



## Why Didn't We Spot That?

Stephen Farrell • Trinity College Dublin

**T**he Secure Sockets Layer (SSL) protocol and its standards-track successor, the Transport Layer Security (TLS) protocol,<sup>1</sup> were developed more than a decade ago and have generally withstood scrutiny in that the protocols themselves haven't been found to have security flaws. Until now. In August 2009, Marsh Ray and Steve Dispensa discovered a design flaw in the TLS protocol (and published it in November 2009 due to independent rediscovery of the flaw by Martin Rex)<sup>2</sup> that affects all versions of the protocol up to and including the current version.

Whereas the vulnerability itself is serious, it need not affect many deployments once administrators apply suitable patches to disable renegotiation, leaving TLS sufficiently secure in most cases because exploiting the vulnerability requires the attacker to be an active man-in-the-middle, redirecting traffic between victims (for example, a browser and a Web server). However, because security problems only ever get worse, a change to the protocol is required and is now being developed as a high priority in the IETF (<http://tools.ietf.org/wg/tls>). If all goes well, a new RFC with the fix might be published soon after this article appears.

The vulnerability is an interesting attack in itself, but perhaps more interesting is the question, why didn't we see this earlier? In this article, I explore this question but, unfortunately, can't answer it. Hopefully, simply asking the question might prompt developers to re-examine assumptions they've forgotten they've even made.

### The Recently Discovered Problem

The TLS protocol starts with the so-called "handshake" phase in which two parties agree on the types of cryptography and on the keys to use for protecting application data. The handshake requires a couple of roundtrips, as the client and server exchange and then verify parameters after they've established shared keys. After the handshake, the keys established

during the handshake protect the application data (for example, HTTP traffic). Figure 1 – modeled on figures from Eric Rescorla's Internet draft<sup>3</sup> – provides an abstract view of such an exchange, showing the initial handshake messages that aren't encrypted, followed by protected application-layer traffic between the client and server.

The problem arises due to the fact that TLS also lets clients and servers renegotiate or, in other words, do a second handshake, and this second handshake isn't cryptographically bound to the initial one.

TLS allows this for a couple of reasons. Perhaps its most common use today is to enable protection of clients' identities in the (currently rare) case that the public-key-certificate-based client-authentication option built in to the TLS protocol is in use. (This particular use of renegotiation seems to have first been mentioned on the TLS working group mailing list in mid-1998.) The problem with directly sending the client's public-key certificate in the initial handshake is that the client's identity is exposed in cleartext because the client and server don't yet share a key to encrypt the client's identity. So, sending the client's identity in the second handshake (see Figure 2) solves this problem because all the messages that form the second handshake are protected (encrypted and integrity protected) using the keys established in the first handshake.

Clients and servers can also change the set of cryptographic parameters in use via TLS renegotiation. In the 1990s, US authorities, in particular, tightly controlled the export of strong cryptography, resulting in standard Web browsers that could only use TLS to negotiate weak ciphers. This was a problem for banking applications, and, in 1999, Microsoft and other companies developed what they called server-gated cryptography (SGC) to solve this problem. With SGC, the server certificate, which is sent from the server to the client as part of the initial hand-

## The Transport Layer Security Protocol — A Brief History

In the mid 1990s, the Internet and the Web were exploding in terms of numbers of users and hosts. The advent of the Web led to many new commercial uses of the Internet and also resulted in lots of nontechnical users engaging in Web commerce. As is still the case, credit cards were the main payment mechanism, so there was a widespread concern that people's credit-card information could be misused because it was generally sent unprotected over the Internet between the user's browser and the commerce site's Web server. The fact that this isn't the most likely source of exploits (attacking a server containing a database of credit-card information is much more lucrative) was in a sense irrelevant because the main issue was one of confidence — users

wanted to feel "secure" when they entered their credit-card information, and it seemed that encrypting the data as it transited the network was the required solution. Starting in early 1996, and basing their work on Netscape's Secure Sockets Layer (SSL) protocol, the IETF's Transport Layer Security (TLS) working group was formed to standardize a solution for this problem, resulting in TLS version 1.0 (RFC 2246)<sup>1</sup> being published in 1999, with the current version being TLS 1.2 (RFC 5246) published in 2008.

