TLS Next Protocol Negotiation

Mike Belshe, Adam Langley
March, 2011

Problem:

We want to use a protocol other than HTTP when connecting via TLS.

But...

We don't know if the server supports the other protocol, and we cannot afford the latency hit of a round trip.

Non-solutions: Using a different port.

The classic approach is to use a different port.

Problems:

- Port numbers other than 80 and 443 suffer from discrimination within the network.
- Port 80 is transparently proxied by intermediaries that only understand HTTP. Use of any protocol other than HTTP often breaks (timeouts, errors, or incorrect behavior)
- Port 443 is safe from intermediary tampering. However it can require up to 5 round trips (DNS, TCP, TLS(2), App) before discovering that the application cannot support a different protocol

Non-solutions: Racing connections

We could race two protocol connections and see which one finishes.

Problems:

- Inefficient on the network
- Inefficient on the server
  (at least one connection will be wasted)
- Has an element of unpredictability.

Non-solutions: Upgrading

HTTP provides an "upgrade" mechanism at the application layer.

Problems:

- Burns an extra round trip.
- Without TLS, it still breaks intermediaries outside the control of the server.

Non-solutions: Memory

This covers schemes where clients learn of protocol support via a HTTP header and, for future connections, use the new protocol.

- Servers cannot roll back support.
- It breaks SSL MITM boxes which expect HTTP unless the indication mechanism is a TLS extension (which these boxes will remove.)
- It requires that the server figure out the protocol based on the initial bytes from the client. Complicates the stack and can result in security issues.

Solution: Next Protocol Negotiation

Use TLS's extension mechanism to provide explicit negotiation.

1. Client advertises support as an extension
2. Server echos the extension, optionally including a list of supported protocols.
3. Client sends a NextProtocol handshake message after the ChangeCipherSpec. The protocol name is padded to 32-bytes to avoid leaking the size.

(The protocol that the client selects doesn't have to be one of the ones offered by the server.)

Why not in the clear?

We are forced into this solution, to a large extent, because of network discrimination against TCP port numbers. It seems like a bad idea to repeat the same mistakes.

If we wish this extension to be generally useful, it should serve other uses and, where cryptography is involved, there are often incentives to discriminate.

Example: the Iranian national firewall inspects the Diffie-Hellman group of EDH-* TLS handshakes and blackholes those which use certain groups in an attempt to discriminate against certain applications.

Deployment Status

- Patches exist for NSS and OpenSSL
- Used by 100M+ Chrome users daily to talk to Google HTTPS servers.
- Draft written
  - http://tools.ietf.org/html/draft-agl-tls-nextprotoneg-00.html