted to conduct an initial code review and vulnerability assessment of version 1.2 of the PWN protocol. Following the initial report and remediation from the client this final report is the assessment of the remediation, plus an additional update added since the initial report. Some issues have been reduced in severity after discussion with the client after the intial report.
Overall the protocol is well designed, the code is of high quality and there is evidence of extensive test coverage, using unit tests, fuzz testing and integration tests.
Section 3 details the findings, where possible we have given recommendations for their resolution.

2 | Audit summary

This audit, conducted intially from 1st May to 16th May, then from May 22nd to June 5th, employed a comprehensive approach combining manual review and automated review methods. Our examination aimed to ensure the robustness and security of the PWN Finance codebase.

- The code is taken from the pwn_contracts repository.
- The initial audit was at commit 4b28847681b582667e5791f8387713e4ab2edc2b in the branch named 'v1.2'.
- The final audit was done from commit e37fbc6d00741b9ec1bd0118a173544199829fec in branch extropy-initial-findings

2.1 | Audit scope

The following contracts were audited:

Contract	LoC
PWNLOAN.sol	137
PWNSimpleLoan.sol	1222
PWNSimpleLoanProposal.sol	350
PWNSimpleLoanDutchAuctionProposal.sol	321
PWNSimpleLoanFungibleProposal.sol	242
PWNSimpleLoanSimpleProposal.sol	186
PWNSimpleLoanListProposal.sol	225
PWNFeeCalculator.sol	30
PWNSignatureChecker.sol	84
PWNVault.sol	281
Permit.sol	32
PWNRevokedNonce.sol	209
PWNConfig.sol	262
PWNErrors.sol	28
PWNHub.sol	93
PWNHubAccessControl.sol	47
PWNHubTags.sol	15
IStateFingerpringComputer.sol	26
IPoolAdapter.sol	30
IPWNLoanMetadataProvider.sol	17
IERC5646.sol	17
IPWNDeployer.sol	41
Total	**3895**

2.2 | Issues Summary

ID	Finding	Status
3.1	[MEDIUM] Usage of rebase tokens may alter the normal functioning of the protocol	Acknowledged
3.2	[LOW] More checks are required generally for the validity of parameter inputs, including calldata inputs	Acknowledged
3.3	[LOW] Possible DoS attack makes the protocol unusable through ‘repayLOAN()‘	Acknowledged
3.4	[LOW] Borrower can avoid paying loan interests if ‘loan.fixedInterestAmount‘ is zero	Acknowledged
3.5	[LOW] ‘bytes32 tag‘ accepts any input without an input validation check	Acknowledged
3.6	[LOW] Unbounded call in ‘setTags‘ can exceed the block gas limit	Acknowledged
3.7	[LOW] Extending loans may invalidate existing nonces incorrectly when using the same nonceSpace	Acknowledged
3.8	[INFO] Refactored EIP-712 Implementation	Acknowledged
3.9	[INFO] Literal ’3’ used instead on a constant	Acknowledged
3.10	[INFO] ‘getCreditAmount()‘ implementation does not reflect the documentation provided	Resolved
3.11	[INFO] Unnecessary Inheritance of Administrative Functions in PWNLOAN Contract	Resolved
3.12	[INFO] Comparison to a boolean constant	Resolved
3.13	[INFO] Unnecessary require statement	Resolved
3.14	[INFO] Typo in ‘PWNConfig‘	Resolved

2.3 | Methodology

2.3.1 | Risk Rating

The risk rating given for issues follows the standard approach of the OWASP Foundation. We combine two factors :

- Likelihood of exploit
- Impact of Exploit

The Categories we use are *Critical*, *High*, *Medium*, *Low* and *Informational* These categories may not align with the categories used by other companies.
The informational category is used to contain suggestions for optimisation (where this is not seen as causing significant impact), or for alternative design or best practices.

2.4 | Approach

The project was assessed mainly by code inspection, with auditors working independently or together, looking for possible exploits.
Tests were written were possible to validate the issues found.
Static analysis tools were also used, and summary results are attached to this report. These tools are seen as an adjunct to code inspection.

3 | Findings

3.1 | [MEDIUM] Usage of rebase tokens may alter the normal functioning of the protocol

- **Location(s):** -
- **Description:** Protocol allows to use rebase tokens that may have bad impact on the normal flow.

  A rebase token is a type of cryptocurrency token whose total supply is dynamically adjusted based on certain criteria such as the token's price or market conditions. The goal of a rebase token is to maintain its price stability relative to a target price. When the token's price deviates from the target, the protocol automatically adjusts the token's supply through "rebases" to bring the price back in line with the target.

  It's possible to have: - Positive Rebase: If the token price goes above the desired reference price and the price stability protocol aims to increase the token price, it may execute a positive rebase. In this case, it may be necessary to "mint" new supply to increase the total token supply and raise the price. Specifically each user token balance will be increased proportionally. - Negative Rebase: If the token price goes below the desired reference price and the price stability protocol aims to decrease the token price, it may execute a negative rebase. In this case, it may be necessary to "burn" existing supply to reduce the total token supply and lower the price. Specifically each user token balance will be reduced proportionally.

  Due to positive and negative rebase the following scenarios are possible.

  Scenario A: 1. Loan is created where Alice uses 100 $stETH rebase tokens as a collateral. 2. Due to positive rebase after some time the balance of $stETH in the vault grows to 106 tokens. 3. Alice repays the loan. She expects to get 106 $stETH back. But she gets only 100 tokens. 6 tokens are stuck in the vault forever.

  Scenario B: 1. Loan is created where Alice uses 100 $stETH rebase tokens as a collateral. 2. Due to a negative rebase after some time the balance of $stETH in the vault decreases to 94 tokens. 3. Alice repays the loan. The collateral repayment fails as full amount of collateral can't be transferred to the borrower. Alice can't claim the collateral either.

