A Security Vulnerability in SSL

Description: This week Steve and Leo plow into a recently discovered serious vulnerability in the fundamental SSL protocol that provides virtually all of the Internet's communications security: SSL - the Secure Sockets Layer. Steve explains exactly how an attacker can inject his or her own data into a new SSL connection and have that data authenticated under an innocent client's credentials.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-223.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-223-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 223 for November 19, 2009: The Trouble with SSL.

It's time for Security Now!, the show that covers security in great detail, sometimes excessively geeky detail. And that's because this guy's a geek, Mr. Steve Gibson of GRC.com. And that's why we love him, because unlike the others, on this show you learn how it works.

Steve Gibson: Can you have too much detail?

Leo: No. No, you can't.

Steve: Not for our audience.

Leo: Well, that's the point.

Steve: Not for these guys.

Leo: That's exactly the point. These guys love that stuff. Hello, Steve. How are you?

Steve: Leo, it's great to be with you again, as always.

Leo: You have a good week, I hope?

Steve: Had a great week. And we have a great episode. This one, this is just classic, perfect Security Now!. It's tech-y; it's relevant; it's important; it's a real problem. The industry has been secretly scurrying around and meeting and trying to figure out what to do about it. The good news is there's something that can be done about it. But, I mean, it incorporates all aspects of sort of the security world that we've been talking about for more than four years. So this is the recently discovered problem in SSL, which turns out to allow bad guys to insert their own badness into connections that everybody at each end thinks is secure, but it's not.

Leo: That's pretty serious.

Steve: It's bad, yeah.

Leo: That's pretty serious.

Steve: It's not good.

Leo: And before we get to that, I know you probably have some errata and security news to cover, as well.

Steve: Yup.

Leo: I want to remind people that your show is nominated in the Podcast Awards. And they have till November 30th to vote, so you're kind of down to the end now. In the technology section; am I right? [www.podcastawards.com]

Steve: Yes, the technology section. Although, I mean, we really do need to make a determination here, Leo, you and I and...

Leo: I'm going to let people vote their conscience.

Steve: Okay.

Leo: Vote for your favorite. Well, the reason you're saying this, I know, is because there are two TWiT shows nominated.

Steve: Right.

Leo: Security Now! and FLOSS Weekly. And by the way, there are other excellent shows also nominated, like Skepticality, which is really great; The Naked Scientist; Bwana.TV. So there are plenty of...

Steve: No. If they're not on the TWiT network, forget about it. No. No. No, that would be really interesting. It would be great if the listeners to this podcast really did vote for their favorite podcast, this one or FLOSS. I'd love to know. I'd be - that would make the number more useful and interesting than just overwhelmingly, okay, we told everybody to please click this button.

Leo: Yeah. We don't want to lobby.

Steve: No.

Leo: And by the way, you won this category I think last year.

Steve: It was year before last, yeah.

Leo: Year before last. You've won it a couple of times.

Steve: I think it was the first year.

Leo: First year you won. You won again the year before last. It's not like...

Steve: At one point...

Leo: At this point it's emeritus. So, but we do want people to vote.

Steve: The old man on the hill is still up there on the hill.

Leo: Exactly. We do want people to vote, mostly because it's a great way of bringing awareness to podcasts. And vote for your favorite podcast, whatever it is, at PodcastAwards.com. You'll have till November 30th. You can vote every day.

Steve: You can?

Leo: Yeah.

Steve: It's not one per person?

Leo: No. You can vote every day.

Steve: Oh, goodness. Oh.

Leo: So stuff the ballot box. Be my guest. Actually that's one way you could solve this. You could vote for Steve one day, Randal the other day, and mix it up. They're all my children. I'm not going to select. Just vote your favorite.

Steve: You like all your children equally.

Leo: I love you all deeply.

Steve: That's important, Dad.

Leo: So what do you want to do first, news or errata, Mr. Gibson?

Steve: News, we've got some security news.

Leo: Okay.

Steve: Not surprisingly. One of the - I was going to say the "baddest." One of the worst problems that Microsoft patched last Tuesday, last week, the second Tuesday of November, is heavily expected by the security community to be exploited very soon, if it's not even - if it's not already being done. So I wanted to further encourage - I know that the fact that Microsoft patches typically require you to reboot your machine, I know that I've been in a position where I've got so many things open and set just the way I like, and I'm in like the middle of things, that rebooting the system right then is a problem. But there's this problem with the embedded open type fonts, EOT fonts, which, well, there's a couple lessons here. It's a font-parsing bug which allows remote code execution. The problem is that it's a kernel bug.

Leo: Ooh.

Steve: So it's an overflow that occurs in the kernel. Well, since these EOT files can be compressed and encrypted by their spec, their spec supports encryption, that makes it extra difficult for antiviral software to see what's going on, because it's an encrypted payload. And so that's expected to thwart AV. And so it'll be systems which simply load a page. This is the other reason it's expected to be a big deal is that it's the classic drive-by problem where you just get some text on a website, and it can take over your machine. Interestingly, because this is a kernel-level problem, Vista's IE7 and IE8 sandboxes, which are designed to protect the system, offer no protection for this exploit. And again, because it's a font-rendering problem in the kernel, this is not helped by disabling JavaScript. So even turning JavaScript off will not help.

So the thing that I find annoying is that fonts are being rendered in the kernel. There's something fundamentally broken about that. And we know where this happened, and we know when. Because, I mean, the idea is the kernel is your holy, sacrosanct - it's the kernel. I mean, it's the OS. You want to keep application sorts of things out of it. It provides core services. It handles the abstraction of your I/O so that various apps can vie for the peripherals, and the kernel manages that. It typically abstracts the file system so that applications are able to talk to NTFS or FAT files in a uniform fashion. It handles memory management so that applications are able to request memory resources, which the kernel juggles. If it runs out of RAM, the kernel swaps things out that are not being used and brings new, empty memory in from swap space, I mean, all those really low-level things.

