Stacks, Registers & Recursion

Description: After a significant security news update, Steve and Leo continue their description of the operation of computers at the raw hardware level. This week Steve explains why and how computers have multiple accumulators, and also how a computer's "stack" operates and why stacks have become a crucial component of all modern computers.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-239.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-239-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 239 for March 11, 2010: Stacks, Registers, and Recursion.

It's time for Security Now!, the show that covers all the things you need to know about keeping yourself safe online - security, privacy, spyware and viruses - and, lately, kind of a fundamental education in the way computers work. Here he is, the star of our show, Mr. Steve Gibson of GRC.com.

Steve Gibson: Hey, Leo, I ran across a new term in the security terrain this week. We now have a new term that's come into - sort of a new term of art in security: "weaponized email."

Leo: Oh, dear. That doesn't sound good at all.

Steve: Yeah, so...

Leo: Weaponized anything's bad. But weaponized email, oy.

Steve: Weaponized email. So we'll be talking about that as part of our security news, and then continuing our journey through the how computers actually work down in the basement, talking about stacks, registers, and recursion this week.

Leo: Oh, good. Oh, all three together, huh? Because I thought maybe we'd take one at a time.

Steve: Well, we need to - I've been wanting to introduce the notion of multiple registers. And I want to kind of hold people in this awareness that it's all about bits in the machine language. So, and registers are often being stored and saved on the stack, which is an important use for it. So I thought, well, I really can't talk about the stack meaningfully until we have that. We need to talk about registers anyway. Then I was noticing that there have been many people talking about all of the pads that are coming out following the Apple iPad announcement, and how they're all ARM-based.

Leo: A million of them, yeah.

Steve: And there's a reason why everyone is choosing ARM processors. And so one of the cool things is, as a result of having laid this foundation of understanding, we'll be able to really grok RISC and CISC and power consumption and these things that really shape our world of computing.

Leo: Good. And they all do go together, don't they.

Steve: Yup.

Leo: Do you think you can do it in one piece? More power to you, Mr. G. We had a Patch Tuesday, didn't we.

Steve: Yup.

Leo: Snuck right by me.

Steve: Although that was not - it was sort of an anticlimactic Patch Tuesday. But lots of other goodies.

Leo: All, Steve. Let's talk patches.

Steve: So probably the least interesting little tidbit of security news this week is Microsoft's second Tuesday of the month. They didn't have a completely dead month, but nothing really very exciting.

Leo: That's good.

Steve: That's, well, yeah.

Leo: That's what you want.

Steve: Last second Tuesday of the month our listeners will remember that one of the patches that Microsoft released was inexplicably crashing machines with the Blue Screen of Death problem, which it turned out was related to people having a trojan installed on their machine which was misbehaving because Microsoft's patch changed the structure of some of the core kernel files, which changed the locations of jump points and structures in memory, which the trojan had been written to in sort of a hard-coded fashion. So when the update was applied, these critical locations changed. When you rebooted your machine, the new kernel components were loaded in memory, then the rootkit that was hiding also in the machine would come in and attempt to install itself. But because these core structures had changed, its modifications to the kernel caused the system to crash. So one of the things Microsoft did this month is update last month's patch to check for the rootkit.

Leo: Great.

Steve: And it will no longer install itself. If you've already been infected, then you might as well just give up and go home. Actually the MSRT, Microsoft's software removal tool, has also been updated for awareness of this. So it will catch it and remove it from your system, and then last month's patch can be installed without crashing your system, which is a good thing.

And then there were just two updates, neither of which were critical in Microsoft's severity ranking scale. They were just ranked as important. Windows Movie Maker, not Live Movie Maker, which is in Vista and Windows 7, but the Movie Maker which was in earlier versions of Windows, and Microsoft Producer 2003, were both found to have an exploitable problem such that, if somebody could induce you to open a Windows Movie Maker or a Microsoft Producer 2003 project file, that could cause your machine to run code of their design. I'm not sure why they didn't consider this critical. Maybe we're just sort of - there are so many things that are really bad now that something that's just bad no longer rates critical. I'm not sure. Maybe it's just because these are sort of obscure things.

Leo: It's like grade escalation.

Steve: Exactly.

Leo: It's creeping up. Ah, that's not so bad.

