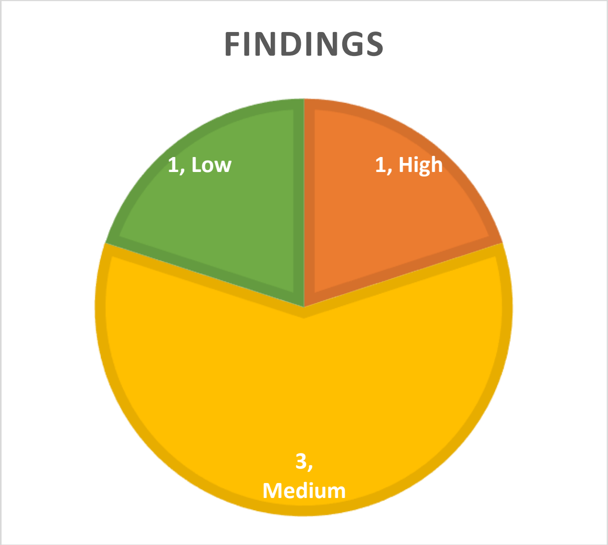
Trust Security

Smart Contract Audit

Story Protocol v1.1

09/07/2024

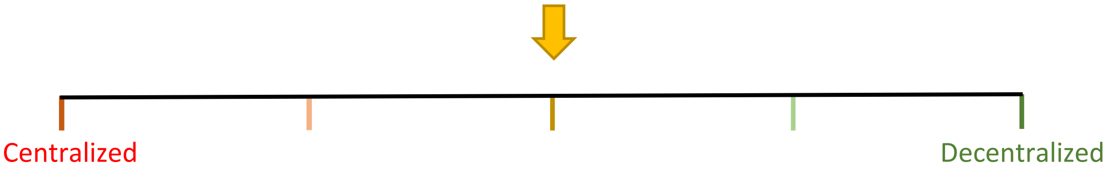
Executive summary



Category	IP management
Auditor	Bernd Artmüller, rvierdiiev
Time period	20/05-24/05

Severity	Total	Fixed	Acknowledged
High	1	1	-
Medium	3	2	1
Low	1	1	-

Centralization score



Signature

Digitally signed by Trust
Date: 2024.07.09 13:46:29 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	24.05.2024	Client report
0.2	09.07.2024	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

The scope included all changes introduced between the two commits:

- V1.0 - 888bb8b3dc0eb706715594591fbb8a5b25013c1f
- V1.1 - ba6b0e88e66f110925354f18037686a7ab44eb93

Also covered is the following pull request below:

- Consider making Upgradeable CoreMetadataModule #99
    - Commit: 773967a7e34bbb419018f6df848bfb7a3021473d

Repository details

- **Repository URL:** https://github.com/storyprotocol/protocol-core-v1
- **Mitigation review commit hash:** efd2008e259fac75d7bff5f7c495538c87eebf17

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

Bernd is a blockchain and smart contract security researcher that has made the transition from a successful full-stack web developer career. His ability to quickly grasp new concepts and technologies and his attention to detail have helped him become a top auditor in the blockchain space. Having conducted 50+ audits, Bernd has identified numerous vulnerabilities across a wide range of DeFi protocols, wallets, bridges, and VMs. He currently splits his time between audit competitions, private audits and bug bounty hunting.

rvierdiiev is a Web3 security researcher who participated in a large number of public audit contests on multiple platforms and has proven track record of experience.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	**Rating**	**Comments**
Code complexity	**Good**	The code is well modularized to reduce complexity.
Documentation	**Good**	Project is mostly very well documented.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Moderate**	Project introduces several points of centralization vectors (e.g., upgradeability, maintaining allow-lists, etc.)

Findings

High severity findings

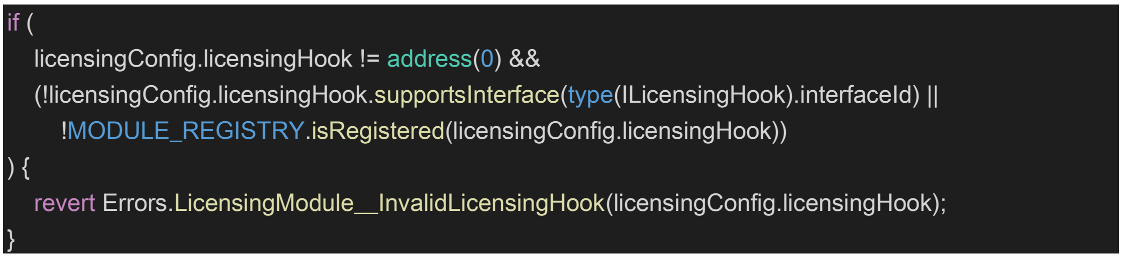
TRST-H-1 Custom ILicensingHook contracts cannot be configured as licensing hooks

