Poking Holes in TCP

Description: Steve and Leo tackle two new and interesting threats to Internet security. First, the newly discovered "Flame" / "Flamer" / "Skywiper" malware dwarfs Stuxnet and Duqu in capability and complexity. Then they examine the work of two University of Michigan researchers who have detailed a collection of new ways to attack the TCP protocol. They inject malicious content into innocent web pages and add malicious links to online chats.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-355.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-355-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here, and we're going to talk about what he says is the most sophisticated malware ever written: Flame. That's next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 355, recorded May 30th, 2012: Poking Holes in TCP.

It's time for Security Now!. Time to protect yourself. Batten down the security hatches, if you will, with this man right here, our Explainer in Chief, Mr. Steve Gibson of GRC.com. And a good day to you, Steve.

Steve Gibson: Hey, Leo. Great to be back here with you again, as always. And speaking of battening down the hatches, we've got two big stories. In our news is the details which are known so far, and they're still a little scant because this has just happened, and that is the most sophisticated piece of malware or malware system ever found...

Leo: Whoa.

Steve: ...was recently uncovered by Kaspersky. When they were looking for something else, they stumbled over this thing.

Leo: That's how penicillin was found. And Post-it Notes. So it's a longstanding tradition.

Steve: Not quite the glue we were looking for, but we think we can use this.

Leo: So what makes it so sophisticated? Now, remember, Stuxnet, and I presume Duqu, were written by governments; right?

Steve: Well, and they - yes. And what makes this so sophisticated is the complexity and the power of it. It uses heretofore unknown injection techniques. I mean, basically it has a ton of new technology which has never been seen before. It is believed that this has to have been generated by a nation state. It may be more than five years old. Pieces of it have turned up. Now that they know what they're looking for, they can go back and realize that VirusTotal was seeing some instances of this. And it renames itself at one point, and those names have been picked up in archives from 2007.

So, whereas Stuxnet and Duqu, well, Stuxnet specifically was extremely focused on, as we now know, on upsetting the centrifuges of Iran's nuclear enrichment program, this is a comprehensive, general purpose, high-power espionage tool. Now, what we don't yet know is how widespread it is. Apparently it is all over the Middle East. Sort of anecdotally, that much we have. But anyway, we're going to talk about that.

And then also, what I mentioned we would talk about last week is the main topic, and that is, I titled this "Poking Holes in TCP." This came to my attention from a bunch of people who were tweeting it to me when it was in the news. And this was two researchers at University of Michigan have found some new ways of injecting malicious code - or malicious content, to make it more generic, because it is - malicious content into TCP connections without needing a man-in-the-middle presence.

Leo: Ooh.

Steve: So they can be standing to the side, essentially, and interfere with TCP. So this is - that, of course, caught my attention. Now, the good news is it's more limited than the title and the synopsis makes it sound. So it's less of a concern. But it's a perfect topic for us on a technical podcast like this because it gives us a chance to look more closely at TCP, which is the Internet's most-used protocol. More data moves across TCP than everything else summed up. Well, maybe streaming now with video and audio over UDP offsets that. But all of our web pages, all of the connection-oriented traffic is TCP based. And we've talked about, in podcasts past, the way TCP works. We'll look at it a little more closely relative to the glitch that these guys have found, which is interesting and provides some lessons for us. So I think a great podcast today.

Leo: Oh, very interesting.

Steve: And a little bit of random miscellaneous news.

Leo: And all of that, thanks to Mr. Steve Gibson. Let's kick it - we don't have a commercial. So just go, baby, go.

Steve: Let's get into Flamer, or Flame.

Leo: Flame. I don't know if I like "Flamer" quite so much.

Steve: It's been called "Flamer." It's been called "Flame." And in fact a Hungarian research group who I'll be talking about, Crysys, they named it - before they realized what it was internally named - they named it "Skywiper." So various people were tweeting to me about Skywiper as if it was something new, but it's the same as this super-sophisticated malware that blows everything else that we've seen before away.

Kaspersky was interviewed. They were the people who first discovered this, having been asked by, I think it was the ITU, the International Telecommunications Union, to look into this. They were looking for something which was wiping the drives of machines throughout the Middle East. And so, in looking at those machines, they found something else that was before this unknown to them, and that caught their attention. The guys at Kaspersky are calling it the most sophisticated malware ever found, dwarfing Stuxnet and Duqu. It is 20MB of code in total for the whole thing. And get this, Leo. It bundles SQLite as its database backend.

Leo: So when you install it, you get SQLite?

Steve: You get an instance of SQLite running in your system because - I'll go through the means...

Leo: That's hysterical.

Steve: ...of data gathering.

Leo: They need a database?

Steve: It builds a comprehensive structured database of everything it collects on its espionage.

Leo: No wonder it's 20MB. Holy cow.

Steve: Yeah, it's like a little OS that has just set up residence inside of your main operating system.

Leo: Geez.

Steve: So here we have 20MB, whereas Stuxnet was only a couple hundred K. So it's 10 times the size of Stuxnet, one of the reasons being that, as we saw, it has a complete database architecture. So, now, Iran got into it. And their equivalent of CERT, their Internet security agency is called MAHER. They have a posting that I'll quote from. They said, "Having conducted multiple" - and one reason is it's apparently rampant within

Iran.

So, "Having conducted multiple investigations during the last few months, the MAHER center, the Iranian CERTCC, following the continuous research on the targeted attacks of Stuxnet and Duqu since 2010, announces the latest detection of a new attack for the very first time. The attack, codenamed 'Flame'" - and that's due to some names contained within the reverse-engineered code. "The attack, codenamed 'Flame,' is launched by a new malware. The name 'Flame' comes from one of the attack modules located at various places in the decrypted malware code. In fact, this malware is a platform which is capable of receiving and installing various modules for different goals. At the time of writing, none of the 43 tested antiviruses could detect any of the malicious components. Nevertheless, a detector was created by MAHER center and delivered to selected organizations and companies in first days of May, and now a removal tool is ready to be delivered."