### Reference

1. T. Dierks and C. Allen, *The TLS Protocol, Version 1.0*, IETF RFC 2246, Jan. 1999; [www.ietf.org/rfc/rfc2246.txt](http://www.ietf.org/rfc/rfc2246.txt).

shake, included a special flag, indicating that the server belonged to a financial institution. The server certificate was then used as a signal for the client (a Web browser) to enable otherwise disabled strong ciphers. Because the client only saw the SGC flag during the initial handshake, a second handshake was required to enable the stronger ciphers.

Thus, the two main uses of TLS renegotiation (client identity protection and enabling strong ciphers) solved problems that were essentially protocol artifacts. Or, put another way, both uses of renegotiation were afterthoughts. In fact, the original purposes of renegotiation were to allow rekeying, once an enormous number of bytes had been sent in a long-lived TLS session (using the same keys for too long is bad cryptographic practice), or to reset message sequence numbers once they had reached a bit boundary (to prevent possible replay attacks that might occur if sequence numbers were allowed to repeat). Importantly, neither of these original purposes are relevant for the application making use of TLS, whereas the main current use for renegotiation (client identity protection) is really relevant only to the application and not to the TLS protocol itself.

In any case, the recently disclosed problem with renegotiation is that no binding exists between the

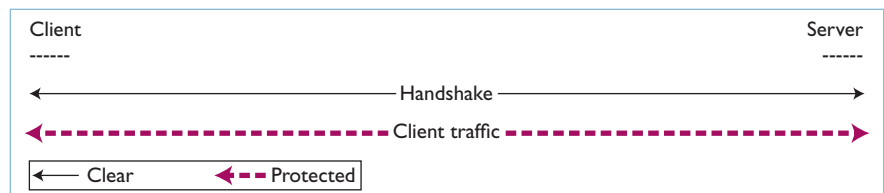


Figure 1. Transport Layer Security handshake. This figure provides an abstract view of such an exchange, showing the initial handshake messages that aren't encrypted, followed by protected application-layer traffic between the client and server.

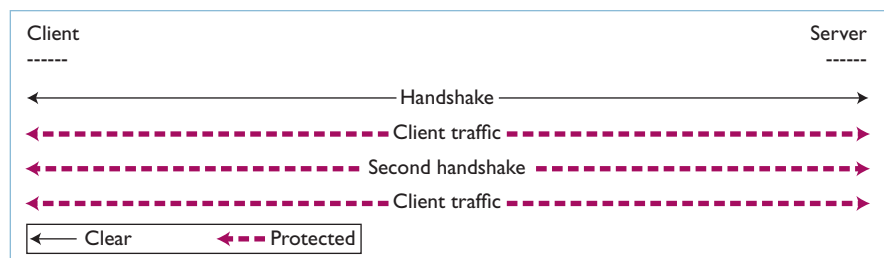


Figure 2. Transport Layer Security renegotiation. Sending the client's identity in the second handshake allows that identity to be protected (encrypted and integrity protected) using the keys established in the first handshake.

different handshakes. So, a man-in-the-middle attacker can carry out the first handshake with the server and then trick the victim-client into doing the second handshake. Figure 3 shows the man-in-the-middle attacker first doing a handshake with the server and then tunneling the victim-client's handshake through the session established with the initial handshake. Because the attacker is a man-in-the-middle, in the important case of HTTP running

over TLS, the attacker can wait until it sees a client attempt to access a given site and can then trigger the attack. From the server's viewpoint, it just sees the initial handshake and then the renegotiation, so it doesn't detect the attacker. From the client's point of view, it doesn't see the initial handshake at all and doesn't know that what it (the client) thinks is an initial handshake is actually a renegotiation.

The problem with this is that the

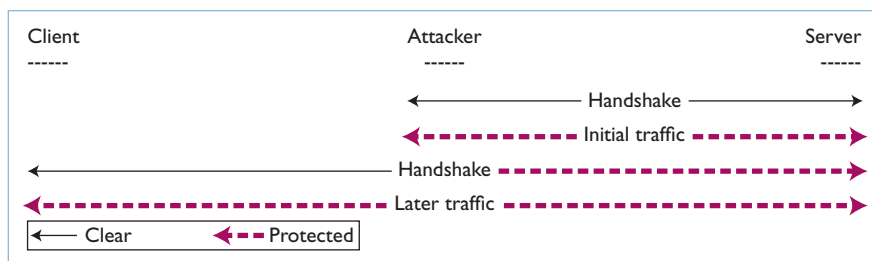


Figure 3. Transport Layer Security handshake under attack. This shows a man-in-the-middle attacker first doing a handshake with the server and then tunneling the victim-client’s handshake using the initial handshake’s session keys.

