Routers & Micro Kernels

Description: This week, Leo and I catch up with the past week's news. Did the Shadow Brokers hack the NSA's Equation Group? Apple's Bug Bounty gets quickly outbid. A critical flaw is discovered in the RNG of GnuPG. The EFF weighs in on Windows 10. The Chrome browser is frightening people unnecessarily. A Johns Hopkins team of cryptographers, including Matthew Green, disclose a weakness in Apple's iMessage technology. We discuss surprisingly and sadly unused router hardware capabilities and then answer the question: "What's a microkernel?"

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-574.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-574-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. He'll talk about that leak of the NSA hack tools from the Equation Group. What does it mean? What does it mean? What could they be? He'll also give us a little insight into the microkernel, how it works, what it is. And a look at a very interesting router operating system. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 574, recorded Tuesday, August 23rd, 2016: Routers & Micro Kernels.

It's time for Security Now!, the show where we cover the latest security, keep you safe and sound online with this guy right here. I feel like I'm sitting next to you. This is nice. Steve Gibson.

Steve Gibson: So normally that screen used to be much further behind you.

Leo: Yeah.

Steve: So you sort of couldn't catch me out of the corner of your eye.

Leo: No, I had to kind of turn completely. And I did a TV turn, which was cheat and look over there. It would look like I was looking at you, but I'm really looking over there.

Steve: Ah.

Leo: We are temporarily discombobulated because we're going to be moving back into my office next time. But for now we're in the main studio because, as you know, I bet you can imagine, moving a whole studio...

Steve: I don't know how you did it, basically on the fly.

Leo: On the fly. And so I made completely impossible constraints on these guys, and I feel so bad for it, and I'm sorry, John. Because first of all they said, "Well, it's going to take us five days." I said, "I can't miss five days of shows." "Three days?" I said, "I'm not going to miss any shows. I'm not going to miss any shows." And they said, oh, okay, okay, okay. And then to add to the horror I said, "And we're not going to buy" - and they said, "Well, we can buy duplicate gear; right? Or rent it or something?" I said no.

Steve: Long cables.

Leo: No. You get one of everything. And then, not my fault, but the tenant improvements by the owner, well, first of all the building got sold in between, during the tenant improvements. So we leased it from one guy, and now it's a different guy owning it. So the tenant improvements dragged on. We were supposed to get this six weeks ago. We got it one week ago, basically. So there was little we could do ahead of time. And then most of that would have been studio building. Right? And like the bricks that are supposed to be there and all that stuff. So we could only do a barebones studio. But I said, no, don't worry. A, everybody's going to understand that for the first two weeks in a new place we're going to be, you know, it's going to be like in a new house. You're still finishing up the stuff, painting. And, B...

Steve: Stuff in boxes still.

Leo: Oh, lots. Lots. And then, but secondarily, most people just listen. And as long as - so what I said is, if we can do a show, if we do audio shows that look okay, I'll be happy. And they don't have to be - we can do everything at the round table, which we are, until the other studios are ready. And the last studio to go is mine because that was the last studio used in the Brick House. Actually, it's surprisingly complete. The desk is over, the backdrop is over, the lights are in. We just have to wire the sound. That'll be ready on Saturday for the radio show. And then after that I'll be doing this show and Windows Weekly from my office again.

Steve: So not only is this the first Security Now! in the new studio...

Leo: Yes?

Steve: This is the first Security Now! of Year 12.

Leo: So we'll always know when we moved. We moved in Year 12.

Steve: That's right.

Leo: Year 11, Year 12. Wow.

Steve: That's right.

Leo: Year 12.

Steve: Ten years from now...

Leo: This show's going to be a teenager next year. Junior high school. Wow.

Steve: So lots of stuff to talk about. Our main topics that we will get to is I discovered something very surprising in the hardware of all consumer routers, almost without exception, which is, I mean, it's distressingly unused capability that just isn't - it's physically there in the hardware, but isn't surfaced to a user interface. DD-WRT is beginning to make some inroads into it. So I want to talk about that a little bit. And I wanted also to do a little bit of just a little sidestep into the topic of microkernels because we're all living on top of operating systems, and there's been a lot of microkernel discussion in the news. So I want to, toward the end of the podcast, talk about those things.

But there was a lot of news of the week, of course. The question, we'll answer the question, or look at it at least, about whether the so-called Shadow Broker Group hacked the NSA's Equation Group. Note that Apple's bug bounty was quickly outbid. A critical flaw has been discovered in the random number generator of GnuPG. The EFF has weighed in on Windows 10. Chrome browser is frightening people unnecessarily, and I've had a bunch of reports about that.

Then a Johns Hopkins team of cryptographers led by Matthew Green presented a paper at the 25th Annual USENIX Conference a couple weeks ago, disclosing a series of weaknesses in Apple's iMessage technology, which, for example, just to give you a little tease, allows for retrospective decryption of encrypted iMessages. So there's that.

And then somebody posed a question actually this morning through Twitter that I really liked. And I thought, this is a perfect puzzler of the week for our listeners. So we will finish the podcast with the question this guy asked because - and it's just something for our listeners to think about for a week, and then we'll talk about it next week. So I think lots of fun stuff.

Leo: All right, Steve. I'm listening with all ears.

Steve: So our Picture of the Week, which is in the show notes, is just a fun T-shirt which

our listeners will appreciate and which would confuse pretty much any normal people who would think, what? How does that make any sense? And of course we all remember the "Hundred Bottles of Beer on the Wall"...

Leo: I just saw it. I love it.

Steve: Isn't that great? So we all remember the "Hundred Bottles of Beer on the Wall" song, where you take one down and pass it around and then there's 99. So this T-shirt reads: "99 little bugs in the code, 99 little bugs. Take one down, patch it around, 117 little bugs in the code."

Leo: Oh, lord.

Steve: And so it's like, yes, that's the lesson we learn. You have to be so careful when you think you're fixing something.

Leo: Oh, so easy.

Steve: It's just as likely that you're going to add some more problems.

Leo: 'Tis indeed.

Steve: So, okay. Probably the top story of the week was this whole NSA hacker group Shadow Brokers deal. I've read in as much as I can from what's available publicly. And attribution is famously difficult. You'll remember that I was reluctant for the longest time on the topic of Stuxnet to ascribe this to the U.S. and Israel, who we now - it's sort of, again, no absolute proof, but the consensus has sort of been, yeah, I mean, we're as sure as we could be that that's, you know, that it was state sponsored and probably the U.S. and Israeli intelligence groups, the cyber groups.

So what happened here in this case is that a group calling themselves the Shadow Brokers posted a bunch of data, but only a taste of what they have, 256MB of compressed stuff, predominantly batch scripts and what was regarded as unimpressively coded Python. So the people looking at it were unimpressed by it. And in fact I saw one massive compound IF statement, checking the version of the Cisco ASA software that was running. And I have to say it's not the way I would have written the code. So who's to know? But they posted this with the claim that they had hacked into the NSA's Equation Group.

Now, one of the things that immediately sort of caught my attention was that, if you actually read the posting, and I'm going to let everyone see what you think because I'm going to read the introduction paragraph exactly as it's written. Tell me if you think it's actually somebody who can't speak English, or somebody who does, who's doing a bad pretend, a bad emulation of a non-English speaker.

So they said: "How much you pay for enemies cyber weapons? Not malware you find in networks. Both sides, RAT and LP, full state sponsor tool set? We find cyber weapons

made by creators of Stuxnet, Duqu, Flame. Kaspersky calls Equation Group. We follow Equation Group traffic. We find Equation Group source range. We hack Equation Group. We find many, many Equation Group cyber weapons. You see pictures. We give you some Equation Group files, you see. This is good proof, no? You enjoy? You break many things. You find many intrusions. You write many words. But not all. We are auction the best files." Now...

