TCP Pt. 2 - Attacking TCP

Description: After catching up with the week's news, Steve and Leo return this week to their "How the Internet Works" fundamentals series. They examine the operation of the various attacks that have been made through the years against the Internet's most popular and complex protocol: TCP.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-323.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-323-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 323, recorded October 19, 2011: TCP Attacks.

It's time for Security Now!, the show that covers your security needs online, your privacy protection, and throws in a little sci-fi to boot. Mr. Steve Gibson is here. He's our man about town, our security guru with GRC.com, creator of SpinRite and many useful utilities. And we're so glad to have you here, Steve. We're going to continue on with our conversation on how the Internet works today.

Steve Gibson: Right. We know enough now about the low-level plumbing and the way TCP initiates connections to share a really fun topic, which I think is going to really be interesting to our listeners, which is how TCP is attacked. Everyone is familiar with the term "denial of service attack." There are, it turns out, some particular vulnerabilities that TCP has always had, which are not defects in it, but just, as always, ways that attackers can use anything complex in order to leverage things to their advantage that the designers just didn't think about. So this is attacking TCP today. Of course we've got a bunch of interesting news of the week, which we'll cover, and a real short little SpinRite blurb, a little surprising use of SpinRite.

Leo: And I'm here to report that I am really loving "On Basilisk Station."

Steve: Oh, yay.

Leo: Not quite finished. I'm reading it on the Kindle. But what's interesting to me, and one of the things I know, I was a huge fan of, well, the Horatio Hornblower tales

and later the Patrick O'Brian's Aubrey-Maturin series, which is a 21-volume seafaring novel that follows the exploits of Captain Jack Aubrey, "Lucky Jack" they call him, and his bosom buddy, the ship's doctor, Maturin, Stephen Maturin. And it's during the Napoleonic Wars. They're a British man-of-war. And it starts very similarly.

And I'm really wondering if the author of these Honor Harrington novels - if I had said to myself, gee, I love these Patrick O'Brian novels, I wonder if I could set them in the future. Because it's exactly, not plot-wise, well, there's somewhat a similar plot. I mean, just as Honor is getting her first command, the Aubrey-Maturin books begin with Jack getting his first command. But also a lot of - all the naval tactics are very similar. It's like they have to broadside these ships and so forth. And even to the naval hierarchy is identical to the British naval hierarchy.

Steve: Oh, that's a good point. I hadn't thought about that. But, for example, the cannons are aimed out of the side, and that's...

Leo: They broadside each other because they're protected by the pattern from their impellers. Their high-speed impellers protect them. It's clearly crafted to be a direct analog to British naval warfare. So I thought it was - and even down to they have the Captains of the Green and the Blue, or Admirals of the Green and Blue, they have two different. And they have the list. All of this is very much out of the - you should read some of the Aubrey-Maturin books. I'd be very curious what you think. There's no lasers in it, though; I'm sorry.

Steve: Well, we did have, last week with Tom, some discussion of Honor Harrington. I shared three representative notes. First of all, we've been getting fabulous feedback from our listeners who have followed us into this new series also. And so I shared that. And I said, okay, I'm not going to pound everyone over the head. But one of the pieces of feedback talked about how someone started to read it, then put it down. Then I'd mention it again the week after, and he picked it up again, he read it for a while, then he put it down again. Then I'd mention it again the next week. And he finally, like, he said, two pages after I started the third time, he said, it finally got me, and I have not been able to put it down. So, and a number of people have talked about Audible, saying that it gets better, or that maybe the readers of the later books are better or something. But so I've been hoping that...

Leo: I'll give Audible another chance. Yeah, I'll give Audible another chance. And I only got - I didn't even get through the preface, I thought the reader was so poor. But maybe she does get better, so I'll give it another chance. You know, that's me. If it's an audio book, it's much more likely that I'm going to get to read it because I have more time to read audio than I do - but I was on vacation, as you know, last week. By the way, thanks to Tom Merritt for filling in. So that's when I got a lot of reading done on the beach, which was nice. All right, Steve. Let's launch into the security and attacks portion of the show.

Steve: Well, speaking of attacks, we pretty much covered all of the major player updates last week. We had our second Tuesday of the month standard Microsoft updates. Apple, of course, across the board, updated iTunes and Mac OS X and...

Leo: Huge updates for - mostly not security. Well, some of them security, but the Apple stuff was also about iCloud and iOS 5. And it was half a gigabyte.

Steve: Right, and there were also some Safari updates that were security, and also some performance, actually, of Safari's JavaScript processing. They were claiming they'd made it faster. Also I mentioned, you and I talked the week before that, our topic was BEAST, the browser exploit against SSL and TLS. We shared the fun chronology of the development of that hack. And our listeners will remember that, at the time, I described that the developers of this looked into the Java source for a vulnerability to fit their particular needs and found one. And Mozilla blogged shortly afterwards that they were considering blocking Java in order to protect people from the BEAST. What ended up happening was that Oracle, who now bought Sun and controls Java, immediately, I mean, relatively quickly updated Java to what is now Version 6, Update 29.

So I wanted to let our listeners know - oh, and Mozilla did update their blog saying, okay, now that there's a new Java, we're not going to do anything as draconian as just shutting down all Java in Firefox. I was at 1.6.0 Update 26, which is Version 6, Update 26. So I went up to 29. You can just go, if you Google "verify Java version," then the first link you get is to sort of the official page that Oracle is maintaining at Java.com that will allow you to click a button to have Java test itself and tell you what version you're using. You can also open a DOS box, it turns out, and if you have a DOS prompt you can just type "Java -version," and it'll dump out - in my case it dumped out three lines of various version information, so plenty of detail there. But you do want to be up at Version 6 Update 29.

Technically there is a Version 7 now, which is, as it sounds like, a major update, adding a bunch more features and so forth. But it has not yet been widely deployed. It's available for developers and offers some additional performance enhancements, some security enhancements, and even support for using the Java Virtual Machine with non-Java languages, so that you'd have languages other than Java that would compile to the Java byte code for speedy interpretation by the Java Virtual Machine. So new stuff is coming. And that's really all I had in the way of updates.

