The SSL/TLS Protocol

Description: Steve and Leo plow into the detailed operation of the Internet's most-used security protocol, originally called "SSL" and now evolved into "TLS." The security of this crucial protocol protects all of our online logins, financial transactions, and pretty much everything else.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-195.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-195-lq.mp3

INTRO: Netcasts you love, from people you trust. This is TWiT.

Leo Laporte: Bandwidth for Security Now! is provided by AOL Radio at AOL.com/podcasting.

This is Security Now! with Steve Gibson, Episode 195 for May 7, 2009: SSL. This show is brought to you by listeners like you and your contributions. We couldn't do it without you. Thanks so much.

It's time for Security Now!, the show that covers all things security and privacy related, with our host Steve Gibson of the Gibson Research Corporation, GRC.com - creator of SpinRite, discoverer of spyware, and man about town. Soon to be man about town with...

Steve Gibson: Well, man about Starbucks.

Leo: Man about Starbucks. Yeah, that's a good...

Steve: I don't really get out on the town. I'm pretty much at Starbucks. In fact, today I was only there for five hours this morning. As I was walking out at 10:00, Jackie, who opened the store at 5:00, and I helped unlock the front door because I'm always there at opening, she said, "You're leaving early."

Leo: So wait a minute. Let me get this straight. You go there at 5:00 a.m.?

Steve: Yeah, every morning I'm there at opening at 5:00 a.m.

Leo: And that's breakfast for you.

Steve: Well, it gets me up and going. And my new deal now I'm so excited about is I've got my Northgate OmniKey 102 keyboard that I have under my other arm. And I've been spending 10 to 11 hours a day there.

Leo: What? Oh, so you're programming there instead of at home.

Steve: Yeah, yeah. In fact...

Leo: Why is that? You like the people around and the bustle?

Steve: Well, I spent 10 years sort of, you know, all by myself at home. And that was good. But there's actually more distractions here than there are at Starbucks, even though...

Leo: Interesting, yeah.

Steve: Even counting the UCI cuties who come across the street and hang out at Starbucks. I mean...

Leo: That's just good for the soul.

Steve: Oh, yes, it's - it gives me a little bit of a break.

Leo: When you get to our age, folks, it's not as much of a distraction as it used to be. It's like, uh-hmm, and...

Steve: Oh, okay.

Leo: ...back to programming. Now, are you going to bring - I know you've ordered the new Kindle DX, the big one. Are you going to bring that around? Is that going to be your new, like, newspaper that you carry under your arm with your Northgate keyboard?

Steve: I don't - I'm not sure. I've actually put my Kindle down. I haven't looked at it much for the last few weeks. I mean, I've really been having a ball, getting a ton of coding done.

Leo: On, that's so neat.

Steve: And there's a very good chance that all of CryptoLink will be written at Starbucks, which would be kind of fun.

Leo: That's really cool. Well, you wouldn't be the first program to be written in a coffee shop.

Steve: Nope.

Leo: It's very common. In fact, we've kind of been looking at larger spaces for - maybe not for another year or so. But at some point...

Steve: Like some retail space you were talking about.

Leo: Yeah. And one of the things I would like to do is kind of have a co-working space there so that people in the neighborhood who are smart techies would have a place they could come, they could use the WiFi, we'd probably have a little coffee bar there. Because people do, I think you're right, I think even though you're working on your solitary thing, you don't want to be completely alone. It's nice to have a little - people wandering around and making noise around you.

Steve: Yup. And there's a bunch of Starbucks regulars there. There's another guy, Robert, who's also always there at 5:00. He's studying French, having mastered Spanish.

Leo: Oh, that's neat.

Steve: And he goes out and paddles after a couple of hours, and so he's there. So, you know, we - and there's a bunch of regulars. And it's sort of a little bit of social dip. But I just can't believe how much I'm getting done. I'm just getting so much work done. And I'm a little giddy afterwards. I think, my god, this is what I'm supposed to be doing. It's not like I'm playing hooky.

Leo: Isn't that nice. That's a nice feeling. And not only that, you're doing it, and nobody's making you do it. You're not on deadline. You get to really enjoy the creative process, knowing that you're going to make something that people will use.

Steve: Yup. Wait till you see it. I think in two weeks I'll be, hopefully, unveiling this thing that I've been working on, this DNS benchmark.

Leo: That soon. Oh, cool. That's cool. Well, let's get down to business, to brass

tacks, as they are wont to say.

Steve: Yes, this is Episode 195.

Leo: Wow. What are talking about today?

Steve: Well, I've been talking about what we're going to be talking about as the SSL Protocol, which is really a misnomer because it really is TLS.

Leo: I see that setting in my email program sometimes. It'll say SSL via TLS, or TLS/SSL or something like that.

Steve: Right. It's really - TLS is Transport Layer Security, which is the renaming of the SSL, the Secure Sockets Layer, which was the name originally given to it by Netscape back in '94 when they originated this. So with everything that we've learned, all of the background that we have in the previous 194 episodes, I'm going to describe today in sufficient detail that everyone listening who has been following along will really have a good sense for what the number one most popular, most used security protocol on the Internet is, how it works, in lots of fun detail.

Leo: Cool.

Steve: So that's today's topic.

Leo: Any - oh, goodness. There's two of me. Any errata to address before we get to the news of the day?

Steve: Well, having discussed the Kindle - I had that on my notes to discuss.

Leo: That's pretty - we didn't discuss it - now, we should say we discussed it before we started the show.

Steve: Oh.

Leo: So let's go through just a little bit about why you think this is an important thing. First of all, why did Amazon, hard on the heels of announcing Kindle 2, do another Kindle? I mean, I just bought a Kindle 2. I just bought one for my mom a day before the announcement. Should I send it back? Should I say "Don't open it, Mom, send it back"?

Steve: Well, okay. I don't have the DX yet, so I don't know.