  Scenario C: 1. Loan is created where Alice uses 100 $stETH rebase tokens as a collateral. 2. Second loan is created where Bob uses 100 $stETH rebase tokens as a collateral. The vault has now 200 tokens. 3. Due to a negative rebase after some time the balance of $aSTETH in the vault decreases to 188 tokens. 4. Bob repays his loan. Bob gets back its full collateral of 100 $stETH . 5. Alice repays her loan. She should get 100 $stETH tokens. But there is only 88 $stETH tokens in the vault. Alice's collateral repayment fails. Alice can't claim the collateral either. Tokens belonging to Alice has been transferred to Bob

- **Recommendation:** Consider separating vault for each loan and return the full contract balance on collateral repayment.
- **Status:** Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "We are aware of this issue but decided to keep it unresolved. We don't allow any identified rebalancing tokens to be used on the platform. Rebalancing tokens are known for poor DeFi integration and the need for wrappers/bundlers. Anyone who uses the protocol directly should be well aware of this behavior. We can "solve" the issue of negative rebalance by transferring the missing balance to the vault directly in case of an honest mistake."

3.2 | [LOW] More checks are required generally for the validity of parameter inputs, including calldata inputs

- **Location(s):** All
- **Description:** There appears to be too much trust in the parameter values, including calldata struct values, which leads to not enough checks for valid inputs e.g. proposalSpec, lenderSpec, callerSpec. For all inputs, it is best practice to take a defensive programming approach and run checks for every input to ensure that it is valid. This will reduce the risk of unexpected behaviour or errors.

  An example of unexpected behaviour is bypassing PWNHubTag modifiers. When setting a tag in PWNHubTags, it does not check whether the loan address is not address(0). This leads to bypassing the tag check in createLOAN.

```
 function setTag(address _address, bytes32 tag, bool _hasTag) public onlyOwner {
// address can be address(0)
        tags[_address][tag] = _hasTag;
        emit TagSet(_address, tag, _hasTag);
    }

 function createLOAN(
        ProposalSpec calldata proposalSpec, // relying on calldata
        LenderSpec calldata lenderSpec,
        CallerSpec calldata callerSpec,
        bytes calldata extra
    ) external returns (uint256 loanId) {
  if (
            !hub.hasTag(proposalSpec.proposalContract,
// checking calldata address which could be address(0)
PWNHubTags.LOAN_PROPOSAL)
        ) {
            revert AddressMissingHubTag({
                addr: proposalSpec.proposalContract,
                tag: PWNHubTags.LOAN_PROPOSAL
            });
        }
```

- **Recommendation:**
  - Add input validation checks for all constructor values, e.g. input is not address(0), before storing values in state
  - check for address(0) in all function params that pass in address
  - for loanId, check is valid load id
  - for strings, check that the string is not empty
  - for any protocol addresses, check that it is a valid protocol address
  - for any bytes32, check that it is valid and not empty
  - for all bytes32[], check that it is valid and not empty
  - for all uint types, check that it is valid and within range / as expected
  - for decoded bytes e.g. proposalData, check the decoded parameters
  - check the values in calldata structs for proposalSpec, lenderSpec, callerSpec, and other structs e.g. Terms, Permit etc
  - for all bools, check if true, do x else do y.

- **Status**: Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: "We will not redeploy PWNHub, so I cannot change that contract. However, I don't understand how that can bypass the access control. In the example, the call will fail if address(0) finds its way into the hub (which would be a DAO error) and is passed to the loan creation function. Can you also list more examples?"
  - `[Extropy, 29/05/2024]`: Other examples that would fall under INFO/LOW category:

1. PWNConfig.sol#182. `setLOANMetadataUri` does not check for empty string `metadataUri` before storing in mapping.

It appears from the function `loanMetadataUri` that the uri can be empty, however this can lead to undesired outcomes being stored on chain. Therefore `setLOANMetadataUri` should only be called if `metadataUri` is not empty for the loan contract address.

```
function setLOANMetadataUri(address loanContract, string memory metadataUri)
 external onlyOwner {
     if (loanContract == address(0))
         // address(0) is used as a default metadata uri.
         // Use `setDefaultLOANMetadataUri` to set default metadata uri.
         revert ZeroLoanContract();

     _loanMetadataUri[loanContract] = metadataUri;
     emit LOANMetadataUriUpdated(loanContract, metadataUri);
 }
```

2. All contract constructors. Check constructor addresses are not address(0) before storing on-chain. Example with PWNSimpleLoan.sol constructor.

```
constructor(
        address _hub,
        address _loanToken,
        address _config,
        address _revokedNonce,
        address _categoryRegistry
    ) {
        hub = PWNHub(_hub);
        loanToken = PWNLOAN(_loanToken);
        config = PWNConfig(_config);
        revokedNonce = PWNRevokedNonce(_revokedNonce);
        categoryRegistry = IMultiTokenCategoryRegistry(_categoryRegistry);
    }
```

3. ProposalBase.creditAmount - no sanitisation check for 0 amount.

ProposalBase could be signed by acceptor however there should be more checks to ensure that the proposal values are valid. For example, it is possible for `ProposalBase.creditAmount` to be 0. The credit amount is checked in `PWNSimpleLoanProposal` #312

```
if (proposal.availableCreditLimit == 0) {
            // Revoke nonce if credit limit is 0, proposal can be accepted only once
            revokedNonce.revokeNonce(proposal.proposer, proposal.nonceSpace, proposal.nonce);
        } else if (creditUsed[proposalHash] + proposal.creditAmount <=
                    proposal.availableCreditLimit) {
            // Increase used credit if credit limit is not exceeded
            creditUsed[proposalHash] += proposal.creditAmount;
        } else {
            // Revert if credit limit is exceeded
            revert AvailableCreditLimitExceeded({
                used: creditUsed[proposalHash] + proposal.creditAmount,
                limit: proposal.availableCreditLimit
            });
        }
```

