// Security Assessment

03.20.2025 - 03.20.2025

Proof-of-Creativity Protocol - Periphery Contracts - setTokenURI function

Story Protocol

HALBORN

Proof-of-Creativity Protocol - Periphery Contracts - setTokenURI function - Story Protocol

Prepared by: HALBORN

Last Updated 03/27/2025

Date of Engagement: March 20th, 2025 - March 20th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	0	1	0

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Caveats
5. Risk methodology
6. Scope
7. Assessment summary & findings overview
8. Findings & Tech Details
    8.1 Potential cross-site scripting(xss) attack
9. Automated Testing

1. Introduction

Story engaged Halborn to conduct a security assessment on the new functionality that they have added to the SPGNFT.sol contract. The new functionality allows owners to update their token URIs via the setTokenURI() function. The security assessment is beginning on March 20th, 2025 and ending on March 20th, 2025. The security assessment was scoped to the PR request provided to Halborn. Further details can be found in the Scope section of this report.

2. ASSESSMENT SUMMARY

Halborn was provided 1 day for the engagement and assigned 1 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 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 did not identify any significant improvement.

3. TEST APPROACH AND METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, 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 smart contracts and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Local testing with custom scripts (Foundry).
- Fork testing against main networks (Foundry).
- Static analysis of security for scoped contract, and imported functions(Slither).

4. CAVEATS

The current security assessment was limited to the changes in the following PR: https://github.com/storyprotocol/protocol-periphery-v1/pull/189.

5. 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.

5.1 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:

$$E = \prod m_e$$

5.2 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:

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

5.3 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

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
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
Informational	0 - 1.9

6. SCOPE

FILES AND REPOSITORY

(a) Repository: protocol-periphery-v1

(b) Assessed Commit ID: 2f14a63

(c) Items in scope:

- contracts/SPGNFT.sol
- contracts/interfaces/ISPGNFT.sol
- contracts/lib/Errors.sol

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

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

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

INFORMATIONAL
0

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POTENTIAL CROSS-SITE SCRIPTING(XSS) ATTACK	LOW	NOT APPLICABLE

8. FINDINGS & TECH DETAILS

8.1 POTENTIAL CROSS-SITE SCRIPTING(XSS) ATTACK

// LOW

Description

The `setTokenURI()` function allows a malicious user to set an arbitrary URI to be returned when the `tokenURI()` function is called. The URI can be completely arbitrary if the `_baseURI` is not set, as can be seen from the below code snippet:

```
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    ERC721URIStorageStorage storage $ = _getERC721URIStorageStorage();
    _requireOwned(tokenId);

    string memory _tokenURI = $._tokenURIs[tokenId];
    string memory base = _baseURI();

    // If there is no base URI, return the token URI.
    if (bytes(base).length == 0) {
        return _tokenURI;
    }
    // If both are set, concatenate the baseURI and tokenURI (via string.concat).
    if (bytes(_tokenURI).length > 0) {
        return string.concat(base, _tokenURI);
    }

    return super.tokenURI(tokenId);
}
```

Allowing owners to change their `tokenURI` has a couple of security implications:
1. An attacker can create an NFT that mimics another more expensive NFT depending on the different traits of an NFT.
2. An attacker can trigger Cross-Site Scripting(XSS) attack if the website that displays the NFT doesn't properly sanitize the data.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (3.4)

Recommendation

Consider whether allowing owners to set the token URI of their NFTs is really required. Consider allowing owners to set the URI only if it follows a certain schema, for example: the domain is pre-set, and the JSON files that are created by users undergo a verification process. Additionally, make sure the `_baseURI` can't be an empty string.

Remediation Comment

NOT APPLICABLE: The **Story protocol team** has deemed that the issues is not applicable as XSS attack vectors should be handled on the front end, and the NFTs represent ownership of an ERC-

6551 token-bound accounts, whose value is not based on visual traits like typical NFTs.

References

storyprotocol/protocol-periphery-v1/contracts/SPGNFT.sol#L218

9. AUTOMATED TESTING

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.

```
Constant SPGNFT.SPGNFTStorageLocation (src/SPGNFT.sol#44) is not in UPPER_CASE_WITH_UNDERSCORES
Variable SPGNFT.DERIVATIVE_WORKFLOWS_ADDRESS (src/SPGNFT.sol#48) is not in mixedCase
Variable SPGNFT.GROUPING_WORKFLOWS_ADDRESS (src/SPGNFT.sol#52) is not in mixedCase
Variable SPGNFT.LICENSE_ATTACHMENT_WORKFLOWS_ADDRESS (src/SPGNFT.sol#56) is not in mixedCase
Variable SPGNFT.REGISTRATION_WORKFLOWS_ADDRESS (src/SPGNFT.sol#60) is not in mixedCase
Variable SPGNFT.ROYALTY_TOKEN_DISTRIBUTION_WORKFLOWS_ADDRESS (src/SPGNFT.sol#64) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:. analyzed (29 contracts with 100 detectors), 54 result(s) found
```

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.