Opportunistic Encryption Everywhere

Adam Langley
Google Inc
San Francisco, California
agl@google.com

***Abstract*—Many of the core Internet protocols were designed for a less hostile network than the average packet finds itself in today. Many hosts are connecting over shared-key or open wireless networks where eavesdropping is trivial. On a larger scale, ISPs are experimenting with institutional sniffing where the clickstreams of the majority of users are observed and sold for their marketing value.**

Like the shift away from RSH and open mail relays, it's our belief that all traffic should now be encrypted. As a practical matter this means opportunistic encryption, with all its complex trade-offs. In this paper, we explore these compromises and the design space of such a goal.

I. Introduction

Traditionally, most network protocols have been unencrypted. Encryption, if any, is often an optional and secondary concern. There are many good reasons for this. Unencrypted protocols are easier to debug, easier to experiment with and computationally cheaper. Additionally, sniffing on networks has previously been the exception, not the rule.

In recent years, the computational cost of cryptography has been reduced in two ways. As ever, processors become faster and cheaper on the one hand. On the other, cryptography and research into cryptographic implementations has advanced. On an Intel Core2, one can perform AES-128 encryption at 12 cycles/byte[1] and an elliptic-curve key agreement can be performed within 400K cycles[2]. Polynomial message-authenticators can be as cheap as a couple of cycles/byte on a Pentium 4[3].

As some of the costs of encryption have diminished, the motivations have increased. Companies such as Phorm and Nebu-Ad have announced and trialed schemes where web-traffic is sniffed or intercepted at an ISP level to extract information about users. As economic times get tougher, the incentive for ISPs and other network operators to engage in such dubious practices will only increase.

On the small scale, the methods by which end hosts are connected are becoming increasingly wireless. These wireless networks are inherently more open to sniffing, especially given the dominance of Wi-Fi networks using weak cryptography[4] with no key or shared keys.

A major hindrance to the mass deployment of encryption has been the problem of trust. Although key-agreement protocols can establish a secure channel between two end-points, additional information is need to verify that the other end is who they claim to be.

TLS[5] solves this problem using a public-key infrastructure where central authorities are trusted to link real-world and cryptographic identities. SSH has achieved wide-ranging success using imprinting, where cryptographic identities are assumed to be correct on first sight and remembered for future transactions.

Despite ubiqutous support for HTTP over TLS (HTTPS), general adoption has been limited due to a variety of roadblocks:

Human intervention is required to generate keys, install them on the server and submit them to the certificate authority. These authories require a small, annual fee and the administrator must remember to renew the certificates in a timely manner. This process must be repeated for each domain name which points to the server. Also, support for HTTPS virtual hosting[6] is not as widespread as HTTPS support itself.

Once the transport layer aspects are working correctly, websites may need work to avoid mixed-content warnings and explicit `http` URLs in links which would direct the browser to the unsecured version of the site.

II. Opportunistic Encryption

Opportunistic encryption is the act of setting up a secure channel without verifying the identity of the other host (or, at best, imprinting may be used). Although it defeats mere eavesdroppers, an attacker who can modify packets in transit can perform a man-in-the-middle attack and persuade both legitimate endpoints that he is the other.

Because of this, opportunistic encryption is insufficient when the secrets being protected are of high value. However, it can operate without human action or knowledge and this appears to be critical for any design which aspires to large scale deployment. Despite near ubiquitous support for HTTPS, and the low cost of certificates, adoption has been sadly limited.

Perhaps most critically, opportunistic encryption provides a method for detecting large-scale sniffing. Currently, sniffing can occur silently and none of the users need ever be aware that it's happening. In the face of widespread opportunistic encryption, a sufficiently powerful attacker could perform man-in-the-middle attacks on all connections at an ISP level. However, any connections where identities are validated would detect the subterfuge.

Thus, a small set of nodes, distributed across different networks, performing validated connections between each other at regular intervals could serve as canaries for any network machinations. As long as the attacker can not differentiate opportunistic connections from validated ones at the point

where the attack has to be performed, a large-scale attack would inevitably be detected.

III. Design Space

If the goal of opportunistically encrypting traffic is accepted, there are still many details to decide upon. In general, the details break into three parts:

- The cryptographic protocol and primitives to use.
- The layer of the stack to insert them at.
- How to signal support for a new protocol and provide a smooth upgrade path.