Leo: Okay, so you're not reviewing it.

Steve: Right. And I only learned about it minutes before we began recording. A friend of mine said, "So, do you have your DX yet?" And I said, "My D who?" So what I know about it is that it has a 9.7-inch diagonal screen instead of the 6.something that both the first and second-generation Kindles have; that it has a higher pixel resolution, so the individual pixel pitch is tighter; and, significantly, whereas both of the first two Kindles did not have native PDF reading, this one does. Which is a big feature for me.

Leo: This is their - I understand this is their way of getting into the textbook market, basically. You need to have this size.

Steve: Yes. Presumably. I mean, I, for what I was doing this morning during my five hours at Starbucks, was re-reading the RFC. It's RFC 5246, which it's the largest RFC I think I've ever seen. It's 104 pages long. And this is the detailed specification for v1.2 of TLS, the Transport Layer Security, which is the topic of this week's podcast. And I printed it out. I printed it out double-sided to save paper. So there's 52 sheets of paper. I would have much rather written it to a PDF, stuck it on something that I could read. Now, yeah. I could do it on a laptop. But this DX, the fact that it is able now to render PDFs, and the screen is large enough that you can actually see the content of the PDF that's rendered, now I'll have a portable PDF reader. And...

Leo: So when I email a PDF to my Kindle, as I have just - as I've done recently, it's converting it.

Steve: Correct. It converts it to its native format, to the Mobi book, the Mobipocket format.

Leo: And why is that not okay?

Steve: Well, look at the PDF, and you'll find that it just - it didn't make it. There aren't enough pixels to render a typical full page that the PDF assumes, and PDFs are non-reflowable. Whereas books can be dynamically reflowed because really a book, a textual book is just a stream of characters. And so when you change the font size, for example, you're able to reflow so that you repaginate. But, for example, you can't change the font size on a PDF because - you can scale the whole page, but you can't change the font size because the PDF is a page layout specification.

So anyway, I'm excited. And I know a number of my other techie friends, for example Mark Thompson, the one question he had about - of AnalogX. The one question he had about the Kindle was will it display PDFs? And I said no. This one does. The DX does. And it's...

Leo: Because its page size is big enough to show an 8.5x11 or an A4 page without reflowing it.

Steve: Well, with enough resolution.

Leo: So you can just put the - basically, yeah, because a PDF is a picture of a page, basically.

Steve: Right, exactly.

Leo: You can put the picture of the page on it.

Steve: You can certainly put that on your Kindle. And I did. Kindle 1, which has the same resolution as the Kindle 2, I did put a PDF on it. And I thought, okay, well, I'm not doing this again.

Leo: Okay. Well, that's kind of intriguing. But I'm a little worried. I'm going to wait till you get yours because I'm a little worried that it's just too big because I mostly read my Kindle in bed or on an airplane and stuff like that. And I'm a little concerned that it might be a little big.

Steve: Yeah. I'm reading textbooks that often have diagrams. And the diagrams don't survive very well on Kindle 1 or 2.

Leo: Yeah. Yeah, I'll give you that, yeah.

Steve: You have to get out a magnifying glass or squint and so forth. And so that's another example where this DX, it will be much better for books containing diagrams.

Leo: Well, I'm glad you're the guinea pig.

Steve: Hey, for the first time ever, Leo. Normally you're...

Leo: Normally I'm the early adopter. If I hadn't just bought that Kindle. I'm a little miffed, frankly. But I guess I could have Mom send hers back. Because she does want the big type. I just don't know - she tried my Kindle 2 and loved it.

Steve: Yeah.

Leo: [Grumbling]

Steve: Well, it's hard to say. I mean, for your mom, she's probably, what, in her 80s, like mine?

Leo: Right. No, late 70s, yeah. Mid 70s.

Steve: Yeah. At that point, you know, having a bigger page with bigger fonts, I think there's some advantages there.

Leo: Yeah. Well, she had stopped reading books, and she couldn't read even papers for very long. And when she tried the Kindle, she read a whole book while she was visiting last week because she could make the font size a little bit bigger. But it's big enough on the Kindle 2. So I don't know. Well, I'm going to watch you. Thank you for being the guinea pig.

Steve: Happy to be. And it's pre-order time. It's not like they're shipping them yet. They're saying they'll be shipping...

Leo: Right, summer.

Steve: ...sometime this summer.

Leo: Academia time.

Steve: I immediately logged myself in to get in the queue. So I'm excited.

Leo: You know what, this really is exciting for school books, textbooks. They're very, very expensive. Kids carry 20 pounds, 30 pounds, 40 pounds' worth of books in their backpack. This is a much better solution. If Amazon can get those book prices down.

Steve: Well, and Leo, I mean, I just - I recently purchased a tablet PC as part of my new Starbucks kit because I'm using an external keyboard, the old Northgate OmniKey 102. So I got them - I bought the tablet from Motion Computing. They're a well-known, reputable tablet maker. They sort of like do the higher end industrial tablets for, like, vertical markets, for hospitals and construction workers and things, really well built. I really like it. It's a 12.1-inch screen, 1400 by 10-something resolution, so lots of resolution. Of course it's color. It runs Windows. I mean, it's a perfectly adequate PDF viewer. You can run it in portrait mode instead of landscape. And, I mean, so it's not like solutions don't already exist which would be totally adequate, except that that doesn't run for two weeks without a charge.

Leo: The other thing that's going to be interesting...

Steve: [Indiscernible] run for two hours.

Leo: ...is we haven't heard Apple, from Apple yet. And next month Apple's going to come out with something, I suspect. And many rumors are that it might be a reader, or at least have reading functionality. So we'll see. We'll see. It's all very interesting. What else is in the news?

Steve: Well, we have a significant Adobe zero-day flaw.

Leo: Oh, a second - another one, like the one in February.