Okay. So some features of the malware: It can be distributed - and I should comment that, understand, this is compiled code. There's no source for it. They're dealing with 20MB. And, I mean, that itself makes the task of reverse-engineering this behemoth very daunting because they've got to watch it operate in a controlled mode with their own spy tools running. And these are the people at Crysys and Kaspersky and the various organizations that have received this are now, I mean, like right now, as we're recording this, they're working together, sharing their information, and working to understand it. But this task of understanding it is a matter of both watching it to see what it does and, like, watch it hook things, watch it inject code into different modules, run rootkit checkers against it to see which low-level kernel hooks it has grabbed in order to hide itself, I mean, so there's the behavioral side. Then they also have to reverse-engineer this. They have to decompile, disassemble this code and then figure out what's in there whose behavior they haven't seen. So we'll be - I imagine this will be peeling layers off of a very large onion for some time to come. And we'll be touching back on this as we learn more.

So we do know that it can be distributed via removable networks and local area networks. It's capable of sniffing the network, detecting network resources, and collecting lists of vulnerable passwords as they pass by in the clear. It can scan disks of the infected system looking for specific file extensions and contents. It is able to perform screen captures of the infected machine when specific processes or windows are active. It's able to capture the contents of any fields filled out, even when obscured by asterisks or dots so that they're password fields. It can turn on the system's attached microphone and record over a long period of time any sounds in the environment.

All of this data is saved in an SQLite database which it is then able to transfer en masse to control servers. There are more than 10 domains that have been identified as the command-and-control, the C&C servers. It establishes secure, encrypted connections with those servers through SSH and HTTPS protocols, so it's encrypting its traffic, as well, point to point. It bypasses all known antivirus detection, antimalware and other security software. It's able to infect XP...

Leo: Wait a minute. How could it do that? I mean...

Steve: Well, first of all, it wasn't known. So none of these things knew to look for it.

Leo: Right. But now that they know, won't they find it?

Steve: Well, oh, absolutely. But what it was doing also is it uses five different encryption technologies and three different compression technologies.

Leo: Code. See, that's scary. If a virus can bypass antivirus detection, that would be a scary technology to have available.

Steve: Well, and remember that many of these new AVs have a problem with false positives, specifically because they are looking for things, not behavior and not specific known signatures, but they're looking for suspicious code. And, for example, every maybe half a year or so, someone will report that some random update of an AV package now thinks that one of my EXEs, one of GRC's freeware is suddenly viral. And it's like, no, it's not.

Leo: It's usually pretty crappy stuff that does that.

Steve: Well, but again, this is - it's difficult to do because you'd like to catch things you don't know about, but you don't want a false positive. So what's significant is that neither, I mean, even behavioral analysis didn't catch this, apparently for five years. Nothing saw this hooking DLLs. For example, Stuxnet and Duqu have relatively straightforward code injection technology. This is completely new. This is using technology that we've never seen before. So nothing knew how to look for it. So it can infect XP, Vista, and Windows 7. And it's apparently expert at infecting large-scale local networks. Quoting from one of the reports, they said, "According to file naming conventions, propagation methods, complexity level, precise targeting and superb functionality, it seems that there is a" - oh, I'm sorry, this is still from Iran - "it seems that there is a close relation to the Stuxnet and Duqu targeted attacks."

Now, that actually turns out not to be the case. Kaspersky and Crysys, who have looked at this independently, both think there's no connection. Whatever this is, it was created by a different organization or group than Stuxnet and Duqu. Whereas those were targeted for specific purposes, this is a hugely comprehensive general purpose espionage tool which has been out there doing its job in secret maybe for five years. So they...

Leo: Five years. Well, maybe it does get around antivirus. Holy cow.

Steve: Yeah. Yeah, that's what I'm saying. I mean, that's what has shocked people is they have found the...

Leo: But if it's a spear-phishing attack that's going after Iranian government installations, it might not be detected because it's just not out in the wild. Or is it?

Steve: Well, for example, I mean, scroll down to my notes, one of the files, the core file which is downloaded by the initial loader is called mssecmgr.ocx. So that's an ActiveX

control of Windows. It renames itself, or rather that file is renamed by the installation component to wavesup3.drv. That file has been observed, was observed in Europe on December 5th of 2007, so five years ago nearly; in the UAE on April 28th in 2008; and in Iran for the first time on March 1st of 2010. So, and it did not raise alarm. It didn't trip any alarm. No behavior was seen.

Leo: Geez Louise.

Steve: I know. I mean, it sounds like science fiction that we're talking about. And it's as real as any of these malware threats that we've seen recently. So they said, "The research" - this is Iran continuing. "The research on these samples implies that the recent incidence of mass data loss in Iran could be the outcome of some installed module of this threat." Now, again, that's at this point hypothetical. "A list of the major infection components of this malware is presented below. These samples would be available for security software vendors." And they list some, but that's no longer unique.

Then the Crysys guys - who are in Hungary, they're the Laboratory of Cryptography and System Security - they're maintaining a paper that is tracking this. They're at Crysys.hu, and their paper is crysys.hu/skywiper/skywiper.pdf. And that was just the name they assigned it randomly before they realized there was a name embedded in the code. So they have a technical report that they've put together, noting that there's been deliberate obfuscation of dates. For example, the EXEs themselves have dates set back to '97 for whatever reason, to make them look like they're not new. But tearing these things apart, they have found instances that are much newer. For example, the version of SQLite which is installed inside is v3.6.22 from January 5th of 2010. So this is in fact much newer, although the date stamps are deliberately set back to make it look older.