4. ProposalBase.expiration - sanitisation check required for 0

PWNSimpleLoanProposal.sol #291 should also check that the expiration value is valid.

```
        if (block.timestamp >= proposal.expiration) {
            revert Expired({ current: block.timestamp, expiration: proposal.expiration });
        }
```

3.3 | [LOW] Possible DoS attack makes the protocol unusable through 'repayLOAN()'

- **Location(s):** PWNSimpleLoan.sol#708
- **Description:** Within `PWNSimpleLoan.sol` the function `repayLOAN()` is used to repay a running loan by only specifying the `loanId`. Anyone can call `repayLOAN()` and trigger the repayment process on behalf of other users. The only conditions that should be met are:
  - the loan is a running loan, namely a loan which is in the status 2;
  - `loan.defaultTimestamp` is greater than the current `block.timestamp` to ensure that the loan is not defaulted.

```
    function _checkLoanCanBeRepaid(uint8 status, uint40 defaultTimestamp) private view {
        // Check that loan exists and is not from a different loan contract
        if (status == 0)
            revert NonExistingLoan();
        // Check that loan is running
        if (status != 2)
            revert LoanNotRunning();
        // Check that loan is not defaulted
        if (defaultTimestamp <= block.timestamp)
            revert LoanDefaulted(defaultTimestamp);
    }
```

However since anyone can trigger this function without the consensus of the borrower the following scenarios are feasible: - a malicious address or bot monitors the running loans and calls the `repayLOAN()` function for every new loan started. This scenario causes the protocol to be unusable; - a malicious address or bot monitors the running loans of a specific user and calls the `repayLOAN()` function for every new loan created by that specific borrower. This scenario causes the protocol to be unusable for the targeted borrower;

- **Recommendation:** Consider ensuring that if `repayLOAN()` could be called by any user, the borrower involved in the loan agrees on loan repayment and that no one is able to trigger the repayment process for every new started loan.
- **Status:** Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "Even though it is true that anyone can call repayLOAN and pass any running loan id, the repayment amount is transferred from the caller, not the borrower. It would be very costly for an attacker to try to execute a DoS attack this way, and both lender and borrower would probably exploit such behavior immediately after spotting it. The borrower gets free money, and the lender gets free interest on his funds."
  - `[Extropy, 24/05/2024]`: "Even if the repayment amount is transferred from the caller and not the borrower, it will still be possible for an attacker or a competitor with a large amount of funds to perform such DoS attack by triggering `repayLOAN()` causing the protocol to not work as intended. "

3.4 | [LOW] Borrower can avoid paying loan interests if 'loan.fixedInterestAmount' is zero

- **Location(s):** PWNSimpleLoan.sol#792 [commit 28ef5325fc7ad87d69be3acf79bb7c5e97788e44[1]]
- **Description:** Within `PWNSimpleLoan.sol` the function `_loanAccruedInterest()` calculates the interests that the borrower will pay when closing a loan, namely when repaying it or refinancing it. The function calculates the value `accruedInterest` and returns it after adding to `loan.fixedInterestAmount`. However if `loan.fixedInterestAmount` was set to zero when creating the loan, only the `accruedInterest` will be returned and under some circumstances it can be zero, allowing the borrower to not pay any interest.

```
function _loanAccruedInterest(LOAN storage loan) private view returns (uint256) {
    if (loan.accruingInterestAPR == 0)
        return loan.fixedInterestAmount;

    uint256 accruingMinutes = (block.timestamp - loan.startTimestamp) / 1 minutes;
    uint256 accruedInterest = Math.mulDiv(
        loan.principalAmount, uint256(loan.accruingInterestAPR) * accruingMinutes,
        ACCRUING_INTEREST_APR_DENOMINATOR
    );
    return loan.fixedInterestAmount + accruedInterest;
}
```

The computed `accruedInterest` will be zero if the numerator of the calculation is lower than the denominator.
Since the denominator `ACCRUING_INTEREST_APR_DECIMALS` is a constant and its value is 5.256e12, it may happen that the numerator will be lower than that.
Suppose the scenario where:

- `loan.fixedInterestAmount` = 0
- `loan.principalAmount` = 100 USDT = 100e6 , since USDT has 6 decimals

[1]https://github.com/PWNFinance/pwn_contracts/pull/77/commits/28ef5325fc7ad87d69be3acf79bb7c5e97788e44

- `loan.accruingInterestAPR` = 1e3
- `block.timestamp` - `loan.startTimestamp` = 3000 , loan duration is 50 minutes

In such case the numerator will be `5e12`, then `accruedInterest` will be 0 and the `_loanAccruedInterest()` function will return 0.
In this context, a borrower can set up a script/bot to keep refinancing the loan each 50 minutes and not pay any interests on each. At the same time if a borrower repay a loan with above characteristics (low values for `loan.accruingInterestAPR` and `loan.principalAmount`) will be exempted from interests.

- **Recommendation:** Consider ensuring that that each time a loan is refinanced or payed back , if interests are expected, the borrower will pay for them.
- **Status:** Acknowledged.
- **Updates:**
  - `[PWN Finance, 04/06/2024]`: The client acknowledged the issue and stated: "When running the numbers, the interest in the described scenario would be less than 1 wei. We cannot do anything when the small USDC decimals and very low APR lead to accrued interest being smaller than the smallest EVM unit. "

3.5 | [LOW] 'bytes32 tag' accepts any input without an input validation check

- **Location(s):**. PWNHub.sol#48, PWNHub.sol#64
- **Description:** The functions `setTag()` and `setTags` are used to store whether a contract has a bytes32 tag, as shown in below code snippet.

```
function setTag(
        address _address,
        bytes32 tag,
        bool _hasTag
    ) public onlyOwner {
        tags[_address][tag] = _hasTag;
        emit TagSet(_address, tag, _hasTag);
    }

function setTags(
        address[] memory _addresses,
        bytes32[] memory _tags,
        bool _hasTag
    ) external onlyOwner {
        if (_addresses.length != _tags.length) revert InvalidInputData();

        uint256 length = _tags.length;
        for (uint256 i; i < length; ) {
            setTag(_addresses[i], _tags[i], _hasTag);
            unchecked {
                ++i;
            }
        }
    }
```

The functions accept any bytes32 value, which could lead to an inaccurate bytes32 tag being stored as true.
Consequently in this scenario, when creating a loan `PWNSimpleLoan.createLOAN` `hub.hasTag` may return false unintentionally. A valid loan contract could mint a LOAN token, however the modifier `onlyActiveLoan` could fail if the incorrect bytes32 tag was stored.