For now, we'll largely ignore the choice of cryptographic primitives (ciphers, MACs etc). The weakness of opportunistic encryption against active attackers means that we might wish to choose less secure primitives to gain computational efficiency. This is a rare situation to be in, but a discussion here would detract from the more important points below.

We will first consider where in the stack we should work.

A. IPSec

The FreeS/WAN project[7] attempted, and failed[8], to implement opportunistic encryption at the network layer. The reasons for this failure are myriad and include the complexity of the solution and the additional latencies introduced. The work is being carried on by OpenSWAN, StrongSWAN and the IETF Better Than Nothing Security working group[9].

We wish them the best of luck, but consider the failure of FreeS/WAN to be instructive. For a solution at the network layer, their various designs are largely reasonable. Many of the reasons for their failure (complexity, difficultly of working with the variety of networking hardware, latency) are endemic to the layer at which they were working.

Because of this, we don't try to solve the problem at the network layer and carry with us a renewed conviction that any solution must be transparent and painless to the user.

B. TCP

If we are aiming for all traffic to be opportunistically encrypted, then adding the capability to TCP would seem to be almost as ideal as IPSec. By working at this low level, all higher protocols would automatically gain the benefits without having to alter the thousands of user-land programs that implement them.

Providing an upgrade path is easily done using the TCP options mechanism to advertise support.

Also, by working at the TCP layer, we could protect the TCP metadata itself. This would render harmless simple RST injection attacks[10] and other such TCP manipulations.

There are significant technical difficulties with altering TCP to implement worthwhile opportunistic encryption, although we can overcome them at the cost of some complexity. The details can be found in Appendix A.

Most significantly, changes at this layer are difficult to achieve. The addition of TCP options requires the agreement of the IETF. More troubling is that important TCP stacks are closed source and it may take many years to effect a change in them[1].

There is also an aesthetic argument that such behaviour does not belong in such a low-level, critical, and already complex protocol. A prototype patch to implement this in the Linux 2.6.26 TCP stack added 1500 lines to the TCP implementation (not counting cryptographic primitives), supporting this argument.

C. Transport Layer

If we decide to work at the transport layer, above the kernel, we must solve the advertising issue to provide an upgrade path and decide what the transport layer protocol should look like.

There are two basic options for transport layer cryptography: use TLS or roll our own. Many of the caveats about designing one's own transport layer protocol are rendered moot when considering a protocol that is not designed to be secure against man-in-the-middle attacks.

TLS requires two additional round trips to establish a connection. (This is reduced to one for connections where the client has cached state from a previous connection.) By rolling our own protocol we can gain one significant advantage over TLS: no additional round trips when we obtain the server's Diffie-Hellman public value via some side-channel.

The costs of additional latency have not been well researched, publicly. Instead we have to rely on anecdotes like the following: "In A/B tests (at Amazon.com), we tried delaying the page in increments of 100 milliseconds and found that even very small delays would result in substantial and costly drops in revenue."[11]

By rolling our own transport layer protocol we could also use faster cryptographic primitives: elliptic curve Diffie-Hellman, reduced round ciphers etc. (TLS *could* use all of these. However, modifying TLS and its implementations negates some of the advantages of its ubiquity.)

An argument favour of TLS is that it is already universally supported with solid implementations. Every serious web server includes well tested TLS support. TLS does have a reputation of being slow but this is usually the result of conservative default configurations. When using OpenSSL with no special configuration, connecting to `mail.google.com` (which is expertly configured) results in a ciphersuite of `RC4-SHA`. Connecting to a default Apache installation uses the dramatically slower `DHE-DSS-AES256-SHA` suite.

While the latter suite provides additional security, we consider it unnecessary for HTTPS. It's our opinion that the costs of TLS are greatly exaggerated in the minds of many administrators and a little configuration would go a very long way.