Well, at one point Microsoft, hopefully before they got the security religion, I mean, because you wouldn't ever want them to do this after they were concerned about security because it completely breaks security, Microsoft said, well, we need - we want faster display performance, so we're going to move GDI, the Graphics Device Interface, from user space, where it had always been, into the kernel, in order to minimize the user-to-kernel transitions because it's expensive to cross between user space and kernel space. So they said, well, let's move GDI, this complex rendering code which includes the whole font system, down into the kernel because won't that be a good idea. Yeah.

Leo: Oh, boy.

Steve: And as a consequence, this is what you get. You get more complexity. You get little mistakes. But rather than it being a mistake in user space that just causes a much more limited problem, now it's a mistake down in, you know, God Central of the computer. I mean, this is where everything happens. And as we've seen, for example, with rootkits, I mean, the reason rootkits are such a problem is that they're down in the kernel, able to literally do things like hide files from the directory system so you can't see them, and AV systems can't see them.

So anyway, I wanted to encourage people, I wanted to further explain this particular vulnerability, which was fixed last week, and just make sure - and also explain or reinforce how trivial it will be for this to be exploited. Again, it's in the public domain now, what this problem is, how to exploit it, how it can be used. It will be anyone who touches a website, whose Windows system renders fonts on a website, that hasn't patched this can get their machines, at the kernel level, taken over.

Now, Microsoft, to their credit, has done some things in Vista and later, like Address Space Layout Randomization, ASLR, where the chunks of the kernel are located in sort of semi-random locations, making kernel-level exploits more tricky. But there's lots of instances where even Address Space Layout Randomization can be worked around. So if you haven't by any means yet rebooted your machine with last week's patches, delay only as little as possible because this is a bad guy.

Leo: That's really too bad.

Steve: Also in news following from last week, and you probably already have heard of this, Leo, the jailbroken iPhone problem that we discussed has, not surprisingly, escalated. We're no longer changing wallpaper to some random singer from the past.

We're now stealing phone data, including contacts, music, photos, email, text messages, and pretty much everything. There are two known and more expected current worms that are sucking personal content off of iPhones that have been jailbroken where their SSH server password has not been changed.

Remember from last week we discussed this, that the problem is that jailbreaking installs an SSH server, and it has a default password that everyone knows. If you jailbreak your phone and don't change that password, then your phone literally can be contacted over the Internet, just like you were running a little web server. In this case it's an SSH server. Someone can log onto your phone and do pretty much what they want. The original problem, which was a worm constrained to Australia, was just changing wallpaper. Not surprisingly, that quickly escalated into much more damaging attacks. So if you do have a jailbroken phone, do make sure you change that SSH password because it's getting bad quickly.

There's a recently acknowledged by Microsoft zero-day problem with Windows 7 and Server 2008 Release 2. This is another problem with Server Message Blocks, the SMB protocol. Microsoft is suggesting people block ports 139 and 445, which are, you know, the Windows filesharing, file and printer-sharing ports, which is used for the SMB protocol and all other kinds of things. They've acknowledged this problem. Exploit code has been posted in blogs on the Internet so people are aware of it. Most people are not going to have those ports exposed.

As one of our Q&As from last week asked, if I'm behind a router, and I'm using XP, aren't I behind two firewalls? It's like, yes. So unsolicited packets are not coming in from outside. However, there are some exploits involving, not surprisingly, web browsers, where you go to a web page; the page you receive contains a specially formatted URL which can cause your machine to reach out to a malicious SMB server. In that case, blocking your ports isn't going to help you because you're initiating an outbound connection to a hostile SMB server. All it needs to do is send back some bad data.

Now, the good news is, if this happens, the only consequence is that you need to pull the plug on your machine. It completely locks it up. It puts it into an infinite loop in the kernel. The machine won't respond to anything - keystrokes, mouse movements, nothing. You cannot shut your machine off. You just have to - even the power button apparently doesn't work. You literally have to pull the plug out of the back and then wait, you know, count to five and plug it back in, in order to get control of your machine again. So it's just a denial of service attack. But it's something that Microsoft, I'm sure, well, hopefully they will patch it during Patch Wednesday. Because the fact that this happens on Server 2008 is a problem. You don't want, you know, a main corporate server to get locked up. I mean, it completely shuts it down. It's a complete kernel-level denial of service. It would be necessary to somehow trick the server to going to a malicious SMB server. But we know that hackers are clever. I wouldn't be at all surprised if this ends up happening.

Leo: Those nasty boys.

Steve: Now, I did want to follow up a little bit on our port-knocking discussion from last week also because I forgot about one other reason that I don't like port knocking. I talked about how the problem is with port knocking you have multiple packets arriving at a given firewall, for example, where they just die because they're not admitted. But the fact of their arrival at a particular port in a particular sequence is like the secret combination. One of the problems that we talked about was that, sure, anyone who is

listening to your conversation could replicate the knocking sequence. And then that's what led me into a discussion of ways you could cleverly implement essentially a one-time authentication-style system that would prevent that.

I forgot one of the other big problems with it, and that is it's extremely susceptible to a denial of service, that is, not a flood, but technically a denial of service where - that is to say, a denial of knocking sequence. If anybody else knew that this was what you were doing, for example, if a corporation were depending upon this, or if you were for some reason a high-value target, or high value to just even one random person, all they have to do is send random packets at your IP address every so often. And that will look like failed knocking sequence packets and cause the software, which is looking for the proper sequence, to constantly think, oh, look, there's somebody trying to get in, and we're not going to let them.