- **Recommendation:** Consider importing `PWNHubTags.sol` and checking whether the bytes32 input is a valid tag, before storing in the mapping.
- **Status**: Acknowledged.
- **Updates:**
- `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "That would mean that PWNHub must know all tags on deployment, which will prevent its future extensibility. It intentionally does not check the tag to allow the introduction of new tags without the PWNHub redeployment."
- `[Extropy, 29/05/2024]`: We would still recommend to create a mapping of valid bytes32 tags on chain, for example:

```
mapping(bytes32 => bool) _validTags;

function addTag(bytes32 tag) external onlyOwner {
// check tag input is not empty
_validTags(tag) = true;
// emit event
}
```

3.6 | [LOW] Unbounded call in 'setTags' can exceed the block gas limit

- **Location(s):**. PWNHub.sol#64, myFile2#87
- **Description:** The function `setTags()` is used to set the tags and status for a multiple number of addresses, as shown in below code snippet.

```
function setTags(
        address[] memory _addresses,
        bytes32[] memory _tags,
        bool _hasTag
    ) external onlyOwner {
        if (_addresses.length != _tags.length) revert InvalidInputData();

        uint256 length = _tags.length;
        for (uint256 i; i < length; ) {
            setTag(_addresses[i], _tags[i], _hasTag);
            unchecked {
                ++i;
            }
        }
    }
    }
```

The function accepts a dynamic array of addresses and a dynamic array of bytes32 tags. The gas costs for the unbounded loop in this call risks exceeding the block gas limit.

- **Recommendation:** Consider limiting the array size below the block gas limit for the chain that it will be deployed on.
- **Status**: Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "The function is callable only by the owner. Thus, this finding is acceptable to us."

3.7 | [LOW] Extending loans may invalidate existing nonces incorrectly when using the same nonceSpace

- **Location(s):**. PWNSimpleLOAN.sol#974
- **Description:** The function `extendLOAN()` revokes the nonce used for the extension denoted by three factors: the `extension.proposer`, `extension.nonceSpace` and `extension.nonce`. If the same user previously created a loan with `availableCreditLimit != 0` , meaning that after it was accepted the nonce is not revoked as it happens for loans with `availableCreditLimit == 0`, using the same nonceSpace and the same nonce, problems occur. Indeed if the same nonce in the same nonceSpace was used previously for creating a proposal with a non-zero `availableCreditLimit`, no other users can accept that proposal to increase the value of `creditUsed[proposalHash]` since `_acceptProposal()` will revert when validating nonce usability.

```
function extendLOAN(
    ExtensionProposal calldata extension,
    bytes calldata signature,
    bytes calldata permitData
) external {
    ... // other code
    revokedNonce.revokeNonce(extension.proposer, extension.nonceSpace, extension.nonce);
    ... // other code
}
```

- **Recommendation:** Consider ensuring that the nonce and the nonceSpace used by the `extension.proposer` for extending a loan were not previously used by the same user to create a loan with a non-zero `availableCreditLimit`.
- **Status:** Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "Nonces are signed off-chain, and no information about used nonces is on-chain until they are revoked. Unlike nonce spaces, nonces don't have to be strictly incremental. It's up to the user to pick a nonce randomly or through some function and decide whether to share the same nonce between loan or extension proposals. Simply not sharing them can prevent this issue. So yes, it can happen, but in that case, it's more of a feature than a bug. "
  - `[Extropy, 24/05/2024]`: Even if this is a remote possibility, it may happen that a user inputs the same nonce and nonceSpace when extending a loan that were already used for creating a loan.

3.8 | [INFO] Refactored EIP-712 Implementation

- **Location(s):**. PWNSimpleLoan.sol
- **Description:** The refactored `PWNSimpleLoan.sol` smart contract enhances the existing EIP-712 implementation to streamline structured data handling and reduce code redundancy. The introduction of a generalized hashing function (`_hashTypedData`) and the consolidation of state properties into a structured `TokenState` improve both the efficiency and security of the contract:

  Introduction of `_hashTypedData` Function:

  - **Before:** The previous version of the contract used EIP-712 in the `getExtensionHash` function, but each use case required separate, repetitive code for hashing.
  - **After:** The new `_hashTypedData` function centralizes the hashing logic, removing duplication by providing a reusable method for any structured data requiring EIP-712 hashing.
  - Benefits:
    - **Code Reusability and Maintainability:** Centralizes EIP-712 hash calculations into a single function, reducing redundancy and simplifying modifications or audits.
    - **Security Enhancement:** Consistent application of hashing logic reduces the risk of errors or omissions in data handling, increasing overall contract security.

  Structured Data for Loan States (`TokenState`):

  - **New Addition:** The `TokenState` structure aggregates several loan attributes that were previously handled individually.
  - **Benefits:**
    - **Data Organization and Integrity:** By organizing loan data into a single structure, the contract ensures that related data is processed together, enhancing data integrity and simplifying state management.
    - **Enhanced Auditability:** Structured data improves transparency and auditability of state changes, facilitating easier verification and validation using EIP-712 signatures.

