SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.22, the SlowMist security team received the Story Protocol team's security audit application for Story Protocol, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Story Protocol is making the legal system for creative Intellectual Property (IP) more efficient by turning IP "programmable" on the blockchain. That is, creating an API-like system where people or programs alike can license, remix, and monetize IP according to transparent terms set by creators themselves.

The Story protocol intellectualizes NFT by allowing creators to set their own licensing terms, recombine and derive their IP, and profit from it by setting different royalty terms. The user creates an NFT-bound account, and registrates the IP account through the registration section, the licensing module registers and sets up the appropriate licensing template contracts and licensing contracts, and the royalties module registers the royalties and shares the assets with the parent IP for multiple participants in the derivatives chain. The Dispute Module provides a means for users to raise and resolve disputes through arbitration for a wide range of NFT asset tagging behaviors.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	NFT ownership issues on different chains	Design Logic Audit	Medium	Acknowledged
N3	Missing function calling logic	Design Logic Audit	Medium	Fixed
N4	Missing zero address validation	Others	Suggestion	Fixed
N5	Missing the event records	Others	Suggestion	Fixed
N6	Redundant code	Others	Suggestion	Fixed
N7	Missing the whitelist	Others	Suggestion	Fixed
N8	Unimplemented function logic	Others	Suggestion	Acknowledged
N9	External call reminder	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/storyprotocol/protocol-core-v1

commit: 01354084010c33735a0ad88e1669c8b50197bf90

FIxed Version:

v1.1

commit: 773967a7e34bbb419018f6df848bfb7a3021473d

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

IPAccountImpl			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
<Constructor>	Public	Can Modify State	IPAccountStorage
supportsInterface	Public	-	-
token	Public	-	-
isValidSigner	External	-	-
owner	Public	-	-
_isValidSigner	Internal	-	-
executeWithSig	External	Payable	-
execute	External	Payable	-
onERC721Received	Public	-	-
onERC1155Received	Public	-	-
onERC1155BatchReceived	Public	-	-
_execute	Internal	Can Modify State	-

IPAccountStorage			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setBytes	External	Can Modify State	onlyRegisteredModule
getBytes	External	-	-
getBytes	External	-	-
setBytes32	External	Can Modify State	onlyRegisteredModule
getBytes32	External	-	-
getBytes32	External	-	-
supportsInterface	Public	-	-
_toBytes32	Internal	-	-
_toBytes32	Internal	-	-

LicenseToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
setLicensingImageUrl	External	Can Modify State	restricted
mintLicenseTokens	External	Can Modify State	onlyLicensingModule
burnLicenseTokens	External	Can Modify State	onlyLicensingModule
validateLicenseTokensForDerivative	External	-	-
totalMintedTokens	External	-	-
getLicenseTokenMetadata	External	-	-
getLicensorIpId	External	-	-

LicenseToken			
getLicenseTermsId	External	-	-
getLicenseTemplate	External	-	-
isLicenseTokenRevoked	Public	-	-
tokenURI	Public	-	-
_update	Internal	Can Modify State	-
_getLicenseTokenStorage	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

IPAssetRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IPAccountRegistry
initialize	Public	Can Modify State	initializer
register	External	Can Modify State	whenNotPaused
ipId	Public	-	-
isRegistered	External	-	-
totalSupply	External	-	-
_getNameAndUri	Internal	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted
_getIPAssetRegistryStorage	Private	-	-

IPAccountRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

IPAccountRegistry			
registerIpAccount	Public	Can Modify State	-
ipAccount	Public	-	-
getIPAccountImpl	External	-	-
_get6551AccountAddress	Internal	-	-

LicenseRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
setDefaultLicenseTerms	External	Can Modify State	restricted
registerLicenseTemplate	External	Can Modify State	restricted
setExpireTime	External	Can Modify State	onlyLicensingModule
setMintingLicenseConfigForLicense	External	Can Modify State	onlyLicensingModule
setMintingLicenseConfigForIp	External	Can Modify State	onlyLicensingModule
attachLicenseTermsToIp	External	Can Modify State	onlyLicensingModule
registerDerivativeIp	External	Can Modify State	onlyLicensingModule
verifyMintLicenseToken	External	-	-
isRegisteredLicenseTemplate	External	-	-
isDerivativeIp	External	-	-
hasDerivativeIps	External	-	-
exists	External	-	-
hasIpAttachedLicenseTerms	External	-	-
getAttachedLicenseTerms	External	-	-