Well, the problem is, if at the same time a valid person is sending the proper knocking sequence, their packets will be intermixed with the ones deliberately designed to screw up that knocking sequence, and you'll never get in. So it would deny someone access to the service that they're trying to use to get in, just by spraying some deliberately wrong packets every so often. So there's lots not to like about it. It's, I mean, it's sort of a poor man's clever way of getting in. But it's certainly far from optimal.

And I have, since we seem to be on the theme of pirated SpinRite software in the last...

Leo: Last few weeks, yeah, yeah.

Steve: I have another confession from a guy named Troy Starkey who said, "Hi, Mr. GRC." He sent this to the sales email address. And the subject of his email was "I wish to thank you for your software." He said, "Hi, Mr. GRC. Firstly, let me commend you on your fantastic software." Of course he says this after he purchased it. But actually he thought it was fantastic beforehand. He says, "It has repaired a few of my drives in the past. So I truly wanted to show my appreciation by finally purchasing your great software. I regret to admit that when my first drive went on the 'fritz'" - he has "fritz" in quotes. I don't think that's his drive's name, I think that's where the drive went.

Leo: Drive went on fritz.

Steve: It went on the fritz. "I went searching the Internet desperately, trying to get it fixed, as I had a lot of digital photos on it, and I came across a copy of SpinRite 6 registered to someone else."

Leo: Oh, boy.

Steve: "To my amazement, SpinRite worked perfectly, and I was able to save all of my data. At that stage I was under a lot of financial pressure and could not afford to purchase your software. But I vowed to buy it when I was on my feet financially. Well, I'm on my feet now, and I offer you my support for a fantastic product that saved my irreplaceable photographs. So please accept my sincere apologies." I certainly do. He says, "I feel horrible that I used that copy, but I have always supported products that I use. And now I can add you to my list of great software. Let's hope I don't need to use it

in an emergency again. I do have a word of advice for people who try to use SpinRite on the dreaded clicking hard drives, the ones that sound like the heads are trying to bash their way out of the hard drive enclosure" - which makes it sound a little dramatic, but, you know...

Leo: That's an apt description, yeah.

Steve: We've talked about clicking hard drives before. And he said, "and won't be detected even in the BIOS. Place your clicking hard drive in an antistatic bag and place the drive in your deep freezer. I left mine in for about 60 minutes. Then remove from the freezer and the antistatic bag and tightly wrap it in a towel or similar absorbent cloth." That's to prevent moisture from condensing on it, right. He says, "Quickly attach it to your PC and power it up whilst everything is still frozen. Drive worked well enough for me to then correct it with SpinRite 6 and transfer the data to a healthy drive. Kind regards, Troy Starkey."

Leo: You know, I've heard this. And I know you have, too.

Steve: Oh, yeah. The refrigerator trick is a great standby.

Leo: But doesn't the drive immediately heat up? I mean, come on.

Steve: Well, it gets hot very quickly. But, see, but that initial clicking is the drive trying to initialize itself and get out onto the surface and find some servo data. So all you really need to do is to kind of give it a little bit of a help over that first hump to get itself going. And once it's going, then you're often able to stay out there over the disk surfaces. The drive's initialized, the BIOS sees it, SpinRite can see it, and you're off to the races.

Leo: Got it. I see. So it's really just that first thing that you want it to go. Okay.

Steve: Yup, exactly.

Leo: That makes a lot of sense. All right. We are - I'm very interested in this subject.

Steve: Oh, it's really a good one.

Leo: We all have kind of a little bit of a vested interest since SSL is the...

Steve: Yeah, just a little.

Leo: ...technology that secures, you know, all secure web interactions on ecommerce sites and so forth. So is this going to be one of those propeller episodes, Steve?

Steve: Eh, kinda. It's not going to hurt people. And I don't think it'll require multiple listens. It's up there. But it's going to be good. Okay. So while I was rereading some of the RFCs that specify the way SSL and TLS operate, I appreciated yet again how so excruciatingly careful the specifications are and how well and carefully they were written. So those people who were involved in putting it together, they must, given that it's so clear how careful they were, they must be thinking, ooh, crap.

Leo: We worked so hard.

Steve: We were so careful. And we missed one little thing.

Leo: So it's the nature of that they missed something.

Steve: Yes. And I'm going to explain what they missed. Everyone's going to get it. And we'll talk about the consequences and how that's leveraged and ultimately what it's going to mean. It does mean a revision of TLS, the Transport Layer Security. It's got to be changed in order to fix this. And there's really no workaround.

So, okay. A little bit of review of the way SSL and TLS hook up to each other. When the client wants to initiate a connection to the server, it establishes a TCP connection. And then the first packet it sends, it's called a Client Hello packet. That packet contains the highest protocol version that it supports; a blob of randomness that it has made up for itself; a session ID; a list of the ciphers, that is, the cryptographic ciphers that it is equipped to use, which it's offering to the server; the compression method that it proposes to use. The server receives that Client Hello packet containing all of that. And it responds with its Server Hello, which contains a proper subset of some of those things. It knows what protocol version the client has offered as the best it can do. It knows what protocol version it has as the best it can do. So it chooses the highest that they can both do. So it sort of negotiates the best, the latest protocol that they both are aware of in terms of number.

And so, for example, we are currently at TLS v1.2. I imagine we'll be at 1.3 before long. And so everyone will, you know, both ends will breathe a sigh of relief when they exchange, oh, you know 1.3? Oh, thank goodness, because I do, too. And now we can solve, we don't have to worry about this new problem that we have. So they agree upon the latest revision that they both are aware of. The server generates its own chunk of randomness, which it sends back, along with the session ID that it confirms that it received from the client. And the client, remember, sent its list of ciphers, cryptographic ciphers that it knows. The server looks at those and weeds out any that it doesn't know about, so it sends them all back.