I did just - this is completely random, but I thought I would share it with my listeners because at some point I'm going to mention this, and people are going to go, what? I'm back to v3.6.23 of Firefox, having come all the way back from 6, because I just got fed up with its memory problems. Firefox, I don't know what happened, but at 4...

Leo: It's horrible. It's horrible.

Steve: Yes. At 4 they broke it. And when I would unblank my machine in the morning, Firefox would have grown to 1.6 gigabytes. It was basically taking over my machine because I had left it up with some tabs open, which is ridiculous. And so, as I was experimenting with it, if I would open a new tab and then close it, that would sort of wake Firefox up somehow, and then I would watch its memory consumption drop back down. Not back where it should be. It would go back to about half of that, about 800MB. And then if I closed it and then reopened it so that it would just reload freshly those tabs that I had left open, it would be back at maybe 250MB. So something was really wrong, and I just objected to it in principle.

So I went back to 5. I tried 4. I went to the very first 4. That was still doing it. So I thought, okay, forget this, and I went back to 3.6.23, which is the last one prior to 4,

which they are still maintaining, and no problem at all. It's a beautifully behaving, every morning nothing has grown, it's taking up much less memory. And it's like, okay. I'm on the verge of abandoning it altogether in favor of Chrome. But I like the tabs on the side. Chrome on Windows gives me tabs on the side, as we've talked about. But they are much taller tabs, so they just take up more space.

But, and I like - obviously I like having NoScript and a number of the other plug-ins that are available for Firefox, just because it's been around and is so much more mature than Chrome is. But, boy, I don't know. They've got to get their act together, or they're losing me. It's annoying. So I just thought I would share that. If any listeners have been seeing the same things, for what it's worth, 3.6.23 works just beautifully. Yeah.

So, security news. The EFF, our friends at the Electronic Frontier Foundation, were as concerned as we were about Amazon's announcement with their Kindle tablet and their so-called "Silk" technology, which in our careful parsing of their privacy agreement and technology statements - you'll remember a couple weeks ago, Leo, you and I looked at this very closely, trying to figure out what it was they were doing, that is, were they going to be proxying SSL? We had talked before about how Opera behaved and how the Silk browser or the browser in the forthcoming Kindle tablet might have a certificate for Amazon's cloud which would allow it to filter and enhance the performance of SSL encryption.

And so the EFF was as concerned as we were, got a hold of the lead developer for the whole Silk project. And I wanted to give a tip of my hat to Fabrice Roux in Marseille, France, who tweeted me the link that put me onto this. So I was really glad to have the information. So here's the deal. Absolutely never, not under any situation, circumstances or configuration, will Amazon interpose themselves with SSL connections. Period. They're never doing it.

Leo: Oh, yay, yay.

Steve: Yes.

Leo: That's the right thing to do. Fantastic. Of course, that breaks all caching on secure sites.

Steve: Yes. Well, it means, yes, it means that the cloud services are not available for SSL.

Leo: That's fine.

Steve: But I really agree with you, Leo. I think that is absolutely the right thing to do.

Leo: Ironically, it's Amazon's own site that won't be sped up.

Steve: Right. Maybe they can...

Leo: Actually, you know what, that site they could speed up because they legitimately have the certificates for that.

Steve: Yeah. There was a group that I talked to a couple months ago, they have a project called "Cocoon." And I spent some time on the phone with their techie guy. And they are proxying SSL. And I just said, you know, that's just not okay. I mean, they're doing it so they can filter and interpose and add AV and protect you from downloading stuff. And I said, I'm sorry, but that's a problem. That's something that I just cannot get comfortable with, the idea that, well, the idea that some service like that, which is free, is seeing my username and login when I'm logging into things, and also seeing secure session cookies. I mean, it just, as we all know, everyone who follows this podcast knows it is too difficult to be perfect. And perfection is what's required in order to have perfect security. And the bad guys are just looking for the weakest link in the chain, just one little notch, one little niche somewhere where a mistake was made, and that's all they need.

So it is just - it is better for no third party to try to get itself involved in our secure connections, so they never have that responsibility. I'm sure that's what Amazon was feeling was we don't want that responsibility. So, first of all, SSL is never intercepted by Amazon with their Silk cloud services. Secondly, all Silk optimization can be completely disabled on the very first browser options page. So it is right in your face. I haven't seen it myself because I don't yet have the Kindle.

But the EFF, in talking to the developer, he made it very clear that for people who were concerned about the privacy, the potential privacy, tracking, logging implications of having Amazon interposing itself, sort of as an active or overactive caching service, I mean, they are deliberately looking at the page you are requesting and going and fetching all of the third-party and successive content in the page, even before your browser gets it and then turns around and asks for it. So they're actively participating for the sake of speeding things up. That does make people nervous, as I think it should. So the good news is, if you don't want that, you can turn it off. And then your individual Kindle browsing will go direct to the third-party, well, first-party and third-party sites, and Silk and Amazon are not in the way at all. So from your connection, your system will resolve DNS and make direct connections, not running through Amazon.

Now, on the good side of Silk, something that I had guessed and was hoping for was confirmed, which is when you are using Silk, which is by default enabled using the Kindle tablet, they do create a single persistent connection, and it is always encrypted. So what that means is users in WiFi hotspots end up with complete protection. When you're using SSL, you get the encryption of SSL, and Amazon is not a third-party proxy in your connection. There's no man in the middle. When you're not using SSL, that is, the rest of the time as you're poking around the 'Net, just doing other things on nonencrypted web pages, and you have Silk enabled, and you've left it enabled, then Amazon establishes a single connection. They're using a protocol that we'll talk about soon called SPDY, which is short for "speedy." And that's some of the work that Google's been doing with the Chromium project to make the web faster.