So Crysys in their report says that they have found evidence of five different encryption methods. And I happened to note one of them is as simple as XORing with FF, so that just flips, it just inverts all the bits of the byte. So that's very fast, and it's simple.

Leo: And it's effective. Unless you're looking for it. It's easy to decrypt; right? But only if you're looking for it.

Steve: Yes, right. Then but there is, for example, also RC4 encryption, which is strong when it's applied correctly. It was what was encrypting WEP protocol originally, and in fact still encrypts TKIP when it's being used because there's nothing wrong with it when you use it correctly. So five different methods of encryption. Three different methods of compression. Five different known file formats plus some proprietary formats. And, for example, the SQLite database is one of the well-known formats, which it just uses unmodified. It uses, as I mentioned before, completely new, previously unknown code injection technology. It locally stores information which it gathers in a highly structured SQLite database. Get this, Leo. It uses the LUA scripting language.

Leo: Oh, yeah, excellent choice. LUA, I actually have studied LUA. It's a really nice scripting language. You know what's written in LUA is, well, a lot of games use LUA, yeah, exactly, for a command kind of code. And then Lightroom, Adobe Lightroom is written in LUA. It's a scripting language. It's a good glue language.

Steve: That's true. It's a nice language. It's also free. So it's nice to know...

Leo: It's open.

Steve: ...that these people were not pirating.

Leo: Use open resource, yeah.

Steve: Yes. They were not...

Leo: Wordsworth had an interesting hypothesis in the chatroom. He says it sounds like it was engineered in small modules, deployed independently, to see whether they could be detected, prior to being integrated into a larger malware package. Is that - does that - is that the feeling you're getting? It's interesting, yeah.

Steve: Yeah, it's certainly feasible. Apparently, like all of these late-model malware, there is a component that installs itself and hides itself and arranges to run. So you've got to have that.

Leo: A little stub program. It's not very big, I'm sure.

Steve: Yes, exactly. And then it knows how to go check in with headquarters. And we've seen various types of command and control. The most recent ones are those where it uses the time and date and a cryptographic algorithm to dynamically generate domain names on the fly. So they're not built into it. You can't just, like, look at a list of domain names. But at a certain time and data, it dynamically generates a domain name, and the bad guys have preregistered that domain and arranged DNS so that it's able to look up where they now are. So, I mean, this has gotten very sophisticated. And unfortunately, the 'Net provides this kind of power. It's the benefit of the Internet, but also a substantial liability.

Leo: This is the weak point, the weak link, I guess, in these viruses is they need to contact a server for command and control. And knowing that information lets you block the server. So they're being very clever about delaying the activation of that server and obscuring. But as soon as they connect, isn't it obvious? I mean, don't we then know, ah, we got 'em? Because they have to make an explicit connection.

Steve: Well, we know there are also - sometimes they use TOR. Sometimes they bounce through relays a few times. But yes, your point is it's very much the same way that our domestic FBI follow the money because the bad guys who are extorting have to somehow get paid. That's their weak link. And so it's possible, similarly, to track that and catch them, which is often what happens.

So information gathering by this master espionage program are keystrokes on the keyboard; screen captures which are grabbed and compressed and then stuck into the

SQL database; the microphone, so audio being recorded; contents of the drive, and not just all of it, but specific searches and requests are fulfilled. It's sniffing the network and collecting data from the network. It knows about WiFi and is able to exploit that when available; it knows about Bluetooth, and, if Bluetooth radio is on, it will sit there and monitor the Bluetooth channel; and USB and system processes. And it contains rootkit injection and hiding techniques, also new, never seen before.

So Crysys said, "The result of our technical analysis support the hypothesis that Skywiper," which is their name for the same thing, "was developed by a government agency of a nation state with significant budget and effort, and it may be related to cyberwarfare activities." Yeah, you think? Uh-huh?

Leo: Mm-hmm, mm-hmm.

Steve: "Skywiper is certainly the most sophisticated malware we encountered during our practice. Arguably, it is the most complex malware ever found."

And, finally, as I've mentioned before, it looks like it's been around for at least five years doing something, doing its thing. And part of the modular nature of this is the fact that, once you get that kernel installed and propagated, you are able to evolve this over time. So today's Flame, or as Crysys calls it, Skywiper, may not be what was there five years ago. These capabilities can be added over time, and these things are able to evolve.

So as I mentioned, it's been found around the globe for many years. The whole package is 20MB. This one key, that mssecmgr.ocx file, is generally around a meg, but there have been different versions of it each time that it's been seen, sometimes a little larger than a megabyte, sometimes a little smaller than a megabyte. And Crysys said, since they were heavily involved in the discovery and analysis of Duqu, their preliminary analysis suggests that Skywiper was NOT, they said in bold caps, made by the same development team as Stuxnet and Duqu.

Leo: Interesting, wow.

Steve: Really, really interesting. Now...

Leo: There's a novel in here. There's a nonfiction - I would love to see somebody like John Markoff really dig his teeth into this and figure out what happened and the whole story because, boy, that'd be a fascinating book. Fascinating.

Steve: It's interesting, too, how with the archives of the past that we're now beginning to accumulate, just because we've got virtually limitless hard drive storage, that it is possible to turn back the clock and find things that once existed.

Leo: Right.

Steve: So it's possible to look at the state of the Internet five years ago.

Leo: Archeology. Internet archeology.

Steve: Yeah. And do true analysis, like not only reverse engineering of this, but reverse engineering of its history before it was known to exist. So it really can be done. So anyway, I will certainly be looking for people who tweet any new findings about this, and I'll track them down and figure out what they mean and report to people.

Leo: Fascinating, fascinating.