Leo: That's Chinese, by the way. That is almost certainly Chinese syntax.

Steve: Okay. To me it reads as fake.

Leo: Or if you were faking Chinese syntax.

Steve: Oh, exactly. But, for example...

Leo: For instance, when you say "good," you often say "hau hau." Which is too good.

Steve: Well, but, see, "We find cyber weapons made by creators of." That, to me, like...

Leo: Is Russian.

Steve: ...some correct English slipped in there when they were trying to make it seem sort of jilted and stilted. So I don't know.

Leo: You know, I was a Chinese major. My Chinese isn't great. But I do kind of recognize a Chinese-style syntax. There's not a lot of, for instance, Chinese doesn't have tenses. It feels to me a little bit like it would be either Chinese or somebody pretending, you're right. And hackers obviously want to obfuscate who they are.

Steve: I've read a lot of English by non-English speakers, and it feels different than that does.

Leo: Right, right. Feels like a fake, yeah.

Steve: It really does. And, you know, you could understand that that may be what they're [crosstalk].

Leo: It's the equivalent of a ransom note.

Steve: Okay. So the files that have been made available are dated no more recently

than 2013. So the most recent are three years old. So that's, you know, it makes people think that this has been held for a while after it was grabbed. No one understands what that means or why. I think I heard somebody on TWiT suggest that it was like a field tool set, like an archive that may have been lost or left somewhere. That is to say, there are other feasible ways that these tools could have been obtained other than this rather romantic, "We found their IP range and hacked them." Okay, maybe.

But if in fact these are field tools, and they have sort of a feel to them of that, then it's very often the case that NSA people have to leave the Puzzle Palace and venture out in order to go to specific locations on the Internet in order to get the position on the network that they need. And if they're physically roaming around, you know, thumb drives get lost, or laptops get stolen from airports. You know, that kind of thing. So there are other ways this set of tools could have escaped.

Now, all of that notwithstanding, there is some - this was eyebrow-raising for the security industry. There was a whole bunch of previously unknown things that were contained here. So even though they were three years old, everyone on this podcast knows how lumberously - lumberously? Anyway, how slowly...

Leo: That's a good word you made up.

Steve: ...we move forward with security standards. So 2013 is - especially problems that have been persistent for a long time. For example, we'll be talking a little bit later about this flaw in the random number generator of GPG. It's been there, I think, since the late 1990s. So for decades. Because if it's sort of following the logic or the wisdom of that T-shirt, if you don't know it's broken, you're really better off not messing with it because leave it well enough alone.

So, similarly, for example, the news just today is that one of these cyberweapons which was specifically aimed at the Cisco firewall, the ASA line of firewalls, and that's actually where this crazy compound IF statement was located, it was individually stepping through individual IF-THEN clauses, looking at version numbers that it had retrieved from the SNMP protocol. And it's a flaw in the SNMP protocol, Simple Network Management Protocol, which we've talked about before. It's a UDP, typically UDP-based protocol that allows you to query network gear for its status. And so things like the number of bytes received on interfaces and transmitted and, I mean, you can - if you have write privileges, you can reconfigure SNMP devices over that protocol. So it's very powerful, you know, as it sounds, Network Management Protocol.

Anyway, the point is that the code stopped checking versions at 8.something, and I didn't bother to remember to write it down, it wasn't important, which was some years ago. And if it didn't match any of the known versions, it returned an error saying "unsupported." Well, some researchers said, huh. ASA is now at 9.something. What happens if we tell it that it's compatible with that? And sure enough, it works. So, and I'm wondering, and I didn't have a chance to look, when it was that that version of the firmware was published, and if that corresponds with the date of this tool.

Because the point is they may have - that tool may have been current when it was last edited, which was when that 8.something was the most current version of the Cisco software. Because this compound IF statement does nothing except turn an SNMP short version string into a full English statement, saying this is version zum zum zum of the Cisco something-or-other firewall. Then the next line checks a different SNMP version and then says the same thing with a slightly different bit of text.

So again, this is part of what made people feel like, wow, this is - whoever was writing this was being paid by the character rather than to create inefficient code. So anyway, what were found were implants, exploits, and other tools for controlling routers and firewalls, including those from Cisco Systems, Juniper, FortiGate, and a Chinese manufacturer named TopSec. And a whole bunch of other stuff. Again, batch scripts and Python-written tools.

The Shadow Brokers imagined that they were going to - I guess they imagined they were going to make a windfall from this because they were asking for as much as a million bitcoins. Okay, but a bitcoin this morning was $582 US. So that would be $582 million. And nobody expects that that's going to happen. In fact, I saw a posting...

Leo: So it's just theater, in other words, that they're doing this. It has nothing - yeah.

Steve: Yeah. And I saw posting somewhere that said the highest bid we're aware of so far, and they're only 999,999 bitcoins shy of their one million goal.

Leo: Well.

Steve: So I think someone said, "Yeah, I'll give you a bitcoin for that." So they're saying they're holding back, like, way more, the bulk of this. And for what it's worth, it would be valuable to certain entities. I think the price has to come way down for it to be sufficiently, I mean, for the benefit to match the cost. But still, if these guys didn't give us just the good parts, if there was a lot more of similar quality, despite the fact that it's kind of unimpressively written, somebody somewhere knows stuff that is not public.

Cisco immediately put out a notice and quickly came up with a short-term block for this patch. They have not yet updated their firmware. But this is an example. Wherever this flaw was in the SNMP protocol, sending that to the most recent version of firmware after you removed that check to see whether it was a known version, it crashed the firewall. And as we know, the crash is where you begin the development of an exploit.

So the problem is still there in some form in the most recent firmware. So Cisco's got a job to do because SNMP, it's not a super secure protocol. You're able to put a username and password on it. But it's much better to filter the port so that - it's port 161, I think, as I recall. You're much better off not letting anybody see your SNMP ports because that's just generally a good thing. And many ports need to be public: port 80, port 443 for the web have to be. But SNMP, you just don't want everybody to see that. You're asking for trouble.

So with any luck, most organizations that are properly configured won't have their SNMP protocol exposed. On the other hand, maybe they do internally. And so this would create the ability for a hacker who was able to get in through somewhere else to then access the firewall and get up to some mischief in order, for example, to open the front door after coming in through the backdoor. So, wow. And again, if there really is an equal or much greater amount of similar quality, that's kind of frightening that somewhere there is a group that knows, at this level of exploitation into who knows what devices.

We talked a couple weeks ago about how at the recent, it was either Black Hat or DEF

CON, Apple gave a presentation where, among other things, they unveiled their bug bounty program. We talked about how they're going to be offering security researchers up to $200,000 if they privately disclose serious critical holes in Apple's software, rather than taking such vulnerabilities and exploits elsewhere. However, there is an existing commercial exploit broker named Exodus Intelligence. And they immediately upped the ante, raising Apple's bid to as much as half a million dollars for valid Apple software bugs. And so, first of all, as I read this, I'm thinking, half a million dollars. How can they possibly afford that?

Well, get this. Their business model is that corporations that want early preview or access for whatever reason can subscribe for $200,000 a year to receive comprehensive reports, proof of concepts, demos and packet captures of these exploits which this Exodus Intelligence Group are basically purchasing from hackers who find them, pulling them all together, and then they've set up this subscription model. And it's, I mean, it's hard to imagine that there won't be some U.S. front company that says, yeah, we need exploits for only the best purposes for our intelligence services. So $200,000, eh, chicken feed. And I guess that's the model that these guys are using.