Steve: Another one. And in fact this is bad enough that it's now time for all of our listeners to do the following: Click on any PDF that you've got on your system that will open the Reader. And this affects both Reader and the full Acrobat product. Go to the Edit menu, which is the second one over. File is the first; Edit is the second. Choose Preferences. There's a, now, you're going to have a big dialogue with bazillion categories down the left-hand side. Fortunately they're alphabetical so you can go to "J" for, not surprisingly, JavaScript.

Leo: I know where you're going on this one.

Steve: My number one nemesis, JavaScript. Select that, and the first checkbox is Enable JavaScript. Unfortunately, it's on by default. Thus the problem. Just turn it off.

Leo: You know what's sad, there is no reason for you to have JavaScript in a PDF.

Steve: Exactly. PDFs do not need JavaScript. And so here again we've got - found in the wild was a remote code execution exploit, another one. I mean, Adobe's been having a bunch of problems, not only in that but in Flash. I mean, so all current versions of Reader and Acrobat, which is - I'm still on v8, as we've discussed. So mine, the one I've got, 8.1.4, is vulnerable, 9.1, and 7.1.1. So both 7, 8, and 9 releases on both platforms, Reader and Acrobat, are vulnerable to this. It was a zero-day exploit, meaning that they discovered it because it was being used to run code on people's computers when they viewed PDFs.

Fortunately I had already taken my advice, and I had JavaScript disabled in my Reader. Everyone listening to this should just turn off JavaScript. You'll lose nothing because who even knows, I mean, I don't know that I've ever seen a PDF that had JavaScript in it. You know, it's for reading documents. So turn it off, and then you don't have anything to worry about. And we'll let you know. Apparently next week from this podcast, so here we are, this is May 7th is the date of this podcast. So a week from now they should be pushing out a fix for this. They're working on it feverishly. Fortunately our listeners, who are smart enough to just disable JavaScript, can protect themselves right now.

Leo: Many of our listeners use Foxit and don't use Adobe Reader or Acrobat.

Steve: And boy, Reader is 200-and-some megs in size. It's just - I installed it...

Leo: It's the JavaScript engine. Why?

Steve: ...installed it on this new little tablet of mine, and I downloaded the latest one, 8.1.4. And it was - I think it was 214 is the number I'm remembering. It's like, yeow, just to read PDFs I've got 214 - well, that was the download size. Who knows how large it blew itself up to when it landed on my hard drive. So anyway...

Leo: It's just - it's remarkable. And I don't use - I stopped using Reader after the last problem. But just, boy, you know...

Steve: And so have you been using Foxit?

Leo: Yeah, Foxit's great. I've used Foxit for a long time.

Steve: I think maybe - maybe I'll just switch. I mean, you know...

Leo: Not that they're immune from exploits, either. But this is two in a row.

Steve: Well, and I'd rather have something smaller. I just don't want 200-plus megs of bloat in order to read a PDF. That's nuts.

Leo: Yeah. No, I'm with you on that.

Steve: So the only other thing I had was a really short little note from a happy SpinRite user, Doug Davis, who actually wrote to us on Halloween, and I had his in my list of testimonials to get to. He just said, "I'm a very happy owner of SpinRite. I have a laptop that would just shut down whenever the virus scanner tried to scan the disk. Other scanners and disk checks caused the same result. At first I thought a virus was compromising my machine, thus shutting it down so that it couldn't get scanned. But finally I bought SpinRite on a hunch. I ran it, and it magically fixed the disk, and my virus scanner now works again. Another happy ending. Thanks for this great product. Regards, Doug Davis."

Leo: Excellent.

Steve: So you never, I mean, I wouldn't have even recommended SpinRite in that case. But that's what the problem was, and so he was able to fix his system.

Leo: I'm hearing from our chatroom that Foxit also has JavaScript enabled by default.

Steve: Eh, well.

Leo: Not the same - there isn't the same bug, of course. That's a bug in the Adobe implementation.

Steve: Right.

Leo: But there must be something that's going on in PDFs that there's a need for JavaScript. I can't...

Steve: No, not necessarily. Could just be bullet point-itis. That's what bloated ZoneAlarm up to the point that it was no longer useful or recommendable.

Leo: They've got JavaScript, so we have to have JavaScript.

Steve: Well, in the case of ZoneAlarm, of course, they were competing with Symantec and McAfee that were just - that were turning their own personal firewalls into kitchen sinks. And, I mean, I was complaining to Gregor. I said, "Gregor, don't screw up ZoneAlarm by putting all this other nonsense in it." He says, "I hear you, but I have to. We have to compete." And so it may be that Foxit went out and got some JavaScript engine from somewhere and stuck it in, just so they had it, too, even though it'd be nice if they didn't, and it would be smaller if they didn't.

Leo: It's crazy. Oh, well. These are the things we have to live with. All right. Let's talk about SSL, Steve Gibson.

Steve: Well, we really need to talk about TLS. So SSL is, because it was what came first, it was - it's what we still refer to this protocol as, commonly, although it's really not the case actually that SSL is in use. TLS has formally and completely replaced it. It is substantially more bulletproof. SSL has an interesting, somewhat spotty history, more so than is commonly known. I mean, we've beaten up the early WiFi protocol, WEP security, pretty badly. We've never really done the same for SSL.

SSL is a specification for securing any reliable protocol on the Internet. Meaning, for example, PCP, as we've discussed, is a reliable protocol because even though the connection itself is not reliable, meaning that routers that are briefly overloaded have, by permission, have the ability to drop whatever packets they're unable to route. They only make a best effort. Routers make no guarantee. So an unreliable protocol, that is to say a non-reliability-guaranteed protocol, for example, like UDP, which just sort of sends packets out, you cannot run, for example, SSL or TLS over UDP.