Steve: And speaking of tweeting, I got a couple interesting little bits from the Twitterverse. A couple tweets from people who do work at ISPs who have been working to remediate DNSChanger troubles. We talked about this a week or two ago, and you remember you had strong feelings about the FBI having commandeered and given control over to the ISC.

Leo: Yeah. The FBI-in-the-middle attack, I call it.

Steve: Yes. So I think there was a little bit of stubbed toes from ISPs feeling like, well, we were saying they weren't doing anything. It turns out it's just a difficult thing to do. So they are working to detect when their users have computers that are checking with these malicious DNS servers and somehow trying to notify them of that fact. So that was good to know.

And I mentioned an alternative to Permit Cookies, a Firefox extension I have loved for years and used until it seemed, for me, to stop working. But Todd Eddy in Ohio sent me a tweet noting that he is maintaining the abandoned Firefox Permit Cookies add-on, but only to repair critical bugs, and he continually updates the version compatibility tag. I don't know what the problem was. I did try to update mine to this one that he's maintaining, I think under Firefox 12, but it might have been 11. But it no longer functioned, and so I abandoned it for that pushbutton cookie gizmo [Cookie Whitelist With Buttons] that basically replaces it that I told our listeners about last week.

And finally, I got a note from someone who's written before, Christian Alexandrov, who apparently suffered a 5.8 magnitude earthquake in Bulgaria, and many hard drives were damaged, not surprisingly, by that. He said, "Hello, Steve. I want to share a SpinRite story." Oh, the subject was "When earthquake strikes, SpinRite strikes back." He said, "I want to share a SpinRite story with Security Now! listeners. Recently, a few days ago, earthquake struck our country. Earthquake's magnitude was 5.8." That's a big earthquake, Leo.

Leo: Yeah. I mean, well, I mean, okay. Big for Bulgaria. I mean, here in California, well, that's just a little temblor.

Steve: Yeah, we would just have our overpasses rocking a little bit.

Leo: No, that's a big earthquake. That's big, yeah.

Steve: "Bulgaria is in seismological calm area, and such earthquake..."

Leo: That's why it's a big deal, see, because it's unfamiliar to them. They're not prepared.

Steve: He says, "Such earthquakes are rare. The last earthquake equal to this one was 167 years ago."

Leo: Whoa.

Steve: Yeah. I wonder how they know that because did we have the Richter Scale 167...

Leo: Oh, yeah, yeah. Well, something like it.

Steve: Yeah. "The earthquake created a lot of work for me and for SpinRite. I go all over Sofia City to help to revive hard disks that suffered damage caused by the earthquake." So clearly, pedestal-based PCs fell over while they were spinning.

Leo: Wow. Wow.

Steve: And that's not good for a hard drive.

Leo: That actually is quite a bit of shaking, if your PC fell over.

Steve: Yeah.

Leo: Geez.

Steve: And so he said, "Within that entire range I maintain, only four PCs were not damaged at all, regardless of the fact the PCs were working during the earthquake." And then I'm going to skip this paragraph where he talks about the four that weren't damaged. He says, "However, many places I had to go with SpinRite as my tool of choice. A lot of hard drives were damaged. SpinRite gave a lot of drives that special care they needed to make them good for work again. Some people called me to try and fix their hard drives before they asked for help from PC repair services, or sometimes they called me after."

Get this. "Of all the range of PCs I maintain, I was the only one who used SpinRite and brought over 180 damaged hard drives back to life. It was a wide variety of hard

drives..."

Leo: Holy cow.

Steve: "...from 20GB UATA/33 to 1TB SATA3 drives that were saved and brought back to life by SpinRite. Needless to say, there were a lot of 'U' and 'R' red squares on all drives on the first Level 4 scan, and a lot of 'B' icons on second Level 4 scan, where the 'U' and 'R' icons were before. And also needless to say, what large number of files were saved and a large number of systems now work fine.

"Suddenly, I become a PC hero for saving PCs and bringing hard disks back to life and saving files and OSes. Work here is still in progress. Many drives still to fix. And many are taken care of by SpinRite. I do somewhere between 10 and 15 drives at a time, and many computers have two or three physical hard drives in them. Steve, once again, thank you for your great piece of software. And once again, thank you, Steve and Leo, for the great Security Now! podcast. I wish best of luck to both GRC.com and TWiT.tv. A happy SpinRite user."

Leo: Wow. Well, there you go. Now you have a new slogan: "Built for earthquake country." Geez. That's fascinating.

Steve: Very cool.

Leo: That is really interesting. Let us continue on poking holes in TCP, shall we?

Steve: Speaking of poking holes. In TCP.

Leo: Yes.

Steve: So some researchers, two security researchers in Michigan - I have to warn everybody, if you did not have your propeller hats on...

Leo: It's time?

Steve: ...maybe this is a good spot to pause and go refresh your coffee or open a new can of - I hope you're not opening a can of soda because that's not good for you.

Leo: No. Sardines.

Steve: Oh, that's - there you go. A can of sardines.

Leo: Yeah. Go get a can of sardines.

Steve: Because this is going to get a little hairy, but it's good hairy. I mean, it's important hairy. And a perfect, perfect topic for this podcast that pushes the limits of what we're able to do over an audio channel. We're going to do that today.

So a couple researchers were looking at the security of primarily cellular networks, that is, their platform focus was Android, and Android connected to the Internet via cellular was the scenario they set up. What we'll talk about is broader than that, really, but they've only focused on that. And it's broader because it's really about the transmission control protocol, TCP, which is the basis for the web and all kinds of apps which are now using TCP behind the scenes. They may not look like a web browser, but they're actually setting up persistent connections back to a server in order to communicate. For example, when I have TweetDeck up, it's got TCP connections open through which it's using the Twitter API to communicate.