  Refinement of `DOMAIN_SEPARATOR` Calculation:

  - **Consistency Improvement:** While the original contract used EIP-712, the refactoring ensures that the `DOMAIN_SEPARATOR` is consistently applied and integrated optimally for structured data signing.
  - **Efficiency and Security:** The immutable setup of the `DOMAIN_SEPARATOR` ensures it is only computed once, reducing gas costs and eliminating potential discrepancies in its calculation.

- **Recommendation**: This self-contained contract containing all suggested changes is at:

```
// SPDX-License-Identifier: GPL-3.0-only
pragma solidity 0.8.16;

import { MultiToken, IMultiTokenCategoryRegistry } from "MultiToken/MultiToken.sol";

contract PWNSimpleLoanEIP712 {

    string public constant VERSION = "1.2";
```

```
/**
 * Mapping of all LOAN data by loan id.
 */

/// @dev `keccak256("EIP712Domain(string name,string version,
    uint256 chainId,address verifyingContract)")`.
bytes32 internal constant DOMAIN_TYPEHASH =
    0x8b73c3c69bb8fe3d512ecc4cf759cc79239f7b179b0ffacaa9a75d522b39400f;

bytes32 public immutable DOMAIN_SEPARATOR = keccak256(abi.encode(
    DOMAIN_TYPEHASH,
    keccak256("PWNSimpleLoan"),
    keccak256(abi.encodePacked(VERSION)),
    block.chainid,
    address(this)
));

bytes32 private constant TOKENSTATE_TYPEHASH = keccak256(
    "TokenState(uint8 status,uint40 defaultTimestamp,uint40 accruingInterestDailyRate,
    uint256 fixedInterestAmount)"
);

bytes32 private constant EXTENSION_PROPOSAL_TYPEHASH = keccak256(
    "ExtensionProposal(uint256 loanId,address compensationAddress,uint256 compensationAmount,
    uint40 duration,uint40 expiration,address proposer,uint256 nonceSpace,uint256 nonce)"
);

struct TokenState {
    uint8 status;
    uint40 defaultTimestamp;
    uint40 accruingInterestDailyRate;
    uint256 fixedInterestAmount;
}

struct ExtensionProposal {
    uint256 loanId;
    address compensationAddress;
    uint256 compensationAmount;
    uint40 duration;
    uint40 expiration;
    address proposer;
    uint256 nonceSpace;
    uint256 nonce;
}


struct LOAN {
    uint8 status;
```

```
        address creditAddress;
        address originalSourceOfFunds;
        uint40 startTimestamp;
        uint40 defaultTimestamp;
        address borrower;
        address originalLender;
        uint40 accruingInterestDailyRate;
        uint256 fixedInterestAmount;
        uint256 principalAmount;
        MultiToken.Asset collateral;
    }
    /**
     * Mapping of all LOAN data by loan id.
     */
    mapping (uint256 => LOAN) private LOANs;

    uint256 public lastLoanId;
    mapping (uint256 => address) public loanContract;

    /**
     * @notice Get the hash of the extension struct.
     * @param extension Extension proposal struct.
     * @return Hash of the extension struct.
     */
    function getExtensionHash(ExtensionProposal calldata extension) public
    view returns (bytes32) {
        return _hashTypedData(
            keccak256(abi.encodePacked(
                EXTENSION_PROPOSAL_TYPEHASH,
                abi.encode(extension)
            ))
        );
    }

    function getStateFingerprint(uint256 tokenId) public view returns (bytes32) {
        LOAN memory loan = LOANs[tokenId];

        if (loan.status == 0)
            return bytes32(0);

        TokenState memory state = TokenState({
            status: _getLOANStatus(tokenId),
            defaultTimestamp: loan.defaultTimestamp,
            fixedInterestAmount: loan.fixedInterestAmount,
            accruingInterestDailyRate: loan.accruingInterestDailyRate
        });

        /// @dev returns `hashStruct(s : ) = keccak256(typeHash ‖ encodeData(s)) where
        /// typeHash = keccak256(encodeType(typeOf(s)))`
        return _hashTypedData(keccak256(abi.encode(
            TOKENSTATE_TYPEHASH,
            abi.encode(state)
            )
        ));
    }

    /// @dev Returns the hash of the fully encoded EIP-712 message for this domain,
    /// given `structHash`, as defined in
    /// https://eips.ethereum.org/EIPS/eip-712#definition-of-hashstruct.
    ///
    /// The hash can be used together with {ECDSA-recover} to obtain the signer of a message:
    ///
    ///     bytes32 digest = _hashTypedData(keccak256(abi.encode(
    ///         keccak256("Mail(address to,string contents)"),
    ///         mailTo,
    ///         keccak256(bytes(mailContents))
    ///     )));
    ///     address signer = ECDSA.recover(digest, signature);
    ///
    function _hashTypedData(bytes32 structHash) internal view returns (bytes32 digest) {
        // We will use `digest` to store the domain separator to save a bit of gas.
        digest = DOMAIN_SEPARATOR;
        /// @solidity memory-safe-assembly
        assembly {
            // Compute the digest.
            // <------------------ 0x18 bytes ----------------->
            // 000000000000000000000000000000000000000000000001901000000000000
            mstore(0x00, 0x1901000000000000) // Store "\x19\x01".
            mstore(0x1a, digest) // Store the domain separator.
            mstore(0x3a, structHash) // Store the struct hash.
            digest := keccak256(0x18, 0x42)
            // Restore the part of the free memory slot that was overwritten.
            mstore(0x3a, 0)
        }
    }

    /**
     * @notice Return a LOAN status associated with a loan id.
     * @param loanId Id of a loan in question.
     * @return status LOAN status.
     */
    function _getLOANStatus(uint256 loanId) private view returns (uint8) {
        LOAN storage loan = LOANs[loanId];
        return (loan.status == 2 && loan.defaultTimestamp <= block.timestamp) ? 4 : loan.status;
    }


}
```