Turns out there is lately a variant that is a packet-based, a so-called datagram-based SSL which is designed to run over a non-reliable protocol. But SSL or TLS, as it's been renamed, does assume that anything that it sends will get to the other end. So that is to say that the underlying protocol, which is normally TCP, if packets are lost in transit, TCP will take responsibility for retransmitting them. And at the receiving end the TCP protocol will assemble them in the right order and end up presenting to the next level up, which is

TLS or SSL, it'll end up presenting sort of a reconstructed error-corrected stream. And in fact it's worth talking a little bit about this notion of layering because that's inherent in the way SSL/TLS works. In fact, instead of using both acronyms from now on I'm going to try to say TLS, since that is the protocol that we're all using.

All contemporary web servers support the latest version, which is TLS 1.2. Firefox 2 and 3, IE7, and the latest versions of Opera, all the current browsers support TLS, as do servers. There's a nice technology that we'll be talking about for sort of dropping back. So the idea is that you will - the browser and the server will always negotiate the latest version and the best strength of available ciphers and hashing and signature and public key technology that each of them know about. So that there's a - it's been really well designed so that the resulting connection is the lower of the highest versions that both support. So it ends up doing the right thing.

But we have this notion of the physical connection, the actual - the protocol on the wire is, for example, often Ethernet, where we know that addressing is by MAC addresses. And then running on top of that protocol is the IP layer, which provides - where addressing is by IP addressing. But so IP protocol runs on top of the Ethernet protocol. And then it is the host for the IP protocol suite. For example, ICMP, which is like the ping and traceroute uses that, is a protocol running on IP. UDP and TCP that we've talked about are two other protocols running on IP. So in this case we've got Ethernet at the bottom, sort of on the physical wire, then the IP protocol to allow IP addressing. Then that hosts TCP, which provides both this abstraction of an IP address and then this notion of port numbers so that you have a 16-bit port number. So that's where portage sort of comes from.

Then normally you would then run your application layer, that is, your application protocol, which would be, in the case of a web browser and server, it'd be HTTP. Or it could be FTP or Telnet or whatever. That is, those are protocols, application-level protocols, that run on top of TCP. They all assume a reliable TCP connection so that they're not worrying about lost packets and so forth. They rely on the underlying protocol, TCP, to do that for them.

Well, what TLS does is it inserts itself in a transparent way between the TCP layer and the application layer. So that essentially it's another wrapper, or sort of another little shim in this layered stack of protocols. And so it relies, as I was saying before, on the underlying layer, TCP, to provide it reliability. So it needs and assumes that it's running on a reliable protocol, TCP. And what it does by inserting itself between TCP and the application layer is it provides a number of services to the application layer, to HTTP, turning it into HTTPS, or FTP if you have, for example, secure FTP or secure Telnet. That is, any application is able to run on TLS and get all the benefits of security that TLS provides to any client application.

Now, this was originated by Netscape, and they pretty much did a good job. They started off in 1994, which is 15 years ago our time, at this point, and it's interesting. They're not security people, but they were smart people and tried to do a good job. They came out with SSL v1.0 that was, unfortunately, so broken that it never saw the light of day. They produced a formal specification and fortunately got some feedback from the Internet community before just going public with it. And that really got the crypto guys involved, who looked at what they'd done and said, oops, you can't use this. The crypto guys immediately saw a bunch of problems with the things that Netscape was proposing. And so SSL 1.0 never happened. Sometime later, in fact I think it was February of '95, Netscape took advantage of all the feedback they had received on the never-released 1.0 and did produce v2.0. So SSL v2.0 is the first version of SSL, Secure Sockets Layer, that anyone ever saw.

So what does it provide? Well, we sort of know from all the discussion that we've had over the last nearly four years what we rely on SSL for. We know that we rely on it for confidentiality, that is to say, encryption. It has ciphers in it such that no one monitoring the line is able to obtain any information, is able to know what it is that we're sending back and forth. One of the nice features of both this original SSL and today's TLS is that all of the connection and handshaking is done before even the first byte of the application layer data is transferred.

So TCP, the layer below this security layer, TCP does its SYN, SYN-ACK, and SYN, the standard three-way handshake, typically involving three packets, which establishes sequencing and confirmation of a connection. At that point SSL performs its handshake and negotiation, which is what we're going to talk about in detail here in a couple minutes. And all of that gets done, and both endpoints of the security protocol are - they need to be satisfied and happy and established, essentially, prior to the application layer that's been waiting all this time. The application layer initiated this, but no actual traffic is emitted at either end until this secure layer has been established. So nothing of what the application is doing is able to leak out into, you know, in any way out of the LAN or the WAN, the Internet, anywhere. So it really does wrap the entire dialogue in a confidential tunnel.

It also, as we know, both SSL and now TLS provide authentication. We've talked a lot about server-side certificates, and we understand that a certificate is identification information and a public key which has been formally signed by somebody that the recipient of the certificate trusts, and that allows them to verify that nothing has been tampered with. And that is a way for them to obtain the, for example, in the case of a typical web browser and server, it's a way for them to obtain the public key that matches the secret and private key of the server, which is used by this protocol in a way that we'll see here in a minute. So that provides authentication and some cryptographic credentials.

It also provides tamper proofing, which is important because it's one thing to know that the message is secret and that it's authenticated. But if it was possible to tamper with the message, then we're not so happy because there are various hacks that can be used if you could tamper with the stream that could still cause trouble, even though the message is confidential and authenticated. And in fact that's where SSL 2.0, it's one of the places where it tended to fall down. It also provides proof against message forgery so that nobody is able to, for example, do a replay by sending packets again and forging packets from either of the endpoints. So all of the packets are serialized through the entire connection, and both endpoints make sure that they never get a duplicate serial number, no serial numbers are missing. Because again, another - you could imagine some sort of a clever attack where somebody would - who, again, who can't see into the packet, who can't pretend to be someone generating the packets, might take some out and in that way again compromise the communication. So SSL and TLS provide proof against that, as well.