So this is a - it's a protocol different from and faster than HTTP, which is of course, as we know, the hyper text transfer protocol that everyone has always been using for web surfing. So the Chromium project, the Chromium guys, as we've talked about this in a couple different contexts in the past, is really looking carefully at where time goes, how delays creep into web page loading - and, for example, we talked before about how one of the things they've experimented with is coming up with ways to improve the speed of

SSL, to cut down on the number of roundtrips that data have to make over connections. That's one of their make-the-web-faster aspects.

Well, this SPDY protocol that they have developed and have a preliminary spec for is what Amazon is using. So it is an encrypted end-to-end protocol which allows Amazon to stream the content of pages over the single connection. And what it means is that, even in open WiFi hotspots, when you're using the tablet with Silk enabled, nothing that you do can be eavesdropped on, whether you're using SSL connections or not. But the EFF, being the EFF, is still concerned about logging because they recognize that with Silk on, for non-SSL connections, we're running through Amazon, and Amazon is busy, sure, on our behalf. But their question was, what's being logged? So I want to read exactly from what the EFF wrote so that I don't misquote this. They said:

"For the persistent SPDY connection between the device and Amazon's servers, Amazon assures us that the only pieces of information from the device that are regularly logged are the URL of the resource being requested, a timestamp, and a token identifying the session. This data is logged for 30 days. The token has no identifying information about a device or user and is only used to identify a particular session." So that's a so-called "fully opaque nonce," or an opaque cookie, essentially, but not a cookie in the normal sense. It's their own session token.

Then they say: "Indeed, Amazon's director of Silk development Jon Jenkins said, 'Individual identifiers like IP and MAC addresses are not associated with browsing history and are only collected for technical troubleshooting. We repeatedly" - we the EFF - "repeatedly asked if there was any way to associate the logged information with a particular user or Amazon account, and we were told there was not, and that Amazon is not in a position to track users. No information about the ongoing requests from the AWS servers is logged. With respect to caching, Amazon follows caching headers, which offers some protection against caching sensitive information sent over HTTP." Meaning that, for example, there are headers that say "No caching, please." And so Amazon will honor such headers, which is the right behavior. It's the only proper behavior for any sort of a cache on the web. It's very important for anything that's caching to obey the headers which talk about expiration and non-caching and so forth because otherwise all kinds of things break.

Once upon a time, years ago, Leo, you may remember, I'm sure you do, that there were strange sort of caching things where you'd get obsolete copies of pages, and you'd have to do, like, control-shift-refresh or something. Or you'd bring it up, even with a different browser, and now that browser would show a different instance because of, like, local caching that wasn't working right. But those are just sort of people not implementing old or original protocols properly, which they have since fixed.

However, the EFF page says Amazon does not operate as an anonymizing proxy. So we should not assume it does. For example, even the IP addresses of the user are not shielded by Amazon. We might think that they were because, if we're making requests which go to Amazon, and Amazon is in turn making those requests on our behalf, which is what a caching proxy does, you might say, okay, that means that sites that we're technically visiting are actually being visited for us by Amazon, so those sites would be seeing Amazon's IP. Except that proper behavior for such a web forwarder or web proxy is to add their own header. There's a header called X-Forwarded-For, so it's a forwarded-for header. And Amazon is putting that into their queries on our behalf, and that means that our IP address is forwarded by Amazon to the destination. That can help prevent the breakage of things which are IP based, which is why this is done.

So Amazon is providing no anonymizing services. And again, it's like they're playing by

the rules as a standard web proxy, not wanting to get in the anonymizing business. So I guess users should accept the idea that there will be a privacy versus performance tradeoff. If any of that upsets you, you can easily turn off Silk and see how it feels. I mean, I can't wait to play with this and see what kind of benefit this much-touted technology with Amazon in the cloud actually provides us, and whether it just feels like it's worth the modest, what I think is probably a modest loss of privacy, and probably not a big deal.

Google, also in the news security-wise, has taken another step towards SSL. They announced on Tuesday, that's just yesterday for when we're recording this on Wednesday, that they are going to be defaulting to SSL connections for users.

Leo: Woohoo.

Steve: And I wanted to make - yes.

Leo: Although it seems more like a publicity - well, I'll let you tell me whether it is or not. Come on, really? Do we need to have secure searches?

Steve: Well, the concern has been that without secure search, anybody who's looking at your connections is able to see what you're searching for and the results you get. And so it's like, yeah, I agree, Leo, it doesn't seem to me like a big deal. To me, what I like about this, and the reason I mention it, is it's just - it's movement in the right direction. And I also noted that both on Amazon and at the EFF and in looking at all this Google news, there's never a mention of TLS.

Well, so I wanted to formally say to our listeners that I'm also going to stop constantly saying SSL/TLS because that's just too obnoxious. Nobody else seems to be talking about TLS. We know that technically SSL's been obsoleted in favor of an upgrade which is TLS protocol. But everyone calls it SSL. It's just simpler. So I'm going to be saying SSL from now on. And from now on our listeners will know that, yes, that's technically the Transport Layer Security instead of Secure Socket Layer acronym. But SSL's what everyone is calling it. I just think it's easier and less confusing.

So what Google is doing is for their logged-on users. That is, if you have logged into Google Services, so that when you go to Google it knows who you are because your browser is returning a persistent cookie, a session cookie identifying you, what Google will do is, if you just enter Google.com into the URL, or www.google.com, then your browser will be redirected to a secure instance of that, so security will come up, so that the searching that you then do and the query you make will be over an SSL connection. Well, I tried that this morning, and it didn't work for me. So they may still be in the process of rolling it out. I don't understand why it didn't because...

Leo: That's not unusual. That's not unusual.

Steve: Yeah. I did log in; and it's like, okay, now it knows who I am, and I tried it, and it didn't work. Now, this has caused some concern for educational institutions because they're a perfect example of an organization that has an arguable need to block queries that they feel are inappropriate from their students and to filter the results for

inappropriate content. And searching Google over SSL prevents them from doing that. Or at least it makes it much more difficult. So being sensitive to that, and this is of course an issue that Google understands and has dealt with, Google on their pages which describe this move that they're making, they provide a workaround for such educational institutions.