So, you know, as I said when we talked about this before, I like the idea of Apple offering this official bounty program because it allowed researchers to dig into Apple's code, which is about as closed as it could be, and actually when we talk about what Matt Green and his group at Johns Hopkins found, the real lesson there is about this notion of having closed protocols and the danger of it. But I like the idea that a true white hat hacker could support themselves, if they were sufficiently skilled, by finding problems and having Apple pay them a fair price for their find. The problem is half a million dollars is a lot more than 200,000. And you'd have to have your ethics screwed on real tight in order to say, you know, I'm going to stay with Apple.

So it certainly is competition for exploits. And not just Apple, by the way. Exodus is looking for - I wrote it down. Oh, so iOS, Google Chrome, Microsoft Edge, and Adobe Flash are their official sort of platforms. So of course Chrome, now the majority browser on the web; Edge, the browser that Windows 10 people are using by default; Flash, that refuses to go away and is probably a ripe territory. I don't imagine you get so much money for those Flash exploits because they're not that hard to...

Leo: A dime a dozen, literally.

Steve: Unfortunately. So GPG. One of the other things we've often talked about is the critical need for - I want to say all, I think it might be all - crypto software to have a source of high-quality random numbers. There are probably some modes where you don't need them. But for most things you do. So, for example, back when I was talking about SQRL, we did a podcast on the Entropy Harvester, which is, like, the first thing I wrote for SQRL because I wanted to get that right.

And what mine is doing, the code that I wrote for SQRL, is it's sucking, continuously sucking in all kinds of noise that the system produces - exact packet counts, exact packet arrivals, hard disk transfer rates, bytes transferred, I mean, like, when anything happens, keystroke timing, mouse moves, everything that's going on in the background, it's just sucking them in, pouring them into a big hash. And the idea is that - oh, and all kinds of, at the nano level, the individual things going on in the processor, the branch mispredictions, the cache hits and misses. You know, our current processors maintain an incredible repertoire of their own internal analysis and management platform. And it's available to the programmer. And it's completely unknowable outside. So, I mean, and

highly unpredictable.

So the point is, because there's no lack of true entropy in everything going on in the system - oh, also how much memory every process is using and so forth. I mean, there's just all this, I'm just scraping all of this, dumping it into a hash, and that churns until SQRL needs a random number. And many crypto systems are very, very hungry for randomness. SQRL actually isn't. It doesn't need much. So that also helps to increase its integrity because the more you need, the more difficult it can be to provide it. What typically happens is that - and that's the case here in GPG, is that a hybrid is created of something that harvests real entropy, which is then used to seed an algorithmic pseudorandom number generator. And the problem is that it's easy to make these work wrong. It is very difficult to make them work right.

And so the problem with a pseudorandom number generator is, once you're able to determine its state, if you're ever able to get a snapshot of, like, of its pool of entropy, and if it algorithmically generates output by churning that pool, then you're able to predict the future of outputs. And that's exactly what this does. So some security researcher discovered what they call a "critical vulnerability." And we'll back off of that a little bit here when I explain what it is because it isn't the end of the world, and it's been fixed, in the random number generator inside GnuPG, and also Libgcrypt. And those apps have been around since 1998. And so essentially it's just been patched. But any version of GPG earlier than August 17th, that is, last week, is vulnerable. This has always been there. They're all vulnerable. But again, they're calling it critical because it is a flaw in the random number generator.

Turns out it's very difficult to exploit. So that's why it's not the end of the world. The vulnerability allows an attacker who can arrange to obtain 4,640 bits of entropy from the random number generator, to then trivially predict the next 160 bits of output. So this is, as I said, if you can - in this case they're not getting a snapshot of the internal state. But it turns out that, due to a flaw in the mixing functions of the pseudorandom number generator part of this, if you were to get a little over 4K bits out, you can then compute the next 160. And that's not something that anybody wants. That's just, by definition, it's cryptographically broken, if you're able to do that. So GPG 2.1.15 is fully patched as of the middle of last week. And all platforms were affected because this was in the common core code of that product.

Now, the researchers indicated that, although the vulnerability is critical, users should not immediately start revoking private keys created with vulnerable versions. And I wrote down exactly what the researchers said. They said: "A first analysis on the impact of this bug in GnuPG shows that existing RSA keys are not weakened. For DSA and ElGamal keys it is also unlikely that the private key can be predicted from other public information." They wrote: "This needs more research." But they suggested do not be overly hasty in revoking keys. Of course, you know, revoking your GPG key would cause some upheaval for everyone who would then need to update themselves in order to communicate with you. And so they found a problem.

The good news - and it is a true design flaw that's always been there. But it's more of an internal issue. It's fixed. And even they say, eh, they call it "critical" because, again, these things tend to, you know, clever people look at this longer and find some unseen way of extracting the earlier entropy and then can compute backwards. So it's a little bit of a flaw in randomness, which, as we said, crypto absolutely needs a good supply of.

I tweeted this last week, and there was one piece of new information here that I wasn't aware of. The EFF has weighed in on Windows 10. And their headline was "With Windows 10, Microsoft Blatantly Disregards User Choice and Privacy." Now, I took a softer tone in

my tweet. I just said "Win10 holdouts, not to mention corporations, would likely appreciate the EFF's take on choice and privacy tradeoffs." Meaning that, if you're on the fence, or if it's for some reason painful for you not to have moved to Windows 10, or if you just want additional justification, the EFF basically, I mean, they rant for a while. They're not being gentle.

The first part is user choice, which they were really unhappy about. And there's no news there. That's everything we were talking about for the last year, things like what seemed like deliberately confusing dialogues leading up to click the red X to close it, and that essentially doesn't deschedule the upgrade and everything. So of course the EFF thinks all of that is a bad idea.

But then of course the second side is what Windows 10 itself does. And here was something I didn't realize, that is, something that was news to me. They wrote, under "Disregarding User Privacy," they said: "The trouble with Windows 10 doesn't end with forcing users to download the operating system. Windows 10 sends an unprecedented amount of usage data back to Microsoft, particularly if users opt in to 'personalize' the software using the OS assistant called Cortana. Here's a non-exhaustive list of data sent back: location data, text input, voice input, touch input, web pages you visit, and telemetry data regarding your general usage of the computer, including which programs you run and for how long." Then they go into talking about the ways it's possible to use options to reduce that level of reporting, that of course we've covered previously, extensively.

But here was what I didn't know: "Unless you're an enterprise user, no matter what settings you choose, you have to share at least some of this telemetry data with Microsoft, and there's no way to opt out of it. Microsoft has tried to explain this lack of choice by saying that Windows Update won't function properly on copies of the operating system with telemetry reporting turned to its lowest level. In other words, Microsoft is claiming," writes the EFF, "that giving ordinary users more privacy by letting them turn telemetry reporting down to its lowest level would risk their security, since they would no longer get security updates." And then they said, in parens, "(Notably, this is not something many articles about Windows 10 have touched on.)" And I never remembered encountering that before.

So anyway, they conclude, saying: "There's no doubt that Windows 10 has some great security improvements over previous versions of the operating system. But it's a shame that Microsoft made users choose between having privacy and security." And that's no surprise from the EFF.

I got a tweet that reminded me of several of these that I've received that I just sort of wanted to talk about sort of a checkpoint here: In August of 2016, after the January 1st, midnight, New Year's Eve sunset of SHA-1 signed certificates. And so the tweet that I got said: "Steve, I work for a hospital, and we have an online bill pay system in place using a third-party site. When visiting the payment site in Chrome and looking at the certificate information, Chrome is telling me that the security on the site is weak." And he says, "See attached image. What are your thoughts on this?" And in fact the image that he attached has Chrome saying that this may not be a private connection. Now, what it's doing is warning that that site that he was going to was signed with SHA-1 cert, claiming that the connection might not be secure. Which we know is utter nonsense.

Leo: It's just scaring people.