- **Status**: Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The client acknowledged the issue and stated: "The alternative implementation looks good. The state fingerprint hash doesn't have to be ERC712 data typed and simple hash is enough. So to not make things more complicated I will keep the original implementation, but keep this on the table in case more ERC712 structs appear in the contract "

3.9 | [INFO] Literal '3' used instead on a constant

- **Location(s):** PWNSimpleLoan.sol#L767, #L847, PWNSimpleLoan.sol#L874, PWNSimpleLoan.sol#L984
- **Description:** Literal '3' used 4 times within the codebase:

```
loan.status = 3;
```

```
else if (loan.status == 3)
```

```
if (loan.status != 3)
```

```
if (loan.status == 3) // cannot extend repaid loan
```

- **Recommendation:** Define and use `constant` variable instead of using a literal to improve readability.
- **Status**: Acknowledged.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: The team acknowledged the issue and opted to make no changes.

3.10 | [INFO] 'getCreditAmount()' implementation does not reflect the documentation provided

- **Location(s):** PWNSimpleLoanDutchAuctionProposal.sol#180
- **Description:** The function `getCreditAmount()` is supposed to calculate the `creditAmount` of a specified Dutch Auction and in order to do so it performs some checks on the `proposal` struct passed as input. Specifically `proposal.auctionDuration` is checked to be greater than or equal to 1 minute (= 60) to avoid reverting with `InvalidAuctionDuration` error.

```
function getCreditAmount(Proposal memory proposal, uint256 timestamp) public
 pure returns (uint256) {
    // Check proposal
    if (proposal.auctionDuration < 1 minutes) {
        revert InvalidAuctionDuration({
            current: proposal.auctionDuration,
            limit: 1 minutes
        });
    }
```

However the check does not match the docs[2] where it's stated that auction duration `must be min 2 minutes`.

- **Recommendation:** Consider changing the mentioned check in `getCreditAmount()` to match the documentation or changing the documentation to reflect the actual code.
- **Status:** Resolved.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: Fixed documentation[3]

3.11 | [INFO] Unnecessary Inheritance of Administrative Functions in PWNLOAN Contract

- **Location(s):**. PWNLOAN.sol
- **Description:**

The PWNLOAN smart contract, currently inherits from PWNHubAccessControl. This inherited contract provides the PWNLOAN with comprehensive access control functionalities, including administrative capabilities such as setting and managing tags (roles) associated with different addresses in the PWNHub.

```
contract PWNLOAN is PWNHubAccessControl, IERC5646, ERC721 {
```

Upon review, it has been observed that PWNLOAN primarily utilizes the onlyActiveLoan modifier from PWNHubAccessControl to restrict access to its mint function. This modifier ensures that only addresses tagged with ACTIVE_LOAN can execute the minting function. However, the complete set of functionalities inherited from PWNHubAccessControl includes additional administrative methods (setTag, setTags) that are not used or needed by the PWNLOAN contract for its intended operational purposes.

- **Recommendation:**

If only the modifier is needed, then implement it directly on PWNLOAN the modifier and encode the external `hasTag()` call to pwnHub with a function selector

- **Status**: Resolved.
- **Updates:**
  - `[PWN Finance, 22/05/2024]`: Fixed in commit 1e7d561b397ea97632eddc7ed1f733a69388af97[4]

3.12 | [INFO] Comparison to a boolean constant

- **Location(s):**. MultiToken.sol#443-445, PWNHubAccessControl.sol#27, PWNHubAccessControl.sol#33
- **Description:** Within the entire codebase at specified locations, boolean values are compared to constants `false` or `true` which is unnecessary.
- **Recommendation:** Consider avoid using `== true` or `== false` when checking a boolean `var` and simply check for `var` or `!var`.
- **Status:** Resolved.

[2]https://www.notion.so/pwndao/Dutch-auction-1164a6719d27416695ff62d54e520c5e
[3]https://www.notion.so/pwndao/Dutch-auction-1164a6719d27416695ff62d54e520c5e
[4]https://github.com/PWNFinance/pwn_contracts/pull/77/commits/1e7d561b397ea97632eddc7ed1f733a69388af97

- **Updates:**

  - [PWN Finance, 22/05/2024]: Fixed MultiToken.sol in commit 59d67a5e[5]. PWNHubAccessControl.sol instead was deleted in commit 1e7d561b397ea97632eddc7ed1f733a69388af97[6]

3.13 | [INFO] Unnecessary require statement

- **Location(s):** PWNConfig.sol#119
- **Description:** Within PWNConfig.sol the function initialize() checks if the input _feeCollector is the zero address before setting it as the feeCollector address through _setFeeCollector.

```
function initialize(address _owner, uint16 _fee, address _feeCollector)
external initializer {
    ... // other code
    require(_feeCollector != address(0), "Fee collector is zero address");
    _setFeeCollector(_feeCollector);
    ... // other code
}
}
```

The check is superfluous since it's already made within _setFeeCollector().

- **Recommendation:** Consider removing the mentioned require statement from the initialize() function in order to save gas.
- **Status:** Resolved.
- **Updates:**

  - [PWN Finance, 22/05/2024]: Fixed in commit b3faccc5e400803bb6fad1b9d8fb0a40c75a44b6[7]

3.14 | [INFO] Typo in 'PWNConfig'

- **Location(s):**. PWNConfig.sol#14
- **Description:** Within PWNConfig.sol the word intendet is written instead of intended.

```
* @dev Is intendet to be used as a proxy via `TransparentUpgradeableProxy`.
*/
contract PWNConfig is Ownable2Step, Initializable { ... }
```

- **Recommendation:** Consider fixing the typo in order to improve code readability.
- **Status:** Resolved.
- **Updates:**

  - [PWN Finance, 22/05/2024]: Fixed in commit 824c5eb6aad35b3862e774417e405e41bad93dd6[8]