So basically it says, okay, it says to the client, of those you sent me, these are the ones I also know about. So choose from among those. And it agrees on a compression method, if the client wanted that. It also sends its certificate, which is the server's assertion that, hey, you've connected to whatever - PayPal.com, Amazon.com, eBay.com - over a

secure connection. Here's my certificate, which I'm using to prove that your connection has not been spoofed and that I really am Amazon, eBay, PayPal, whatever.

So the client then takes the agreed-upon cipher set, chooses a cipher from it, which it now knows that they both understand. It operates at the protocol version that the server has returned, which it knows that they both understand. It takes the randomness that it generated, the randomness that the server provided, and it then does a key negotiation. Basically in that certificate, and the certificate is signed, so it will take - it'll look at the list of certificate authorities that it has, see that it was signed by a certificate authority that it has in its list of verified valid authorities, and the certificate contains the server's public key. So it will encrypt the data composed from the randomness that they both have, using the server's public key, and send that back in what's called the client key exchange packet.

And so what that's doing is, that's saying, okay, here's the master key for this session that I'm proposing. Since it encrypted it with the server's public key, only if the server contains the matching private key will the server be able to decrypt that in order to get the same data that the client has. So nobody listening in the middle is able to do that. They won't have the server's private key because the server guards that with everything it's got. And the fact that the server can decrypt it proves to the client that it has the private key matching the certificate. Otherwise, for example, anybody could record the certificate during a connection to a secure site and then play back that certificate, pretending to be that site. But the certificate does not contain the private key. It contains the public key. So that wouldn't help somebody who was trying to spoof the site. So, you know, so basically that's the way this all works.

Then the next packet is what's called the "change cipher spec" packet, which is the client saying I've given you everything you need. Now I'm going to switch to secure mode using the ciphers that we have agreed on. So it sends this change cipher spec packet. And that is the last packet sent in the clear. It then sends, using the agreed-upon encryption, it sends a finished message which ends the handshaking, which is it contains the master secret encrypted under the cipher and a hash of all the preceding handshake messages. So all of their conversation up to that point is hashed. All of that is encrypted under the current encryption, and that's sent to the other end.

So what's significant here is that, until that cipher change spec message, everything is in the clear. That is, you don't bring up the cone of silence on this dialogue until this change cipher spec message goes in each direction. The server does the same thing. It also sends a change cipher spec message to the client saying, okay, here we go. The next thing you're going to receive from me is the finished message encrypted from my end, proving that I have everything that we've each received. So this is a very nice, lockstep approach for agreeing on protocols, agreeing on ciphers, agreeing on session ID, compression, proving to each other that we've each got - we made up randomness, and we've exchanged it, and we're all on the same page, and then switching into encrypted mode. So the guys that designed it designed it very, very carefully.

Now, it is possible for either end to request renegotiation for - and this could happen for various reasons. For example, it might be that the client is at a public website, XYZCorp.com. And they're browsing through some publicly available pages, but that this client is an employee of XYZCorp.com and wants to go into a specially protected, extra secure region of the XYZCorp.com site. So they browse there. And when they go to a certain directory, the web server says, whoa, wait a minute, that's going to require more security. You need to have a client certificate which would have been issued by the XYZCorp IT staff. But at this point, even though they had a secure connection, the server had offered its certificate, as it always does, but the client had never been asked to

present a certificate because up until then it was an anonymous client. It could be - it might be a non-employee, might be anybody.

So the point is that during an already negotiated secure connection, there are situations which can arise that can require a renegotiation of the security context for a number of reasons. One, for example, in this case might be that the server says, okay, if you want to see this page, we need more credentials from you. We need additional authentication. So what can happen is that upon attempting to connect to that special page, the server is able to send a hello request message to the client, basically saying, hey, we're going to start over again here. I want you to send me a new hello message, just as if we were just connecting. And let's do this again because this time when I send my certificate, I'm also going to send - the server's thinking to itself - a certificate request which requires that the client return a certificate with its next set of packets, which otherwise wouldn't be required. So that's a way of the server saying, okay, we've got to up the ante here on authentication. So this notion of renegotiating the security context of an already established SSL connection has been around for quite a while. That's not big news.

Now, the reason the developers of this protocol weren't concerned about a security problem with renegotiating is that this hello request and the follow-on exchange of hello messages would all be contained in the secure tunnel. That is, those messages, all of this renegotiation, it would all be performed in a secure context, that is, with the previously negotiated security, which wouldn't be upgraded until, once again, that change cipher spec message was sent, and then a finished message was sent for each side to end their handshaking relationship, essentially to end the whole negotiation. So the developers said, hey, that's fine because we can have a renegotiation any time because it will always be under the existing security context. So, and the protocol makes sure that no man in the middle can mess with this. So what happened in the last couple weeks is that some very clever hackers thought about this some more and figured out a way around it within...

Leo: Oh, isn't that nice, yeah.

Steve: Yeah, it's extremely cool. So here's how it works. First of all we need to - I need to make sure people understand that man-in-the-middle insertion is truly trivial. I mean, it's the reason that many of our listeners have reconfigured their wireless networks in a Y configuration with routers. Because unless you do that, if you want to offer open WiFi, you want to have an open WiFi and an encrypted WiFi, like open for guests or even for neighbors, the problem is that that exposes your entire network to man-in-the-middle attacks. And we know that that comes from playing games with the ARP system, the Address Resolution Protocol.

Remember that the reason there's this awkwardness, this ability for a bad guy to insert themselves into a conversation, which is what we mean by man-in-the-middle, not somebody just eavesdropping passively, but somebody who has arranged to receive, sort of to have the traffic flowing through them so that it's subject to, not only their observation, but their modification and addition and subtraction of packets. The reason we get into this is that the Ethernet LAN is based on MAC addresses, yet the Internet protocol is based on IP addresses. So there needs to be a mapping made between which LAN MAC adapters have been assigned which IP addresses. So that's what the ARP table is. It's just a table of simple associations. This IP address is this MAC address. That IP address is that MAC address.