Steve: Yes, exactly. And Google's security engineers know it, as well. On the other hand we know, it's one of the underlying topics of the podcast is how reluctant and difficult it is to move the industry forward, and that SHA-1, it was time for it to die. But it has died. For example, you can't get an SHA-1 cert any longer. But the ones that were issued still exist. I tweeted back to him, and I said, well, for what it's worth, there isn't any known actual problem with SHA-1. Everyone's just being cautious and moving away from it before it becomes a problem because everything we've learned about the way security progresses says that someday that may not be strong enough. And I also said, but look at the expiration date. When that cert expires, that company will have no choice but to move to SHA-256 because, as of midnight of last year, of New Year's, no CA will synthesize and sign an SHA-1 certificate anymore.

So I'm of two minds. The problem is, Google is doing this - and this is not the only person I've heard this from. I get this constantly. It's like, what does this mean? And it's like, well, you know, Google knows that companies using SHA-1 certs will get some blowback from their users, their web server visitors, who are worried by the certificate that the company is using. And so Google knows that will incent the company to move off of SHA-1. To me, that seems like overkill. Unless a problem were known, just let them expire. No one can get them anymore, so within a couple years they'll all be gone. So anyway, I guess this is the tension you're going to have when it is so difficult to change things that aren't broken. SHA-1 actually isn't a problem. We just think it's a good idea to move away from it. And Google's pushing, as we know.

So this USENIX paper from our friend Matthew Green, who is the professor of cryptography at Johns Hopkins, along with four, I assume they're postdocs or grad students - and you can get a lot of work done if you have enough smart grad students that are interested in a project. The paper was titled "Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage." And I'm just going to quote from the abstract because this gives you a sense for what Matthew Green and his group found.

Their paper, which is, I think, 19 pages long, it opens: "Apple's iMessage is one of the most widely deployed endtoend encrypted messaging protocols. Despite its broad deployment, the encryption protocols used by iMessage have never been subjected to rigorous cryptanalysis." Why? Because they're secret, and we'll be talking about that once we catch up on this news. "In this paper, we conduct a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks." Now, of course, this could have been done easily, if Apple had published the protocol for verification by experts. But they didn't.

"Our analysis," the abstract continues, "shows that iMessage has significant vulnerabilities that can be exploited by a sophisticated attacker. In particular, we outline a novel chosen ciphertext attack on Huffman compressed data, which allows retrospective decryption of some iMessage payloads in less than 218 queries." Which is nothing. "The practical implication of these attacks is that any party who gains access to iMessage ciphertexts may potentially decrypt them remotely and after the fact. We additionally describe mitigations that will prevent these attacks on the protocol, without breaking backwards compatibility. Apple has deployed our mitigations in the latest iOS and OS X releases."

Okay. So these guys, this work was done last year. And in November of last year, 2015, they responsibly disclosed, privately to Apple, what they found. So Apple then revised their protocols in, as they said, in a backward-compatible fashion. And we got those in March of this year in iOS 9.3 and Mac OS X 10.11.4. So the problem is solved. So by the time this got any light, everyone should have been updated, and this wouldn't be a problem. But I want to just now go over what they described as their high-level protocol

analysis because it's interesting what they found.

Under the "Key server and registration," and our listeners will - this will sound very familiar to people because it's what I've been complaining about from the beginning: "iMessage key management uses a centralized directory server [they call] IDS which is operated by Apple. This server represents a single point of compromise for the iMessage system. Apple, and any attacker capable of compromising the server, can use this server to perform a man-in-the-middle attack and obtain complete decryption of iMessages. The current generation of iMessage clients do not provide any means for users to compare or verify the authenticity of keys received from the server."

Just to pause for a second, remember, this is why, for example, I'm so bullish about Threema, or even Signal. Both of those give you explicit key management. Now, on the one hand, most users don't want that. But if you actually care about the security of your communications, I mean, if, for you, security is more than just, oh, yeah, it's secure, then iMessage doesn't provide that because, as I've always said, they're managing the keys. And if a third party manages the keys, and the third party can be subject to any kind of coercion, or bad employees, then you don't actually have security. You have the feeling of security.

Then they continue: "Of more concern, Apple's 'new device registration' mechanism does not include a robust mechanism for notifying users when new devices are registered on their account. This mechanism is triggered by an Apple push message, which in turn triggers a query to the Apple-operated server. Our analysis shows that these protections are fragile." Then they have an Appendix A where they implement attacks against both the key server and the new device registration process successfully. That is, they demonstrate its exploitability.

Further, "Lack of forward secrecy," which is a property that we know is important because what that does is it means that the symmetric key being used to encrypt messages is constantly changing so that, if you capture the key in the future, and you had stored ciphertext in the past, you cannot use a key captured in the future to decrypt old messages, which is otherwise a problem. Until recently, SSL, the pre-TLS protocol itself was not - did not offer forward secrecy. iMessage doesn't either. "iMessage does not," they write, "provide any forward secrecy mechanism for transmitted messages. This is due to the fact that iMessage encryption keys are long-lived and are not replaced automatically through any form of automated process." Where, for example, as we know, Signal has the key ratchet mechanism that is constantly moving keys forward.

"This exposes users to the risk that a stolen device may be used to decrypt captured past traffic. Moreover, the use of long-term keys for encryption can increase the impact of other vulnerabilities in the system. For example, in Section 5, we demonstrate an active attack on iMessage encryption that exposes current iMessage users to decryption of past traffic. The risk of such attacks would be greatly mitigated if iMessage clients periodically generated fresh encryption keys. See Section 7 for proposed mitigations."

I'll skip over their discussion of the fact that there is no prevention for replay and reflection attacks, and finally just get down to Apple's use of nonstandard encryption. They write: "iMessage encryption does not conform to best cryptographic practices and generally seems ad hoc. The protocol" - which they diagram earlier in the paper - "insecurely composes a collection of secure primitives, including RSA, AES, and Elliptic Curve DSA. Most critically, iMessage does not use a proper authenticated symmetric encryption algorithm and instead relies on a digital signature to prevent tampering. Unfortunately, it is well known that, in the multiuser setting, this approach may not be sound. In the following sections, we show that an on-path attacker can replace the

signature on a given message with that of another party. This vulnerability gives rise to a practical chosen ciphertext attack that recovers the full contents of some messages."

And I had in my notes here some additional detail. But everyone gets the idea. Essentially, what Apple did was, unfortunately, they rolled their own. They invented something they did not need to invent. Now, I want to back off from that a little bit, saying I'm not sure when iMessage's protocol was put together. So some of these things may not have been around. Some of these primitives may not have been available. But we've covered on this podcast in the past the danger of using a signature rather than a MAC, a Message Authentication Code, the danger of using a signature to authenticate a message because, if there's some way for you to use a valid signature, even if it's not the original person's valid signature, but if the signature will still validate, then you can make any changes you want and then sign the message with, for example, your own signature. So long as it's part of the system, it'll be accepted at the other end. And attacks like that have existed, and iMessage is vulnerable to that attack.

So the takeaway is, once again, the clear and present danger of closed protocol security design. In my opinion, it's not necessary to see the source, that is, Apple's source code. But documentation of the protocol would have allowed anyone who understands crypto to glance at it, and this problem would have been fixed a long time ago. Because Apple is as closed as they are for, I guess, corporate commercial proprietary reasons, they didn't disclose the protocol. So what that does is it hugely raises the bar of difficulty beyond the expertise of just being an expert in crypto. You also then have to reverse-engineer an undocumented protocol from scratch.

Now, there were a couple previous attempts, partial reverse-engineerings of iMessage. So Matt Green and his group used those, but they were incomplete. So they had to do packet captures, watch this thing work, use what little was known publicly, and basically reverse-engineer all of how it works in order to then look at it. And then what they realized was what Apple had done had a lot of problems. So I can't think of a more perfect example of the danger of a protocol being closed, different than the source being closed because, as we know, being able to look at the source code in theory would let you find bugs.