- **Category:** Logical flaws
- **Source:** LicensingModule.sol
- **Status:** Fixed

Description

An IP owner or authorized user can specify a hook contract address with *LicensingModule::setLicensingConfig().* This contract's *beforeMintLicenseTokens()* and *beforeRegisterDerivative()* functions are called before minting license tokens via *mintLicenseTokens()* and before registering derivatives via *registerDerivative(), respectively.*

However, contrary to the *ILicensingHook* interface documentation, which states that custom contracts can be used as hooks, *setLicensingConfig()* asserts that the provided **licensingHook** address must support the *ILicensingHook* interface and be a registered module.



```
if (
    licensingConfig.licensingHook != address(0) &&
    (!licensingConfig.licensingHook.supportsInterface(type(ILicensingHook).interfaceId) ||
        !MODULE_REGISTRY.isRegistered(licensingConfig.licensingHook))
) {
    revert Errors.LicensingModule__InvalidLicensingHook(licensingConfig.licensingHook);
}
```

Recommended mitigation

We recommend removing the *MODULE_REGISTRY.isRegistered()* check and, instead, only validate if the **licensingHook** address supports the *ILicensingHook* interface. This ensures that the hook contract always supports the interface, regardless of whether it is a custom contract or a registered module.

Team response

This is by design, we are going to gradually open *ILicensingHook* to developers, but we want to review the first ones, hence the need for registration as a module.

Mitigation review

The team has updated the interface documentation which results in the implementation adhering to the intended behavior and thus fixing the issue.

Medium severity findings

TRST-M-1 IP owner can front-run with changing minting configs to make license buyer overpay

- **Category:** Logical flaws
- **Source:** LicenseRegistry.sol
- **Status:** Acknowledged

Description

There are three sources from which the minting price is determined, listed by priority:

1. *MintingFeeModule*,
2. *MintingLicenseConfig*, and
3. License terms

If an IP intends to override the fee configuration of a specific license, an individual minting configuration containing the minting fee can be set as the **mintingLicenseConfig**. Alternatively, the configured license terms minting fee is used.

However, the IP owner is able to front-run license-buying users by changing the minting license configuration to the buyers' disadvantage, such as increasing the minting fee.

Recommended mitigation

We recommend adding an additional **maxPayment** parameter to *LicensingModule::mintLicenseTokens()*, allowing a license token minter to specify a maximum payment amount.

Team response

Acknowledged. Since it would have a significant downstream impact due to interface change (SDK, websites, etc.), we will track the issue internally and tackle it in subsequent development iterations.

TRST-M-2 Outdated EXECUTE type hash breaks EIP-712 compatibility and signature verification

- **Category:** Logical flaws
- **Source:** MetaTx.sol
- **Status:** Fixed

Description

In v1.1, the **Execute.nonce** type is changed from uint256 to bytes32. But the execute type hash, **EXECUTE**, is not updated and is still using uint256 for the **nonce**.

```
/// @dev Execute type hash.
bytes32 public constant EXECUTE =
    keccak256("Execute(address to,uint256 value,bytes data,uint256 nonce,uint256 deadline)");
```

As a result, the execute type hash is incorrectly calculated, breaking EIP-712 compatibility and potentially causing the signature verification in *IPAccountImpl::executeWithSig()* to fail.

Recommended mitigation

We recommend updating the **EXECUTE** type hash to use bytes32 for the nonce.

Team response

Fixed.

Mitigation review

The issue has been fixed by changing the type for the nonce from uint256 to bytes32.

TRST-M-3 Inability to reset an IP's overridden license minting fee to use the default fee defined in the license terms again

- **Category:** Logical flaws
- **Source:** LicensingModule.sol
- **Status:** Fixed

Description

An IP owner or authorized user can specify a license minting configuration with *LicensingModule::setLicensingConfig()*, containing a license token minting fee and a hook contract address. This minting fee, or the fee returned by the optionally called hook function, will take precedence over the fee defined in the license terms.

However, once an IP's minting configuration is set, resetting it is impossible, so the default behavior of using the license term's minting fee is restored again. This is because the **isSet** flag, which is used to determine which minting fee to use, remains **true** after the configuration is set and cannot be set back to **false**.

Recommended mitigation

We recommend allowing the minting configuration to be reset to the default behavior by setting **isSet** appropriately.

Team response

Fixed.

Mitigation review

The issue has been fixed by updating *setLicensingConfig()* to allow the IP owner to reset the **LicenseConfig** by setting the **isSet** flag to **false**.

