Rhinestone Nexus (2025-07) Security Audit

: Rhinestone Nexus

July 30, 2025

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Beginning on June 25, 2025, ChainLight conducted a 2-day security audit of the Rhinestone Smart Contract. The audit focused on identifying potential issues in the initialization process of the ERC-7702-compatible Smart Account, including signature replay threats.

Summary of Findings

The audit revealed a total of **1** issue, categorized by severity as follows:

- **High:** 1 issue

Audit Overview

Scope

Name	Rhinestone Nexus (2025-07) Security Audit
Target / Version	<ul><li>Git Repository (`https://github.com/rhinestonewtf/nexus`)<ul><li>PR1 (https://github.com/rhinestonewtf/nexus/pull/1): `0b7a7c0f87677ed2abd0d4d3910290c59423c496`</li></ul></li></ul>
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**1**	• `Nexus-001`
Medium	**0**	• N/A
Low	**0**	• N/A
Informational	**0**	• N/A
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`Nexus-001`	Signature Replay Vulnerability in `Nexus.initializeAccount()`	High	Patched

#1 `Nexus-001` Signature Replay Vulnerability in `Nexus.initializeAccount()`

ID	Summary	Severity
`Nexus-001`	`Nexus.initializeAccount()` is vulnerable to signature replay attacks, allowing a malicious relayer to reuse a valid EOA-signed initialization payload in unintended contexts	**High**

Description

In `initializeAccount()`, a relayer may submit an initialization payload signed by an EOA on the EOA's behalf. Since the signature is not bound to the specific Nexus implementation contract address, replay attacks become possible.

```
    function initializeAccount(bytes calldata initData) external payable v
irtual {
        if (msg.sender != address(this)) {
            if (_amIERC7702()) {
                bytes calldata signature = initData[0:65];
                AccountStorage storage $accountStorage = _getAccountStorag
e();
                // Remove the signature  from the initData
                initData = initData[65:];
                // Calculate the hash of the initData
                bytes32 initDataHash = initData.hash(_IMPLEMENTATION);
                // Calculate the digest (excluding chainId as it's implici
tly checked)
                initDataHash = _hashTypedDataSansChainId(initDataHash);
                // Make sure the initHash is not already used
                require(!$accountStorage.erc7702InitHashes[initDataHash],
AccountAlreadyInitialized());
                // Check if the signature is valid
                require(ECDSA.recover(initDataHash, signature) == address(
this), InvalidSignature());
                // Mark the initDataHash as used
                $accountStorage.erc7702InitHashes[initDataHash] = true;
            } else {
                Initializable.requireInitializable();
            }
        }
        _initializeAccount(initData);
    }
```

Because the signature is not bound to the specific Nexus implementation contract address, it may be replayed in contexts the EOA never intended. For example, data an EOA previously signed for an unrelated purpose outside of Nexus, such as a transaction or message from another dApp, could be submitted to `initializeAccount()`. In such a case, `_initializeAccount()` would parse the bootstrap address from the signed data and perform a delegateCall, enabling arbitrary code execution defined by that bootstrap instead of the expected initialization logic.

Additionally, if the EOA later changes its delegation address to another SmartAccount, an attacker could reuse an old signature. This risk also exists if the EOA's new delegation points to a Nexus fork deployed by a third-party service with a modified `_STORAGE_LOCATION` in `Storage.sol`. In that scenario, the replayed signature could still pass the `require(!$accountStorage.erc7702InitHashes[initDataHash])` check, enabling unauthorized execution without the EOA's consent.

Impact

High

An attacker with access to a previously signed initialization payload could execute arbitrary code during the victim's SmartAccount initialization without the EOA's consent, potentially performing other unauthorized actions.

Recommendation

Implement ERC-7739 signature binding so that the signed payload explicitly references the Nexus implementation contract address in the `smartAccount` field. This ensures signatures are cryptographically tied to the intended contract and cannot be replayed in different deployments or altered storage configurations.

Remediation

Patched

The signature data is now bound to the Nexus contract address to prevent replay attacks. By design, the implementation introduces a `chainIdIndex` parameter. When `chainIdIndex` is set to `0`, signatures are intentionally allowed to be reused across multiple chains.

Revision History

Version	Date	Description
1.0	July 30, 2025	Initial version

ChainLight