This has to be rather large because it requires that such an organization be running their own DNS. So this is probably university scale, I would think, more than just elementary school scale. And what they tell such people who want to defeat this automatic, and I would say this forthcoming automatic switch to secure search is to put in what's called a CNAME record for Google.com, pointing it to nosslsearch.google.com. So what will happen is, when your browser asks for - when you put Google.com in, and I guess also www.google.com, your system will look for the IP address, look up the IP address of Google.com. Instead, it will be told, oh, that's actually a nickname for, sort of a short name for nosslsearch.google.com. So then your system will ask for, oh, then give me the IP address of nosslsearch.google.com, and that will return a special IP address.

To check all that out, I checked what IPs come back for Google.com. And I get a list of six IPs: 74.125.73. and then 99, 103, 104, 105, 106, and 147. So those five are returned. And normally those will be rotating in order to do load balancing, so different users will get a list which is in a different order, and typically your browser takes the first one in the list. And if there's a problem with that, it takes the second one and so forth. So that sort of spreads queries being made by users out over all those servers. But if you look up the IP address of nosslsearch.google.com, you get just one, which is 74.125.127.114. So it's in a different C class network, off by itself, obviously expecting to have lower traffic because there's only one and not six IP addresses. And queries made to that IP don't get bounced around, that is, don't get redirected into a secure mode over SSL. They're just left alone. So that provides a means for anyone who doesn't want searching by default to be protected inside of an SSL secure tunnel to avoid that.

So anyway, mostly, in answer to your question, Leo, I just think it just represents the direction we want to be going in. We'd like to have SSL on all the time...

Leo: Everywhere, yeah, yeah.

Steve: ...pervasively for everything. And this is like, okay. And Google being very careful, they're sort of just moving us forward. It's not non-logged-on users, for example, it's only logged-on users. So they're trying not to break anything as they sort of cautiously move us forward.

Leo: It's more of a statement than anything else. It's putting a flag...

Steve: I'm sorry?

Leo: It's more of a statement than anything else, I think. It's putting a flag in the ground, saying others should do this, too. Facebook is who needs to do it. And they kind of have.

Steve: Yes, although they're still - they also sort of incrementally moved forward. We

talked about how, if you ran across an app which could not run over SSL, then you would be forced to drop back to non-SSL. And then we discovered it reset the setting of you asking for SSL.

Leo: I think they fixed that a little bit, or at least they notify you, because I've noticed lately that it's turning itself back on. So they're - I think it's good. They're paying attention.

Steve: Okay. So here's one where you just kind of have to roll your eyes and say, okay, are you really kidding me? And a little tip of the hat to Jimmy LaMaster in Fort Wayne, Indiana, who tweeted this to me, and I got a kick out of it. He picked up a story about security researchers at Georgia Tech who have managed to achieve 80 percent accuracy on word detection rates for words typed into a keyboard when an iPhone was resting next to it on the desk.

Leo: Some special app they wrote?

Steve: Well, turns out that the 3GS iPhone, the earlier pre-4 iPhone, only had accelerometers. And it didn't provide them with quite enough information. Basically it's picking up the vibration of the user typing keys on the keyboard. But as of the iPhone 4, which added the gyro, that's all they need. And so with an iPhone 4 or later sitting on your desk next to your keyboard, they're able, with 80 percent accuracy, to determine what you're typing. What they do is they break each - so every key you press is going to transmit a little bit of vibration via the desktop to the phone. And what they do is they take each pair of letters at a time.

So, for example, in the example they gave in this Wired article, take the word "canoe," C-A-N-O-E. They take CA, then AN, then NO, then OE. And for each pair they determine whether it's on the left-hand side or the right-hand side, and whether it's far or near. So this CA-AN-NO-OE translates, from a vibrational standpoint, they turn that into left-left-near, left-right-far, right-right-far, and right-left-far. Then they use that to reference a dictionary where they have completely translated all the letters of the dictionary into these codes based on where the letters are on the keyboard. And so they search the dictionary with these codes, and 80 percent of the time they can figure out what word was typed.

Leo: Wow. That's impressive.

Steve: This is, you know, we've talked about side-channel attacks often where, for example, crypto algorithms are themselves secure, but weird things they do, like the amount of power they pull depending upon the actual data that they're processing or the amount of time they take to respond, where it's like the algorithm itself is doing everything right, but something completely off-axis can be observed to see into what's going on. So here's somebody typing on a keyboard, and just the vibration patterns picked up by their phone laying next to them is able to give away what they're doing. And they said that microphones, which are able to sample at a much higher sample rate, typically I guess 44KHz, give them much better access to what's happening.

And there have of course been plenty of articles about how listening to someone typing

can reveal often what they're typing. And they said that, by comparison, the sample rate from the phone's inertial and gyro positioning systems has a much lower resolution. It's about a hundred samples per second is all they can get. But they said microphones are better secured. That is, apps will ask for permission before allowing a microphone to be used; whereas accelerometers, not so much because you think, oh, what can an accelerometer give away?

Leo: What could it do; right, right.

Steve: How could it spy on me? Well, turns out it can. And I did want to also mention, for people who are Android users and Firefox users on Android, that NoScript, our favorite scripting control tool for Firefox on other platforms, is now available for the Android platform.

Leo: Really. Oh, that's neat.

Steve: Yes.

Leo: That's really great.

Steve: Yes. There are a few features that don't make sense over on the Android platform, but all of those that do have been moved over. So anyone who's using Firefox on Android, as I may be at some point if that's available to me as an option through Amazon with my forthcoming Kindle Fire...

Leo: Oh, I guess you will be, won't you, yeah.

Steve: I'll want to do that. And then, just closing this, I wanted to give everybody a quick update on my Off The Grid project. Many people have said, hey, Steve, whatever happened to that? Well, I'm working on it. If you do go to the printing page of the Off The Grid pages, which are still not linked to the main menu because I haven't gone official yet, you'll see lots of changes. The ultra high-entropy pseudorandom number generator that I created and talked about is all integrated now, in and working. A ton of entropy is coming from GRC to initialize it, and entropy from all other sources is being mixed in. There's now a visible high-entropy key which you can use. And if you always use the same one, you always get the same grid.