LicenseRegistry			
getAttachedLicenseTermsCount	External	-	-
getDerivativeIp	External	-	-
getDerivativeIpCount	External	-	-
getParentIp	External	-	-
isParentIp	External	-	-
getParentIpCount	External	-	-
getMintingLicenseConfig	External	-	-
getExpireTime	External	-	-
isExpiredNow	External	-	-
getDefaultLicenseTerms	External	-	-
_verifyDerivativeFromParent	Internal	-	-
_isExpiredNow	Internal	-	-
_setExpirationTime	Internal	Can Modify State	-
_isDerivativeIp	Internal	-	-
_getMintingLicenseConfig	Internal	-	-
_getIpLicenseHash	Internal	-	-
_hasIpAttachedLicenseTerms	Internal	-	-
_exists	Internal	-	-
_getLicenseRegistryStorage	Internal	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

ModuleRegistry			
Function Name	Visibility	Mutability	Modifiers

ModuleRegistry			
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
registerModuleType	External	Can Modify State	restricted
removeModuleType	External	Can Modify State	restricted
registerModule	External	Can Modify State	restricted
registerModule	External	Can Modify State	restricted
removeModule	External	Can Modify State	restricted
isRegistered	External	-	-
getModule	External	-	-
getModuleType	External	-	-
getModuleTypeInterfaceId	External	-	-
_registerModule	Internal	Can Modify State	-
_getModuleRegistryStorage	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

ProtocolPausableUpgradeable			
Function Name	Visibility	Mutability	Modifiers
__ProtocolPausable_init	Public	Can Modify State	initializer
pause	External	Can Modify State	restricted
unpause	External	Can Modify State	restricted
paused	Public	-	-

ProtocolPauseAdmin			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManaged
addPausable	External	Can Modify State	restricted
removePausable	External	Can Modify State	restricted
pause	External	Can Modify State	restricted
unpause	External	Can Modify State	restricted
isAllProtocolPaused	External	-	-
isPausableRegistered	External	-	-
pausables	External	-	-

AccessController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setBatchPermissions	External	Can Modify State	-
setPermission	Public	Can Modify State	whenNotPaused
setAllPermissions	External	Can Modify State	whenNotPaused
checkPermission	External	-	-
getPermission	Public	-	-
_setPermission	Internal	Can Modify State	-
_encodePermission	Internal	-	-
_getAccessControllerStorage	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

AccessControlled			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_verifyPermission	Internal	-	-
_hasPermission	Internal	-	-

ArbitrationPolicySP			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
onRaiseDispute	External	Can Modify State	onlyDisputeModule
onDisputeJudgement	External	Can Modify State	onlyDisputeModule
onDisputeCancel	External	Can Modify State	onlyDisputeModule
onResolveDispute	External	Can Modify State	onlyDisputeModule
governanceWithdraw	External	Can Modify State	restricted
_authorizeUpgrade	Internal	Can Modify State	restricted

DisputeModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessControlled
initialize	External	Can Modify State	initializer
whitelistDisputeTag	External	Can Modify State	restricted
whitelistArbitrationPolicy	External	Can Modify State	restricted

DisputeModule			
whitelistArbitrationRelayer	External	Can Modify State	restricted
setBaseArbitrationPolicy	External	Can Modify State	restricted
setArbitrationPolicy	External	Can Modify State	verifyPermission
raiseDispute	External	Can Modify State	nonReentrant whenNotPaused
setDisputeJudgement	External	Can Modify State	nonReentrant whenNotPaused
cancelDispute	External	Can Modify State	nonReentrant
tagDerivativeIfParentInfringed	External	Can Modify State	whenNotPaused
resolveDispute	External	Can Modify State	-
isIpTagged	External	-	-
disputeCounter	External	-	-
baseArbitrationPolicy	External	-	-
disputes	External	-	-
isWhitelistedDisputeTag	External	-	-
isWhitelistedArbitrationPolicy	External	-	-
isWhitelistedArbitrationRelayer	External	-	-
arbitrationPolicies	External	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted
_getDisputeModuleStorage	Private	-	-