Steve: And then somebody reported seven different vulnerabilities in Excel to Microsoft some length of time ago, lord knows how long ago. But Microsoft has fixed those. And that's the other change, which they also ranked as important, even though somebody targeting you, who sent you an Excel spreadsheet and could induce you to open it, could potentially run their own code in your machine. So that's also not good. Those are fixed. And that's pretty much all we had from Microsoft for this Patch Tuesday.

Opera, however, is not singing right now. A really bad vulnerability, bad in the sense that it's so easy to exploit, has been discovered and confirmed. So Opera's own advice, there's no update for it as of the date of our recording of this podcast, which is the 10th of March, and not certain how soon there will be one. Opera has said that you should enable DEP for Opera, which is not enabled by default. But it's not clear that even that would be sufficient. The security industry is just telling people don't use Opera until...

Leo: It's so sad. I mean, this just came out.

Steve: Yeah. And the reason it's disturbing is that we've talked about how the HTTP protocol works, with headers, where the query to the server can include headers which the user never sees. They are so-called "metadata." They're not part of the query; or, in the case of a server's reply, they're not actually part of the content being returned. But they're additional data that helps with the interchange. For example, there might be an expiration date on the contents so that the client is able to cache what it receives without continually asking for it over and over and over until some length of time has passed or some date has passed.

So there's, like, sort of an expiration date on the content. Or the content length, where the server says, okay, you asked for a JPEG. The JPEG itself doesn't specify the length. So a header is added, the so-called "content length" header, so that the recipient knows how long it's going to be. That's handy, for example, if we wanted, like, a progress bar, and the server's downloading something big. If it knows how large it is, it knows how much it's received. So it's able to present the user with a valid progress bar saying, okay, we're getting something big here, and this is the percentage of it that we have received. You only know that if you know how long it is overall. So this is useful additional so-called metadata that is received from the server.

Well, as it turns out, the specific problem is in, in fact, the content-length header. If the content length header is representing a 64-bit quantity, and the most significant 32 bits have their sign negated so that it looks like a large negative quantity, that trips a buffer overrun. It trips a misbehavior in Opera, which is believed to be exploitable. And so the reason this is bad is that it's so easy to do. I mean, this is potentially, you know, you don't need any fancy, content-specific, download this Excel file or download that. Apparently any web object with a content length that was invalid could trip up Opera.

Now, we are currently at Opera v10.5. That's known to be vulnerable, and it's presumed that all previous versions are, as well. So Opera users are advised not to use Opera until they've got something past v10.5. I'm sure the Opera folks are working on fixing it. It should be a trivial fix. I mean, that's - the downside is that it's so easy, I mean, this is something so intrinsic to the HTTP protocol. But the good news is that, because it's that way, it ought to be instantaneous to fix. So I think it's just a matter of them not having had a chance yet to respond. Maybe even by tomorrow, when this podcast goes live, there will be a new one. So, if so, it'd be good to fix that.

Leo: Are there exploits in the wild? Is it a zero-day exploit, or just - somebody just...

Steve: No.

Leo: No.

Steve: It was a vulnerability discovered, and it's been confirmed. And what I just told our listeners, this is how you do it, is well known, too, on the Internet.

Leo: Everybody knows that, yeah.

Steve: So the problem is, bad guys with that information will be able to look at the nature of the crash and say, ooh, I know how to execute code with this. Now, it's interesting, too. I'm going to tell a story here at the end of an interesting exploit involving social networking, which is increasingly becoming part of sort of a multiphase blended attack. And this notion of weaponized email that I mentioned is exploited there, and used. But remember that it could be that people think, oh, well, Opera only has a couple points of market share, so nobody is going to - it's unlikely that I'm going to be clicking on a link which is going to take advantage of an obscure Opera bug. Except that what we're seeing more and more is that exploits are no longer being sprayed at random across the Internet for these kinds of problems. Instead, the bad guys are targeting specific companies or specific groups or individuals. And, for example, if they knew that a company had standardized on Opera, and it's arguably possible to determine what, like, web browser technology...

Leo: Oh, yeah, you can see, yeah.

Steve: ...a company uses, yes.

Leo: Yeah, it's easy.