server could (and in fact does in real Web servers) treat the “initial traffic” and the “later traffic” as belonging to one session, even though one comes from the attacker and the other from the victim-client. In the client certificate in the second handshake use case, real Web servers do treat the “initial traffic” as if it were from the client who in fact is only authenticated as part of the second handshake. This lets the attacker insert an application-layer request, which is treated as being from the victim-client. In the case of HTTP running over TLS, the attacker gets to send the initial HTTP request that could be exploitable or could damage the victim-client – for example, by deleting something or causing a purchase to occur.

In fact, there are also HTTP-specific ways in which the attacker could exploit the second handshake to attack a victim-client who has previously authenticated via a password and has received a cookie, which the browser presents with subsequent HTTP requests to demonstrate that the user is authenticated. In this case, the attacker essentially splices together two HTTP sessions via some rather clever HTTP-specific “glue.”

So, a couple of things are going on here. One is that the TLS protocol doesn’t bind the different handshakes (and there could be more than two), and the other is that the higher-layer protocol (for example, HTTP) treats the initial and later traffic equivalently, even though they might not be from the same

source. And to make matters worse, most TLS implementations don’t make it easy for the server application to see the difference between the initial and later traffic, handling renegotiation transparently to the application, as the TLS specification perhaps implies.

The longer-term fix here is relatively straightforward and binds together the previous handshake with the current one so that if an attacker is in the middle, then the current handshake fails at the stage at which the previously exchanged handshake messages are verified. This is a relatively simple change to the TLS protocol, and participants in the TLS working group are currently discussing the details of exactly how to embed this binding into the protocol, with one proposal<sup>3</sup> looking like it will form the basis for the standardized solution to the problem. In the meantime, many SSL and TLS deployments could simply turn off renegotiation because they don’t have a real need for it, and server vendors will undoubtedly distribute patches that provide this control to administrators.

It’s also worth noting that the attacker in all these cases doesn’t get to fully control the victim-client’s TLS session because the attacker can’t decipher the later traffic (see Figure 3). The bottom line is that most client application data remains protected, even in the face of a successful attack. However, there are potentially many Web applications in which the attacker can guess or calculate damaging values to include

in the initial traffic. Thus, server administrators should give a high priority to deploying fixes for this vulnerability – for example, this kind of attack has been reported against Twitter ([www.theregister.co.uk/2009/11/14/ssl\\_renegotiation\\_bug\\_exploited](http://www.theregister.co.uk/2009/11/14/ssl_renegotiation_bug_exploited)). Application developers should also consider how their applications are structured – and whether an attacker could calculate or guess damaging values – and consider making appropriate changes at the application layer.

Finally, because TLS also protects other applications (for example, those using the Internet Message Access and Lightweight Directory Access protocols, IMAP and LDAP), we should expect variations on the attack that will affect some uses of those protocols. Administrators and developers using TLS should be on the lookout for reports of such attacks and should, of course, be diligent about updating their application and security infrastructures.

### How Did We Miss It?

Having described the newly published vulnerability, we come to the main question: why didn’t the security community spot this problem sometime over the past decade of widespread TLS use?

This might partly be due to a split between those who develop and use security protocols (such as participants in the IETF) and those who analyze security protocols. There are generally few analyses of security protocols presented to IETF participants because its focus is generally on either producing new protocols or fixing known problems in existing ones, as in this case. Although several analyses of TLS have been published in the literature<sup>4,5</sup>, they mainly seem to focus (as we would expect) on the security of key establishment and how applications subsequently use those keys. To date, I haven’t seen any security analysis

that directly addresses using renegotiation in TLS.

People doing (in particular) formal security analysis presumably didn't realize that applications using TLS used renegotiation as I've described, and, because they rarely meet with the protocol developers, there were few opportunities to communicate this fact. There's also the fact that it's often hard for protocol developers to fully understand the assumptions built into the security proofs presented in the literature – for example, typical IETF participants might not properly understand the conclusion that “the TLS protocol framework securely realizes secure communication sessions,”<sup>5</sup> and typical application developers depending on TLS to secure their applications are probably even less well-placed to understand such conclusions.

