// Security Assessment

03.24.2025 - 03.25.2025

ERC20 For SPL

Neon Labs

HALBORN

ERC20 For SPL - Neon Labs

Prepared by: HALBORN

Last Updated 04/04/2025

Date of Engagement: March 24th, 2025 - March 25th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

TABLE OF CONTENTS

8. Automated Testing

1. INTRODUCTION

`Neon Labs` engaged Halborn to conduct a security assessment on their smart contracts beginning on March 24th, 2025 and ending on March 25th, 2025. The security assessment was scoped to the smart contracts provided to the Halborn team.

2. ASSESSMENT SUMMARY

The team at Halborn was provided 2 days for the engagement and assigned a security engineer to evaluate the security of the smart contract.

The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn did not identify major issues during the assessment.

3. TEST APPROACH AND METHODOLOGY

`Halborn` performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract 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 the 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.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).
- Local or public testnet deployment (`Foundry` , `Remix IDE`).ontent goes here.

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
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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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

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

SEVERITY	SCORE VALUE RANGE
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY
(a) Repository: neon-contracts (b) Assessed Commit ID: d96bd1d (c) Items in scope: • erc20forspl_factory.sol • erc20forspl.sol
Out-of-Scope: Third party dependencies and economic attacks. All code modifications not directly related to the issues included in this report. (e.g., new features)
Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING CONTEXT CONTRACT IMPLEMENTATION	INFORMATIONAL	ACKNOWLEDGED - 04/03/2025

7. FINDINGS & TECH DETAILS

7.1 MISSING CONTEXT CONTRACT IMPLEMENTATION

// INFORMATIONAL

Description

The contract does not inherit from OpenZeppelin's `Context` contract, which is standard practice for ERC20 implementations.

```
contract ERC20ForSplBackbone {

// ... //

contract ERC20ForSplMintable is ERC20ForSplBackbone, Ownable {

// ... //

contract ERC20ForSpl is ERC20ForSplBackbone {
```

The absence of `Context` means the contract doesn't support meta-transactions (transactions that can be signed by users but submitted by others who pay the gas). This limits interoperability with certain protocols and tools in the Ethereum ecosystem that rely on ERC2771 and similar meta-transaction standards.

BVSS

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 to consider inheriting from `Context` and replacing `msg.sender` with `_msgSender()` throughout the contract.

Remediation Comment

ACKNOWLEDGED: The **Neon Labs team** acknowledged this issue.

8. AUTOMATED TESTING

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 contracts. 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.

```
➜  neon-contracts git:(update/erc20forspl-factory) ✗ slither . --include-paths contracts/token/ERC20ForSpl/erc20_for_spl.sol --include-paths contracts/token/ERC20ForSpl/erc20_for_spl_factory.sol --exclude-informational --exclude-low
'npx hardhat clean' running (wd: /Users/liliancariou/Desktop/Halborn/audits/neon-contracts)
'npx hardhat clean --global' running (wd: /Users/liliancariou/Desktop/Halborn/audits/neon-contracts)
'npx hardhat compile --force' running (wd: /Users/liliancariou/Desktop/Halborn/audits/neon-contracts)
INFO:Slither:. analyzed (35 contracts with 63 detectors), 0 result(s) found
➜  neon-contracts git:(update/erc20forspl-factory) ✗
```

All issues identified by `Slither` were proved to be false positives or have been added to the issue list in this report.

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.