So those things are done. I just now need to finish up the printing aspect. I want to allow people to choose fonts, set the sizes, turn off the border if they don't want to use the border, change the character padding and other things. So just user interface stuff is where I am. So I just want to let people know it's not dead by any means. It's where I am spending my time.

Then I promised a strange but happy short SpinRite story from Sam Cannon, who on October 5th, a couple of weeks ago, sent to me, he said, "My wife lost her Android phone SD data, and I simply plugged it into my laptop. SpinRite saw it."

Leo: Wow.

Steve: "I ran SpinRite 6.0, and it was all back." He said, "Another unexpected SpinRite miracle, when the estimate for professional recovery was greater than $800. Good investment for a retired UNIX guy. Happy wife, happy life. Thanks."

Leo: That's true, as well.

Steve: Exactly.

Leo: Wow. What a great...

Steve: So thanks for sharing that.

Leo: So that's interesting. I didn't know that you could work on Flash memory.

Steve: It works on everything, Leo. It's just, I mean, it does things I frankly am surprised by.

Leo: Shocked.

Steve: But I don't complain.

Leo: No, no, no. All right. I'm ready to begin, Steve. I'll put on my beanie. The lecture is about...

Steve: Spin up your propeller, my friend.

Leo: The lecture is about to start.

Steve: Okay. So we last talked about TCP in the context of how connections are established and the use of sequence numbers which maintain and establish the notion of a virtual connection. So that the point of sending this so-called SYN, or synchronized packet, is for the sender of the SYN to declare the starting 32-bit numbering of its packets. And the reason that we don't always start at zero - well, actually, as we'll learn in a second, there's a security and attack-oriented reason also. But the main reason is that a connection is identified by the source IP and source port, the destination IP and destination port. Those four items uniquely establish the identity of a point-to-point connection between two different machines on the Internet, that is, a TCP connection.

And if we were, for example, to number packets always at the beginning of a connection

starting from zero, there would be the possibility that we would create a connection, send some data, drop the connection, create a new connection, start numbering it also from zero and then, like, for example, send some data. But just due to the vagaries of the way the Internet functions, it might be that packets from our prior connection were mistaken for packets from our new connection because they had overlapping or similar numbering. So you could see that there's a need. And since we've got 32 bits of packet sequence space, we're able to sort of jump forward and make sure that when we start a new connection we say, okay, we're going to be numbering it from here, which is unique numbering for this endpoint, so that if any packets came in, then we would ignore them.

What that also implies is that there's some window during which we would believe incoming packets' sequence numbers. Meaning if they're, like, lower sequence numbers than when we began, we ruled them out. But also, if they were, like, much higher than is reasonable to assume could be valid, we would just discard them also. Well, that's important because it gave us some protection for the first kind of attack on TCP. And what we're going to be talking about here are attacks which are sort of inevitable due to the protocol. They're not vulnerabilities created by mistakes anyone made. They're just the demonstration of what clever hackers can do when they look at a protocol and think, okay, how can we get up to some mischief?

One of the important links between routers, and we've talked about routers in this How the Internet Works series, is them sharing their routing tables with each other. So a router in a given location has networks that its interfaces are connected to. And so it also has an interface connected to another router, and it wants to advertise to that router the networks that it services. That way the other router, when it receives a packet which is intended for networks that that first router has access to, it'll go, oh, I know which of my interfaces to send this to in order to forward it on its way.

So this is called - they use a protocol called BGP, Border Gateway Protocol, which uses TCP connections for security. That is, as we've discussed, there is an automatic spoof prevention in TCP as opposed to, for example, UDP or ICMP protocols, because of this three-way handshake. When TCP sends its SYN packet over to the other endpoint that it wants to connect to, that endpoint needs to acknowledge that by returning the packet. No hacker is able to know what sequence number a given SYN packet that's establishing a connection will use. So that gives their dialogue some protection against TCP/IP spoofing.

So BGP was always assumed to be safe for its communications among routers. Except that, as we know, attackers are very clever. And in some routers and in some regular old-school UNIX systems we're not doing a good job with this initial sequence number randomization. So bad guys figured out a way of actually taking over, splicing into an existing TCP connection. By sending some connection requests to a router, they would get back some - and by that I mean they would send SYN packets, synchronized packets to the router. They would get back the router's answering SYN/ACK packet, which contained its SYN, that is, contained its direction synchronized number. So that allowed them to figure out where the router's own internal stack was at that point in time. And by doing some guessing and being clever, they were able to surmise when routers were talking to each other, to probe a router, to have it divulge the current state of its internal sequence counter, which, for example, might always move up by 64K.

There were some simple algorithms originally used where - because, again, the developers said, well, sequence numbers are to protect packets from being confused with older ones, so here's what we'll do. And they did something that solved that need but wasn't attack proof. And so hackers were able to notice the pattern of the way initial sequence numbers were being allocated by a router and then guess what the sequence

numbers would be in existing dialogues and essentially splice themselves into an existing TCP connection; using BGP, this Border Gateway Protocol, poison the routing tables of routers; and end up being able to cause routers to divert traffic to faraway places. And we've heard that this was being done before. I've never gotten into the details of how it was often being done because we weren't at that level of detail. Now we can understand how that's possible.

So you have TCP connections between routers which are assumed to be secure, but they're not because this initial sequence number can be determined. If the router is not really using cryptographically secure determination of sequence numbers and actually was using a very simple approach, bad guys were able to inject packets into the other router, simulating spoofing the router whose sequence numbers they've been able to probe, and get that router to accept bogus routing table information which immediately the router would adopt, and suddenly all traffic would go off somewhere else. I mean, maybe across the planet somewhere. So when it was found out that hackers had figured out how to do this, of course, there was a big kerfuffle, and people updated their routers' operating systems in order to prevent this from happening, and we stopped having that kind of problem. But that was one of the earliest protocol-level attacks on TCP. Now, the more famous attacks - say what?

Leo: I said nothing.

