Story IP Derivative Agent Audit Report

Prepared by Cyfrin

Version 2.1

Lead Auditors

Immeas

JesJupyter

January 15, 2026

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

`IPDerivativeAgent` is a thin delegation wrapper around Story Protocol's `LicensingModule::registerDerivative`. Its owner maintains a whitelist of exact registrations identified by `(parentIpId, childIpId, licenseTemplate, licenseTermsId, licensee)`, where `licensee = address(0)` enables a global allowlist for that tuple.

A whitelisted caller can invoke `registerDerivativeViaAgent`, which predicts the minting fee via `predictMintingLicenseFee`, pulls the quoted ERC20 fee into the agent, temporarily approves the `RoyaltyModule` to pull that amount during registration, calls `registerDerivative` for a derivative using conservative royalty caps, then resets allowance back to zero. The agent is `Pausable`, and while paused the owner can `emergencyWithdraw` ERC20s.

The owner controls whitelist policy and emergency withdrawals, so the owner key should be a well-secured multisig with strong operational controls.

5 Audit Scope

The audit scope was limited to:

```
src/IPDerivativeAgent.sol
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Story IP Derivative Agent smart contracts provided by Story. In this period, a total of 5 issues were found.

During the audit, we identified one low-risk issue where the predicted minting fee may differ from the amount charged, which can revert registrations or leave excess tokens in the agent. We also reported three informational items and one minor gas optimization.

Summary

Project Name	Story IP Derivative Agent
Repository	story-ecosystem
Commit	92096853dbea...
Fix Commit	b5a3b1376eff...
Audit Timeline	Jan 9th - Jan 13th, 2026
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Informational	3
Gas Optimizations	1
Total Issues	5

Summary of Findings

[L-1] Actual minting fee can differ from predicted fee	Resolved
[I-1] `IPDerivativeAgent` only supports single parent IP, limiting `Multi-Parent Derivative` use cases	Acknowledged
[I-2] Fee-on-transfer ERC20 tokens not supported	Acknowledged
[I-3] `CommercializerChecker` Access Control Bypass via `IPDerivativeAgent`	Acknowledged
[G-1] `IPDerivativeAgent::constructor` `owner` check redundant	Resolved

7 Findings

7.1 Low Risk

7.1.1 Actual minting fee can differ from predicted fee

Description: `IPDerivativeAgent::registerDerivativeViaAgent` relies on `predictMintingLicenseFee(...)` to determine `tokenAmount`, transfers that amount from the caller, and approves the Royalty Module for exactly that amount. However, the fee ultimately paid during `registerDerivative(...)` may differ from the predicted value (e.g., due to hook logic or other execution-time conditions). The agent does not reconcile the predicted amount with the actual amount spent.

Impact:

- If the actual fee is higher than the predicted amount, the Royalty Module may attempt to pull more than the agent approved/holds, causing `registerDerivative(...)` to revert (DoS) despite the user providing a sufficiently high `maxMintingFee`.
- If the actual fee is lower than the predicted amount, the excess tokens remain in the agent contract with no automatic refund path, potentially leading to stranded user funds (recoverable only via privileged/admin withdrawal, if at all).

Recommended mitigation: Consider approving (and fund) `maxMintingFee`, then refund any remaining token balance to the caller after a successful registration:

```
        // Handle token payment if required
        if (currencyToken != address(0) && tokenAmount > 0) {
            IERC20 token = IERC20(currencyToken);

            // Transfer tokens from licensee to this contract
            token.safeTransferFrom(msg.sender, address(this), tokenAmount);

            // Increase allowance for RoyaltyModule to pull tokens during registerDerivative
+           token.safeIncreaseAllowance(ROYALTY_MODULE, maxMintingFee);
-           token.safeIncreaseAllowance(ROYALTY_MODULE, tokenAmount);
        }

        // ...

        // Clean up any remaining allowance for RoyaltyModule
        if (currencyToken != address(0) && tokenAmount > 0) {
            IERC20 token = IERC20(currencyToken);
            uint256 remainingAllowance = token.allowance(address(this), ROYALTY_MODULE);
            if (remainingAllowance > 0) {
+               token.safeTransfer(msg.sender, token.balanceOf(address(this)));
                token.forceApprove(ROYALTY_MODULE, 0);
            }
        }
```

Story: Fixed in PR#5

Cyfrin: Verified. Allowance is done for `maxMintingFee` together with transfer to the agent. Left over tokens are then returned after the call to the license module.

7.2 Informational

7.2.1 `IPDerivativeAgent` only supports single parent IP, limiting `Multi-Parent Derivative` use cases

Description: The `IPDerivativeAgent::registerDerivativeViaAgent` is hardcoded to support only a single parent IP by constructing fixed-length(`1`) arrays for `parentIpIds` and `licenseTermsIds`.

```
        // Prepare arrays for LicensingModule call (single parent)
        address[] memory parents = new address[](1);
        parents[0] = parentIpId;
        uint256[] memory licenseTermsIds = new uint256[](1);
        licenseTermsIds[0] = licenseTermsId;
```

However, the underlying `LicensingModule::registerDerivative` explicitly supports registering derivatives with multiple parent IPs.

