IS IT TIME TO BRING BACK THE HOSTS FILE?
INELEGANT ENCRYPTION SOLUTIONS FOR AN INELEGANT NETWORK?

Peter Eckersley,
Technology Projects Director
Electronic Frontier Foundation

Recent revelations have made clear that the widespread unencrypted communications channels on our current network are being ruthlessly exploited by intelligence agencies around the world. We know that HTTP and SMTP in particular are enormous windows for dragnet data collection.

Pragmatically, our hopes are slim for defending the entire network against adversaries with the sophistication and versatility that intelligence agencies have demonstrated. Anyone specifically targetted by these actors will need help beyond anything that the Internet engineering community can provide. The task the community needs to tackle -- and the one where we at least have a hope of success -- is ensuring that communications are encrypted by default in such a way that only active, targetted attacks can collect them.

THE OBSTACLES TO A DEFAULT-ENCRYPTED INTERNET

The practical problems we face in trying to close the HTTP and SMTP dragnet data collection windows are quite daunting.

One is the the "PKIX" ("Public Key Infrastructure via X.509"), the system of certificate authorities for issuing TLS certificates has, perhaps through little specific fault of most participants, become a vast and obstructionist bureaucracy that is doing more to undermine Internet security than to increase it.

The second obstacle we face is that, as Internet deployment has grown, the half-life for old software versions has stretched to the point where communications end points cannot safely assume that their peers support protocol security features that are less than ten years old. In fact, expecting that features launched in 2010 will be universally available by 2030 might be optimistic. We are going to be living with clients running Windows XP and servers running minimally-maintained LAMP stacks and MTAs for the near-infinite future.

The third obstacle we face is that network operators and hardware manufacturers have failed to heed the old arguments that the network should avoid interposing itself in end points' communications and computations. Protocol designers today face a formdiable gauntlet of NAT devices that make erroneous assumptions about the traffic they carry; "captive portal" networks that assume all HTTP (and even HTTPS) requests are made by humans with Web browsers and are appropriate targets for hijacking; ISPs that are keen to prioritise some protocols and origins over others, and network administrators who concluded that everything they didn't understand should be blocked.

INELEGANT CIRCUMSTANCES WILL CALL FOR INELEGANT METHODS

Given the obstacles arrayed against clean, from-scratch designs for better communications protocols, we should not expect that such designs are capable of solving the dragnet surveillance problem in a timely fashion.

We should certainly work toward clean solutions, and proposals like DANE DNSSEC or CurveCP might eventually become those. In the mean time, we are going to need to be more pragmatic. Faced with an adversary that fights dirty, we are going to need to fight dirty too.

Here are a couple of ideas:

1. Bring back the hosts.txt file.

We have an actually-engineered solution for forcing HTTPS connections on a given domain. It is the HTTP-Strict-Transport-Security header, and as of this writing, it remains inadequately deployed and still unsupported by one of the major Web browsers. There is no equivalent for other protocols that require upgrading to TLS, including SMTP, XMPP (and to a lesser degree, IMAP, POP,

IRC, and many others).

One solution to this problem is to bring back something like the hosts file
that existed before the Domain Name System: a single global repository of
every domain. Unlike the original hosts file, which included an IP address
for each host, a modern one could include information about TLS support and
deployed keys.

Unlike the 1980s, many of the systems we have today are perfectly capable of
storing a database of all of the servers that speak a given protocol (like
SMTP or XMPP). Unlike the 1980s, we have extremely good tools for
distributing such files (like git or mercurial). We even have some basic and
workable authentication methods for site operators to update their keys
(github credentials, signed PGP tags).

Given these tools, it shouldn't be more than a few days' work to force TLS
and pin keys for any major SMTP or XMPP server that wishes to be upgraded.
A few days' further work could get us installable packages for common server
OSes to ensure that MUAs making outbound SMTP connections can fetch that
pinning information via git, and enforce it.

2. Yes, let's do opportunistic encryption (and let's make it count)

Intensive debate is ongoing about whether HTTP/2.0 should include
opportunistic encryption, or whether we should force every server to obtain a
valid X.509 certificate before it gets the benefit of HTTP/2.0.

This is a hard and subtle question; following either fork in the road could
lead to good or bad outcomes. The "high road" is certainly to require valid
X.509 certificate chains, and try to make it easier for people to get them.

We should think about "low roads" too. One option would be to declare that in
HTTP/2.0, all X.509 chains are valid, not only for opportunistically upgraded
http origins, but actually *for https origins*. The exception would be any
public suffixes for which valid X.509 certificates have been recorded in
certificate transparency logs. This list of suffixes for which a certificate must
be checked is not that large: a few megabytes with a well chosen compression
algorithm. Operators who care a lot about security can easily get their certs
checked (just make a valid one!); operators who don't get a weaker form of
HTTPS -- but one that's still good enough to prevent dragnet surveillance.