IV. Providing an Upgrade Path

If we decide not to work in the TCP stack, we have to provide some way for clients to know when a given server

[1]It took eight years between the publication of RFC 1323 and the implementation of TCP timestamps in Windows 2000. It took several more years before Windows 2000 and later kernels came to account for a significant fraction of running Windows systems.

supports our transport layer modifications. At this point we have to start dealing with the specifics of application-level protocols, and for simplicity, we only consider HTTP. If an opportunistic encryption scheme ever managed reasonably wide deployment with HTTP, we would consider it a stunning success.

Such an advert needs to provide confirmation that a given server supports opportunistic encryption. It may provide an alternative port number for encrypted connections, a list of supported protocols, the server's Diffie-Hellman public value etc.

A. DNS

The first advertising possibly that we consider is putting information in DNS. Since HTTP clients start DNS resolution before the HTTP connection is started, DNS provides the possibility of supporting opportunistic encryption on the first connection to a given server.

By far the easiest method of encoding information in DNS is via CNAME records. If a given name (say `www.example.com`) is a CNAME to `xyz.example.com` (where `xyz` is a longer, base64 encoded string) then this information is typically returned in the Additional section of the reply[12] and is available via the standard `gethostbyname` API call.

This neat trick has a major limitation in that it doesn't work when using a name without a `www` (etc) at the beginning because such a name cannot be a CNAME.

Using other record types is certainly possible but DNS packets cannot carry more than one request[2], so accessing the extra records means additional DNS requests. The requests for the non-standard records will also have much lower cache hit rates. If the HTTP client doesn't wait for the reply to this additional request before making the connection, then we lose the ability to act on the first connection. If it does wait, then we introduce additional latency for all websites, regardless of their support for opportunistic encryption.

Non-standard record types are also more difficult to resolve using standard APIs, difficult to configure with some DNS registrars, and require human intervention to setup.

B. HTTP Based Imprinting

Since we are only considering HTTP at this point, let us imagine that there's a distinguished path that HTTP clients can request from a given server, containing information about the server's support for opportunistic encryption schemes (like `/favicon.ico` for icons). The client then remembers this information for future connections to the same server.

For latency reasons we wouldn't want to wait to resolve this, so the client's first connections to a given server could not be opportunistically encrypted.

However, such a scheme has two important advantages:

[2]DNS packets can have more than a single request in them. However, due to an apparent oversight, there's no way for the server to signal the status of the different requests and so, to our knowledge, every DNS server returns `SERVFAIL` in this case.

Firstly, an HTTP client can check the validity of the claims in the advert. A client may find a valid advert for a server but then find that encrypted connections actually fail. For example, network policies in place near the client may deny outgoing TCP connections to non-whitelisted port numbers. The HTTP client is perfectly placed to check that encrypted connections are possible when fetching the advert.

Secondly, it can be implemented automatically. The default configuration of web servers can advertise support without any administrative action. Usually the presence of firewalls/proxies etc precludes the enabling of protocols without an administrator verifying its correct functioning. However, if clients verify this before accepting the advert they will not be adversely affected by an over-optimistic claim of support.

V. Related work

We have already mentioned FreeS/WAN[7] and its progeny, OpenSWAN, StrongSWAN and BTNS[9], all of which add opportunistic encryption to IPSec.

One notable success for opportunistic encryption has been `STARTTLS` support in SMTP[13]. Since SMTP already had a standard method for advertising additional server capabilities, organic deployment of `STARTTLS` support has meant that opportunistically encrypted hop-to-hop transport will eventually be the norm for e-mail.

VI. Conclusions

In order to gain understanding of the issues involved, we have built working prototypes of all the possibilities outlined above.

In light of that work, we currently believe that the most minimal solution stands the best chance of success. That would be an HTTP specific solution, using imprinting and TLS. We have initial implementations for Firefox and Chromium at this time.

Although several details still need to be resolved, the rough outline of the scheme is this: Upon connecting to an HTTP server for the first time, the browser requests the path `/host-meta`. If present, this document[14] contains information about the server's support for opportunistic encryption, the fingerprint of the certificate and the duration of time that this information should be cached. If the server believes that it is capable of opportunistic connections, the client immediately verifies this.

For future connections to such a host, the HTTP stack connects using TLS on the standard HTTPS port (443) and skips certificate verification.

This design emphasises ease of deployment above all else. It requires changes to web browsers; however, open source browsers now account for over 20% of users[15]. Implementation on the server side is as simple as adding a file to the server's document root and setting up a self-signed certificate. Opportunistic encryption can also be configured by default for new web server installs.