Steve: Then the bad guys would go digging around in their Opera bag of tricks and say, oh, look, here was a bad problem in Opera. What are the chances that one or more of the people in the company haven't fixed it? So then they would go about sending email with links that would launch Opera to open a web page which would be deliberately crafted to be malicious.

So, I mean, this is - it's really - it's come to the attention of the security industry. And this was a topic in much of RSA's conference last week, that we're really seeing a transformation in the way vulnerabilities are being exploited in machines. And so it no longer means that really obscure problems aren't something, unfortunately, to be worried about. And it's specifically because they can be so powerful that every one of them is a potential entry point.

Leo: Is the problem also that the tools exist for people who are not necessarily so skilled that they could take advantage of this to take advantage of it?

Steve: Oh, absolutely. In fact, I was reading, in the last week or two, exactly that comment. There were some people editorializing about the nature of security problems.

And one of the things that really upset them was that - I think it might have been the three guys that were nabbed a week or two ago for running one of the large botnets. And in looking at - in talking to them, and in looking what they were doing to be running, essentially, a large multi-hundred-thousand-machine botnet, these guys were not technical.

Leo: Yeah.

Steve: I mean, they weren't capable of writing any of this stuff themselves.

Leo: They don't have to be.

Steve: They picked up pieces that were readymade and put them together and basically were sort of orchestrating other people's code. And so that was a serious concern was that it's now no longer - you don't need deep voodoo hacking skills in order to do this. It's becoming more canned. Very much like I was talking about Metasploit, that system which allows people to grab these exploits when they're very new and see how they run and do them themselves.

Leo: Yeah. Good news.

Steve: Yeah. Speaking of penetration, there's been news on Aurora. Aurora was the name given to the series of attacks which Google first publicized, but which were actually, as the investigation expanded, it looked like it was on the order of 30 or more corporations were penetrated. They did trace them back to a couple locations in China. And what they found - McAfee gave a presentation at RSA last week about the Aurora attacks. And this is where we were first seeing the term, people using "weaponized email." Used to be called "spear phishing."

Leo: Ah.

Steve: So what we used to be calling spear phishing, where you would be deliberately aiming email at people, we're now calling it weaponized. What they found was that the Aurora attacks were going after the penetrated corporation's stores of intellectual property, the so-called "source code repositories." And so, for example, they found that Google's source code repository was not adequately secured, and that the bad guys were able to get in there and literally browse around. And so...

Leo: Through mail aimed at somebody who had access to it already?

Steve: Exactly.

Leo: Okay.

Steve: Or email, see, this is all sort of incremental. Email aimed at somebody else who might have access to somebody else who had access to it. I mean, once you're in, then you're able to bounce around until you find what you're looking for, and able to see the contents of someone's machine, see who he or she, that employee, is corresponding to, look at their email and say, oh, wait, that looks like they're writing to a developer to ask them a question about something. Let's see if we can trace over to that developer's machine because they'll probably have the access we want. And of course you could do three things if you are able to gain access to a company's IP, intellectual property repository. You can download the entire source tree. You can get write access to it and subtly alter existing proven source code...

Leo: Ooh, right.

Steve: ...so that future products which are produced using that source code - and remember that, I mean, products are so large now, I mean, that millions of lines of code goes into these products which ends up shipping as megabytes, you know, tens of megabytes of code, that subtle alterations that go undetected in source code then end up being bundled into future products which are exploitable by those people who made the changes. So it's like a way of tricking well-meaning, good companies with as much integrity as any to shipping trojan horse code, trojan horse products that have backdoors built into them that nobody but the bad guys who were able to slip in in the dead of night and make some subtle changes to the source code and then get out unnoticed. Then this stuff gets compiled and shipped.

Leo: Right. A lot of, now, I don't know which source code repository they're talking about. If they're talking about Google code, a lot of that's, like, you know, websites and open source projects, things like that. I wonder what they mean when they say the database was compromised.

Steve: Well, and not just Google; but, I mean, other major corporations. Anybody who's got intellectual property, apparently. There's a common set of utilities that they're all using. I'm trying to think of the name. It was, like, Perforce? I think it's Perforce is one of the very, very popular source code management tools.

Leo: Oh, so it's like a code repository software. I get it. I get it.