[5]https://github.com/PWNFinance/MultiToken/commit/59d67a5e7424d774b09a15e83bcc956cdab0b197
[6]https://github.com/PWNFinance/pwn_contracts/pull/77/commits/1e7d561b397ea97632eddc7ed1f733a69388af97
[7]https://github.com/PWNFinance/pwn_contracts/pull/77/commits/b3faccc5e400803bb6fad1b9d8fb0a40c75a44b6
[8]https://github.com/PWNFinance/pwn_contracts/pull/77/commits/824c5eb6aad35b3862e774417e405e41bad93dd6

A | Disclaimers

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purpose

A.1 | Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

A.2 | Proprietary Information

The content of this document should be considered proprietary information. Extropy gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

B | Slither results

```
INFO:Detectors: Deployments.deploymentsSubpath (src/Deployments.sol#28) is never
initialized. It is used in: - Deployments._loadDeployedAddresses()
(src/Deployments.sol#55-68) Reference:


PWNSimpleLoan._loanAccruedInterest(PWNSimpleLoan.LOAN)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#815-824) performs a
multiplication on the result of a division

INFO:Detectors: PWNSimpleLoanProposal._acceptProposal
PWNSimpleLoanProposal.sol#233-348) uses a
dangerous strict equality: - proposal.availableCreditLimit == 0
(src/loan/terms/simple/proposal/PWNSimpleLoanProposal.sol#309)

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) - PWNSimpleLoan.getLOAN
(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1108-1136) -
PWNSimpleLoan.getStateFingerprint(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1199-1217) -
PWNSimpleLoan.loanRepaymentAmount(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#789-796) -
PWNSimpleLoan.repayLOAN(uint256,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#708-739) -
PWNSimpleLoan.tryClaimRepaidLOAN(uint256,uint256,address)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#868-912) 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) - _tryPermit(permit)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#440) - IERC20Permit
(permit.asset).permit({owner:permit.owner,spender:address
(this),value:permit.amount,
deadline:permit.deadline,
v:permit.v,r:permit.r,s:permit.s})
(src/loan/vault/PWNVault.sol#182-192)
State variables written after the call(s): - _updateRepaidLoan
(callerSpec.refinancingLoanId)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#449) - loan.status = 3
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#767) - loan.fixedInterestAmount
= _loanAccruedInterest(loan)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#770) -
loan.accruingInterestDailyRate = 0
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#771) 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) - PWNSimpleLoan.getLOAN
(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1108-1136) -
PWNSimpleLoan.getStateFingerprint(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1199-1217) -
PWNSimpleLoan.loanRepaymentAmount(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#789-796) -
PWNSimpleLoan.repayLOAN(uint256,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#708-739) -
PWNSimpleLoan.tryClaimRepaidLOAN(uint256,uint256,address)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#868-912) Reentrancy in
PWNSimpleLoan.extendLOAN(PWNSimpleLoan.ExtensionProposal,bytes,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#974-1069): External calls: -
revokedNonce.revokeNonce
(extension.proposer,extension.nonceSpace,extension.nonce)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1041) State variables written
after the call(s): - loan.defaultTimestamp = originalDefaultTimestamp +
extension.duration
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1045) 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) - PWNSimpleLoan.getLOAN
(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1108-1136) -
PWNSimpleLoan.getStateFingerprint(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1199-1217) -
```

```
PWNSimpleLoan.loanRepaymentAmount(uint256)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#789-796) -
PWNSimpleLoan.repayLOAN(uint256,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#708-739) -
PWNSimpleLoan.tryClaimRepaidLOAN(uint256,uint256,address)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#868-912) Reference:
https://github.com/crytic/slither/wiki
/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors: PWNSimpleLoanProposal._acceptProposal
(address,uint256,bytes32,bytes32
[],bytes,PWNSimpleLoanProposal.ProposalBase).currentFingerprint
(src/loan/terms/simple/proposal/PWNSimpleLoanProposal.sol#325) is a local
variable never initialized
Reference:https://github.com/crytic/slither/wiki
/Detector-Documentation#uninitialized-local-variables
INFO:Detectors: 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)
↪  Reference: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: -
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
PWNVault._supplyToPool(MultiToken.Asset,IPoolAdapter,address,address)
(src/loan/vault/PWNVault.sol#143-153): External calls: -
asset.transferAssetFrom(address(this),address(poolAdapter))
(src/loan/vault/PWNVault.sol#146) - poolAdapter.supply
(pool,owner,asset.assetAddress,asset.amount)
(src/loan/vault/PWNVault.sol#147) Event emitted after the call(s): - PoolSupply
(asset,address(poolAdapter),pool,owner)
(src/loan/vault/PWNVault.sol#152) Reentrancy in PWNVault._withdrawFromPool
(MultiToken.Asset,IPoolAdapter,address,address)
(src/loan/vault/PWNVault.sol#125-132): External calls: - poolAdapter.withdraw
(pool,owner,asset.assetAddress,asset.amount)
```

```
(src/loan/vault/PWNVault.sol#128) Event emitted after the call
(s): - PoolWithdraw(asset,address(poolAdapter),pool,owner)
(src/loan/vault/PWNVault.sol#131)
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)
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)
- loanId =
_createLoan(
{proposalHash:proposalHash,
proposalContract:proposalSpec.proposalContract,
loanTerms:loanTerms,lenderSpec:lenderSpec,extra:extra})
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#428-434)

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) - _tryPermit(permit)

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#440) - IERC20Permit
(permit.asset).permit({owner:permit.owner,
spender:address
(this),value:permit.amount,
