TLS Metadata for Load Balancers

draft-schwartz-tls-lb
Ben Schwartz, Google LLC
IETF 105
What is a “load balancer”?  

- **Definition**: A load balancer is a server that forwards connections from clients to appropriate backends.  
- **Balancing loads** is the main purpose but not the only purpose: also DDoS defense, access control, etc.  
- A load balancer can operate at many levels in the protocol stack. We use different terms depending on which protocol the load balancer “terminates”:  
  - IP: “ECMP router”  
  - TCP: “TCP load balancer”, “reverse tunnel”  
  - SNI: “SNI proxy”  
  - TLS: “TLS load balancer”, “TLS termination proxy”  
  - HTTP: “Reverse proxy”, “CDN”  
- **Goal**: Make it easier and safer to use **SNI proxies**, to **reduce the need for TLS termination** by load balancers.
Motivating use case: **Full Split-mode ESNI**

- **Shared mode**
  - Client → Server (CDN)
  - Have to give your private key to the server!
  - ESNI Private Key
  - TLS Private Key

- **Partial split mode**
  - Client → Load balancer → Backend server
  - ESNI private key is a widely shared secret!

- **Full split mode**
  - Client → Load balancer → Backend server
  - Backend can’t reply without the ESNI nonce & decrypted SNI
State of the art: **PROXY protocol**

PROXY TCP4 192.168.0.1 192.168.0.11 56324 443
GET / HTTP/1.1
Host: www.example.com

- For TCP reverse tunnels.
- Prepends info in cleartext (even if the contents are encrypted)
  - Not encrypted or authenticated in any way.
- Originally only carried the client IP and port.
  - Now extended to forward ALPN, SNI, etc. (when used by a TLS-terminating load balancer)
- Implemented by HAProxy, NGINX, Stunnel, Postfix, Squid, Jetty, etc.
- Deployed by Amazon Elastic Load Balancer, Google Cloud Load Balancing, etc.
Proposed architecture: Like PROXY but encrypted

1. Load balancer and backend share a long-lived symmetric* PSK.
2. On each connection, the load balancer packages the needed metadata into a ProxyData struct.
3. Load balancer prepends ProxyData, AEAD encrypted and bound to the ClientHello.
4. Backend decrypts and uses the info.

*Static Diffie-Hellman is also possible.
Alternative architectures: TLS-in-TLS

- Defends ESNI privacy against a trivial two-point surveillance attack.
  - Makes padding possible, for stronger defenses in the future
- Potentially immune to replay attacks (after 0-RTT)
- Offers a clear way to implement a public, free load balancer (no PSK)
  - Greatly increases load balancer costs
    - 16-32x based on AWS prices
  - Not clear how to extend to QUIC
Alternative architectures: ESNI Oracle

- Excellent architectural clarity
- Imposes a latency penalty OR creates a lot of potential complexity
  - e.g. H2 PUSH and a cache on the backend
- Not clear how to support other metadata (e.g. client IP)
  - Might require a very fast database on the load balancer
Questions for the group

- Should we try to standardize a solution?
- Which architecture should we pursue?