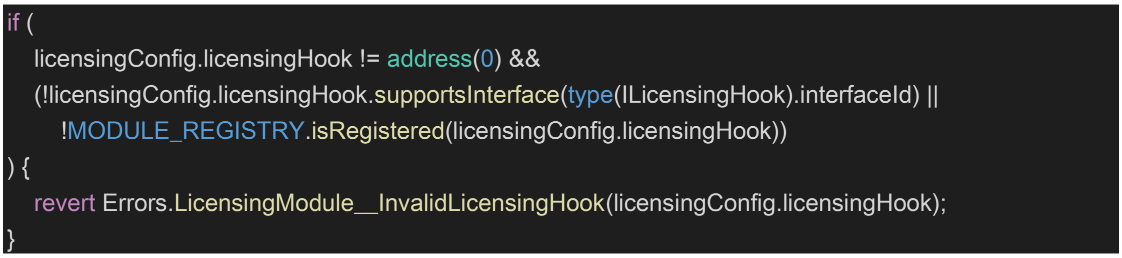
Low severity findings

TRST-L-1 User can set undesired module as hook when contract is paused for maintenance

- **Category:** Logical flaws
- **Source:** LicensingModule.sol
- **Status:** Fixed

Description

In v1.1, *setLicensingConfig()* allows a user to register the licensing configuration for a given license template and terms ID. As part of this configuration, the user can provide an optional **licensingHook** address**.** The function ensures that the provided address supports the **ILicensingHook** interface and that it is a registered module.



```
if (
    licensingConfig.licensingHook != address(0) &&
    (!licensingConfig.licensingHook.supportsInterface(type(ILicensingHook).interfaceId) ||
        !MODULE_REGISTRY.isRegistered(licensingConfig.licensingHook))
) {
    revert Errors.LicensingModule__InvalidLicensingHook(licensingConfig.licensingHook);
}
```

It is a possible scenario that after some time, the Story protocol team discontinues support for a registered module and thus removes it from the registry. This is likely accompanied by pausing the protocol. But because *setLicensingConfig()* does not have the **whenNotPaused** modifier it is possible to set the module that is going to be removed as a valid hook.

Recommended mitigation

We recommend adding the **whenNotPaused** modifier to the *setLicensingConfig()* function.

Team response

Fixed.

Mitigation review

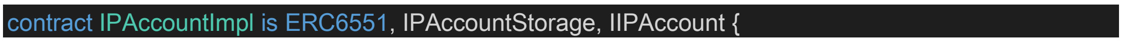
The issue has been fixed by adding the **whenNotPaused** modifier to *setLicensingConfig()*.

TRST-L-2 IERC6551Executable interface is not checked in IPAccountImpl::supportsInterface()

- **Category:** Validation flaws
- **Source:** IPAccountImpl.sol
- **Status:** Fixed

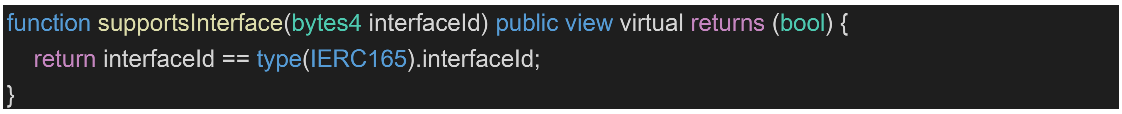
Description

The *IPAccountImpl* contract implements the EIP-165 standard and overrides *supportsInterface()* to include additional interface checks. However, the *IERC6551Executable* interface check is missing because the ERC6551 parent contract’s *supportsInterface()* is not called. This is caused by the order of inherited contracts, which lists *IPAccountStorage* after *ERC6551*.



```
contract IPAccountImpl is ERC6551, IPAccountStorage, IIPAccount {
```

As *IPAccountStorage* implements *ERC165*, *ERC165:: supportsInterface() takes precedence. Due to ERC165:: supportsInterface() not calling super.supportsInterface(), the call chain stops.*



```
function supportsInterface(bytes4 interfaceId) public view virtual returns (bool) {
    return interfaceId == type(IERC165).interfaceId;
}
```

Consequently, *ERC6551:: supportsInterface()* is not called, preventing the *ERC6551Executable* interface from being checked.

Recommended mitigation

We recommend adding the *IERC6551Executable* interface check to *IPAccountImpl:: supportsInterface()*.

Team response

Fixed.

Mitigation review

The issue has been fixed by adding the missing *IERC6551Executable* interface check.

Additional recommendations

TRST-R-1 Misleading ILicensingHook::beforeRegisterDerivative() NatSpec documentation

The NatSpec documentation for *ILicensingHook::beforeRegisterDerivative()* states, "This function is called when the LicensingModule mints license tokens." while this function is actually called before registering a derivative with *LicensingModule::registerDerivative()*. We recommend updating the documentation to avoid confusion.

Team response

Fixed.

Mitigation review

The NatSpec documentation has been updated to better reflect its behavior.