So 2.0 had some problems. It was good, and we used it for a long time. And there weren't any highly publicized breaches in SSL 2.0 because it was well designed. But, for example, it uses the MD5 hash.

Leo: Whoops.

Steve: Well, that wasn't a problem back then.

Leo: We know it's a problem now.

Steve: Yes, exactly. We know that there are all kinds of problems with MD5. So we had to move past using an MD5 hash. SSL v3 sort of fixed it a little bit by coming up with a clever solution. They hashed both - they produced both an MD5 hash and an SHA-1 hash, and they XORed the result. So that means that the result sort of hybrid hash depended upon both, which was an interesting solution because it meant that either one could be compromised; but since the result was an XOR with the other, presumably uncompromised hash, you'd still get the full strength of the surviving uncompromised hash.

Leo: That's pretty clever, actually.

Steve: It is, really clever. However, the sense was, eh, you know, that's kind of a kludge. And TLS, which is where we are now, it started off as SSL 3.0. So it was sort of just a renaming of 3.0 into TLS. And it was - when it went to TLS is when the IETF, the Internet Engineering Task Force, took it on as a formal standard. They said, okay, this is important. We're going to take this under our wing. We're going to rename it in the process. But 3.0 looks pretty - SSL 3.0 looks pretty good, so that's what 1.0 - that's what TLS 1.0 is going to be.

Well, we're now at 1.2 because a number of improvements and refinements have been made. For example, the use of SHA-1 and MD5 is gone completely. We're now using just SHA-256, which is state of the art, much stronger than either the MD5 or the SHA-1 hashes. And those, again, can be supported in the case of downgrading your connection to an earlier protocol. But given that both endpoints support TLS, that won't happen.

SSL 2.0 also had a problem in that it used identical cryptographic keys for both message authentication and encryption. And while it's not a horrible thing to do, it's just not good. The crypto guys worry that using the same keys for two different purposes is fundamentally less secure than using separate keys for separate purposes. So one of the things that 3.0 did, SSL 3.0, is that it said, okay, we can make keying material at will. So let's make more keying material and use separate keys for authentication and for encryption. So that's one of the other things where we moved forward from v2 over to 3.0.

There was also an interesting attack that was discovered. And again, it's sort of a theoretical problem where you could - a man in the middle could intercept and mess with the handshaking phase of SSL 2.0 where the two endpoints are negotiating the cipher that they're going to use and the authentication. Basically - and I'll go into this in more detail in a minute. But essentially the client sends a list of all the ciphers and hashes and public key and key exchange technology that it knows about. And so it sends this list off to the server and says, here's how smart I am. And basically it's - so the server knows all the different suites of protocols that are being offered to it, essentially, to use by the client.

The server then looks at its own list and in every case chooses - and this is a hierarchy - chooses the best of each of these categories that it also knows about. So that's how the client and server are able to arrive at, like, the best way they each know of speaking to each other, yet in a way that allows a less capable endpoint to still establish a connection. It's not as good, as strong a connection necessarily, if you wanted to get

really picky at crypto. I mean, even, like, the least secure is, like, way secure. But it's not state of the art. So that allows endpoints with different knowledge of cryptographic suites to still be able to negotiate a connection.

Well, it turned out that unfortunately this negotiation was not adequately protected. So there was an attack possible where a man in the middle could essentially trick the server into believing that the client was much less capable, in fact virtually incapable of any useful security and therefore essentially really interfere with the resultant strength of the connection. So one of the other things that was fixed in 3.0 was that that initial handshake, that we'll again be talking about in a minute, was strengthened to prevent any kind of modification and man-in-the-middle attack.

Essentially there's now a Finish message that each end exchanges which involves the hash of everything that they've said to each other, which is really very clever. And that's encrypted based on the encryption keys and protocols that they have established. And so what that does is each end is hashing everything it sends and hashing everything it receives and concatenating that. And then the idea is that each end is able to verify that hash, which is a hash over the entire exchange so far. So if anything was altered or modified in transit, the other end won't - the other end will have something different than the local end believes it sent. Thus the hash won't match, and the endpoints completely shut down their connection and begin a renegotiation. So it's much stronger now than it was before.

And it's interesting because SSL 2.0 didn't have this notion of any sort of a finish. It just, well, it wasn't providing that kind of handshake protection. And also the overall connection didn't have a "okay, we're finished with this connection." And it turns out that there were some clever attacks that involved truncating a communication. And in SSL 2.0, when TCP shut down, that is, the underlying protocol shut down, then the protocol above, that is in this case SSL 2.0, it said, okay, I guess we're done, and it sort of wrapped itself up. Well, now there's a formal end-of-message communication added in at SSL 3.0 in order to say, okay, this is really the end. I've been told by the application layer that we're done now, so it's okay to shut down. Which was missing from 2.0, that was added in 3.0.

And finally, there's a feature that still hasn't been implemented. It's funny because I was, in reading some of the commentary on the spec, there was some commentary that mentioned that, gee, you know, this feature hasn't - no one's used it yet, but it's in there, and it'd be really nice to have. And that is, there is a provision that appeared in 3.0 and has survived in TLS that allows more than a single service. That is to say, SSL 2.0, and actually all practical implementations because we're still not using it, as I said, assumes that everything is known about the remote endpoint, that is, that the identity of the remote endpoint is known. So that when, for example, the client connects to a server at a given IP, that server responds with its certificate. But this causes problems with virtual hosting because, in a virtual hosting environment that we have discussed in the past, you can have many different websites located at the same IP. And it's the host header in the HTTP protocol that specifies which hosting website you want to connect to at that IP. If you don't have that, if you're unable to use that, which is the case now, you can't - you have a problem with SSL certificates because you can only bind a single certificate to a single IP. And so there isn't a good way to do virtual hosting.