So, okay. So what they discovered was that there were what they called "middleware firewalls" installed in these mobile networks in - they detected them, I think they checked 147 mobile networks, and they found them in 31 percent of them, so about one third of them. The firewalls were there to block unwanted traffic, just to sort of, like they just stuck a firewall sort of in the middle of their network and said, okay, at this point anything going in either direction that looks funky, let's get rid of it.

Well, these are stateful firewalls which connections have been made through them. And that way they're very much like our NAT routers that we've talked about that are stateful routers, where incoming packets are just dropped because they don't correspond to a connection that was initiated from the inside out, which is what allows connections to come back, or packets of connections to come back in. So these are firewalls which, when a SYN packet, a TCP synchronized packet encounters a firewall on its way to a destination on the other side of the firewall, the firewall makes note of it. It looks at the source IP and port, the destination IP and port, and adds that to its "okay, a connection is being set up through me" table. Then, when the answering SYN/ACK comes back to the firewall, it makes sure that that SYN/ACK is expected. And, if so, it updates its tables and passes the SYN/ACK on towards its destination. So it's sort of a - it's a blockade just sitting at an arbitrary middle point in a cellular provider's network because, if SYN packets are just, like, from a denial of service attack, if a SYN packet hits it that it isn't expecting, or an ACK packet or something bogus in one way or another, it just ignores it.

Well, one of the things that these firewalls do is they validate the traffic on an ongoing basis. They watch the setup. And people who have followed along with our talk about how TCP works will remember that the reason we call a SYN packet a "SYN" is SYN is short for synchronize. And this is the connection initiator saying I'm going to number my packets starting at this number. And there's a sequence, a 32-bit sequence counter in all TCP packets which numbers the first byte of the payload in that packet. So, for example, if a sender were to send out packets of a thousand bytes, then successive packets would contain these sequence numbers that were a thousand larger each time.

Well, that's done because, as we well know, packets don't always arrive in order at the other end. The route that packets take is not predetermined by either the sender or the receiver. Those are determined just by routing tables among all the routers that link these two points. So it's possible that one router could be busy. An interface might be backed up, so it routes a packet somewhere else, so a later packet could get to its destination sooner. So it's necessary to number the bytes in the packet so that they can

be reassembled in the proper sequence when they arrive.

So TCP has been a target of attack in the past, and the sequence numbers are one of the weaknesses of TCP. So if we were to step back for a minute and come up with a lesson that we're about to learn, it is that, if you have a fundamentally unsecure protocol, there isn't a way to add security later. It was an argument that I made against Backblaze, where they sort of tried to add a user password in a hokey way to give people a greater sense of security, but it actually achieved nothing in terms of true security for the user, or it achieved very little. That is, they had a broken model where they were doing the decryption at their server end, and that couldn't change, no matter what they added on top of it.

In the case of TCP, it was designed, as we know, literally decades ago, back when numbering packets with a 32-bit count seemed like more than enough. It was that 32 bits gives us 4 billion. And there's nothing wrong with these counters wrapping, that is, counting up to 4 billion and then wrapping around back to zero. They do that all the time. TCP understands that, if it has a high packet number, and then suddenly it's got one very near zero, that it didn't go back 4 billion, it just went ahead a few. So it allows and handles gracefully that wraparound from the maximum back to the minimum, as all the binary bits, all 32 binary bits fill up with ones and then click back over to zero again. So the designers thought, well, 4 billion.

Now, the reason we have sequence numbers is several. One is, as I mentioned, we want to number the bytes that we're sending in each direction. So a TCP packet has a sequence number for its data that it's carrying. And it also has a sequence number which acknowledges the most recently in order, received in order packet that it has received. So what normally is happening is, after the SYN goes, and then the SYN/ACK comes back, a final ACK is sent, an acknowledgement, acknowledging the SYN of the SYN/ACK packet. From then on, all the packets going back and forth have their ACK bits set, each end acknowledging what they've received so far from the other end. So the system works nicely. 32 bits to number the data. So that's plenty. It means that, if packets come in out of sequence, it's obvious how they should be arranged.

The other purpose for this, though, is because packets wander around the Internet, we know that they're clipped off by their TTL field, the Time To Live, which decrements, is decremented by every router which forwards the packet. And any router which decrements it to zero will send back an expired notice to the sender of the packet, saying, well, I tried to send this, I wanted to, but TTL went to zero. And if we didn't honor that, the Internet absolutely would crash because packets would never die. They'd just roam around forever and fill up the Internet. So the fact that they die is crucial.

But it still means that, if you have a very high bandwidth connection and a long delay between endpoints, it's difficult to see how you could have 4 billion bytes in flight at any time. And the designers were very comfortable with the idea that 32 bits would never wrap around within the lifetime of a single packet on the 'Net. And that's the key is that, because connections, TCP connections come up and down and up and down, they're often very short-lived connections. They establish, send something, and then drop again. And then the connection may be reinitiated between the same two endpoints.

Connections are identified by the source IP and port and destination IP and port. And so if you brought the connection up, sent some traffic, took it apart, and then brought it back up again and made another connection, what the designers of TCP wanted to guarantee against was the possibility that lost packets wandering around from the previous session, the previous connection, might be confused with packets from this session. So the way they solve that problem is by advancing the sequence numbers

forward, actually for the whole system.

They normally - they do it system-wide so that any connection that is created uses forward-moving sequence numbers so that all of the numbered traffic that is leaving it will never have the same numbers as older traffic. And that's a little confusing because you would think that we're always counting from zero. The idea is that that SYN packet, the sequence packet says we're going to count from this number. So this number equals zero, and a thousand plus this number equals a thousand. So the idea being that these sequence numbers move forward, and in what was once a uniform fashion.

