

Enforcing mTLS

This is where the policy becomes real. The two configurations below, one on the calling side, one on the serving side, are what turn "both ends should authenticate" into "no valid certificate, no connection." Everything before this page was setup, this is the enforcement.

Recall the two ends of an internal connection: We'll use a WAF as an example, but this can apply to any communication. The WAF must present its client certificate and verify the backend's server certificate, and the backend must present its server certificate and require the WAF's client certificate. Both halves have to be configured.

The calling side (WAF)

On the WAF, inside the `location` block that proxies to the backend:

```
location / {
    proxy_pass https://service.int.example;

    # Verify the backend's server certificate against the CA
    proxy_ssl_server_name on;
    proxy_ssl_name service.int.example;
    proxy_ssl_trusted_certificate /etc/ssl/certs/example-ca.crt;
    proxy_ssl_verify on;
    proxy_ssl_verify_depth 2;

    # Present the WAF's own client certificate (this is the "mutual" half)
    proxy_ssl_certificate      /etc/ssl/certs/waf-to-service.crt;
    proxy_ssl_certificate_key /etc/ssl/private/waf-to-service.key;
}
```

Line by line, what each directive enforces:

- `proxy_pass https://...`: the upstream is HTTPS, not HTTP. The connection to the backend is itself TLS, separate from the public TLS the user terminated at the WAF.
- `proxy_ssl_server_name on` / `proxy_ssl_name`: send the backend's name during the handshake and verify the certificate matches it. This is what makes the name the identity: the WAF checks it reached `service.int.example` and not something else answering on that

IP.

- `proxy_ssl_trusted_certificate`: the CA root. The backend's certificate is trusted only if it chains to this.
- `proxy_ssl_verify on`: actually enforce the check. Without this, nginx would happily connect to a backend presenting any certificate. This directive is the difference between "encrypted" and "verified."
- `proxy_ssl_verify_depth 2`: allow the chain to be two links deep (host cert → intermediate → root), matching the CA's structure.
- `proxy_ssl_certificate` / `_key`: the WAF's own client certificate. This is the half that makes it *mutual*: the WAF proves its identity to the backend, not just the other way around.

The serving side (backend)

On the backend host, the internal gateway server block:

```
server {
    listen 443 ssl;
    server_name service.int.example;

    # This host's server certificate
    ssl_certificate      /etc/ssl/certs/service.crt;
    ssl_certificate_key  /etc/ssl/private/service.key;

    # Require and verify the caller's client certificate
    ssl_client_certificate /etc/ssl/certs/example-ca.crt;
    ssl_verify_client on;
    ssl_verify_depth 2;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        # hand off to the local application
        proxy_pass http://127.0.0.1:80;
        proxy_set_header Host $host;
        # ...forwarding headers
    }
}
```

The three directives that do the enforcing:

- `ssl_client_certificate`: the CA root, here used to validate *incoming* client certificates. The backend trusts a caller whose certificate chains to this.
- `ssl_verify_client on`: the critical one. This makes the client certificate *mandatory*. A connection without a valid CA-signed client certificate is rejected at the TLS layer, before the request ever reaches the application. This single directive is what enforces "being on the network isn't enough."
- `ssl_verify_depth 2`: same two-link chain allowance as the other end.

The backend terminates the mutually-authenticated TLS, then hands the request to the local application over plain localhost HTTP (`127.0.0.1:80`). The application itself doesn't need to know anything about certificates. The gateway enforces mTLS in front of it. That keeps the application simple and puts all the certificate logic in one place per host.

Gotchas

- **The name and the certificate must agree.** The WAF's `proxy_ssl_name`, the backend's `server_name`, and the name in the backend's certificate all have to refer to the same thing. If the WAF connects under one name and the backend's server block answers to another, nginx rejects the handshake and you get a misdirected-request error that looks like a proxy bug, but is really a name mismatch. When adding a service to a host that already serves others, the name the caller uses is what routes it. Get that wrong and the symptom is confusing.

Adapt this for...

Any internal service-to-service connection, not just WAF-to-backend. Anywhere two internal components talk and you want that traffic authenticated rather than merely encrypted, this is the shape, and `ssl_verify_client on` (or its equivalent on the serving side) is the line that enforces it.

Revision #4

Created 2026-07-20 19:36:33 UTC by Chris Landis

Updated 2026-07-27 14:52:29 UTC by Chris Landis