But in practice, lots of open source code has bugs hiding in plain sight. You just can't see them when you look at the code. That's not the way code works today. But the protocol, that is, what the code was trying to implement, I would call that the policy, as opposed to the implementation. The policy should be able to withstand scrutiny. And that's, for example, that's exactly my position with SQRL, is that the SQRL protocol is absolutely open. I talked about it on the first day that I mentioned SQRL. And other people are implementing compatible clients and servers using their understanding of the protocol because it's, first of all, very straightforward and very simple, but nothing hidden.

And I think, even if we're going to have proprietary closed source solutions, there's just - this is a classic example of why the protocol that that closed solution implements should be public. And that is actually the rule with the rest of the Internet. The Internet is based on open protocol. That's what has allowed it to be as robust as it has been. Well, that and the fact that it was an inspired design from the beginning. It's got problems, things that it wasn't designed to do that we're trying to make it do, that it's having problems with. But that's not its fault, the fact that all these RFCs are written, the documents, how everything glues together.

Look at all the companies, like the router manufacturers, like Cisco and others, that are making a great living writing, creating hardware and software solutions that implement that open protocol. I really think that's the future that we're going to see, that this kind

of, oh, no, we're better than everybody else, we know how to do this right. I wanted to believe that. But this is an expos of even Apple's good crypto people didn't understand as well as true experts who do this, like, for their living.

Leo: It's a classic case, isn't it.

Steve: Yeah, it's perfect.

Leo: Yeah. Is it good enough? Remember we talked about Telegram, and you said it was good enough? This is good enough. It's just not strong.

Steve: Yeah. Oh, it's going to keep, you know, it's going to keep anybody in the neighborhood...

Leo: Everybody but a state actor, probably, out of your pants.

Steve: Right. And, again, even if it were public, if you use centralized key management, you can't trust it. You cannot trust it. I mean, we trust Apple. But Apple could be compelled, as we often said, to add another key to a conversation that would allow a non-Apple actor to have access to that messaging traffic. And in the paper - I didn't have this in my notes, but I read the whole thing - Apple maintains in their database 30 days' worth of everyone's message traffic. And my god, it's an amazing amount of message traffic. It's 200,000 messages per second, 200,000 iMessages per second Apple is currently transiting through their network. So they're maintaining 30 days of back traffic so that devices that are turned off, when you turn them on, they're able to resynchronize themselves. We've all seen that. When you bring a new device online, sync it into your Apple account for the first time, it's able to get all of the back messaging traffic that's available for 30 days.

So Matthew and his group's point was that, with the exploits they found, it would be due to the fact, for example, that keys are not being rotated, and that in fact their chosen ciphertext attack that leverages the fact that Apple did not authenticate messages properly, that would allow an attacker to subpoena the encrypted data from all of the most recent 30 days for a given person, and then do an offline decryption with no other information. So the fact that Apple says, oh, well, yes, we're storing it, but it's encrypted, and we don't have the keys, that's okay. Johns Hopkins does because it wasn't [crosstalk].

Leo: It's okay. Johns Hopkins has it.

Steve: So you shot me a note over the weekend, and I received a bunch of tweets about this. There was a very nice posting, a blog posting in CodersNotes.com, about the "Elegance of Deflate."

Leo: Wasn't that a great - wasn't that interesting? Yeah.

Steve: Really was. And I have a - I just love encryption. I mean, sorry, I do love encryption, but I love compression. I've always been fascinated by compression. It's just been one of the fun puzzles that engages me because it's a closed system. It's very much like the mobile security or the mobile app puzzles that I find and share with our listeners, thanks to them sharing them with me, which are themselves closed systems. Everything that you need is in front of you to work with, and what can you do? And compression is like that.

So I wanted to point our listeners back to our podcast that was titled "Lempel & Ziv." It was Podcast 205 on July 16th of 2009. So, and there's show notes. The audio is there. I'm sure we were doing video by then. So you can get it from TWiT, or you can get it from GRC. Again, Podcast 205, July 16th, 2009. I explained, in Explainer-in-Chief mode. We had a lot of fun visually explaining the way this very clever buffer-based Lempel & Ziv compression works. And I've talked about it through the years. And it was an invention. I think it was 1973 a patent was issued...

Leo: Long time ago, yeah.

Steve: ...to these guys. And I think they were at IBM at the time.

Leo: Or, no, Unisys.

Steve: Oh, you're exactly right, Unisys. And the idea was to compress data over a communications channel. And that's sort of a - it's weird because we don't think of like compressing a file as being over a communications channel. But the idea is that the recipient who is going to decompress it has no advance knowledge of the contents. So the sender is able to look into the future, sort of like upstream of what it's going to be sending, if that helps them; or also look downstream, that is, remember what has been sent, but then chooses to send stuff to the recipient.

And the idea is that the way the algorithm works is they each create state in the form of some buffers, which is kept synchronized, so that each end does the same thing to their state. And so the sender knows what the recipient has in their table, their state table, and that knowledge allows the sender to use shortcuts, essentially, to represent much longer runs of data which happened to appear in the recent past of the data that was sent because that will be represented in this table. And the way this applies to file compression is that we think of the act of compressing the file as sending it. That is, we're sending it to a small file. And the act of decompressing it is we're receiving it. We're receiving the small file into a big file.

And so, anyway, I go into it in great detail. I think our listeners would find it interesting. I know, you know, we have so many of these things that we talked about years ago that are still relevant today. So again, Podcast 205, Lempel & Ziv.

Leo: But you don't cover Deflate there. Deflate is kind of interesting because it's Lempel-Ziv plus; right?

Steve: Well, yeah. Deflate, I did actually talk about...

Leo: That's Phil Katz came up with that, with PKZIP.

Steve: I did talk about Huffman coding of the tokens.

Leo: Oh, okay.

Steve: And so Lempel & Ziv is the core algorithm. And then the idea is that, because not all things will happen with the same frequency, once you have this set of things that you want to send, you're able to represent the ones that occur most often with fewer bits, and the ones that occur less often with more bits. So you get Huffman compression, which is what that's called, variable bit-length compression, on top of the really cool buffer compression. So, yeah, neat stuff. And that's Deflate, you know, Gzip that's been around forever.

Leo: Yeah, PKZIP.

Steve: I just wanted to mention about SpinRite that I got some nice feedback from people who were really glad that I mentioned as I did last week something I had never mentioned before, and that was that it's not necessary to run SpinRite all at once; that, because of the fact that I made the percentage complete, accurate to four decimal places, 37.1234, you're able to run it for a few hours when you don't need your computer, like overnight, and then stop it. It shows you where it was. And then next time you have some time, you're going out, or you're going to sleep again, you start it up where you left off.

And so a lot of people appreciated knowing that. That's just - it's not something that's really apparent. In fact, I did see a little confusion, some people who didn't even know where it was in the user interface. There's one place where you, I mean, and this is one of the things that I look forward to changing in the future is updating the UI. Because at the time it was state-of-the-art. But that was 25 years ago, and the state has changed a little bit.

Leo: We had Alan Cooper on yesterday, on Triangulation, old, good old friend, and kind of king of the UI. And he had written, in the early days, he was working at DOS. And then Windows 1 had come out, or actually he said 0.98 or whatever. And he was writing a project manager, which ended up becoming Super Project, that CA bought. And he said it was graphical. I said, "Graphical." He said yeah. I said, "Were you using like the ASCII art, like..."

Steve: Like the line drawing.

Leo: "...the line drawing in ASCII?" He said, "Oh, yeah, that's how it was graphical." That's how Windows was graphical originally.

Steve: Yeah.

Leo: So that's how you did it, by the way, in SpinRite.

Steve: I think it's no coincidence that we just sold a couple copies.