Well, the hackers figured out that they could take advantage of older operating systems' uniform sequencing. That is, the idea was they figured out that, by talking to a server, for example a router, they could determine the router's current sequence number. And, because they would initiate a connection to it, it would send back its SYN/ACK. That would tell them the current state of the sequence number. Then what these clever hackers did was they were able to spoof TCP traffic because they were able to guess what other connections that either later existed or preexisted, they could guess the probable sequence numbers of those connections.

So this was a huge problem for the Internet, and it resulted in the randomization of TCP sequence, these Initial Sequence Numbers, the ISNs, for communications. So for a while there was a problem because the hackers figured out how to get the sequence numbers of probable connections. The response was to randomize those. And in fact what Linux does is, being 32 bits, that's 4 bytes. Linux takes the highest, the most significant byte and increments it every five minutes. Then, for the lower 24 bits, they just use a random number. So it's jumping all around within one-256th of the total sequence number space, and that's moving forward, jumping forward every five minutes. So that's still believed to be about the best tradeoff you can come up with between random and still meeting some of the goals of moving sequence numbers generally forward over time so that you don't have various types of replay attacks.

So what these guys in Michigan figured out is a new way of determining a connection's packet sequence numbering. And that's bad because, when you think about it, there isn't any protection in TCP. I mean, there really isn't. It's only obscurity which protects TCP. It's the fact that, unless you can see the traffic going back and forth, you don't know for sure what the originating port was. You may know what the destination port, if it's a server, for example, HTTP, it's going to be port 80. If it's going to be SSL, it's port 443. If it's POP, it's port 110 and so forth, SMTP port 25. We know those so-called well-known ports at the server end.

We also know the server's IP. And we may know the originator's IP. So now we have just the question of what port it generated its traffic from. And unfortunately, most OSes still do that in a uniform, from the bottom up, fashion. They just issue traffic sequentially. So if you can get that client to talk to you, you know what its current outbound port enumeration is, and so that gives you lots of information from which you can spoof TCP traffic. And that's what we're talking about here. We're talking about a new way to insert traffic into TCP sessions that gets around this problem that the initial sequence numbers are now being randomized by all contemporary operating systems.

So what these guys do is they have a number of approaches that they take. I'll describe one in more detail because it's representative, and it'll give you a sense for this. Their main exploit is not completely standing off to the side and attacking. It is possible to do that, and they outline how that can be done. I explained how you can know what their IP is, what their source port is. You know what the destination port and the destination IP is. And so there are completely, like, third-party attacks which they have identified and

verified.

But the one that works best is one which is also really interesting because it demonstrates an interesting failure in sandboxing. They implemented this with Android-based phones from, shoot, it was HTC, Samsung, and Motorola, using Android v2.2 and 2.3.4 from those manufacturers. And they installed an unprivileged piece of software, we'll call it malware because it's not doing what you want it to do. It may be offering some service to you, but it's also doing more than that. But this malware is unprivileged, meaning your phone does not need to be jailbroken. It's not even breaking out of the sandbox because they discovered some very clever side-channel leakage about the TCP stack on any contemporary operating system.

And that is, if, for example in Windows, you open a console window and type "netstat," you get an enumeration of all the system-wide TCP connections. And as of, I think, XP and on, you can add a command line switch that will tell you which processes are involved with those connections. So you can see that IE has the following connections open here, and TweetDeck has them open there, and so forth. Well, and in fact there's one other thing. There's netstat -s, which shows you an array of counters which all Internet operating systems, starting with UNIX and since, have maintained. These are things like the total number of packets sent, the total number of packets received, an array of error information which is also available.

Now, Linux has the same thing, which it implements differently. There's something called the proc file system, procfs. And it's sort of a clever way - it's not actually a file system of data that exists in any storage media. It's just a nice, clean way for the operating system to publish real-time information about things going on with it. And, for example, there's a path, /proc/net/snmp, which stands for simple network management protocol, and then InSegs, returns a count of the total number of TCP packets received. The same thing, proc/net/netstat: InErrs is a count of the erroneous packets received, like packets that have been received with a broken checksum, where the checksum doesn't match.

And there's one called, also under netstat, PAWSEstab, that's a count of old timestamp packets. PAWS is an acronym for Protect Against Wrapped Sequences. And that's exactly what I was talking about, the concern of sequences wrapping. The designers decided that they needed more than 32 bits, and they realized that the timestamp was a feature just never really took off and got used, so they could use that as an upward-compatible means of providing some disambiguation. But it generates, it increments this count in the operating system if a packet is received with a bad timestamp.

The point of all this is that these guys in Michigan realized that, if they had some software running in an Android phone, it could detect, by using this common, globally available information - this is an unprivileged read. Everybody, all apps in Linux have access to the proc file system. It's readable. Everybody can have it. So what this provides is a so-called "side channel attack." And we've talked about those in various contexts before, but never in this context, where something leaks information that can be used.

For example, classic crypto has side-channel attacks if it takes differing lengths of time to respond to, like, an encryption operation or a decryption operation, depending upon the data or the key. Because if you then use precise timing measurements, you somehow induce a system to try to decrypt what you're giving it with an unknown key, and you look at the time it takes, if its time varies with the key, then it's leaking information in the time domain, which that's a perfect side-channel example. It's not saying, I mean, it's not giving you your data back properly. But it is leaking information that it didn't intend to.

One of the reasons that AES was adopted was it is inherently power and time stable. It does not allow those kinds of side-channel attacks, which is one - in fact, I don't think any of the submissions do, or allow that, because it's now well understood that you just can't have crypto do that. What has not until now been understood is that doing something as benign as allowing an app to look at counters from the system-wide TCP stack is all it takes to launch exploits against other applications on the same OS. And that's what these guys have showed.