Steve: Oh. That was actually an echo coming back to me.

Leo: Hellooo. Sorry about that. Continue on, my friend.

Steve: The more famous attacks, of course, are denial-of-service attacks, which have been plaguing the Internet and the industry for a long time. And there were several types of denial-of-service attacks which are a little more clever and which systems turned out to be more vulnerable to than most people appreciated. We talked about how a TCP connection is established. The user, for example, of a browser wants to connect to a server. So we, the user, send a remote server a SYN packet to declare our interest in connecting to it. The server, upon receiving a SYN packet, allocates a chunk of storage to represent that connection, meaning that it fully believes we're good guys. We want to connect to it. So we're declaring, for example, in that SYN packet, this is our initial sequence number. This is the port we're connecting from, and from this IP.

And remember, these are the things which make our traffic to that server unique from all the other people in the world that have connections to it. So it needs to allocate memory, to start some record-keeping, to record the sequence number from that remote IP and that remote port which is coming into it. And it then generates its own sequence number to send back in its SYN/ACK acknowledgment, which also declares its sequence number. So it needs to record what sequence number it's starting with because those are the numbers it's going to be numbering its bytes back to us with. So then, if all goes well, we receive its SYN/ACK and respond with a final ACK, acknowledging the receipt of the SYN portion of its packet in a so-called three-way handshake.

Well, the Internet is a strange place. We've established that. Routers by law, by agreement, have the ability, if they're overloaded, to just drop packets. If their buffers can't handle buffering anymore, if their links are full because a lot of traffic is coming in on some interfaces and all trying to squeeze out of one, but that one doesn't have

enough bandwidth, the router is just able to discard things. Which means that all of the systems have to be tolerant of delays and of lost packets.

So consider what this means for a high-profile server on the Internet. A SYN packet comes in, and it has to, first of all, assume it's valid. It has no reason not to. All the connections that are coming in start with SYN packets, so ours just looks like anybody else's. So it receives it, allocates a chunk of memory, and sends back an answering SYN/ACK.

Well, if we do nothing else, a timer in the server expires because it has to assume that we didn't receive its SYN/ACK, that we're sitting around patiently waiting for a response to our SYN packet. So that means that there also has to be timers that it's allocated as part of our connection. So there's a block of data and responsibility that goes along with this connection-opening process. But it turns out that it is possible - more in the old days than now, more in the original implementation of TCP than so much today - but it was once possible for a single user on the Internet to take down a major web server, just by sending SYN packets and never responding to the SYN/ACKs. And in fact, since we don't have to respond to the SYN/ACK, we can create a fraudulent source IP, that is, we can pretend that this packet is coming from someone else and never even get the return traffic.

So we spoof the source IP on the SYN and just send it. The server, again, has no reason not to believe it's as good as the one we sent before. So it allocates a chunk of memory, sends a SYN/ACK off to where that packet apparently came from, and it waits for the answering acknowledgment. If it doesn't come, it sends another one off. Meanwhile, we keep trickling SYN packets in. We may be sending them as fast as we can, but consider what's happening is every one of those SYN packets it has to treat as valid. Some of them are. Some of them are real users. It can't tell the difference. So it's allocating chunk of memory after chunk of memory after chunk of memory, one for every single incoming SYN packet, until it finally collapses; until finally it cannot allocate any more memory.

Normally that's not going to be gigabytes of server memory because down in the kernel there's no provision for allowing the TCP/IP stack to take over the memory of the system. It's given some reasonable amount of allocation, some reasonable elasticity to that allocation. And at some point it's filled up with what looks like connections which for some reason have never made it to the next stage. And so it turned out that it was originally extremely simple for someone to bring down a major site. It did not require, as we've often seen more recently, huge zombie fleets with tens of thousands of unwitting computers, hopefully on high-speed broadband cable modems, all concentrating their fire onto a single beleaguered server somewhere. It actually was originally possible for a single machine to take down a major site, just by sending it a bunch of SYN packets, forcing it to allocate memory, and it couldn't determine which one was valid or not. And once that memory pool filled up, it wasn't able to create any new connections, meaning that valid SYN packets coming in from people who wanted to get to a site, they had to be ignored.

So when this attack became understood, what was done? Well, a couple of things were done. First was that firewall vendors jumped in with appliances which they stuck on the front end. They stuck these appliances in front of vulnerable servers or server farms, and those appliances took responsibility for this. They would, for example, have much more memory allocated to them. That's, like, all they were was connection pool memory. And they've been hand-optimized to be able to do a better job at this. So they would respond with a SYN/ACK. And then only if an ACK came back would this appliance then turn around and open a connection to the server behind it.

In other words, this was a TCP connection proxy, which was built to be heavy iron, very robust, high performance. And it would, essentially, it would take the attack itself and isolate and insulate one or more servers behind it from this incoming SYN flood. The other thing it could do was be smarter. As it began to see that SYN packets were piling up and were not being completed, it could just start freeing memory from the oldest SYN packets forward. So it was always giving preferential handling to the newest ones, hoping that they might be valid users even amid a SYN flood coming in, trying to push them off the 'Net.

But it wouldn't just fill itself up and collapse. It would say, okay, I'm going to start throwing away the oldest ones because, if they never got - the oldest ones that had not been completed. And so when a three-way handshake got completed, it would say, okay, now we've got a good connection, and would pass it through. Otherwise it would wait until it got that completion, and in the meantime discard the oldest ones so that it was still able to receive new ones. So that was one thing that was done.

Then a very smart security researcher named Daniel Bernstein realized there was a whole different way possible to create a TCP connection. And someone else you guys know well independently realized the same thing. That is, me. I didn't find out about Dan and what he called "SYN Cookies," which existed as an option in Linux, until I proudly produced my own system, which I called "GENESIS," which was a SYN-flooding protection system. And then everyone said, "Oh, nice, Steve, but that's already been done." It's like, oh, okay. Well, I mean, it didn't keep me from having the fun of inventing something independently. And I did mine a little bit different than what Dan did.