IpRoyaltyVault			
Function Name	Visibility	Mutability	Modifiers

IpRoyaltyVault			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
decimals	Public	-	-
addIpRoyaltyVaultTokens	External	Can Modify State	-
snapshot	External	Can Modify State	whenNotPaused
claimableRevenue	External	-	whenNotPaused
claimRevenueByTokenBatch	External	Can Modify State	nonReentrant whenNotPaused
claimRevenueBySnapshotBatch	External	Can Modify State	whenNotPaused
collectRoyaltyTokens	External	Can Modify State	nonReentrant whenNotPaused
_claimableRevenue	Internal	-	-
_collectAccruedTokens	Internal	Can Modify State	-
ipId	External	-	-
unclaimedRoyaltyTokens	External	-	-
lastSnapshotTimestamp	External	-	-
ancestorsVaultAmount	External	-	-
isCollectedByAncestor	External	-	-
claimVaultAmount	External	-	-
claimableAtSnapshot	External	-	-
unclaimedAtSnapshot	External	-	-
isClaimedAtSnapshot	External	-	-
tokens	External	-	-

IpRoyaltyVault			
_getIpRoyaltyVaultStorage	Private	-	-

RoyaltyPolicyLAP			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setSnapshotInterval	Public	Can Modify State	restricted
setIpRoyaltyVaultBeacon	Public	Can Modify State	restricted
upgradeVaults	Public	Can Modify State	restricted
onLicenseMinting	External	Can Modify State	onlyRoyaltyModule nonReentrant
onLinkToParents	External	Can Modify State	onlyRoyaltyModule nonReentrant
onRoyaltyPayment	External	Can Modify State	onlyRoyaltyModule
getRoyaltyData	External	-	-
getSnapshotInterval	External	-	-
getIpRoyaltyVaultBeacon	External	-	-
_initPolicy	Internal	Can Modify State	onlyRoyaltyModule
_getNewAncestorsData	Internal	-	-
_getExpectedOutputs	Internal	-	-
_getRoyaltyPolicyLAPStorage	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

RoyaltyModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
whitelistRoyaltyPolicy	External	Can Modify State	restricted
whitelistRoyaltyToken	External	Can Modify State	restricted
onLicenseMinting	External	Can Modify State	nonReentrant onlyLicensingModule
onLinkToParents	External	Can Modify State	nonReentrant onlyLicensingModule
payRoyaltyOnBehalf	External	Can Modify State	nonReentrant whenNotPaused
payLicenseMintingFee	External	Can Modify State	onlyLicensingModule
isWhitelistedRoyaltyPolicy	External	-	-
isWhitelistedRoyaltyToken	External	-	-
royaltyPolicies	External	-	-
supportsInterface	Public	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted
_getRoyaltyModuleStorage	Private	-	-

LicensorApprovalChecker			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessControlled
setApproval	External	Can Modify State	-
isDerivativeApproved	Public	-	-

LicensorApprovalChecker			
_setApproval	Internal	Can Modify State	verifyPermission
_getLicensorApprovalCheckerStorage	Private	-	-

BaseLicenseTemplateUpgradeable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
__BaseLicenseTemplate_init	Internal	Can Modify State	onlyInitializing
name	Public	-	-
getMetadataURI	Public	-	-
supportsInterface	Public	-	-
_getBaseLicenseTemplateUpgradeableStorage	Private	-	-

PILicenseTemplate			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	LicensorApprovalChecker
initialize	External	Can Modify State	initializer
registerLicenseTerms	External	Can Modify State	nonReentrant
exists	External	-	-
verifyMintLicenseToken	External	Can Modify State	nonReentrant
verifyRegisterDerivative	External	Can Modify State	-
verifyCompatibleLicenses	External	-	-