And so when a computer initially powers up onto a Local Area Network, it sends out a

broadcast packet that says, hey, I'm configured so that my gateway is this IP. Who is that? So it says what adapter has that IP. And the adapter listening for such broadcasts, and all Ethernet LAN adapters listen for broadcasts like that, check to see if that's one of the IPs that they've been assigned. And if so they go, hey, I'm that IP. Here's my MAC address. And so that creates an entry in this newly booted computer's ARP table so that it knows how to address packets to the gateway. And similarly, as machines come on the LAN, they send out broadcasts. They also can listen to other broadcasts just to learn about other adapters, sort of as a side effect of being on the LAN.

So it turns out that because this association between IPs and MAC adapters is dynamic and can be changing, these things also expire. If you don't get a response a couple times, then the computer will say, oh, okay, maybe the IP address changed or the adapter changed. So it'll sort of try to renew the ARP table. As a consequence of this, there's lots of opportunity for mischief. And...

Leo: What layer does this happen at? Is this at the Ethernet layer? Is this - this is a TCP/IP issue.

Steve: Well, it's actually at the Ethernet LAN layer. And this is a protocol that was added to the Ethernet because it wanted - the Ethernet wanted to carry a foreign protocol. It wanted to carry IP protocol on top of the lower level Ethernet protocol.

Leo: It was kind of a necessary handshake so that you could marry the two. You could put TCP/IP over Ethernet.

Steve: Exactly. But it's because it's dynamic, because these tables are being built on the fly, that there's all kinds of ways for bad guys to insert their MAC adapter into other machines' ARP tables. And so that's the trick is if a bad guy inserts their MAC adapter MAC address into the gateway ARP table for a different IP, then the gateway is none the wiser. It will send packets that are intended for somebody else to the bad guy. And all the bad guy has to do is forward them on to the intended target, and that target will be none the wiser. And similarly, if the bad guy inserts its MAC address into the target's ARP table for the gateway, then when that machine thinks it's sending packets back to the gateway, it's actually sending them back to the attacker. And the attacker simply forwards those to the gateway. And so that allows an attacker to splice himself into the conversation.

And there's tons, many more due to the history of this for UNIX and Linux machines than, for example, for Windows and Mac machines. But there's plenty of software around where you can simply bring a laptop to an open caf, and within a few minutes you are now filtering all the traffic going from that open WiFi hotspot to any of the users in the caf. So, I mean, this is real. This is not sci-fi. This is not even difficult. But until now you really couldn't do anything with it. If they were not using SSL, and we've talked about this a lot, if they were not using SSL, they were just logging into their POP server, then their name, their account name and username are going to be in the clear. You can capture them. If they're surfing the web, you can see all the URLs and all of the data that they're transacting once you've spliced yourself in. In fact, you really don't even need to use an active attack. You could just passively monitor all the packets if you just wanted to suck in all this information. It has been believed, though, that an SSL connection protected you completely from a man in the middle. We now know in some circumstances it does not. So here's how this happens.

A user initiates a connection to a remote server after somebody has placed themselves in the middle. Now, this doesn't have to be in a caf with WiFi. It could be in a hotel with a bad employee who's able to go into the closet where the router is and plug their computer in, in which case they're able to do this with every employee, I mean every guest in the hotel who's using the hotel's network. So there are all kinds of other scenarios than just the open WiFi. The idea though is you do need to be in the flow, not passively eavesdropping, but actively able to intercept traffic. So a...

Leo: Well, how is that different from just plugging in? Do you say "I am actively listening"? I mean, is there something you do to signal that?

Steve: Well, no, yeah. What you start doing is, you begin listening to ARP - you listen to the Ethernet traffic, and you send your own ARP packets, malicious ARP packets to the various endpoints to confuse them and get them to put you in their tables instead of each other.

Leo: Right. You respond to their ARP requests with your own MAC address.

Steve: Actively, yes. And before you know it, you're spliced into their conversation. So then you see the client initiating an SSL connection to a remote server. You know that because it's over port 443, the SSL port. You hold that packet for a second, that is, the attacker holds that packet and sends their own to the remote server, establishing their own SSL connection.

Leo: Hence the man in the middle.

Steve: Hence the man in the middle. They go through the protocol negotiation, just as I said. So now the attacker has a secure connection to the remote server. At that point the attacker sends a malicious request to the server. That is to say, like some sort of command that the client hasn't sent, making it malicious. I mean, something that the client never intended. And I'll explain how that can be formed. And, I mean, these things have been done and proven in the last few weeks.

So the attacker sends this command which is unfinished to the remote server and then sends a new Server Hello message. So in one packet it sends the beginning of a command, then sends a Server Hello message, which it's able to send at any time. That is, part of SSL and TLS is either side can request renegotiation at any time. The server receives the - I'm sorry, the client. It's the Client Hello message, not a Server Hello message. So the malicious guy in the middle sends a Client Hello. The server goes, oh, okay, this guy wants to renegotiate. That's fine. So remember that all of this SSL negotiation, up until after the change cipher spec message, is in the clear. So the bad guy sends the Client Hello message that it had been holding and blocking from the real client. And so that it forwards it to the server, and the server believes it is renegotiating the existing connection; whereas the actual good guy, the innocent client, thinks it's establishing a new connection.

So what happened was the man in the middle was able to hold the client's hello message briefly, send its own, negotiate a connection, get a secure connection, send some stuff to the server, then allow the client's hello message to pass through it, and all it then does is

allow the standard handshake to proceed, and essentially knitting the innocent client's SSL connection to the server. Client doesn't know anything about a previous SSL connection. The server sees it as a renegotiation of an existing one. What happened was, as a consequence of this, the attacker was able to send some stuff ahead using the, well, on this little brief sort of a stub of a connection, but over SSL.