But consider this: The goal here is to survive a TCP SYN-flooding attack of this sort, is to keep from letting any resources be depleted. This is a resource-depletion attack because just by doing something at a relatively modest rate, you don't have to burn up the wires, we just take advantage of the fact that some connections can take a while to complete, which means that servers have to be patient. If they have to be patient, then we can leverage that patience against them and burn up all of the resources they have for connections within that patience window, which is what this first type of SYN flood does. It doesn't saturate the bandwidth at all. It saturates the memory, the connection resources.

So what Dan first realized is there was a way to establish a TCP connection such that you responded to a SYN packet with a SYN/ACK and allocated nothing, which was extremely cool. He used a cryptographic approach - and I will say that I did also - to encrypt the information about the remote user's IP, source port, and initial sequence number, and encrypt that into our responding sequence number in our SYN/ACK. So that what that meant was we did a little bit of work upon receiving a SYN packet and sent back a very special SYN/ACK which involved some crypto of the incoming fields, which would still be the same in the server's final acknowledgment packet. And what that meant was that we receive the SYN, we respond with a SYN/ACK, and we remember nothing. No resources are allocated on our end in response to an incoming connection request SYN packet.

The cool thing then is, if our SYN/ACK (which contained essentially our cryptographically encoded sequence number), if our SYN/ACK made it back to a legitimate connection target, which then echoed an ACK (it was the incoming ACK when we did not have a connection established with that remote endpoint), we were able to look then at the remote IP, the remote port, the sequence number, and our sequence number which was being acknowledged; do the same crypto; and verify that essentially this was a cryptographic cookie, which Dan called a "SYN Cookie" because what it meant was the only way that this incoming ACK could be valid is if the person sending it had received

our SYN/ACK, which involved some crypto, to take the IP and remote port and encrypt that with its sequence number.

So what this technology did was essentially to solve the problem of resource depletion attacks. I implemented it independently in the way, what, like maybe 10 years ago, whenever it was that...

Leo: You had to.

Steve: ...GRC was being attacked all the time.

Leo: It was a defense.

Steve: Yes. It was a defensive measure against these kinds of attacks. And now that technology is much more widely available. In some cases, in fact, Windows incorporated it, I don't remember when.

Leo: Oh, interesting.

Steve: I don't think it was Windows 2K. It might have been Server 2003. For the sake of compatibility they're not always operating in that mode. But if they see their connection queue growing to a certain size, then they dynamically decide, whoops, this looks like we're under attack. And so it switches on this non-resource-depleting connection mode in order to respond. And Linux has, last time I looked, but it's been a long time since I looked at this stuff, Linux did not have it enabled by default. But you could certainly build a Linux or maybe use options at boot time in order to turn SYN cookie behavior on. And it didn't break many things. It was a highly compatible solution. But that solved the problem of resource depletion attacks.

Now, I should mention that it was because of all this that I got in Microsoft's face with Windows XP because none of this was possible unless you had something called "raw sockets." That is, remember that the whole point of the TCP/IP stack, as it's called, this is all networking technology in the kernel. At the application layer, everything is much simpler. All of this is being done for you. At the application layer you say, you the application - Opera, Firefox, a browser, email, whatever the application is just says please give me a connection to this remote IP and port. So you ask for that. But all of that packet traffic, all of the SYN, SYN/ACK, ACK stuff is done by the operating system in the networking stack.

You cannot produce - there's no code, there's no API is the technical term, an Application Programming Interface, there's no API for generating a SYN packet. You can't do it. You can send an ICMP packet to ping or to do a traceroute. But you cannot generate a SYN packet unless you have something called raw sockets, which is an API that was originally available in UNIX only. It was also available in Windows 2000. And what I saw Microsoft doing, as they were going to XP, was they went from Win2K to WinXP, basically just changed the UI, gave it a whole candy sugar coating user interface so that it no longer looked like Windows 3.1 and Windows NT, which is what Windows 2K looked like.

Windows 2K had a much more advanced networking technology, including, finally, raw

sockets. Microsoft was always being laughed at by UNIX people, saying, oh, well, it's not really a real operating system because you don't even have raw sockets. So I think in response to that Microsoft said "ouch" and added raw sockets, which was fine for Windows 2000 because it wasn't a consumer operating system. Consumers were still using Windows 95, 98, ME. And it was with XP that Microsoft was going to migrate all of that legacy OS over to Windows XP, which they did. That was the source of my concern because I recognized that Windows XP users were going to get themselves infected, and that malware running on XP that had access to full raw socket API would be vastly more dangerous for the Internet than any malware that had been on any previous consumer platform.

I got all kinds of flak for that because many people did not understand my argument, which ended up being shown to be correct. It took years. And it wasn't until Service Pack 2 of Windows XP that Microsoft said, "Oh, that's what Steve meant," and they removed the raw sockets API from Windows XP. Which, I mean, they never remove things. They're notorious for not breaking stuff. Apple has traditionally been willing to remove things and upset their developers because they wanted to keep things simple and limit their legacy support. Microsoft has really been reticent to do that, historically. But they did pull raw sockets out of XP because they realized they were just too dangerous.

But it turns out there was another type of problem which SYN packets created which was different than resource depletion. And that was routers. A router is designed, as we've talked earlier on in our How the Internet Works series, to accept packets and send them on their way. That's what they do. So a typical packet is around 1500. For some reason 1536 sticks in my mind, but that's also two to the power of [indiscernible]...

Leo: It's got to be a power;, right.

Steve: ...that I've been seeing recently. But anyway, so it's about 1500 bytes. So the idea is, if you've got certain bandwidth on your connections, and packets are coming in at 1500 bytes at that bandwidth, that means that you're going to have packets at a certain rate.

Now, applications are typically giving the TCP/IP stack a big blob. You may send a small query to the server, and that maybe fits in a single packet. But it's going to respond with a page, which is going to be tens, maybe hundreds of K. So the server up at the application level sends this 100K file, probably out of its cache, if other people have asked for that page, down to the TCP/IP stack and says, "Send this off." So the TCP/IP stack, wanting to be efficient, chops this up into the largest packets possible.

