Rhinestone WebAuthnValidator (2025-07) Security Audit

: Rhinestone WebAuthnValidator

July 30, 2025

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Beginning on June 17, 2025, ChainLight conducted a 3-day security audit of the Rhinestone Smart Contract. The audit identified and assessed issues related to WebAuthn credential management, including the uniqueness of credential IDs, sorting order integrity, and potential signature overcounting due to duplicated keys with varying requireUV flags.

Summary of Findings

The audit revealed a total of **4** issues, categorized by severity as follows:

- **High:** 1 issue
- **Low:** 2 issues
- **Informational:** 1 issue

Audit Overview

Scope

Name	Rhinestone WebAuthnValidator (2025-07) Security Audit
Target / Version	• Git Repository (`https://github.com/rhinestonewtf/core-modules/blob/main/src/WebAuthnValidator/WebAuthnValidator.sol`): commit `3bee0973e7bcda0627a95c32ed4dfe972649d0cb`
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	• `WebAuthnValidator-001`
Medium	0	• N/A
Low	2	• `WebAuthnValidator-002` • `WebAuthnValidator-003`
Informational	1	• `WebAuthnValidator-004`
Note	0	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	`WebAuthnValidator-001`	Improper credential binding in `validateSignatureWithData()` enables signature replay attacks	High	Patched
2	`WebAuthnValidator-002`	Mismatch risk in sorted `credentialIds` may break signature verification	Low	Patched
3	`WebAuthnValidator-003`	CredentialID duplication due to `requireUV` may cause redundant signature counting	Low	Patched
4	`WebAuthnValidator-004`	Minor Suggestions	Informational	Patched

#1 `WebAuthnValidator-001` Improper credential binding in `validateSignatureWithData()` enables signature replay attacks

ID	Summary	Severity
`WebAuthnValidator-001`	The `validateSignatureWithData()` function is vulnerable to signature replay attacks due to insufficient linkage verification between credential IDs and their corresponding public key data.	High

Description

The `validateSignatureWithData()` function attempts to prevent duplicate credential usage by applying `uniquifySorted()` to `context.credentialIds`. However, this does not guarantee that each `credentialId` is uniquely and verifiably associated with its corresponding (pubKeyX, pubKeyY, requireUV) in `context.credentialData`. As a result, it is possible to populate `credentialData` with repeated (pubKeyX, pubKeyY) pairs, allowing a single valid signature to be counted multiple times toward the threshold. This effectively bypasses multi-signature requirements. This opens up a replay vector where a valid signature on a single key can be reused across multiple slots, undermining the intended security of the authentication logic.

Impact

High

A malicious actor could reuse a single valid credential to satisfy multiple threshold requirements by duplicating its public key data in `WebAuthnAuth`. This compromises the integrity of multi-factor authentication and allows unauthorized access with fewer actual valid credentials than required.

Recommendation

It is recommended to include the account address in the data and compute `generateCredentialId(pubkeyX, pubkeyY, requireUV, account)` for each entry in `context.credentialData`. Then, verify that the resulting ID matches the corresponding value in `credentialIds[i]`.

Remediation

Patched

The issue has been resolved as recommended.

#2 `WebAuthnValidator-002` Mismatch risk in sorted `credentialIds` may break signature verification

ID	Summary	Severity
`WebAuthnValidator-002`	Sorting of `credentialIds` during verification may break the expected correspondence between credential IDs and their associated authentication data, leading to signature verification failures.	Low

Description

Both `_validateSignatureWithConfig()` and `validateSignatureWithData()` use `sort()` and `uniquifySorted()` on the `credentialIds` array to remove duplicates. However, if the input `credentialIds` array is not already sorted and unique, sorting it in-place alters the original order.

Since WebAuthn signature verification in `_verifyWebAuthnSignatures()` depends on the alignment between `credentialIds[i]` and `credentialData[i]`, any modification in the array order may cause a mismatch between an ID and its corresponding `(pubKeyX, pubKeyY)` pair. This results in legitimate signatures being rejected.

Impact

Low

If the caller provides a valid set of `credentialIds` and `credentialData` in an unsorted order, in-place sorting may break the alignment between them. As a result, correct signatures that match the original input order may fail verification after sorting.

Recommendation

Implement one of the following recommendations:

1. Before calling `credentialIds.uniquifySorted()`, store the original array. After deduplication, compare lengths of the original and uniquified arrays. If no duplicates were removed, use the original (unsorted) array to preserve positional alignment.

2. Enforce that the credentialIds included in the data are sorted and unique by calling `require(credentialIds.isSortedAndUniquified(), "...")`.

Remediation

Patched

The issue has been addressed by applying Option 2 as recommended.

#3 `WebAuthnValidator-003` CredentialID duplication due to `requireUV` may cause redundant signature counting

ID	Summary	Severity
`WebAuthnValidator-003`	Credential IDs that differ only by the `requireUV` flag but share the same public key may lead to the same signature being counted twice during verification.	Low

Description

Currently, credential IDs are generated using a combination of `(pubKeyX, pubKeyY, requireUV)`. However, it is possible, especially due to UI confusion or manual input errors, for a smart account owner to register two credentials with the same public key but different `requireUV` values (`true` and `false`).

Since `WebAuth.verify()` does not check or differentiate based on `requireUV`, a single signature may satisfy both credentials. This results in the same signature being counted twice toward the `validCount`, which could inadvertently help meet the signature threshold.

This does not directly enable an unauthorized signature but can introduce inconsistency in how signatures are counted and may reduce the accuracy of multi-signature enforcement.

Impact

Low

The risk originates primarily from user-side misconfiguration. Although the impact is limited to a maximum of two counts per signature, this could still affect systems with tight threshold settings where accurate accounting is important.

Recommendation

1. Remove the `requireUV` flag from the credential ID derivation logic to ensure that credential IDs are uniquely tied only to the public key.
2. Optionally, introduce a function that allows users to update the `requireUV` flag for an existing credential, avoiding the need to register duplicate credentials with the same public key.

Remediation

Patched

The issue has been resolved as recommended.

#4 `WebAuthnValidator-004` Minor Suggestions

ID	Summary	Severity
`WebAuthnValidator-004`	The description includes multiple suggestions to improve precision, prevent unexpected pool creation, clarify comments, and enhance token distinguishability.	Informational

Description

1. In `validateSignatureWithData()`, it is necessary to verify that `credentialIds.length` does not exceed `MAX_CREDENTIALS`.
2. The `credentialIds` variable in `onInstall()` is not used. It is recommended to remove it.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

It has been patched as recommended.

Revision History

Version	Date	Description
1.0	July 30, 2025	Initial version

ChainLight