What some people have done is used wildcard certificates, where it'll be essentially asterisk dot, and then an underlying domain dot com, and then other domains are defined. So, for example, you might have webservice1.commondomain.com, webservice2.commondomain.com. And so the certificate would be *.commondomain.com. Well, that provides you the ability to establish an SSL connection

to any of the subsidiary different domains. But suddenly now you've lost a lot of protection because you don't know what domain - you've lost the authentication because you're authenticating to, essentially, to a parent domain, not to the subdomains, which is what you really want.

The point is that with 3.0, although it's never been used, and also in TLS, there is the ability for the client to specify which certificate it would like to receive from the other end. So there is really nice native support for a virtual hosting environment. And the reason I was chuckling was that, well, none of the certificate people are anxious to offer something like that. They would much rather sort of force a certificate per IP and sort of leave things as they are. It's interesting to me that this facility exists, yet no one has implemented it.

So all of these things that were wrong with 2.0 have been fixed in 3.0. SSL 3.0 was then, as I said, sort of taken over by the IETF. And that was, in terms of timeline, we got SSL v2.0 in February of '95, 3.0 in '96, and then the IETF, it was in '99 that the IETF took it over and began to sort of take it under their wing and create a more formal specification. So I read, or reread, the specification this morning.

Leo: On your Kindle.

Steve: No, remember, I don't have my big-screen Kindle yet.

Leo: Oh, that's right. On your tablet, yeah.

Steve: So I printed it out in old school - it's 104 pages of really dense stuff. I mean, it's not - it's the exact opposite of light reading. But I have to say it is - at 104 pages, that makes it one of the largest RFCs I think I have ever encountered. But it is just - it's a pleasure to read. It is a beautifully put together specification. I mean, and in the beginning it talks about how this is - we've produced this document for people who actually have to implement a working TLS v1.2. Everything is here that you should need. And it's just - it's laid out just in a spectacular format.

So TLS further improved on v3.0 security. They continued to tweak things. More than anything there was more - it's sort of a more rigorous specification. There's a formal approach used in RFCs where in the interest of improving interoperability, there's this notion of using the terms "may," "should," "shall," "shall not," "will not." And, I mean, they've got real exact meaning in terms - sort of within the RFC specification protocol itself. And so as you're reading RFCs it'll say such and such and such "may" provide this or "should provide" this or "must not." And so - and often these are in capital letters because, I mean, they really intend for readers of the RFC to pay attention to that sort of thing.

Well, there always was some ambiguity that caused implementation problems in previous versions of SSL, so that people tended to copy each other's implementations rather than doing it from scratch themselves because they may not interoperate because the specification really wasn't rigorous enough. It didn't really clearly specify exactly how everything should work, sort of in those edge cases, in the boundary cases where it's like - where a programmer goes, you know, who reads the spec goes, well, wait a minute. You've sort of said "greater than or equal to" here. But over here you said "greater than." So what exactly do you mean?

Leo: Which is it?

Steve: Exactly, which is it? I mean, programmers, and of course who...

Leo: And computers.

Steve: Who at that point are reflecting what the computer is thinking, you can't have it be either way. It's got to be one or the other. And that's typically where code falls down. So TLS has really worked to make that a lot more strong. It discarded in the later versions the use of this hybrid MD5 and SHA-1 approach so that it's just not even in there anymore. There are alert messages where either end can send each other warnings when, like, something doesn't look right. There are many more alerts in TLS. And whereas before there were, like, "should" verbs tied to those, now they are "must" verbs. So it's like you're more sure that, if there's a problem, the other point, the other endpoint will - the other implementation will be notifying you for sure that that's going to happen.

Also the pseudorandom sequence generator, both inherent in crypto, we talk about cryptographically strong pseudorandom number generators. Crypto needs a lot of random numbers. They are used, so-called "nonces" is used once and often encrypted using public key technology in order to provide a secret, even over a channel that may have eavesdroppers listening in, that can be received by the other end, decrypted using some pre-agreed upon cipher, and so the other end is able to get that information.

So again, TLS has a more robust and stronger pseudorandom number generator based on the HMAC. And in fact one of the reasons I had been - we talked about HMAC some time ago, and I declared it as the final thing we needed in order to talk about SSL, or TLS, is that it's TLS that uses the HMAC, that is, the keyed Message Authentication Code. And remember that the way that works, real briefly, is that you take something that you want to hash. You append a particular hex pattern, I think it was 5C, a block of 5Cs, and then you hash that - oh, and I'm sorry, you append your key and then the 5Cs and then the content, which you hash. And then you take the result of that, and you use the key, a different hex pattern, and the result of the first hash which you hash, and that's the result. So that this is the so-called HMAC.

And it is so strong, and people are so happy with it, that they made that the basis for the hashing in TLS and even the source of pseudorandom data. That is, you start with a seed and a key, and you run HMAC on it in order to generate some material, and then you HMAC that along with the same seed to generate a next chunk of material, and you HMAC that with the seed as many times as you need to as your source of pseudorandom data. And this keyed MAC, this HMAC is good enough that what it's producing through just successive iterations of itself has passed all muster among the cryptographic security guys.

So, okay. So that sort of gives us an overview outline of the history of the evolution of SSL to TLS, where we are now. So let's talk about how these guys connect. There is a series of protocols which are just packets in an agreed-upon format that's sent back and forth. So when the client connects to the browser, the first thing that happens is a TCP protocol connection is established. Then, if this is going to be an HTTPS connection, for example, a secure connection, the first thing that happens - oh, the way the server expects that is the port that you connect to. As we know, web servers that are not going

to be using a secure connection will connect by default to port 80. If you are going to use a secure connection, by agreement you connect to port 443. So any incoming TCP connection to port 443 assumes that an SSL, or I should say TLS, connection is going to be made.