And there is technology which we briefly talked about for figuring out how large a packet I can get from one endpoint to the other because one of the ICMP plumbing protocol features says that I was unable to forward a packet from where I am to the next router because the interface, the technology over the interface can't take a packet the size you've given me. So routers can fragment packets, but you can also tell them "do not fragment; instead, send me an error." So it's possible for links to establish the maximum packet size, which is worth doing because you get a lot more efficiency. So what I'm trying to say is that, in general, the Internet works with packets being as large as they can be because...

Leo: Isn't that the MTU size? When I see an MTU setting, is that...

Steve: Yes, Maximum Transmission Unit.

Leo: Okay.

Steve: Yup, that's exactly what that is. And so routers are always being designed to be as inexpensive as possible. And if you've ever looked inside of your typical Cisco router, it's a little old MIPS chip from the 1900s. I mean, just there's not much in there, which is why Cisco was doing so well for so long is they were able to sell something that looks pretty impressive from the outside, but it's hollow on the inside.

Leo: It's all industrial design, yeah.

Steve: Yes. And it actually doesn't have to do very much. It just receives this packet. It looks at the first few bytes. And then it sticks it in an outbound buffer, and off it goes. So what happens with small packets? What happens is the switching rate. At a given amount of total bandwidth coming into the router, routers are rated in terms of the amount, the number of packets they can process per second. PPS, packets per second, is one of the specs for routers. And what it turns out is almost no routers, in fact I could almost, I think I could say no routers are able to process the amount of bandwidth they can handle in small packets.

So if packets are 1500 bytes - and I'm reaching for my calculator because I used to know all this stuff by heart, but I haven't talked about this for a long time. If packets are 1500 bytes for regular data-bearing packets, and SYN packets are 60, which is a number I seem to have in my head also, well, that's a huge difference between a data-bearing packet of 1500 bytes and a SYN packet of 60 bytes. That's a 25:1 ratio, which means if routers are receiving the smallest possible packet, which is a TCP SYN packet, then given the amount of bandwidth connecting them to the Internet, somebody sending SYN packets back to back as fast as possible is able to present the router with 20 times the rate of packet headers. The headers are what causes the router to do work because it's got to look at the header, read the destination source, or the destination IP, and send it on, also check its routing table and figure out which interface it goes to and so forth. So there is substantial per-packet routing overhead.

So just by sending smaller packets somewhere, all the routers between you and the destination, if you send lots of small packets, are doing far more work, and they're just not rated for it. They're generally rated for a mixture of packet sizes, and you can see that in the specs, too. They'll say they can handle 20 percent minimum-sized packets, 80 percent typical-sized packets. And they'll explain that minimum-sized packets are a lot more work. So one of the not-very-well appreciated, sort of unforeseen consequences of SYN floods is that they were the smallest TCP packets possible, and they were 1/25th the size of the normal payload-carrying packets that routers switched, and they just collapsed the router. The routers just went underwater, could not, I mean, in a way that they were ill-equipped. Routers can handle dropping packets by design if their interface buffers are overfull. But if you just pour in packets at 25 times the rate they expect, the router just collapses. I mean, it reboots, it crashes, it freezes up, it does horrible things.

And so what the DoS attackers learned was that they may not even be reaching the actual server they're trying to attack. If they do have a number of bots, and again, not even that many, all concentrating their fire at a single IP, their traffic will be aggregating around the Internet as it begins to filter closer and closer toward the target IP. And at

some point the routers along the way will just collapse under the strain. So the server is sitting around with no traffic because the routers serving it have collapsed. And of course users, valid users who send innocent TCP/IP SYN packets or just want to transact regular data, they can't because the routers have been pushed off the Internet by the packet-switching rate, the number of packets that it's possible to send a router if each packet is very small.

And so that ended up being sort of an unforeseen side effect of these denial of service attacks. It was small packets that routers - it could be 25 times the number of packets per second that DoS attacks were creating. And the result was the same as if the server itself had been blown off the Internet, but the attack never even got to the server because it collapsed the Internet's infrastructure upstream of the server. And that's our episode on attacking TCP.

Leo: There you go. From a man who has had some experience with this, direct.

Steve: Unfortunately, yes.

Leo: Yeah, alas. Very interesting stuff. And very practical, if you have a website. Now you know what's happening.

Steve: Yeah, and really no one's fault. I mean, these were...

Leo: Right.

Steve: It wasn't because of implementation problems. Many people have scrambled around, and there are robust routers now that talk about their packet processing rate, and then they say they're able to handle the bandwidth of their interfaces at minimum-sized packets. But that required more cost and more expense, and it wasn't the case in the old days.

Leo: So you would use this to attack an individual, as well, I guess. You could attack their router.

Steve: Oh, gosh, an individual has none of the higher end hardware. So, yeah. You could easily take one of these little blue box routers off the 'Net by sending a bunch of SYNs to it. And in fact, that's what the script kiddies were originally doing when they were in IRC chatrooms. They would want to blow somebody who had IRC privileges off of the 'Net so that essentially they would capture the flag, and then they would be running the IRC chatroom.

Leo: Slapping them.

Steve: Yup.

Leo: If you want to know more about this, well, Steve's got a whole website just full of good stuff at GRC.com. That's where you'll find SpinRite, his bread and butter, the world's finest hard drive maintenance and recovery utility. Heck, hard drive, you can do compact Flash, too, apparently, phone memory. He also has a lot of freebies there which you should check out: his Perfect Paper Passwords, Password Haystacks, great information, lots of things. GRC.com. He's on Twitter @SGgrc. And we'll do a feedback episode next week, so you want to email a question, best way to do it is go to the website, GRC.com/feedback. And there's a form there you can fill out. There's very active forums there, too, though. Good place to go to ask questions.

Steve: Yeah.

Leo: Thank you, Steve.

Steve: Thank you, my friend.

Leo: We'll be back Wednesday at normal time, again, 11:00 a.m. Pacific, 2:00 p.m. Eastern, 1800 UTC at TWiT.tv. And we'll see you then.