The problems with this design are that it is specific to HTTP traffic only, it adds additional latency, it doesn't account for the

first connection to a given host, and TLS is not an optimised protocol for opportunistic encryption.

VII. UI Considerations

There should be no changes to the UI of any client application because of these proposals. Because we have discussed HTTPS, we find that people often conflate our suggestions with issues of secure UIs and user-interaction. Opportunistically encrypted transport is no more pertinent to users than compression, and HTTP clients must not suggest otherwise in their UIs. TLS with PKI is the HTTPS gold-standard and we don't wish to dilute that.

Appendix
Opportunistic Encryption in TCP

The major technical problems with performing opportunistic encryption in TCP come from the small size of the TCP options space.

The TCP options space is, at most, 40 bytes long. In a modern TCP stack, 20 of those bytes are already used in a SYN frame. If we leave four bytes for future options, that leaves 16 bytes for a Diffie-Hellman public value. After taking two bytes for the option header we have only 14 bytes (112-bits).

The most space efficient Diffie-Hellman schemes are based on elliptic curves (we are discounting groups based on hyper-elliptic curves[16] because they are still research topics). The best general algorithm currently known for solving the Diffie-Hellman problem on elliptic curves is Pollard's Rho. The expected number of operations required is $O(\sqrt{n})$, or $\approx 2^{56}$ in this case.

Most importantly, once you have solved a single instance you can precompute tables to speed up breaking more instances. With a petabyte of storage, you could break 112-bit curves in 2^{12} operations, which is a real-time attack.

So, if we concede that we need larger public values we have to employ space outside of the traditional TCP options space. There is an IETF draft to this effect[17], but it greatly complicates matters because the client's public value can no longer be carried in the SYN frame. Instead, we are forced to invent the concept of deferred options which are carried in the client's second frame (and thus must be retransmitted and acknowledged etc).

References

[1] D. J. Bernstein and P. Schwabe, "New AES software speed records," in *INDOCRYPT*, ser. Lecture Notes in Computer Science, D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365. Springer, 2008, pp. 322–336.

[2] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *In Public Key Cryptography (PKC), Springer-Verlag LNCS 3958*, 2006, p. 2006.

[3] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication." Springer-Verlag, 1999, pp. 216–233.

[4] H. Berghel and J. Uecker, "Wireless infidelity II: airjacking," *Commun. ACM*, vol. 47, no. 12, pp. 15–20, 2004.

[5] T. Dierks and E. Rescorla, "RFC 5246: The transport layer security protocol, version 1.2," 2008.

[6] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "RFC 4366: Transport layer security (TLS) extensions," April 2006.

[7] J. Gilmore. (2009, January) FreeS/WAN. [Online]. Available: http://www.freeswan.org

[8] C. Schmeing. (2004, March) FreeS/WAN announcement. [Online]. Available: http://www.freeswan.org/ending_letter.html

[9] IETF. (2009, January) Better-than-nothing security charter. [Online]. Available: http://www.ietf.org/html.charters/btns-charter.html

[10] P. Eckersley, F. von Lohmann, and S. Schoen. (2007, November) Packet forgery by ISPs: A report on the Comcast affair. [Online]. Available: http://www.eff.org/wp/packet-forgery-isps-report-comcast-affair

[11] G. Linden. (2006, November) Marissa Mayer at Web 2.0. [Online]. Available: http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[12] P. Mockapetris, "RFC 1035: Domain names - implementation and specification," 1987.

[13] P. Hoffman, "RFC 3207: SMTP service extension for secure SMTP over transport layer security," 2002.

[14] M. Nottingham and E. Hammer-Lahav. (2009, February) Host metadata for the web. [Online]. Available: http://tools.ietf.org/html/draft-nottingham-site-meta-01

[15] N. Applications. (2009, February) Browser market share. [Online]. Available: http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0

[16] H. M. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, pp. 393–422, July 2007.

[17] W. Eddy and A. Langley. (2008, July) Extending the space availible for TCP options. [Online]. Available: http://www.ietf.org/internet-drafts/draft-eddy-tcp-loo-04.txt