So what they did is they have this piece of malware installed in an Android phone. Three different phones, two different versions of Android, from HTC, Samsung, and Motorola. So it's widespread. They used an undisclosed nationwide cellular carrier. Their report doesn't want to point fingers at anybody. And it's not just one carrier that is doing this. They all do. I do know that it was a GPRS-based technology because they referred to GPRS at one point, some data files at one point in their report.

So they have an attacking server located somewhere else that this malware is communicating with. And understand, in a real-world scenario users would innocently think they had downloaded an Android version of Firesheep or a how-many-calories-have-I-eaten-today app or something that looks like it's absolutely benign. And it never breaks out of the technical sandbox. But it establishes a connection to a malicious server, and then it waits. It waits for some activity on the phone that it wants to intercept. It might be instant messaging. It might be Twitter. It might be web browsing, whatever it's designed to do. It's able to look at the processes running; and it's also able, from netstat, to look at the processes that are communicating. And it knows, because netstat shows it, both of the endpoints. It knows the local IP and port and the remote IP and port. What it doesn't know is the sequence numbering for that connection.

So as soon as it sees a connection coming up, it communicates with a server located remotely that it's time to get busy and try to get into this connection. So the initial sequence packet, opening a connection, goes from the phone toward the true destination. We'll say Google, just to pick an example. And the attacking server now sends a flood of probes back into the network, aimed at the client. Between this attacking server and the client is this firewall, which is intending to do the right thing. It's going to block packets that don't have matching sequence numbers. That is, whose sequence numbers are unexpected. There's a window which moves along, that moves forward, of sequence numbers that will be accepted. And anything outside the window are rejected.

So the idea is this thing, the attacking server, floods the connection back toward the client with guesses. They've even figured out how to binary search the entire 4GB - 4 gigabit, sorry, the 32-bit sequence number. They've figured out how they can do a binary search in order to narrow it down. So the client is - what the client is doing, it cannot see that this - the malware running on the phone cannot see the traffic coming back. But the malicious server is deliberately generating packets, using the timestamping option, which it knows are going to fail. So this one counter in the TCP stack, this PAWSEstab, which is the count of old timestamp packets received, it's normally not used. That counter normally sits at zero. So the point is it's not noisy. They're not have to disambiguate other reasons for it to be incremented than their own.

So what they've essentially done is they've managed to come up with a communications path through the side channel, off the TCP stack, that allows the malicious client in the phone to notify the server when its packets get through the firewall. That tells it, then, that allows it to zero in on the sequence numbering of the connection and then verify that it knows where the connection sequence numbers are. Then it's able to respond as the original server was. It sends a reset packet to the original destination server, which

resets the connection. The reset packet just causes, for example, in this case, it would be Google, causes Google to ignore the connection. And reset packets are honored by all servers. And Google doesn't know why the user changed his mind about connecting to it, doesn't care, just simply drops the connection completely. So Google no longer has any interest in the connection.

In order for the reset packet to be accepted, it has to fit within this window. It has to be a valid sequence numbered reset packet. But the attacking server has been able to determine what that is through this probe. So Google is dropped off the connection. Now the attacker is able to send data in lieu of what Google would send as the response to whatever query the user initiated. Now, this is tricky because remember that the traffic is not going to the bad guy. The traffic is going to Google's IP. And it's always going to go there. The traffic will continue going there. But Google's own firewalls, or its operating systems, will simply drop the traffic. That connection's been reset, so the traffic coming in is not part of a known connection, and it gets ignored.

So the acknowledgments from the original client application running on the mobile phone are ignored. But when you think about it, there's nothing in TCP that actually requires that both ends receive the data that the other one is sending. That is, the malicious guy has a valid sequence number for the data going to the client on the mobile phone. He can assume the data gets there. He can assume packet after packet after packet are being acknowledged. The acknowledgments he never sees. They go to Google, and they're ignored because the connection has been reset. They're just dropped.

So we now have a situation where an app on a mobile phone initiated a TCP connection to some service - Facebook, Twitter, Google, whatever - or maybe has a longstanding connection open if the malware wants to go in and inject things into existing connections. But in this scenario they open a connection. As far as the app knows in the phone, everything's fine. It sends its SYN off to open the connection, and the next thing it gets is a valid SYN/ACK with a sequence number, which it accepts because it's coming from the malicious server. And Google accepts the reset packet because the bad guy figured out what the sequence number was in that direction because that was allowed to get through the blocking firewall in the middle.

Now the app running in the mobile phone receives data. But it could be an HTTP redirect which bounces them to a clone website without them seeing it, and these researchers achieved that. They also achieved installing JavaScript, injecting JavaScript into the connection and causing the JavaScript to run in the context of your relationship with Google, which means, if you were permanently logged into Google, then they're able to send all of the information that your browser would normally never share with anyone but Google, off in any direction that they want to.

So that's the gist of this. It is, I mean, it sounds - it is complex. To me, as I'm reading this, I'm thinking, okay, well, this doesn't sound like the kind of thing that everyone is going to get infected by. But it's perfect for targeted attacks; or, with time, this kind of attack could mature to the point where apps are offered on Android stores, people download them, and the malware starts up in the background and gets up to its mischief.

Now, I guess I feel of two minds about this. I mean, all the details are in this 14-page report that allows somebody as good as these guys are, and there are a lot of smart hackers in the world, to duplicate this. That is, nothing is left unknown. And the reason I'm of two minds about it is that it's not clear how we solve this. I mean, I guess providing the side-channel information from the TCP stack only to root-based apps might make sense. But who knows what you would break? Who knows what apps are actually using that information for some purpose? So, I mean, that fundamentally changes the

way all of our UNIX and UNIX-derived and also Windows systems function. So we have a new problem which has surfaced that, in the context of mobile phones, breaks the sandbox and allows malicious packets to get injected.