Leo: Oh.

Steve: It must be that we have some live viewers who are waiting for me to talk about SpinRite to...

Leo: Yabba-dabba doo.

Steve: ...push the button. So thank you. I didn't have the loud one on next to me. But I did hear yabba-dabba doos in the background.

Leo: That's awesome.

Steve: So it's appreciated. It's what keeps me here.

Leo: Yeah. I ought to do something like that in the studio, where we have, like, little sounds go off every once in a while telling us [indiscernible].

Steve: It's funny, I also got feedback about that. Someone asked, by the way, if I was - and I never had a chance to respond, so I will here - whether I was using some program to generate those. I talked about how, when a process launches and a process stops, that I have various little, just little click-y bonk sounds and things. No. It turns out that's built into Windows. You are able to associate sounds with all kinds of Windows events, like process start, process stop, logging on, logging off, screen blanking, and all kinds of things. So it's all built in. If you just go into the sounds applet and poke around in there, you will see the ability to bind sounds to Windows events, like switching users and logging on and so forth. All kinds of things. And so I have a standard set of sounds that have moved with me through the years. And that's just sort of part of my operating environment.

Leo: All right, Steve. Let's get into it. Routers and microkernels.

Steve: Yeah. So I was astounded a couple weeks ago by what I found in the hardware of most of our dumb, dumb routers. And it was just a trail that I was set onto because a number of our listeners like a different type of router. We've of course been talking about the Ubiquiti EdgeRouter X and the power that it offers. A number of people said, what about the - and no matter how much I practiced saying this, I cannot say it - the MikroTik. Mik-rot-ik. M-I-K...

Leo: I think it's Mi-krot-ik.

Steve: MikroTik?

Leo: Yeah.

Steve: That's a lot easier. Oh, you're right, MikroTik. MikroTik. Okay. Anyway...

Leo: It could be Mikro-Tik. I don't know.

Steve: Could be. Yeah.

Leo: MikroTik. What does that mean? I don't know what that means. You've got a word I've never heard of.

Steve: It is a bizarre word.

Leo: I know "necrotic," which is dying.

Steve: Well, and I was trying to say, okay, "erotic" but with a "mik." So erotic, MikroTik.

Leo: MikroTik.

Steve: But apparently it...

Leo: Actually, the way it's spelled with the inner cap, I think it's MikroTik.

Steve: Okay.

Leo: See the inner cap?

Steve: Oh, look. Oh, Mikro...

Leo: I think it's MikroTik.

Steve: Somehow I missed that. I don't think I saw that page. Nice.

Leo: Yeah, that's on wiki, Wikipedia.

Steve: But actually, okay. So what I have in the show notes, I'm not going to go into this in detail. I just want to put it on people's radar. Basically the show notes are one, two, three, four, five, six, seven, eight, nine interesting links of surprising stuff. It turns out that the heart of, I want to say all, but at least all that I looked at, certainly many, of our rather dumb routers, the ones where we've got a WAN port and four LAN ports, and it's blue and plastic, it turns out in there is an extremely capable chip, the same chip as in the Ubiquiti EdgeRouter that does all of that extra stuff.

Because it turns out that Qualcomm, I would argue, really overdesigned a beautiful chip, and then no one took advantage of it. They used a minimal set of features. But in this Qualcomm chip is not only the ability to individually configure the subnets of individual LAN ports, but a complete hardware-based, packet-processing engine and rule-based filter system. So, I mean, it's a firewall multiport router on a chip. And it's in the dumbest of the routers. They just don't use it because it's not what they were selling. And so when I - first of all, I was trying to understand what features this MikroTik - thank goodness that's the way you pronounce it. Now I can say it. This MikroTik...

Leo: Well, wait a minute, because they're from Latvia. So it's probably MikroTik.

Steve: Oh, MikroTik.

Leo: Isn't that where Andy Kaufman's character Latka came from, Latvia? I don't know. Say it like Andy Kaufman would say it, MikroTik.

Steve: So I was trying to figure out what capabilities these routers had. And I ran across this chip. And the documentation on this MikroTik site is really bad. And I should say, for what it's worth, this is not my favorite router. I looked at it. It's funky. If you have one, fine. But if you don't, get a Ubiquiti EdgeRouter X.

Leo: I have never, never heard of them.

Steve: So, but our listeners do or have. The documentation is very confusing about, like, which router. They make, like, 50 of them, different models, and also antennas and things. But even among the routers it's very unclear what they can do and what they can't. So that forced me to dig down. And what I found when I dug was this Qualcomm chip. And it turns out everybody's using it because it's amazing. But they're not using it for its amazing stuff. And so, for example, DD-WRT, that we've talked about often, which is the alternative, open source, very nice firmware which can be used on many of these routers, it is creeping into this chip further.

There is a page, one of the links that I've got in my show notes, at the DD-WRT wiki. The page is "Switched Ports Separate LAN Ports Into Another Subnet," or "Separate LAN Ports Into Another Subnet." So that says that on this DD-WRT page, they explain it. The web UI, the normal browser-based UI, it doesn't go there. You can't do that there. But at the command level that you're able to get to, you can give commands that DD-WRT will

interpret to program these advanced capabilities that are in the cheapest router around, or older router.

I mean, really, the Ubiquiti EdgeRouter at 49 bucks is hard to beat. And it brings all of those features to the surface so that you can use the standard browser UI in order to create separate LAN ports or separate subnets on different LAN ports. But anyway, the point I wanted to make was that, to my amazement, it isn't a dumb switch or a hub that is doing nothing, even in the cheapest router. It is typically this Qualcomm part.

Now, the other thing is that Qualcomm's got all this locked down in NDA. And the only documentation I could find had some poor guy's name plastered across every single page in light gray, branding him as the person who let this loose on the Internet. On the other hand, it's the complete documentation for this chip, which is otherwise not public. You have to go under NDA in order to get documentation for it. Intel made me do that once when I wanted to write some code for their gigabit chip. I was like, come on, it's just a gigabit chip. I mean, it's supported under FreeBSD. I can go get their driver. But I'd rather have the official documentation. But companies are like that.

So anyway, I wanted just to let everybody know that this exists, that that capability exists. I don't know what you could do with it except I know we've got tinkerers. And if you've got older hardware that isn't a MikroTik router, and isn't a Ubiquiti, for example, but it will run DD-WRT, you may be able to unlock unsuspected capabilities there, which would be kind of fun. I think that'd be cool.

So I was amazed that there's, like, an incredible packet processing computer. And you can tag individual packets for processor-level handling or local handling. So, for example, that chip itself can be a hardware firewall, which in hardware follows a hierarchical ACL, you know, sequential match rule set. All that's built into the hardware. And if, then, more intelligence is needed, there's a rule that says "Forward me to the CPU," in which case - and, for example, the chip itself is called a 7-port switch because five of them are Ethernet ports, and two of them are processor ports. So it's not seven Ethernet ports. But it's technically a seven-ported chunk of silicon.

Anyway, I'm just - I was just amazed that that was in there. And, boy, you know, if I had any free time, that'd be fun to play with. But I know that our listeners may have some fun with it. So I wanted to let everybody know that's there, hiding.

So operating system and microkernels. This comes from a conversation, Leo, you were having a few weeks ago about - was it an alternative OS for Android phones? That's been one of the things that has been happening. And it might have been something that Google was doing. I don't remember now what the genesis was. But I wanted to remind people that, sort of a little bit of history here, the way, like, where we came from with computing.

Because once upon a time there were no operating systems. Those old-school computers with the raised air-conditioned floors flowing cold air up through large boxes of electronics, with the big reels of mag tape going back and forth, and the big huge printer chug-chug-chugging out, line printer, lines of text. Back in those days, when a program ran on the computer, that's all that was running. So, and they were called "jobs," typically. And at the very beginning you would run one program at a time. And so this job would run on the machine. And in general the operators, these things had their own operators who often wore white coats to look official.