PILicenseTemplate			
verifyRegisterDerivativeForAllParents	External	Can Modify State	-
getRoyaltyPolicy	External	-	-
isLicenseTransferable	External	-	-
getEarlierExpireTime	External	-	-
getExpireTime	External	-	-
getLicenseTermsId	External	-	-
getLicenseTerms	External	-	-
getLicenseTermsURI	External	-	-
totalRegisteredLicenseTerms	External	-	-
supportsInterface	Public	-	-
toJson	Public	-	-
_policyCommercialTraitsToJson	Internal	-	-
_policyDerivativeTraitsToJson	Internal	-	-
_verifyCommercialUse	Internal	-	-
_verifyDerivatives	Internal	-	-
_verifyRegisterDerivative	Internal	Can Modify State	-
_verifyCompatibleLicenseTerms	Internal	-	-
_getExpireTime	Internal	-	-
_getPILicenseTemplateStorage	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

LicensingModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessControlled
initialize	Public	Can Modify State	initializer
attachLicenseTerms	External	Can Modify State	verifyPermission
mintLicenseTokens	External	Can Modify State	whenNotPaused
registerDerivative	External	Can Modify State	whenNotPaused nonReentrant verifyPermission
registerDerivativeWithLicenseTokens	External	Can Modify State	nonReentrant whenNotPaused verifyPermission
_payMintingFeeForAllParentIps	Private	Can Modify State	-
_payMintingFee	Private	Can Modify State	-
_getTotalMintingFee	Private	-	-
_verifyIpNotDisputed	Private	-	-
_authorizeUpgrade	Internal	Can Modify State	restricted

CoreMetadataViewModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateCoreMetadataModule	External	Can Modify State	-
getCoreMetadata	External	-	-
getMetadataURI	Public	-	-
getMetadataHash	Public	-	-
getRegistrationDate	Public	-	-

CoreMetadataViewModule			
getNftTokenURI	Public	-	-
getNftMetadataHash	Public	-	-
getOwner	Public	-	-
getJsonString	External	-	-
isSupported	External	-	-
supportsInterface	Public	-	-
_isEmptyString	Internal	-	-

CoreMetadataModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessControlled
updateNftTokenURI	External	Can Modify State	verifyPermission
setMetadataURI	External	Can Modify State	verifyPermission
setAll	External	Can Modify State	verifyPermission
freezeMetadata	External	Can Modify State	verifyPermission
isMetadataFrozen	External	-	-
supportsInterface	Public	-	-
_updateNftTokenURI	Internal	Can Modify State	onlyMutable
_setMetadataURI	Internal	Can Modify State	onlyMutable

TokenWithdrawalModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessControlled

TokenWithdrawalModule			
withdrawERC20	External	Can Modify State	verifyPermission
withdrawERC721	External	Can Modify State	verifyPermission
withdrawERC1155	External	Can Modify State	verifyPermission

BaseModule			
Function Name	Visibility	Mutability	Modifiers
supportsInterface	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.Since most contracts adopt the UUPS upgrade mode and the AccessManager mode, the AccessManager mode can manage and restrict calling permissions and restrict calls to contract functions through the restricted modifier. UUPS upgrade mode can upgrade the contract through the upgrader role.

Code location:

contracts/LicenseToken.sol

contracts/registries/IPAssetRegistry.sol

contracts/registries/LicenseRegistry.sol

contracts/registries/ModuleRegistry.sol

contracts/pause

contracts/modules/dispute/DisputeModule.sol

contracts/modules/licensing/LicensingModule.sol

contracts/modules/licensing/PILicenseTemplate.sol

contracts/modules/royalty/RoyaltyModule.sol

2.In the ArbitrationPolicySP contract, the governance protocol admin can call the governanceWithdraw function to withdraw all the PAYMENT_TOKEN from the contract before the whitelisted arbitration relayers call the setDisputeJudgement function, which will lead to the risk of over-privilege of the owner role.

Code location:

contracts/modules/dispute/policies/ArbitrationPolicySP.sol#99-104

```
function governanceWithdraw() external restricted {
    uint256 balance = IERC20(PAYMENT_TOKEN).balanceOf(address(this));
    IERC20(PAYMENT_TOKEN).safeTransfer(msg.sender, balance);

    emit GovernanceWithdrew(balance);
}
```

3.In the LicenseToken, the protocol admin can modify the licensing image imageUrl through the setLicensingImageUrl function.