```
    function registerDerivative(
        address childIpId,
        address[] calldata parentIpIds,
        uint256[] calldata licenseTermsIds,
```

According to the protocol documentation, an IP Asset can only register as a derivative once. If it has multiple parents, all parent IPs must be registered atomically in the same call, and once registered, no additional parents can be linked later.

> An IP Asset can only register as a derivative one time. If an IP Asset has multiple parents, it must register both at the same time. Once an IP Asset is a derivative, it cannot link any more parents.

Given this constraint, the agent's single-parent design is not a recoverable limitation. Instead, it permanently prevents registering any multi-parent derivative through `IPDerivativeAgent`, even though such derivatives are explicitly supported at the core protocol level.

As a result, the agent abstraction introduces an implicit and restriction that materially diverges from the capabilities and guarantees of `LicensingModule`.

This single-parent assumption also propagates to:

- Fee estimation via `predictMintingLicenseFee()`, which only accepts a single parent IP, preventing accurate fee prediction for multi-parent derivatives.
- The whitelist mechanism, which is keyed by a single `parentIpId`, making it impossible to express authorization rules for combinations of multiple parent IPs.

Impact: This limits the agent's applicability for cross-parent derivative use cases.

Recommended Mitigation:

- If multi-parent derivatives are intended to be supported via the agent, extend `IPDerivativeAgent` to accept arrays of parent IPs and license terms, and update fee prediction and whitelist logic accordingly.
- If single-parent support is an intentional design decision, explicitly document this limitation and clarify that multi-parent derivatives are not supported and must be registered directly through `LicensingModule`.

Story: Acknowledged.

7.2.2 Fee-on-transfer ERC20 tokens not supported

Description: `IPDerivativeAgent::registerDerivativeViaAgent` assumes the fee token is transferred 1:1. It pulls `tokenAmount` from the caller via `transferFrom`, then later the Royalty Module pulls the required minting fee from the agent via `transferFrom`.

For fee-on-transfer/deflationary ERC20s, the agent may receive less than `tokenAmount` on the initial transfer, causing the subsequent Royalty Module pulls to fail due to insufficient balance (and the derivative registration to revert).

Consider restricting fee tokens to non–fee-on-transfer ERC20s, or if add a parameter `amountIn`, transfer this and measure the agent's received balance delta and only proceed/approve when it covers the required fee, while refunding the delta.

Story: Acknowledged.

7.2.3 `CommercializerChecker` Access Control Bypass via `IPDerivativeAgent`

Description: When `IPDerivativeAgent::registerDerivativeViaAgent` is used, the agent contract acts as an intermediary and calls `LicensingModule::registerDerivative` on behalf of the user. As a result, the agent contract address is propagated as the caller / licensee throughout the verification flow.

In LicensingModule:

```
    if (
        !ILicenseTemplate(licenseTemplate).verifyRegisterDerivativeForAllParents(
            childIpId,
            parentIpIds,
            licenseTermsIds,
            msg.sender  // This is the agent contract address
        )
    ) {
        revert Errors.LicensingModule__LicenseNotCompatibleForDerivative(childIpId);
    }
```

In `LicensingModule::verifyRegisterDerivativeForAllParents`, the caller parameter is derived from `msg.sender`, which in this case is the agent contract. This value is then forwarded to the `commercializerChecker` hook and used as the licensee parameter during verification.

```
   // Check if the commercializerChecker allows the link
    if (terms.commercializerChecker != address(0)) {
        if (
            !IHookModule(terms.commercializerChecker).verify(
                parentIpId,
                licensee,  // This is the agent address, not the actual user
                terms.commercializerCheckerData
            )
        ) {
            return false;
        }
    }
```

Consequently, `commercializerChecker` implementations observe and validate the agent address rather than the actual end user initiating the request. Hook logic that assumes the licensee corresponds to the end user (e.g., blacklists, allowlists, or compliance checks) may therefore behave differently when invoked via the agent.

It is unclear whether this behavior is an intentional design decision or an implicit consequence of the agent abstraction.

Impact: This behavior does not represent a direct vulnerability and may be intended. The practical impact is limited to a semantic difference between agent-based and direct interactions with LicensingModule.

In the worst case, a restricted end user could register a derivative through the agent for a parent–child IP combination that is otherwise globally allowed. This does not bypass parent-level authorization, but may differ from expectations of hook implementations that assume end-user–level enforcement.

Recommended Mitigation: The relevant behavior could be explicitly documented, such as " the agent overrides the hook".

Story: Acknowledged.

7.3 Gas Optimization

7.3.1 `IPDerivativeAgent::constructor` `owner` check redundant

Description: The `IPDerivativeAgent::constructor` checks `owner == address(0)` after calling `Ownable(owner)` in the initializer list. Since OpenZeppelin's `Ownable` constructor already reverts on a zero owner, this custom check is redundant and will never be reached. Consider removing it.

Story: Fixed in PR#5

Cyfrin: Verified.