Leo: No, I thought that was to keep things clean.

Steve: Well, to make sure that their salary was clearly justified. Because, well, and the other thing is these computer systems were ungodly expensive. I mean, like an insurance company, a huge insurance company, would have one in their main headquarters. And it would process all of their stuff. And it would run, typically, it would never stop. It was 24 hours a day because they wanted to get as much work out of this thing as they possibly could. The operators ended up after a while knowing how long which jobs took. So, like, the East Coast payroll job would take five hours to run. The West Coast payroll would take four hours to run because there were not as many people on the West Coast back then, and like that.

And so the different things they were doing would - they'd, like, run the deck of cards in that would get this thing programmed. And then they'd mount the reels of tape that the job required. And basically they were just sort of trained monkeys. They weren't programmers. They knew which button to press to start. And then they basically sort of sat around and watched, literally, the wheels spin for a few hours. And they could - also you developed some expertise after a while. When the lights dimmed a certain way, they could kind of tell, oh...

Leo: You've got to listen to yesterday's Triangulation because this is exactly what Alan Cooper did with his System 360 or 370. It's exactly what he was talking about, the disk packs and taking them out. And, you know, he said - I said, "Man, you must - you were like the priesthood." He said, "I was getting $3.25 an hour. It wasn't exactly high-end stuff," as you said.

Steve: That's exactly that era.

Leo: Yeah.

Steve: And so the idea was that, when a job was running, it owned the machine. And the problem was that some jobs needed lots of tape drives, just because they had lots of data that needed to be shuffled around in many more places. Other jobs only needed three. But if a system had 10, then when a job was running that needed only three drives, it sort of bothered the boss that those - he would, like, knock on the glass and ask those technicians why are those seven tape drives not busy? Because he knew what he was paying for those, and he wanted to keep everything busy. And they just sort of shrugged, and they said, you know, go talk to the programmers. All we do is put decks of cards in and press Go.

So what evolved then was, first, was the automated queuing of jobs in order to minimize, to squeeze out the time that nothing was going on because that really upset the boss. But then, because this thing was so expensive, there was such tremendous pressure on keeping this busy all the time, it occurred to people that maybe this thing, this machine could do more than one thing at once. There could be a supervisor that would arrange to have multiple jobs coexisting, and each using the machine when it was available. For example, oftentimes, especially back in the mag tape days, it took a long time to rewind one of those tapes. And so while a tape was rewinding, the other tapes could come alive doing a different job on their drives, using the computer, while the other one was getting

ready for its next phase.

And so we sort of - we moved forward little by little. Basically, because these things were so expensive, there was tremendous pressure to just keep it busy. And then the next change was so-called "timesharing," where we sort of backed off from this notion of having jobs, to this notion of having users on terminals. And a bunch of people would all be connected to this. And it was, you know, these machines were not fast in terms that we're used to thinking of today. I don't have any stats on-hand.

But, for example, when I was at Berkeley, we had two different mainframes. We had a CDC 6600 and a CDC 7600. And, for example, as computer science students we would punch up our cards, a deck of cards of our Fortran program, and pass them to a student assistant through a little cubbyhole, and then later come back in the day, and our card deck with the printout that that deck generated would be in a cubbyhole. And so we would pick up the results. I mean, that's literally the way this worked back then.

But then we added, we developed this notion of timesharing, where many users would be doing things at the same time. Now, not all users had the same priority. And one of the things that you got to know, for example, and this was again at Berkeley, if you really wanted to get access to the system, you would wake up at about 2:00 a.m. and trudge over to the lab where there were terminals and run your project. And it would run in about one one-hundredth the time that it would take during the middle of the day because during the middle of the day the university basically had priority, and/or the chem lab is doing x-ray crystallographic computations or something, and the lowly comp-sci students had low priority.

So you'd hit Enter on the Hazeltine terminal. The cursor would be blinking, and you'd just wait for it to acknowledge that, you know, and you'd think, I did press Enter, didn't I, and hoped that it wasn't waiting for you, that you were waiting for it. And in fact, when I worked in summers while I was in high school at Stanford's Artificial Intelligence Lab, I begged the director, Les Earnest, who was the administrative liaison with the powers that be at the campus, for access to the PDP-10 and the PDP-16 machines they had. And he agreed. But I could only use them in the evenings and weekends because, again, the system was loaded down during the day and was slow, and they didn't want any excess random stuff going on that wasn't for their main purpose.

So back then you were sharing a machine. But consider what that meant. If you had much more pressure on the system, many more people waiting for it, then it was never waiting for anyone. And that meant it was busy all the time. And so that was the thinking at the time. So now coming forward to present-day, we're sitting in front of computers that are ridiculously inexpensive. We've got them in our pocket, and we have them on our wrist. We own multiples of them ourselves, all of them ridiculously more powerful than those huge things on elevated air-conditioned floors that many people were desperately trying to share back in the 1970s.

So the concept of an operating system became important when programs needed to share the same hardware. It was originally, as I mentioned, called a "supervisor." We now call it an OS, an operating system. And the need for an operating system reflects the fact that there are a couple of things that cannot be done by the applications that are running, sort of by definition. For example, the application cannot load itself because it is itself. Even when I was toggling in the bootstrap loader through the switches of those PDP-8s behind me, you had to put in that loader in order to read the paper tape into core memory and then run the core.

But the app itself - and in fact there were some fancy tapes that had a loader on the

front. So you would toggle in a low-capability small loader because it was practical to key it in. It was like the minimum number of instructions, just to suck in the leader of the paper tape. And once that was done, what that leader was, was a much more efficient loader, which the pre-loader would load. And then it would take over and read the rest of the tape in a much more efficient fashion, using a much larger loader that wasn't feasible to key in by hand. But the point is that programs need help to get going.

The other thing that, I was going to say no program could do, but it wouldn't be practical, at least, is scheduling. That is, there are many different resources on a computer system that the programs running on it need to share. And time is one of the key resources. And so every operating system has this - it's called a scheduler, something that schedules the execution of programs. And there may not be anything that has received as much attention over the years as scheduling. There's an art and a science to it. Papers have been written, doctoral theses have been written over how to best - because it's a complex problem. Scheduling in general is a complex problem, how to maximize the amount of runtime given very complex competing requests. And ultimately these days we just throw more cycles at it, or more memory, or more juice, or more cores, and worry about it a little bit less. But it is a big issue.

And then another resource which is inherently global and shared is memory. And once again, you can't leave that up to the program. The program can't load itself, it can't schedule itself, and it can't manage the system's memory by itself. That all has to be done by a third party. It has to be done by the operating system. That is, the program can ask for memory. And then the operating system decides if it can grant that request or not. Based on rules like processes could, in the old days when there was very limited memory, you would have limits, quotas on how much memory a process could ask for. So a request could be denied by the operating system. I says, I'm sorry, but I can't give you as large a buffer as you would like to have.

So the concept of a microkernel, best stated, is the minimum set of services that must be provided, that cannot be provided by the programs running or other services running. And so the loading of them, the scheduling of their use of time, and the managing of the sharing of the system's global pool of memory, something needs to stand back, separate from the workers, and manage them.

Now, of course, there is much more to an operating system than just loading scheduling and memory management. And this is where in my notes I said, "No battle plan survives contact with the enemy," to remind myself that all microkernels start out being perfect, pure, pristine, oh, look, we finally got a beautiful little microkernel. And now the question is, how long will it stay that way? How long will it be before it becomes adulterated?