So the first data that the server receiving this connection expects from the client is the so-called "Client Hello" message. Now, packets and messages are sort of separate things because the packet is the unit of transmission over the Internet, but a packet can contain, I would say one or more, but actually even zero messages. There is a provision, sort of clever, in the current TLS to deliberately allow null traffic to be sent in order to confuse anybody who might be trying to do a traffic analysis. So it is even possible to send no application data, just sort of to send some blob, just to sort of, like, mess with the people who may be trying to figure out what's going on. I got a chuckle out of that.

But so the client sends its Client Hello message, which contains the highest TLS or SSL protocol version that it knows about, that is, that it supports. And this is the way, this is the first step in this negotiating for the best connection, the most secure connection possible. Also in there is a 32-byte random number. It's composed of the UNIX time at the client end. UNIX time is just a 32-bit value which is the number of seconds since January 1st, 1970. So it takes those 32 bits, or four bytes, and it appends that to 28 bytes that it has locally generated randomly. That produces a 32-byte random number, which it includes with the highest protocol level that it understands. It also sends a session ID. It makes up a session ID.

And what's interesting is that there's always been this notion of caching credentials at each end. We've talked often about how expensive public key crypto is. And, for example, in the case of a very busy server, which is inherently using a lot of secure connections, a lot of TLS connections, there's some tremendous overhead associated with the public key aspect of this negotiation. So there is the notion, both in the older SSL and now surviving in TLS, the notion of the endpoints being able to cache the result of that expensive public key work. And so what'll happen is, if the client is connecting to the same endpoint, that is, at the same IP, same web server URL that it had connected to before, so that is to say, if it still has credentials cached from a previous connection, which may have just been seconds or fractions of a second before, it will offer the session ID from that cached credential to the server. It doesn't - there's no guarantee that the server is going to accept it. But it'll make the offer, suggesting that, hey, I still remember you from just a second ago, and maybe we could dispense with some of this if we both agree to do that. So that's in there.

Initially in this same Client Hello packet is a list of all the cipher suites that it knows about, things like Triple DES, AES, various types of cryptographic packages, and also the compression methods that it knows about. There isn't much work that's been done on compression. And I've seen that sort of opinions vary about whether any of it's being done or not. You either have no compression, or apparently there is the ability to use the standard UNIX-style deflate compression. That's of course always done before you encrypt because, as we know, you cannot, by definition, you cannot compress anything that's been properly encrypted because it just looks like randomness.

Leo: It's random, yeah.

Steve: Randomness has no compressibility. And then optionally there is the ability for extensions to be added. One of the nicest things about SSL and TLS is that they've always provided, by careful design, extensibility. So that we don't have to break any of

this in order to move forward. That is, we're able to retire ciphers that, for example, if a cipher were suddenly discovered to have a problem, well, either end could remove it, and it could never again be used by that end or by anybody that it was going to have a conversation with.

Similarly, all of this sort of has a list-based architecture where new ciphers coming onstream can be added. Because this is an IETF standard, the IANA is the sort of the formal enumerator of these things because they end up just sending, like, a list of numbers. So you need to know what ciphers those numbers represent and which hashes and so forth. So that's the IANA establishes all of those things for TLS moving forward. So this packet, this Client Hello packet goes to the server saying this is the specification, the highest level specification I'm aware of.

Oh, and by definition all previous specs are known to state-of-the-art TLS client and server endpoints. But it's felt now that enough time has gone by since SSL v2 - which remember came on in February of '95, so, what, 14 years? - that most endpoints will not downgrade themselves to SSL v2.0. Some can be configured. But by default they'll go back to 3.0, but no further back. Even if they may know how to, they'll just - it's like, eh, no, we don't want to use MD5, for example, under any circumstances. So there's a limit to how far back they will go.

So this packet says this is the version I know of. Here's some randomness, 28 bytes of pure crypto random with this extra four bytes of UNIX time tacked on the front, just to give it a little extra, you know, non-reproducibility from one second to the next; the session ID, either one I'm making up just out of the blue or that may represent cached credentials that we have, that we may share; and then lists of all the different cipher suites, hashes, and compression methods that the client side knows about; plus any optional extensions which the specification formally allows.

So the server gets that and says, okay, now I know a lot about the client who is attempting to connect to me. It looks at the protocol level that the client is offering and looks at its own best protocol level, choosing the highest level that they both can agree on. It looks at the list of cipher suites and compression methods and similarly chooses - picks one, one from the cipher suite and one from compression, that it knows and that is the most advanced, most secure, best that the client knows. It does the same generation of a random number, producing a 32-byte random number. And it looks at the session ID and checks its own cache to see whether it is in agreement with the client that, hey, you know, the client is saying it's got this in its cache.

The server can either return a null session ID saying no resumption is agreed to, and by the way I'm not doing any caching at this end for whatever reason; it can return a new session ID, which is its way of saying, sorry, I don't have that in my cache, so this is the session ID we're going to use instead; or it can return to the client the same session ID that the client sent, which is the server's formal acknowledgment that they're going to have an abbreviated handshake because it still knows and is in agreement with the client about the stuff they had before.

So essentially the Server Hello message, which returns, has all of this decision made - the final protocol that they're going to use that's been agreed upon; its own 32-byte random number; the session ID that they're going to be using, which may be what the client sent, if they both have the prior credentials still in the cache; and the chosen cipher suite and compression method, chosen from among those that the client offered that the server end is able to agree upon. It also, often in the same packet, sends its certificate message, even though it's technically in the protocol as a separate message. As I said, you could have multiple messages in a packet.

So what may be in the same packet, though it doesn't have to be, will be its certificate message, which is essentially its certificate. It's offering its certificate, which normally contains the whole hierarchy of signatures back to the certificate authority. And of course the certificate has bound into it its public key. So that's its way of providing the first stage of authentication, saying here's the certificate that I, the server end, have. It also, well, and so that information goes back to the client end, and the server says hello is done. That is, my job with the hello phase is finished.