Code location:

contracts/LicenseToken.sol#68-72

```
function setLicensingImageUrl(string calldata url) external restricted {
    LicenseTokenStorage storage $ = _getLicenseTokenStorage();
    $.imageUrl = url;
    emit BatchMetadataUpdate(1, $.totalMintedTokens);
}
```

4.In the RoyaltyPolicyLAP contract, the upgrader admin can upgrade the contract by the upgradeVaults function, which will lead to the risk of over-privilege of the owner role.

Code location:

contracts/modules/royalty/policies/RoyaltyPolicyLAP.sol#108-113

```
function upgradeVaults(address newVault) public restricted {
    // UpgradeableBeacon already checks for newImplementation.bytecode.length > 0,
    // no need to check for zero address
    RoyaltyPolicyLAPStorage storage $ = _getRoyaltyPolicyLAPStorage();
    UpgradeableBeacon($.ipRoyaltyVaultBeacon).upgradeTo(newVault);
}
```

5.In the DisputeModule contract, the whitelisted arbitration relayers can call the setDisputeJudgement to judge the IP whether dispute, and the successfulDisputesPerIp can not be called outside any function of the verification tag in the protocol, which will lead to the risk of over-privilege of the whitelisted arbitration relayers.

Code location:

contracts/modules/dispute/DisputeModule.sol#224-248

```
    function setDisputeJudgement(
        uint256 disputeId,
        bool decision,
        bytes calldata data
    ) external nonReentrant whenNotPaused {
        DisputeModuleStorage storage $ = _getDisputeModuleStorage();

        Dispute memory dispute = $.disputes[disputeId];
        ...
       IArbitrationPolicy(dispute.arbitrationPolicy).onDisputeJudgement(disputeId,
decision, data);

        emit DisputeJudgementSet(disputeId, decision, data);
    }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds. When updating a new contract, be careful to maintain compatibility with the old contract in terms of storage structure, do not reorder the state variables in the old contract, and do not insert new variables between the old ones.

Status

Acknowledged; After communicating with the project team, they stated that they want upgradeability up to the point when the protocol contracts can be ossified and governance renounces the right to upgrade. In the meantime, they will be on a decentralization path, starting with a governance multisig holding the admin keys with an emergency pauser team multisig.

Point 2 is fixed in the pull#152.

And for the point 3, the project team stated that they will keep this method. In future releases they might remove it, allow customization by users, or similar.

[N2] [Medium] NFT ownership issues on different chains

Category: Design Logic Audit

Content

When registering a new NFT IP Asset in the IPAssetRegistry contract, the internal function _getNameAndUri will be called to obtain the name, uri, and tokenid of the nft. But when the registered NFT is not the ID of this chain, its owner's ownership will be set as the 0 address in the IPAccountImpl contract, and the ownership and URI cannot be obtained or updated. The ipaccount without owner permissions(ownership is 0 address) cannot perform any operations in the protocol.

Code location:

contracts/IPAccountImpl.sol#91-95

contracts/registries/IPAssetRegistry.sol#75, 114-145

```
    function owner() public view returns (address) {
        (uint256 chainId, address contractAddress, uint256 tokenId) = token();
        if (chainId != block.chainid) return address(0);
        return IERC721(contractAddress).ownerOf(tokenId);
    }
    function register(
        uint256 chainid,
        address tokenContract,
        uint256 tokenId
    ) external whenNotPaused returns (address id) {
        id = registerIpAccount(chainid, tokenContract, tokenId);
        ...
        (string memory name, string memory uri) = _getNameAndUri(chainid,
 tokenContract, tokenId);
        ...
    }

