draft-spaghetti-sidrops-rpki-crl-numbers

SIDROPS, IETF 120, Vancouver

Job Snijders job@fasty.com
Theo Buehler tb@openbsd.org
Ben Maddison benm@workonline.africa

Background

- ATHENE researchers flagged RFC compliance issues related to CRL Number Extension handling in various RP implementations
- Tom Harrison expressed concern about the CRL Number field maxing out
- RFC 6487 mandates that CAs **MUST** include CRL Number extension in every CRL they issue

Prompting investigation: ***what are CRL Numbers even used for (in the RPKI)?***

Recap: what a RPKI Certificate Revocation List looks like

- Defined as X.509 v2 CRL (RFC 5280 Section 5)
- RPKI-specific requirements for CRLs specified in RFC 6487 section 5

```
$ rpki-client -f yqgF26w2R0m5sRVZCrbvD5cM29g.crl
File:                    yqgF26w2R0m5sRVZCrbvD5cM29g.crl
Hash identifier:         Y52MZeUcBdsxx0nhiP7LE53+vgBEf3k9YdqE3+69IWo=
Authority key identifier: CA:A8:05:DB:AC:36:47:49:B9:B1:15:59:0A:B6:EF:0F:97:0C:DB:D8
CRL issuer:              /CN=caa805dbac364749b9b115590ab6ef0f970cdbd8
CRL serial number:       01A2BF
CRL this update:         Sun 21 Jul 2024 11:49:22 +0000
CRL next update:         Sun 21 Jul 2024 17:49:22 +0000
Revoked Certificates:
                         Serial:   01462B   Revocation Date: Wed 29 Nov 2023 16:38:30 +0000
                         Serial:   016C10   Revocation Date: Thu 15 Feb 2024 16:49:45 +0000
                         Serial:   016C26   Revocation Date: Thu 15 Feb 2024 17:21:31 +0000
                         Serial:   016C2B   Revocation Date: Thu 15 Feb 2024 17:22:46 +0000
                         Serial:   016C33   Revocation Date: Thu 15 Feb 2024 17:24:53 +0000
```

RFC 5280 section 5.2.3

The CRL number is a non-critical CRL extension that conveys a *monotonically increasing sequence number* for a given CRL scope and CRL issuer.

This extension allows users to easily *determine when a particular CRL supersedes another CRL*.

In other words, the CRL Number is used to choose amongst multiple simultaneously valid CRLs


I'VE HEARD
THAT BEFORE
NETFLIX

From RFC 9286, Manifests for the RPKI

Section 4.2.1

manifestNumber:

> This field is an integer that is incremented (by 1) each time a new manifest is issued for a given publication point. This field allows an RP to detect gaps in a sequence of published manifests.

CRL Numbers are purposeless for RPs

- Well-formed Manifest contains exactly one entry for the associated CRL
- That named entry contains a collision-resistant digest of the CRL content
- The CRLDP in the CA Resource Certificate points to the CRL by name
- The manifestNumber monotonically increases, helping detect replay for both the Manifest itself and its listed products (such as the CRL)

Together, these properties guarantee that RPKI RPs will always be able to unambiguously identify exactly one current CRL for each RPKI CA.

Thus, in the RPKI, the ordering functionality provided by CRL Numbers is fully subsumed by monotonically increasing Manifest Numbers, **thereby obviating the need for RPKI RPs to process CRL Number extensions at all.**

Solves Case 1: Maxing out the CRL Number

Both manifestNumber and CRL Number can max out.

For manifestNumber's there is a solution described in draft-ietf-sidrops-manifest-numbers

But, what to do if the CRL is maxed out? CA is dead in the water? Keyroll?

Or … just have RPs ignore the CRL number!

Solves Case 2: using the CRL Number is complicated

- Imagine a RP obtains a higher numbered CRL, but not its associated Manifest (could happen if cArepository updates are smeared out over multiple RRDP deltas)
- The new CRL revokes the previous Manifest's End-Entity certificate
- But the RP only has the previous Manifest (which the new CRL revoked)
- But RFC 9286 specifies to use cached versions of the objects in case of failed fetch
- Thus RPs should ignore the higher numbered CRL, until they acquire a complete and valid cArepository state
- Ergo, the CRL Number has no constructive purpose in RP handling

The "Failed fetch" mechanism exists to improve reliability and cArepository availability. The RFC 5280 CRL Number purpose description is incompatible with how things are done in the RPKI.

Again, RPs ignoring the CRL Number solves this

```
/*
 * Check that the CRL number extension is present and that it is non-critical.
 * Otherwise ignore it per draft-spaghetti-sidrops-rpki-crl-numbers.
 */
static int
crl_has_crl_number(const char *fn, const X509_CRL *x509_crl)
{
        const X509_EXTENSION    *ext;
        int                      idx;

        if ((idx = X509_CRL_get_ext_by_NID(x509_crl, NID_crl_number, -1)) < 0) {
                warnx("%s: RFC 6487, section 5: missing CRL number", fn);
                return 0;
        }
        if ((ext = X509_CRL_get_ext(x509_crl, idx)) == NULL) {
                warnx("%s: RFC 6487, section 5: failed to get CRL number", fn);
                return 0;
        }
        if (X509_EXTENSION_get_critical(ext) != 0) {
                warnx("%s: RFC 6487, section 5: CRL number not non-critical",
                    fn);
                return 0;
        }

        return 1;
}
```

Next steps for draft-spaghetti-sidrops-rpki-crl-numbers

- RP implementers to double-check they ignore the CRL Number Extension
- It is a short document, please review and provide feedback!
- Call for Working Group Adoption?