deadline:permit.deadline,v:permit.v,
r:permit.r,s:permit.s})
(src/loan/vault/PWNVault.sol#182-192)
Event emitted after the call(s): - LOANPaidBack({loanId:loanId})
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#776) - _updateRepaidLoan
(callerSpec.refinancingLoanId)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#449) Reentrancy in
PWNSimpleLoan.extendLOAN(PWNSimpleLoan.ExtensionProposal,bytes,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#974-1069): External calls: -
revokedNonce.revokeNonce
(extension.proposer,extension.nonceSpace,extension.nonce)

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1041)
Event emitted after the
call(s):
- LOANExtended(
{loanId:extension.loanId,
originalDefaultTimestamp:originalDefaultTimestamp,
extendedDefaultTimestamp:loan.defaultTimestamp})

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1048-1052)

Reference:https://github.com/crytic/slither/wiki
/Detector-Documentation#reentrancy-vulnerabilities-3

INFO:Detectors: PWNSimpleLoan._settleLoanRefinance
(uint256,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec)

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#601-669)
uses timestamp for
comparisons Dangerous comparisons: - surplus > 0

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#649)
- shortage > 0
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#653)
- newLoanAmount >
repaymentAmount(src/loan/terms/simple/loan
/PWNSimpleLoan.sol#615)
- surplus > 0
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#616)
```

```
PWNSimpleLoan._withdrawCreditFromPool

(MultiToken.Asset,PWNSimpleLoan.Terms,PWNSimpleLoan.LenderSpec)

(src/loan/terms/simple/loan/PWNSimpleLoan.sol#678-691)
uses timestamp for
comparisons Dangerous comparisons:
- credit.amount > 0
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#688)

PWNSimpleLoan._checkLoanCanBeRepaid(uint8,uint40)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#747-757) uses timestamp for
comparisons Dangerous comparisons: - defaultTimestamp <= block.timestamp
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#755) PWNSimpleLoan.claimLOAN
(uint256)(src/loan/terms/simple/loan/PWNSimpleLoan.sol#837-856) uses timestamp
for comparisons Dangerous comparisons: - loan.status == 2 &&
loan.defaultTimestamp <= block.timestamp
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#850) PWNSimpleLoan.extendLOAN
(PWNSimpleLoan.ExtensionProposal,bytes,bytes)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#974-1069) uses timestamp for
comparisons Dangerous comparisons: - block.timestamp >= extension.expiration
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#994) PWNSimpleLoan._getLOANStatus
(uint256)(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1143-1146) uses
timestamp for comparisons Dangerous comparisons: - (loan.status == 2 &&
loan.defaultTimestamp <= block.timestamp)
(src/loan/terms/simple/loan/PWNSimpleLoan.sol#1145)
PWNSimpleLoanDutchAuctionProposal.getCreditAmount
(PWNSimpleLoanDutchAuctionProposal.Proposal,uint256)
(src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#178-228)
uses timestamp for comparisons Dangerous comparisons: - timestamp <
proposal.auctionStart
(src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#199)
- proposal.auctionStart + proposal.auctionDuration + 60 <= timestamp
  (src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#205)
  PWNSimpleLoanDutchAuctionProposal.acceptProposal
  (address,uint256,bytes,bytes32[],bytes)
  (src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#233-319)
  uses timestamp for comparisons Dangerous comparisons: - creditAmount <
  proposalValues.intendedCreditAmount || proposalValues.intendedCreditAmount +
  proposalValues.slippage < creditAmount
  (src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#252-253)
- creditAmount > proposalValues.intendedCreditAmount ||
  proposalValues.intendedCreditAmount - proposalValues.slippage > creditAmount
  (src/loan/terms/simple/proposal/PWNSimpleLoanDutchAuctionProposal.sol#263-264)
  PWNSimpleLoanProposal._acceptProposal(address,uint256,bytes32,bytes32
  [],bytes,PWNSimpleLoanProposal.ProposalBase)
  (src/loan/terms/simple/proposal/PWNSimpleLoanProposal.sol#233-348) uses
  timestamp for comparisons Dangerous comparisons: - block.timestamp >=
  proposal.expiration
  (src/loan/terms/simple/proposal/PWNSimpleLoanProposal.sol#291) Reference:
  https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
  INFO:Detectors: PWNHubAccessControl.onlyActiveLoan()
  (src/hub/PWNHubAccessControl.sol#26-30) compares to a boolean
  constant: -hub.hasTag(msg.sender,PWNHubTags.ACTIVE_LOAN) == false
  (src/hub/PWNHubAccessControl.sol#27) PWNHubAccessControl.onlyWithTag
  (bytes32) (src/hub/PWNHubAccessControl.sol#32-36) compares to a boolean
  constant: -hub.hasTag(msg.sender,tag) == false
  (src/hub/PWNHubAccessControl.sol#33) Reference:
```

```
  https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
  Reference:
  https://github.com/crytic/slither/wiki/Detector-Documentation
  #different-pragma-directives-are-used

Deployments._contains(uint256[],uint256) (src/Deployments.sol#70-76) is never
used and should be removed Deployments._loadDeployedAddresses()
(src/Deployments.sol#55-68) is never used and should be removed
Deployments._protocolNotDeployedOnSelectedChain()(src/Deployments.sol#78-80) is
never used and should be removed Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation
#incorrect-versions-of-solidity

The following unused import(s) in src/Deployments.sol should be
removed: -import { PWNHubTags } from "pwn/hub/PWNHubTags.sol";
(src/Deployments.sol#13) The following unused import(s) in
src/nonce/PWNRevokedNonce.sol should be removed: -import { PWNHubTags }
from "pwn/hub/PWNHubTags.sol"; (src/nonce/PWNRevokedNonce.sol#5) Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Detectors: PWNConfig.VERSION (src/config/PWNConfig.sol#18) is never used
in PWNConfig (src/config/PWNConfig.sol#16-262) Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors: Deployments.deploymentsSubpath (src/Deployments.sol#28) should
be constant Reference:
https://github.com/crytic/slither/wiki
/Detector-Documentation
#state-variables-that-could-be-declared-constant
INFO:Slither:. analyzed (59 contracts with 95 detectors), 68 result(s) found
```

extropy