Now, the one thing that stops this cold is HTTPS because there's no way, if an HTTPS connection is being established, there's no way for the bad guy to have a valid certificate that the browser is going to require and verify. So throughout their paper these guys mention that they did find in a secure connection with somebody, I think it was either Facebook or Twitter, there were two connections that still were not running over SSL. But this, one thing this says to us is that, certainly in a mobile network scenario, where we're assuming that the encryption of the mobile provider is sufficient, is HTTP and unencrypted connections really are not sufficient. We really need to use end-to-end authentication. And that, and pretty much that alone, gives TCP the additional layer of authentication and privacy that it needs. TCP by itself just doesn't do it. It wasn't designed to do it. It was designed to work. And it really works well, but it's not very attack resistant, unfortunately. Isn't that cool, Leo?

Leo: One more reason why HTTPS Everywhere is a good idea.

Steve: Yes, exactly, yeah. I mean, this is some seriously clever hacking, and an interesting way of using just a little bit of information leaking across the OS boundary, which is how many errors of a certain type have we seen, and then deliberately generating those kinds of errors on packets that may or may not get through a middleware firewall, and using that to probe the firewall which in turn releases information about the state of the TCP connection that would otherwise never be available, and then you exploit that in order to hijack the TCP connection. Wow.

Leo: Wow. And is this in the wild? Or is this just now merely an academic exercise?

Steve: Academic exercise. I wouldn't be...

Leo: Okay. So we don't have to fear it.

Steve: There's nothing to fear except...

Leo: Except that it's been documented now, and so...

Steve: Yes. There's also nothing to do, really. I mean, I would love to see Android lock this stuff down. And I would be surprised if they don't. I mean, everybody - I got so many tweets about this, people saying, "Okay, I don't know what these guys said, Steve. Could you explain it to us?" And so...

Leo: What does it mean? Well, now we know what it means. I don't know if you...

Steve: This sounds bad; this looks bad.

Leo: I don't know if we feel any better, but now we know what it means, anyway.

Steve: Yes.

Leo: So this show, as with all of the previous 354 episodes, lives in several places. You can, of course, watch us live every Wednesday, 11:00 a.m. Pacific, 2:00 p.m. Eastern, 1800 UTC, right here on TWiT.tv. But if you missed a show, or I think more likely you wanted to listen again or even read a transcription, Steve makes text transcriptions available, as well as 16Kb audio versions at his site, GRC.com. We also have audio and video and higher quality versions at TWiT.tv. And you can get it on a podcaster like iTunes or Downcast or any of those, as well. In fact, that's the best way, subscribe to the version you like. That way you'll never miss an episode. And you'll own all 355 at absolutely no charge.

Steve also makes a lot of other great things available at GRC.com, including his bread-and-butter, SpinRite, world's best hard drive maintenance and recovery utility. He also has a lot of freebies there, including the world-famous ShieldsUP! and lots of free security stuff, too. GRC.com. Now, when you're there, if you have a question about this or anything in the realm of security, privacy, Vitamin D or fish oil, you can - or ketogenic diets, you can leave those questions for us to talk about next week because every other episode we do a feedback episode, GRC.com/feedback. Did I miss anything there?

Steve: I should mention that I've had a bunch of really neat feedback from our two Over the Sugar Hill episodes, over at GRC.com/health.

Leo: Oh, wow, yeah.

Steve: And there is a feedback page there for health-related stuff. Anybody who wants to share their experiences, I've got a user experiences page and an FAQ page that I will be updating shortly. So, love to hear what people think.

Leo: My dad just sent me an email saying, you know, "You ought to be taking Vitamin D." I said, "Been there, done that, know all about it." And I sent him a link to your health page. I said we did a show about it. And, yeah, in fact I've been talking to my doctor about the ketogenic diet. He's really interested in that "Art and Science of Low Carbohydrate Living" that we're using as our bible.

Steve: Oh, I forgot to say, I got email from Geoff Bond.

Leo: Whoa.

Steve: The author of "Deadly Harvest."

Leo: Whoa.

Steve: Apparently we spiked his sales, and he's now the No. 1 rated preventative health book on Amazon.

Leo: Awesome. The power of the Gibson.

Steve: And so he sent me a note saying, hey, thanks, Steve. He got a kick out of the fact that I wasn't that enamored with the name of the book. But I understood that it wouldn't sell as many copies if it was titled "Nutritional Anthropology That I Have Known." So anyway, it was neat to hear from Geoff, yeah.

Leo: It's interesting, too, because we get people in the studio all the time now who are on the ketogenic diet or have been doing it. A guy was here last week, lost 80 pounds, looked great. Another guy who had celiac disease, so he was compelled to because of his intolerance for wheat and glutens. And he also said, you know - he looked great. He said, "I've lost a lot of weight." So every day almost, somebody comes in here and says, "That Steve." Well, it started with Paul Thurrott, and now it's spread.

Steve: It did, yup. Well, and in fact many people they started with Paul, and in some cases the carbs crept back in because it's just so difficult, I mean, there's so much...

Leo: But it's everywhere. We're inundated with it.

Steve: Yes. We're a carb-based culture. But then they said, when they heard my two podcasts and a lot of the biochemistry of it, they understood. In fact, one person said that ketosis was the only way she'd ever been able to lose weight, but it always starved her to death. Now she knew how she could do it...

Leo: Now she knows why, yeah.

Steve: ...and not be hungry all the time and not have to starve. So, yeah, cool stuff.

Leo: Great stuff. Thank you, Steve. GRC.com. It's all there. Is it GRC.com/health, if people want to read more about that?

Steve: Yup, and also in the menu, if you go under Research, there's Health as a subtopic of the research tab.

Leo: There's a ton of stuff there. Thank you, Steve. We will do this next week, right

here.

Steve: Talk to you then, Leo.

Leo: See you.