    function _getNameAndUri(
        uint256 chainid,
        address tokenContract,
        uint256 tokenId
    ) internal view returns (string memory name, string memory uri) {
        if (chainid != block.chainid) {
            name = string.concat(chainid.toString(), ": ",
tokenContract.toHexString(), " #", tokenId.toString());
            uri = "";
            return (name, uri);
        }
        ...
    }
```

Solution

If the business scenario allows NFT registration from different chains, adding ownership verification for different chains and logic for submitting uri is recommended.

Status

Acknowledged; After communicating with the project team, they stated that they are going to use different means for cross chain interactions.

[N3] [Medium] Missing function calling logic

Category: Design Logic Audit

Content

In the LicenseRegistry contract, three functions have the onlyLicensingModule modifier which can only be called by the `LICENSING_MODULE` contract, the setExpireTime function, the setMintingLicenseConfigForLicense function, and the setMintingLicenseConfigForIp function. The setExpireTime can set the expiration time for an IP and it has an internal function `_setExpirationTime` to be called in the registerDerivativeIp function after registering a derivative IP for its parent IP. If there is no call to set the ExpirationTime and the default is 0, it means that it is permanently valid. The setMintingLicenseConfigForLicense function and the setMintingLicenseConfigForIp function set `mintingLicenseConfigs` and `mintingLicenseConfigsForIp`. When there is currently no code logic call, the license configuration returned by `_getMintingLicenseConfig` in the verifyMintLicenseToken function verification for a given license terms of the IP is both 0, which will cause the LicensingModule contract to be invalid. The `mlc` obtained by verifyMintLicenseToken during the mintLicenseTokens function is all 0, which will cause a series of impacts on subsequent _payMintingFee operations.

Code location:

contracts/registries/LicenseRegistry.sol#67-72, 115-165

```
    modifier onlyLicensingModule() {
        if (msg.sender != address(LICENSING_MODULE)) {
            revert Errors.LicenseRegistry__CallerNotLicensingModule();
        }
        _;
    }

    function setExpireTime(address ipId, uint256 expireTime) external
onlyLicensingModule {
        _setExpirationTime(ipId, expireTime);
    }

    function setMintingLicenseConfigForLicense(
        address ipId,
        address licenseTemplate,
        uint256 licenseTermsId,
        Licensing.MintingLicenseConfig calldata mintingLicenseConfig
    ) external onlyLicensingModule {
        LicenseRegistryStorage storage $ = _getLicenseRegistryStorage();
        if (!$.registeredLicenseTemplates[licenseTemplate]) {
            revert
Errors.LicenseRegistry__UnregisteredLicenseTemplate(licenseTemplate);
        }
        $.mintingLicenseConfigs[_getIpLicenseHash(ipId, licenseTemplate,
licenseTermsId)] = Licensing
            .MintingLicenseConfig({
                isSet: true,
                mintingFee: mintingLicenseConfig.mintingFee,
                mintingFeeModule: mintingLicenseConfig.mintingFeeModule,
                receiverCheckModule: mintingLicenseConfig.receiverCheckModule,
                receiverCheckData: mintingLicenseConfig.receiverCheckData
            });

        emit MintingLicenseConfigSetLicense(ipId, licenseTemplate, licenseTermsId);
    }

    function setMintingLicenseConfigForIp(
        address ipId,
        Licensing.MintingLicenseConfig calldata mintingLicenseConfig
    ) external onlyLicensingModule {
        LicenseRegistryStorage storage $ = _getLicenseRegistryStorage();
        $.mintingLicenseConfigsForIp[ipId] = Licensing.MintingLicenseConfig({
            isSet: true,
            mintingFee: mintingLicenseConfig.mintingFee,
            mintingFeeModule: mintingLicenseConfig.mintingFeeModule,
            receiverCheckModule: mintingLicenseConfig.receiverCheckModule,
            receiverCheckData: mintingLicenseConfig.receiverCheckData
        });
        emit MintingLicenseConfigSetForIP(ipId, mintingLicenseConfig);
    }
```

Solution

It is recommended to clarify the code calling logic to confirm whether the modifier used conforms to the protocol calling logic and whether the function call lacks the implementation of logic code.

Status

Fixed; Fixed in the v1.1 commit: f559e0bb4069ea5d213137fe3ed832fa9a858b81

[N4] [Suggestion] Missing zero address validation

Category: Others

Content

1.In the IPAccountRegistry contract, the contract constructor initialization operation only checks whether the ipAccountImpl contract address is 0, and does not check whether the erc6551Registry contract address is 0.

Code location:

contracts/registries/IPAccountRegistry.sol#26-31