And of course we've talked many times about the decision Microsoft made when they moved GDI, the Graphics Device Interface, API support from outside the kernel to inside the kernel. They did it because crossing back and forth across the kernel boundary was expensive on the Intel architecture. There was a switching overhead that they wanted to minimize because systems were becoming so graphics intensive. So then there are a number of kind of gray areas. For example, there are many other API services beyond loading the program, scheduling the programs, and managing memory, like what time of day is it, for example. Where there are things that many programs would want to share, then it makes sense not to have every program have to write the same thing, to rewrite the same code.

Now, and there's another thing that the OS has to manage, and that's the file system. Somehow a file system is an abstraction of the physical storage device such that the programs refer to entities, files, without regard for how they're stored. It's open this file

that's named this and give me its data. And so notice that that doesn't change, if it's a FAT file system or an NTFS or a ZFS or whatever. Basically you're asking for a file. You're insulated from the details of how the file system does that.

So the question is where should this go? Or where should these things go? We have all of the additional API services. And then of course device drivers. Device drivers is an example of another shared resource. Does that go in the kernel? Or is that a service that runs outside the kernel that applications may be able to get to directly, or may need to run through the kernel to get to? And then, of course, higher level functions. When we were talking about a couple weeks ago, Leo, you were talking about, I think it was the little kernel?

Leo: Yeah.

Steve: Was it LT?

Leo: LK, yeah.

Steve: Yeah.

Leo: This is what Google's Fuchsia is based on, yeah.

Steve: Correct. And it all sounded fine. And then I heard that somebody said, oh, yeah, and it'll have GPU support in the microkernel.

Leo: No, it's not a microkernel.

Steve: Ah.

Leo: That ain't no microkernel.

Steve: And so this is the problem, is that everyone keeps trying to have a microkernel. And for the first week you do. And but then it's just so tempting to put more stuff in there; to just say, oh, you know, let's just add this feature and that feature and another feature. And it takes some real discipline to say no because, as we know, it is very hard to make this stuff work perfectly. And you want that kernel, you want your microkernel to not be buggy, to not have memory management flaws, not have vulnerabilities in the program loader or the scheduler. Or all the other things you add. For example, the GPU, the idea of having - calling a microkernel something that also supports a GPU? It's like, uh, well, they must have had a reason. Clearly it's because the GPU is a widely used resource. Every app that's running on this thing will be using graphics.

Leo: It's probably more efficient, too; right? I would think, if it's in the kernel, it'd be

more efficient.

Steve: Well, yes. But it's a tradeoff because look at all the...

Leo: It's also more complex; right.

Steve: ...all the trouble that Microsoft has suffered from the fact that a JPEG can take over Windows.

Leo: Right.

Steve: That's just ridiculous. And if it were outside of the kernel, it couldn't do that. But the actual code that's interpreting the JPEG has root privileges. It's kernel code. So it can do anything it wants to. So, anyway, I just sort of wanted to create some context. There is not a formal definition. A microkernel mostly represents a wish that sort of the ivory tower academicians have this notion of a microkernel, where - and in a strict microkernel, all that other crap is outside the kernel. You apply the test. Can this be done outside? That's the test. Can the loader be done outside? No, probably. Can the scheduler be done? Well, by definition, no. The scheduler has to be in the kernel. Can memory management be done outside? No, that has to be in the kernel.

So the test is does it have to be in the kernel? If the answer is no, and the decision is therefore it is not in the kernel, then somehow you've managed to maintain a microkernel. But if the decision is, oh, it'd be nice to have it in the kernel, wouldn't it? It's like, yeah, it would, but then it's not a microkernel. It's starting to be, not a maxi kernel, but still, not micro.

Leo: A kernel kernel, because most kernels...

Steve: Okay, so...

Leo: The Mac is still based on a microkernel.

Steve: Mach.

Leo: It was originally Mach. I think it's now XNC or something.

Steve: Well, everything's based on a microkernel.

Leo: Yeah. It all started with a...

Steve: That's right.

Leo: Linux has never been a microkernel. Linux is always a monolithic kernel. That's, by the way, the opposite of micro is monolithic, which is everything's in it, everything but the kitchen sink.

Steve: Yeah, it's kernel bloat.

Leo: Yeah.

Steve: So, okay. We will wrap up with the puzzler for next week. I don't want people to tweet the answer. You can talk among yourselves. If you have some fellow geeks, this might be fun to talk about. But don't send it to me. Don't tweet it. Keep it to yourselves, and we will discuss this next week. And because what's fun about this is there are some subtleties to it. So challenge yourself, not only to get, like, the right answer, but the full answer. So here's the question. And this was a tweet I received this morning: "I've often heard you talk about cryptography and prime numbers. But why can't non-prime numbers be used?" So think about that. We talk about...

Leo: Why can't non - I know why. Go ahead. I think I know why.

Steve: Don't tell us.

Leo: I'm not going to tweet it. I'm not going to Facebook it. I'm not going to semaphore it.

Steve: Just for yourself, it's a self-test. Why can't we use non-prime numbers?

Leo: Why can't we use - oh. I'm going to put my answer in a sealed envelope to prove that I, well, if I'm wrong, then I prove nothing. But if I'm right, to prove that I knew ahead of time.

Steve: Yes. Now, will this be the Schrdinger cat envelope, where...

Leo: Yeah, that's right.

Steve: Where we don't know what it contains.

Leo: It's always right, no matter what happens. No, I think there's a pretty good answer to that question, actually. That's a good question. Why does it have to be prime numbers? Why does it have to be prime numbers?

Steve: Yup.

Leo: Steve Gibson is at GRC.com. You know that. That's where you'll find SpinRite. Make a yabba-dabba doo in his office, anytime of the day or night.

Steve: And again, thank you. There were two, actually, that occurred when I began talking about SpinRite. I'm sure it was live listeners who were waiting to push the button.

Leo: Yeah, that's nice.

Steve: So thank you very much.

Leo: That's really great. Don't wait. Do it right now. SpinRite. You'll also find - that's his bread and butter. It's the only thing he charges for. Everything else is free there, including the latest crypto stuff, information on SQRL, the perfect sleep formula, healthful sleep formula.

Steve: Never10 is down to 4,200 downloads a day.

Leo: Yeah. Well, there's no need for it; right?

Steve: I don't know why. I know. It's like, I still have LeakTest there, and people get it every day.

Leo: Yeah. Hey, well, it's free. I'm going to take it. Let's get some more stuff. Free stuff. GRC.com. You can also go to GRC.com/feedback to ask questions. But probably the best way to do it is go to Steve's Twitter, @SGgrc. And you can tweet him. He takes private tweets, if you want to give him a lead, if you work for the Equation Group, whatever, @SGgrc. You'll also find the audio and human written transcripts of the show at GRC.com. Steve does that every week out of his own pocket, and I appreciate that, Steve. And, yeah, SpinRite's there. We have audio and video at our website, TWiT.tv/sn. You can also subscribe.

By the way, we've got an update. Remember Patrick Delahanty told us that there was a script running out of Australia that was attempting to download every Security Now! episode ever?

Steve: And going into the future, where no one has gone before.

Leo: Yeah. It's still running. He is looking for Episode 577,043 today. Not kidding. This morning...

Steve: It's in somebody's closet somewhere.

Leo: He just ran it and forgot it.

Steve: Too bad because it's not going to get #574, this one. It's shot right past.

Leo: Oh, that's true, it's gone right past it. That's the disadvantage of doing that. That's right. Because you can't retroactively - oh, I found it. That one you were looking for. I found it. Unless he's written some very good code. And I hope he hasn't because his memory's overflowed by now. What else? I think that's about it. We'll be back in our regular studio next week. Although this really worked well, I think. It's, yeah, looks great, and it was very comfortable for me. If you want to join us live, you do that every Tuesday at 1:30 Pacific, 4:30 Eastern, 20:30 UTC. And we'll see you back here next week for Security Now!.

Steve: Thanks, Leo.