The client then receives both the Server Hello from the server and the server certificate and the Server Hello Done messages and says, okay, we're in agreement here about all this. This also tells it whether it's going to be using the same session ID or not. If the session ID comes back that it offered, meaning that the server has retained the prior credentials, then at that point they need to do no more negotiation. The client is able to send what's called a Change Cipher Spec message, which essentially says, okay, this is the last thing I'm going to send you that is not under the cipher data that we have agreed to. Everything else from me henceforth will be. And the server, upon receiving it, echoes the same thing back. It sends the Change Cipher Spec message back to the client saying, okay, now we go under encryption.

At that point they then exchange Finished messages, which since those follow the Change Cipher Spec messages are encrypted using everything that they have agreed to so far. And that's their final way of saying, of proving to each other that, for example, that these caches are valid, that all the encrypted data is present, and that they are in agreement and able and basically proving that they are both able to exchange encrypted, authenticated messages by sending this Finished message. And thanks to the finely evolved design of TLS, they're not - they will not allow any application traffic to pass. It won't begin doing any work on behalf of the layer above them that initiated all this until they have exchanged and verified these Finished messages. Oh, and the Finished messages also contain this hash of everything that - basically the hash of their entire dialogue, everything sent and received, in sequence, hashed together, in order to verify that no packets were lost or inserted or changed. And so that protects the whole handshake from any kind of malicious modification.

Leo: Wow. You came to an end. I thought that was all one long sentence. I heard a period. That is the most complicated thing ever. But it works.

Steve: Well, it does. And, I mean, it is complicated. But when you think about it, I can't see a way of eliminating any of that.

Leo: No, you need it, yeah.

Steve: Yeah.

Leo: No, I can see that, yes.

Steve: Yeah, you absolutely do. You want, I mean, this is the most used cryptographic protocol on the Internet. It's HTTPS. It's what we're all using when we're logging into web servers, and we want to be protected. And so you want a protocol which is going to

be able to grow, which is going to be able to accommodate differing capabilities on the endpoints because lord knows we've got all kinds of crazy clients all over here that are trying to connect to the Internet. So we've got Kindles, we've got phones, we've got iPods, we've got PCs, I mean, we've got refrigerators and all kinds of stuff. It's clear to me that as processing power increases, and as the pervasive threat presented by malicious actions on the Internet become more of a problem, that security and the robustness of these kinds of communications is going to be increasingly important. So, I mean, you can imagine a day where the notion of sending email over an unencrypted connection is just kind of quaint. You know, it's like Scotty trying to talk to the mouse in the early Star Trek...

Leo: "Hello, computer."

Steve: "The Voyage Home." "Hello, computer."

Leo: Use the keyboard.

Steve: The idea of not encrypting everything, which is where we still are today. We only normally jump to encryption when we need it, and often fall back to standard plaintext communication. And you really kind of wonder why.

Leo: Yeah, I wish we could use it all the time.

Steve: Yeah.

Leo: That would be nice, wouldn't it. I mean, it's not that - so it's not so much overhead that you really wouldn't be - not want to use it all the time.

Steve: No, I don't think so. I mean, sure, back when we were...

Leo: There's some overhead.

Steve: ...on 4.77 MHz 8088s, on a PC, that was painful.

Leo: Right.

Steve: But there's only a little bit of packet overhead to reach agreement. There's a little bit of work being done in order to establish this - basically what you're doing is you are establishing a - you are secretly and with full authentication establishing a shared secret symmetric key. Once you have, for example, an AES shared secret symmetric key, which you absolutely know no eavesdropper can access, then you're able to use a state-of-the-art, high-speed cipher like AES, which is, I mean, no overhead. I mean, fundamentally really low overhead and super-strong security for all of your payload traffic. And so once

that final agreement is reached, and the application that's been patiently waiting for all this to happen is allowed to finally send something, it's just encrypted using this shared secret symmetric key which is extremely high speed. And data goes across the wire, and nobody can figure out what you've done, no matter how much they want to.

Leo: Very cool. Very cool. Well, maybe we could just - someday we'll all use SSL all the time. Or TLS, as the case may be.

Steve: As I said, I think ultimately it will - the idea of not having encrypted secure communications will just be regarded as, oh, that's the way you did it then? How did you guys - how did you survive?

Leo: You sent everything in the clear? What's wrong with you?

Steve: Yeah, yeah.

Leo: Very interesting stuff. I really appreciate the effort and time you put into getting it all in detail. Now, I know a lot of people are going to want to listen again. There's good news on that front. You can. Just download the darn thing from the iTunes store or the Zune store. Or you can go to Steve's site and get it in 16 or 32KB - or, I'm sorry, 16 or 64KB versions at GRC.com/securitynow. But you'll also find a very useful tool for shows, particularly shows like this, the transcription. You can read along as Steve talks. And I find that really, really helpful. Those are also at GRC.com, along with all of Steve's show notes.

You can also find more information at our wiki, which is - and in fact we're always looking for people to help out with the wikis. You don't have to have even an account to edit it. So Wiki.TWiT.tv, and take a look at the show notes page there. And if there's something you have to add, links or so forth, we sure appreciate that, too. That makes it more valuable. Also Google searchable, which makes it easier for people to find this great information. Steve, thank you so much.

Steve: Absolutely a pleasure, Leo. And we'll do a Q&A next week, and who knows what the week after.

Leo: Get your questions in. Security Now! feedback is at GRC.com/feedback, so it's easy to do.

Steve: Yup. And I love getting feedback. It's just great to hear from our listeners.

Leo: Thank you, Steve. We'll see you next time on Security Now!.

Steve: Thanks, Leo.

Copyright (c) 2006 by Steve Gibson and Leo Laporte. SOME RIGHTS RESERVED

This work is licensed for the good of the Internet Community under the Creative Commons License v2.5. See the following Web page for details: http://creativecommons.org/licenses/by-nc-sa/2.5/