```
        constructor(address erc6551Registry, address ipAccountImpl) {
            if (ipAccountImpl == address(0)) revert
    Errors.IPAccountRegistry_ZeroIpAccountImpl();
            IP_ACCOUNT_IMPL = ipAccountImpl;
            IP_ACCOUNT_SALT = bytes32(0);
            ERC6551_PUBLIC_REGISTRY = erc6551Registry;
        }
```

2.In the IPAccountStorage contract, the contract constructor initialization operation does not check whether the contract addresses are 0.

Code location:

contracts/IPAccountStorage.sol#36-40

```
        constructor(address ipAssetRegistry, address licenseRegistry, address
    moduleRegistry) {
        MODULE_REGISTRY = moduleRegistry;
        LICENSE_REGISTRY = licenseRegistry;
        IP_ASSET_REGISTRY = ipAssetRegistry;
    }
```

Solution

It is recommended to add zero address validation.

Status

Fixed; Fixed in the pull#158

[N5] [Suggestion] Missing the event records

Category: Others

Content

1.In the LicenseRegistry contract, the setDefaultLicenseTerms function can modify the default license template and license terms addresses for all IPs under the restricted modifier, but lacks 0 address checking and event record.

Code location:

contracts/registries/LicenseRegistry.sol#96-100

```
    function setDefaultLicenseTerms(address newLicenseTemplate, uint256
newLicenseTermsId) external restricted {
        LicenseRegistryStorage storage $ = _getLicenseRegistryStorage();
        $.defaultLicenseTemplate = newLicenseTemplate;
        $.defaultLicenseTermsId = newLicenseTermsId;
    }
```

2.In the RoyaltyPolicyLAP contract, the setSnapshotInterval and setIpRoyaltyVaultBeacon functions can modify the snapshotInterval value and ipRoyaltyVaultBeacon address for all IPs under the restricted modifier, but lacks 0 value and address checking and event record.

Code location:

contracts/modules/royalty/policies/RoyaltyPolicyLAP.sol#91-103

```
    function setSnapshotInterval(uint256 timestampInterval) public restricted {
        RoyaltyPolicyLAPStorage storage $ = _getRoyaltyPolicyLAPStorage();
        $.snapshotInterval = timestampInterval;
    }

    function setIpRoyaltyVaultBeacon(address beacon) public restricted {
        if (beacon == address(0)) revert
Errors.RoyaltyPolicyLAP__ZeroIpRoyaltyVaultBeacon();
        RoyaltyPolicyLAPStorage storage $ = _getRoyaltyPolicyLAPStorage();
        $.ipRoyaltyVaultBeacon = beacon;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed; Fixed in the pull #160, #169

[N6] [Suggestion] Redundant code

Category: Others

Content

1.In the LicenseRegistry contract, the registerDerivativeIp function will be called and can only be called by the upper-level LicensingModule contract in the registerDerivative and registerDerivativeWithLicenseTokens functions. And all the upper-level functions have the `parentIpIds.length` check, which makes the `if (parentIpIds.length == 0)` check redundant.

Code location:

contracts/registries/LicenseRegistry.sol#212-214

```
        if (parentIpIds.length == 0) {
            revert Errors.LicenseRegistry__NoParentIp();
        }
```

2.In the RoyaltyModule contract, the payLicenseMintingFee function will be called and can only be called by the upper-level LicensingModule contract in the _payMintingFee internal function and the _payMintingFee function will be called by the mintLicenseTokens and registerDerivative functions. The upper-level mintLicenseTokens and registerDerivative functions have the `_verifyIpNotDisputed` check and the `if (royaltyPolicy != address(0))` check, these checks make the `if (DISPUTE_MODULE.isIpTagged(receiverIpId))` and `if` `(licenseRoyaltyPolicy == address(0))` check redundant.