Now, how is that useful? One of the ways that we see the web is being used is that requests to servers, like standard HTTP get requests, have these very complex-looking URLs with all kinds of gobbledy-gook in them. Remember, you know, it'll be https:, for example, or just :// and then the domain, then the directory and the page. Then there'll be a question mark which starts the parameters. And then you have all these parameters afterwards which are sophisticated command directions, causing things like PayPal to transfer money or banks to log you in, causing purchases on Amazon and so forth. So there's all this parameterization in this - contained in the URL. That's normally protected within the SSL secure wrapper. So it can't be seen by someone eavesdropping. It can't be synthesized by a man in the middle. It's protected.

After that get request are a series of headers, things like, for example, the host header is required by HTTP 1.1, the current version of HTTP, where it says, you know, www.amazon.com. It'll say this is the host I am trying to connect to. And then things like time and date and also cookies. Every one of these header lines ends with a carriage return line feed and then the next header. CRLF is the abbreviation for carriage return line feed. This hails from way back in the PDP-8 days with a teletype, where you'd be typing along, and then when you come to the line you send a carriage return character, which moves the head back to the beginning, the typing head back to the beginning of the far left of the paper, and then a line feed, which rotates the paper's platen up by one line, and then more characters follow. So the end of each line is this carriage return line feed, and then the next one.

Well, imagine what happens if what the attacker in this scenario sends is a sophisticated command with a URL and all of this extra stuff padded on the end which the receiving server knows this is a command for it to do something. Then it adds a couple headers. But the last header it leaves open. For example, the header is ignore this colon, space. And that's where it ends its insertion. It leaves that last header unterminated. Now it passes the innocent client's Client Hello message through. The client establishes an SSL connection. And the client sends its valid get request, its HTTP get command. Well, because the server - sorry. Because the malicious man in the middle left that last header unterminated, what the server sees is ignore this colon, space, and then the client's request, which looks...

Leo: Oh, wow.

Steve: Isn't that cool, Leo?

Leo: Yeah, yeah.

Steve: Which looks like just a header, and it is a header of the malicious request. Which means the client's actual command is absorbed. But then - here's the good part. The client, because it's got a connection that's secure to a web server it knows, it continues, its command follows with other things, including its cookies.

Leo: Oh, great. We're cool here. I know you. Here.

Steve: Yup. You're over a secure connection. You've proven who you are. I've seen your certificate. Here is my cookie authentication for who I am. So that ends up getting tagged onto the attacker's command, allowing the attacker to perfectly impersonate the client, the innocent client.

Leo: Because that has the authentication cookie. That's all it really needed.

Steve: Exactly. Exactly. Because that was a way for it to get the innocent client to make a request, and the client would automatically include its authentication cookies that it has from dealing with the server in the past, allowing the attacker to impersonate the client and the attacker to do whatever it wants to, whatever it can, given its ability to send a command to the server. So this has got everybody really concerned.

Leo: It seems like that was pretty easy. I mean, okay, maybe...

Steve: No, no. That's exactly the point, Leo.

Leo: Seems like it was kind of simple.

Steve: It's why it's so - as I said, after I understood this, and I was rereading the RFCs to refresh myself, I was like, okay, wait a minute now. Why is this so easy? The RFCs were written so carefully and so deliberately. And it's like, oh, crap.

Leo: So was the oops leaving this hole with the line termination? I mean, what was the oops?

Steve: Well, the oops was that they were relying on the existing security of the previous connections.

Leo: Allowing the continuation.

Steve: Yes. They were assuming that the renegotiation would be protected by the secure envelope that had already been created. There was no provision for carrying forward the security context that had been established as part of the renegotiation. And that's what we're going to talk about next is how they have solved the problem.

Leo: Oh, good. Probably too late for me, but they've solved it.

Steve: Well, actually, I mean, I should say how they will solve the problem.

Leo: How they hope to solve, yeah, yeah, yeah.

Steve: Nobody has the solution yet. We're all vulnerable to this today.

Leo: Oh, very interesting. It does, it did worry me when you said SSL is flawed. I thought, oh, no, you know. But at least they have the encryption before it's sent. So, just to recap, they're using ARP - is it spoofing? Would that be the word? Not poisoning, but spoofing?

Steve: Well, it's any scenario where you have a man in the middle. So it is definitely possible for someone to insert themselves in the middle on an Ethernet LAN by playing games with ARP. But the presumption is that SSL and TLS provide us with an end-to-end safe channel such that we establish the channel. And that's one of the cool things about SSL. This tunnel, this encrypted tunnel comes up first. It's negotiated. Everybody agrees at each end. It comes up, and then everything passing through it is in this cone of silence. It's encrypted securely, specifically so that nobody who is monitoring or intercepting can have access. The only thing somebody in the middle can do is deny service. But if the packets transit through, the whole point of SSL is that there's no way for anyone to mess with you.

Leo: See, I always kind of thought of it like a VPN, like it's an encrypted tunnel. But it isn't really a tunnel, is it.

Steve: Well, it's a - there's no encapsulation of protocol. But the packets themselves are - the payload of the packets themselves are encrypted within the IP wrapper. So...

Leo: Okay. So they look like standard IP packets, but they are encrypted within. The data's encrypted.

Steve: Correct.

Leo: Okay.

Steve: Correct. So what this does is, this is a significant and fundamental flaw which was - just someone tripped over. And in fact, I mean, it's bad enough that the major vendors of SSL protocol have met secretly before this became public knowledge and scratched their heads and said, oh, crap.

Leo: What are we going to do?

Steve: This is really bad. Okay, so it turns out the fix is incredibly simple. The TLS, the Transport Layer Security, we've talked about how it and SSL v3, which was the last version of SSL, how the protocols are virtually identical. That's true, except that TLS

added one feature. It added the ability for what's called extensions. Extensions are a variable list of things which the client, in the Client Hello packet, in that initial packet going to the server, the client can say, oh, and by the way, I'm extra special. I know the following extensions. And it can know them and leave them, leave their payload, the actual extension payload empty, if it just wants to assert to the server that it's aware of these things. Because in some cases the server may need to know that the client is aware of these extensions in order to enable it to use them.