Implicit in what I've just described is the fact that today's uses of the TLS protocol don't actually use renegotiation for the purposes for which it was initially intended (rekeying or wrapping sequence numbers). Renegotiation to handle rekeying or sequence numbers is quite reasonably something that a TLS implementation could handle transparently. However, because renegotiation ended up being used particularly for transitioning between authentication states that are highly meaningful for applications using TLS, it's now clear that such renegotiation shouldn't be transparent to applications when used like this. In the case of SGC, the ciphers that end up being used are in fact visible to the application, but in the case of client authentication based on certificates, the transition from unauthenticated to authenticated is less visible, which leads to the now realized possibility that an attacker could splice together the initial and later traffic into what the application sees as a single session.

Arguably, protocol developers should pay closer attention to features

like this that end up being used for purposes for which they weren't originally intended. We could also conclude that protocol developers should more carefully consider what internal states of the protocol should be visible to that protocol's consumers, rather than simply convincing themselves that the protocol can be used “securely.” Finally, protocol developers should clearly more carefully consider whether and how to cryptographically bind different phases or parts of complex protocols like TLS.

Probably psychological issues also play a role here, in that we tend not to focus on what we perceive as secondary use cases when considering deployed protocols. In this case, the use of certificate-based client-authentication (which uses handshake renegotiation) is perceived as rare and so, presumably, receives less attention as a potential source of vulnerabilities in TLS. One way to counter this might be for security protocol developers to build up a set of antipatterns (such as the man-in-the-middle attack I described earlier) that they could compare to protocol proposals.

In this case, it's hard to reach definitive conclusions because I and a few people I've asked really aren't sure why we didn't spot this flaw in TLS earlier. However, as protocol developers, we might benefit in the future if we try to talk more with people doing more formal security analyses, if we look more closely at things being used for purposes other than those for which they were originally intended, and if we build up a set of antipatterns and occasionally check our work against those. We should also consider whether what we think of as a protocol's internal states should actually be exposed to applications, and we should probably include additional cryptographic bindings between different parts of

complex protocols as a general feature even if we aren't quite sure why we need them at the outset. In the case of this particular vulnerability, we can expect security and application providers to update TLS clients and servers, in the short term, to include relevant fixes. Administrators should plan to deploy these updates to continue to get the real benefits that TLS brings to the Internet for, hopefully, at least another decade. □


### Acknowledgments

I thank Marsh Ray, Steve Dispensa, and Nasko Oskov for comments on a draft of this article. All errors, of course, remain my responsibility.

### References

1. T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol, Version 1.2,” IETF RFC 5246, Aug. 2008; [www.ietf.org/rfc/rfc5246.txt](http://www.ietf.org/rfc/rfc5246.txt).
2. M. Ray and S. Dispensa, *Renegotiating TLS*, tech. report, Nov. 2009; [http://extendedsubset.com/Renegotiating\\_TLS.pdf](http://extendedsubset.com/Renegotiating_TLS.pdf).
3. E. Rescorla et al., “Transport Layer Security (TLS) Renegotiation Indication Extension,” IETF Internet draft, work in progress, Nov. 2009.
4. L.C. Paulson, “Inductive Analysis of the Internet Protocol TLS,” *ACM Trans. Information Systems Security*, vol. 2, no. 3, 1999, pp. 332–351.
5. S. Gajek et al., “Universally Composable Security Analysis of TLS,” *Proc. 2nd Int'l Conf. Provable Security*, J. Baek et al., eds., LNCS 5324, Springer-Verlag, 2008, pp. 313–327.

**Stephen Farrell** is a research fellow at Trinity College Dublin and chief technologist with NewBay Software. His research interests include security and delay/disruption-tolerant networking. Farrell has a PhD in computer science from Trinity College Dublin. Contact him at [stephen.farrell@cs.tcd.ie](mailto:stephen.farrell@cs.tcd.ie).

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

This article was featured in

# computing **now**

ACCESS | DISCOVER | ENGAGE

For access to more content from the IEEE Computer Society,  
see [computingnow.computer.org](http://computingnow.computer.org).



IEEE  computer society

Top articles, podcasts, and more.



[computingnow.computer.org](http://computingnow.computer.org)