Code location:

contracts/modules/royalty/RoyaltyModule.sol#227-228

```
if (DISPUTE_MODULE.isIpTagged(receiverIpId)) revert
Errors.RoyaltyModule__IpIsTagged();
 if (licenseRoyaltyPolicy == address(0)) revert
Errors.RoyaltyModule__NoRoyaltyPolicySet();
```

3.In the RoyaltyPolicyLAP contract, the _initPolicy function will be called by the upper-level onLicenseMinting and onLinkToParents functions. All the upper-level functions have the onlyRoyaltyModule modifier and call the _initPolicy internal function. The internal function also has the onlyRoyaltyModule modifier and it will not be directly called by the RoyaltyModule contract, which makes the modifier in the _initPolicy redundant.

Code location:

contracts/modules/royalty/policies/RoyaltyPolicyLAP.sol#224

```
    function _initPolicy(
        address ipId,
        address[] memory parentIpIds,
        bytes[] memory licenseData
    ) internal onlyRoyaltyModule {
    ...
  }
```

Solution

It is recommended to clarify the business logic implementation, and if it is redundant code, it is recommended to remove it from the contract.

Status

Fixed; Fixed point 1 and point 2 in pull #172 and #173.

[N7] [Suggestion] Missing the whitelist

Category: Others

Content

In the TokenWithdrawalModule contract, it can help the ipAccount to transfer the ERC20, ERC721 and ERC1155 tokens from the ipAccount contract to the ipAccount owner, but it does not have the tokenContract whitelist to protect the ipAccount owner to from harassment of some transferred meme coins.

Code location:

contracts/modules/external/TokenWithdrawalModule.sol#38, 57, 82

Solution

It is recommended to add a tokenContract whitelist mechanism that can be added and removed to the contract to prevent users from being harassed by some meme coins.

Status

Fixed; Removed in the pull#154

[N8] [Suggestion] Unimplemented function logic

Category: Others

Content

In the DisputeModule contract, users can cancel an ongoing dispute or resolve a judged dispute through the cancelDispute and the resolveDispute functions. In these functions, custom logic is further executed by calling the onDisputeCancel and onResolveDispute functions of the arbitrationPolicy contract. However, the onDisputeCancel and onResolveDispute functions in ArbitrationPolicySP do not implement specific logic.

Code location:

contracts/modules/dispute/policies/ArbitrationPolicySP.sol#88, 95

```
    function onDisputeCancel(address caller, uint256 disputeId, bytes calldata data)
external onlyDisputeModule {}

    function onResolveDispute(address caller, uint256 disputeId, bytes calldata data)
external onlyDisputeModule {}
```

Solution

It is recommended to confirm whether the implementation of these functions meets the requirements.

Status

Acknowledged; After communcating with the project team, they stated that the hooks might be used in other

ArbitrationPolicies. The one they initially implemented does not use them. They add the comments in the pull #170.

[N9] [Information] External call reminder

Category: Others

Content

In the LicensingModule and PILicenseTemplate contracts, the mintLicenseTokens, _getTotalMintingFee, _verifyRegisterDerivative, _verifyCommercialUse ,and verifyMintLicenseToken functions will call the verify function of the external contract HookModule. External calls are not within the scope of the audit. You need to pay attention to the design logic and code security.

Code location:

contracts/modules/licensing/LicensingModule.sol#169-171, 445-450

contracts/modules/licensing/PILicenseTemplate.sol#150-152, 410, 452-454

```
if (!IHookModule(mlc.receiverCheckModule).verify(receiver, mlc.receiverCheckData)) {
                revert Errors.LicensingModule__ReceiverCheckFailed(receiver);
            }

if (!IHookModule(terms.commercializerChecker).verify(licensee,
terms.commercializerCheckerData)) {
                return false;
            }
             IMintingFeeModule(mintingLicenseConfig.mintingFeeModule).getMintingFee(
                licensorIpId,
                licenseTemplate,
                licenseTermsId,
                amount
            );

if (!IHookModule(terms.commercializerChecker).verify(licensee,
terms.commercializerCheckerData)) {
                return false;
            }

IHookModule(terms.commercializerChecker).validateConfig(terms.commercializerCheckerDat
a);
```

Solution

It is recommended to clarify whether this external call contract is credible and check the validity of the incoming resolver address and data.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002405170002	SlowMist Security Team	2024.04.22 - 2024.05.17	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 medium risks, 5 suggestions, and 1 information. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.