So an example of an extension, a very useful one, is one that allows multiple homing over SSL. One of the problems with SSL is that you're not, with SSL, able to have a single IP address post multiple different domains because the initial connection and negotiation has to happen before the client is able to send through the tunnel, the SSL channel, the request containing the host header that tells the server which server it wants. So the certificate negotiation that has the server's name in it is tied to a single IP address. But with IP starvation, and how nice it would be, for example, to have multiple domains on a single IP, it'd be nice to have that.

So one of the TLS extensions allows a TLS client to provide the server name in the Client Hello packet. That allows a smart server to say, oh, this client wants this certificate. So it allows the client essentially to ask for the certificate that it wishes at a single IP address. Well, that's very cool. I mean, that solves a big problem for SSL. Another example of an extension would be for, if you had a really lightweight, really small client, like a whole system on a chip that doesn't have much buffer space and much memory, it's possible for large SSL packets to get fragmented, and it's necessary for the client to have enough buffer space to reassemble the fragments into a full packet. But the client might just not have that much RAM available. So one of the other extensions allows the client to negotiate with the server a much smaller maximum fragment size to allow the client to not need so much buffer space. So that is cool.

And then finally I'll give you one last example because it's something we'll all understand. The client is able to tell the server what certificate authorities it knows about. Remember that when the server sends its server certificate to the client, that certificate will have been signed by some certificate authority. Well, if the client is able to say here's the authorities I know about, please give me a certificate signed by one of them, that can prevent an SSL negotiation failure as a consequence of the server handing the client a certificate signed by someone it doesn't know or doesn't trust. So those are some existing extensions. It turns out that all we have to do is add one more. And it is a renegotiation info extension. And all it contains is that finished data from the prior negotiation.

So the idea is that, remember that in this attack scenario the server saw the real client's Client Hello message. It saw that as a renegotiation of the existing connection, whereas the client thought it was its initial contact packet. So all we need to do is enhance the protocol. We add a renegotiation info type of extension which contains the payload from the previous finished message. That finished message is this - basically it's the I'm proving to you everything which has gone on before we're in agreement on. So if renegotiation requires that extension - which it doesn't today. That's the mistake. The mistake is the renegotiation did not require any security context from the previous secure context. It assumed security because it was in the security envelope. But in this particular attack there really wasn't a security envelope.

So if we simply change TLS, probably 1.3, to version 1.3, we simply change TLS so that renegotiation must have the payload from the previous security context as part of the hello message, the problem is solved. The server, who is aware of this, would never accept a renegotiation that didn't have that packet from the previous security context.

And in the scenario we painted, the client wouldn't think it was renegotiating. It wouldn't have any context to share. So the server would deny the renegotiation and drop the connection.

Leo: Perfect.

Steve: So to give you a sense for how careful these guys are, I'm going to read from the draft. There's already an Internet draft for this fix. I mean, these guys have scrambled in order to, like, deal with this. And this is beautiful because the language of this gives you a sense for how careful they have always been, and just how much of a mistake this was. So this is under the category of backward compatibility. And this is the problem we face now because none of us have TLS 1.3. It doesn't exist yet. We all have 1.2. First of all, it's worth saying that SSL v3 doesn't have any provision for extensions. That was added in TLS v1.0. So SSL is completely dying at this point. That is...

Leo: Because you can't fix it.

Steve: Exactly, it's not fixable. TLS can be revved, but SSL can't be. So goodbye to SSL. This, like, gives some reason to no longer allow a fallback to SSL.

Leo: So people have been using TLS, not SSL, even though we call it SSL.

Steve: Exactly, yes. Everybody now has TLS clients and servers at each end. And that's the actual protocol we're using because it's got all these extra cool little features, basically thanks to these extensions. So under backward compatibility they say: "Existing implementations which do not support this extension are widely deployed" - yeah, like that's all we have right now - "and in general must interoperate with newer implementations which will support it. This section describes considerations for backward compatible interoperation." Okay.

On the client side, client considerations: "If a client offers the renegotiation info extension, and the server does not respond, then this indicates that the server either does not support the extension or is unwilling to use it. Because the above attack looks like a single handshake to the client, the client cannot determine whether the connection is under attack or not. If clients wish to ensure that such attacks are impossible, they must terminate the connection immediately upon failure to receive the extension without completing the handshake. Otherwise, they may be performing client authentication and thus potentially authorizing the data already sent by the attacker, even if the client itself sends no data. Note that initially deployment of this extension will be very sparse, and thus choosing to terminate the connection immediately is likely to result in significant interoperability problems."

Meaning that, you know, if we told our browsers not to allow TLS without renegotiation protection, we couldn't talk to anybody today. And it's going to take a while before we're going to be able to talk to anybody. So it's not practical to enforce that. But these guys are being - this is how careful they're being.

On the server side, server considerations: "If the client does not offer the renegotiation info extension, then this indicates that the client does not support the extension or is

unwilling to use it. Note that TLS does not permit servers to offer unsolicited extensions." And that's generally true of the whole spec. Remember that all throughout this I've been saying that the client first makes the offers. The server then chooses from among them. So similarly, if the client doesn't say I know about renegotiation info extension, then the server does not have permission to use it because it might break the client.

Leo: Yes.

Steve: That didn't know about it. So it says, "Note that TLS does not permit servers to offer unsolicited extensions. However, because the above attack looks like two handshakes to the server, the server can safely continue the connection as long as it does not allow the client to re-handshake." So that's a significant point. A server could accept a connection that did not have the renegotiation info extension being offered by the client. It would simply have to flag that connection as non-renegotiable and never allow renegotiation on the fly.

Leo: Which then also prevents this exploit.

Steve: Exactly. So it says...

Leo: Does that break anything else, though? I mean, is that...

Steve: No, no, that's the beauty of this is that right now clients don't often want to renegotiate. But it's in the spec, so the attackers can use that. But it's not common for clients to renegotiate their credentials once they've established it. So a nice solution is for all servers everywhere to get themselves updated. And it's, you know, I was going to say it's easier for the servers to do it, but that's not necessarily the case. Hopefully Microsoft will respond next month, and we'll all have TLS v1.3, and you'll hear about it right here on the podcast. There'll be party streamers and noisemakers and things going off in the background. So, and a crowd of ovation.

So the beauty is that a server that does know about, has been upgraded, can still accept connections from clients that are nave and protect both the client and the server if it simply notices that the client didn't offer it, so renegotiations are disabled for this session and do not allow renegotiation. It says, "If servers wish to ensure that such attacks are impossible, they must not allow clients who do not offer the renegotiation info extension to renegotiate with them and should respond to such requests with a no renegotiation alert, which has now been described. Servers should follow this behavior."

Leo: Great, great.

Steve: So that's the story.

Leo: So how does that get implemented? Do we do a new - what do we do? The extension gets pushed out, or...

Steve: Well, yeah. For all platforms that have any kind of built-in updating deal, I know that the GNU SSL group are up to speed. Microsoft is involved. In fact, they're one of the co-authors of that paper I was just reading from. So everybody who produces SSL is certainly aware of this and is on the ball. And so they will be shortly updating the crypto libs of all the various packages. I mean, you know, we've had our crypto lib updated under Windows several times already this year. So it'll just happen again, and it'll be "Now supports TLS v1.3," and servers will be updated to do the same thing, and the problem will be solved.

Leo: Is there - I guess you could go in, what, the About menu or something? You would look in the browser, or would you look in the - it's the browser, right, that you would look at. Or is it the operating system?

Steve: I would think, well, no, it would be the browser. If you were connecting to a remote system, you'd look under the connection characteristics and see if you were using TLS 1.3 or 1.2. I'm just making it up. It might be 2.0. I don't know what they're going to do with...

Leo: I think it's, yeah, right, I think it's 1.0 right now in Firefox. I'm just trying to find it. Could it be that old?

Steve: It'd be 1.2 probably. That's been the current spec for some time. Although there's nothing wrong with 1.0. They just added some more features to it. And they're about to add another big one. And I imagine they'll just take it to 1.3.

Leo: Right, right.

Steve: So that's the story. It's an interesting kind of oops that is clever, that the designers just didn't consider, which we will shortly, pretty easily, fix. But it does offer some interesting opportunities for exploitation. I imagine hackers will run around and have some fun with it before they get shut down.

Leo: Is there anything we can do until then? I guess not, really.

Steve: I can't see anything. Hopefully this thing will - this will get fixed quickly. There's really no protection from man-in-the-middle attacks. I mean, if you were aware of the MAC address of your gateway, and there was something that could tell you if that ever changed - frankly, that's a feature I've already got logged in for CryptoLink. CryptoLink will have built-in ARP games detection just to alert people for it. It's on my bullet list of things to do. But I don't - I'm not aware of any software now that does watch for malicious ARP traffic and alert you if your gateway changed because that would immediately tell you that something was going on.

The problem is, if you were in, like, an open WiFi hotspot, it might be that the very - and likely that the moment that you got on the air, the attacker was there waiting for your initial broadcast and stuffed his MAC address into your ARP table and beat the gateway to doing so. Gateways tend to be underpowered, and attacker machines tend to be more

powerful. So they're able to get their packet, to beat the gateway in responding to a broadcast. So there you wouldn't see a change. The initial MAC address would just be wrong. On the other hand, if you looked at the MAC address, you might see that it was like an IBM or a Sony machine. Remember that MAC addresses do contain the manufacturer's ID in them. And so it's like, wait a minute, I don't think I have a Sony gateway. I think someone with a Sony computer is playing games with me.

So there are some things you could do, but not easily. So I think we'll just have to hold our breath for a couple weeks and hope that Microsoft gets this fixed quickly. I hope everybody in the SSL, or sorry, the TLS community will be able to respond quickly. They should be able to. The spec is designed with these extensions in mind. They've done a trivial - I can't think of anything cleaner than simply saying use the payload from the finished packet in the renego- as the payload for the renegotiation info extension, and you're done. I mean, that's all it needs to be. It's such a simple fix that it ought to be able to be done quickly.

Leo: Just fantastic. Very good, clear explanation that I, in my limited capacity, even understood.

Steve: Like I said, this was...

Leo: It's fascinating.

Steve: I just knew this was going to be a good episode.

Leo: Yeah. Well, thanks, Steve. Everybody should rush to GRC.com, not only to get a copy of SpinRite, especially those of you who've been using it without paying for it. Stands for Gibson Research Corporation, that's easy to remember, GRC.com. And there's lots of other stuff there. You've seen ShieldsUP! probably, all those free programs Steve offers. And also 16KB versions of this show, show notes, and transcriptions by Elaine, who is so great. She did a transcription for me, a rush, last week. And she's just the best. She's just a great, really great transcriptionist. [Thank you, Leo.] So that's all there at GRC.com. Next week your questions and answers; right?

Steve: Yup, that's what we're going to do, Leo.

Leo: So go right now to GRC.com/feedback if you've got a question. Somebody in the chatroom is saying, well, IPSec allows renegotiation. This apply to IPSec? We'll find out.

Steve: Good question for next week.

Leo: That's a good question for next week. Go to GRC.com/feedback and leave your question. Steve, we'll see you next week.

Steve: Thanks, Leo.