Steve: Exactly, source code repository. So developers, well, so one problem, of course, is the repository. The other is the nature of the way the developer's working. They'll often check source code out from the source code repository onto their local machine, use it, work with it, make some changes, then check it back in. But it stays on their machine. So it's often the case that you don't even need to get to the source code repository. You can alter the code on a developer's machine.

Leo: Sure. And then they check it in and, boom, you're done.

Steve: And so what was found, and this was the gist of McAfee's talk, essentially, at the RSA conference was that there's a presumption of trust of the developers, naturally. And

that has sort of extended to a trust of the network. So, for example, they found that there's no security in - like endpoint security, no SSL encryption being used in the source code repositories when they connect to developers. So all the standard problems of lack of SSL apply.

Now, you think, well, wait a minute, we don't need SSL. It's our Intranet. Except that, when you've got bad guys occupying nodes of your Intranet, now it's like the Internet, where you really do need to worry about point-to-point security, even within your own corporation. So this whole - sort of this transformation changes this notion of the security boundaries. And so once upon a time a corporation could feel comfortable with a firewall that isolated their Intranet from the external public Internet. And they'd have their filters up, and they'd have all their ports closed unless they were using them and say, okay, now we're safe. But this whole next generation of weaponized email is a way for the bad guys penetrating that through something as - what used to be as benign as email getting into the corporation behind those barriers and setting up camp.

Leo: Wow.

Steve: And that's what Aurora did. That's what - it happened to a bunch of companies. There was another interesting report. We talked about Secunia quite a while ago. They had their Personal Software Inspector, PSI, which is a really cool free download, which I know a lot of our listeners grabbed, downloaded. And you run it on your system. Basically it does an inventory of all the software that you've got installed and then checks the version of the software versus the most recent version to let you know, with a very nice and friendly user interface, if you've got stuff that's got known vulnerabilities that you should update. Well, thanks to their doing that, they've been able to accumulate a substantial database of the state of software in the machines of the population, which is many, that ran this free PSI tool.

Leo: Now, I presume that they told people they were doing that.

Steve: Oh, yeah, yeah, absolutely.

Leo: Good.

Steve: And so Gregg Keizer, who reports for Computerworld, reported on a Secunia study which said that the typical, average computer user is now being subjected to 75 patch events per year.

Leo: 70, wow.

Steve: 75. So essentially a patch event...

Leo: That's six a month.

Steve: It's every 4.9 days is, on average, the typical user needs to do something. And, I mean, when you think about it, Leo, I mean, I'm constantly updating all of the software that I use. I mean, it's like more or less a continual thing for me.

Leo: Yeah. That's true, yeah.

Steve: I've got a lot of software, and it's just creeping forward. I'm checking to see if it's the latest version. Oh, look, there's something new, no surprise, and you download it. And...

Leo: Is that a bad thing, though?

Steve: Well, this little blurb was covered in a recent SANS security newsletter. And one of their editors, one of the members of the editorial board, Eugene Schultz, he commented, he said Secunia is right about this being a burden. He said, "There are just too many bugs in too many software products. There's no way that the average PC user, let alone the average organization, can even begin to keep up with all the patches. A single patch installation method would help only to some degree. The far bigger problem is software developers continuing to produce bug-infested software with few, if any, negative consequences to them." Because what happened was...

Leo: You're right, because we don't mind. Oh, yeah, more updates.

Steve: Yeah. I mean, maybe...

Leo: That's good, maybe it'll be better.

Steve: Exactly. There's sort of this holy grail sense of, well, maybe this will make it work better. What happened was that, in last year's RSA conference, in the '09 conference, Secunia made a case for it being too burdensome; that not only were there 75 patch events, but on average the software was coming from 22 different organizations, each with their own patching methodology. And so Secunia made a case for a single, industry-wide patching approach, whatever that would be. And they were just blown off. I mean, everyone said, oh, yeah, okay, no. We're going to do our own.

Leo: Well, because they're never going to agree.

Steve: Exactly.

Leo: It would have to come from Microsoft, probably; right? Have to be OS-based.

Steve: I've noticed that one of the installation tools, InstallShield, seems to be trying to aggregate some of that. I've got a single instance of InstallShield Update, and a couple,

two or three programs all use InstallShield and InstallShield's update management. So it sort of wakes up every couple weeks and says, hey, I need to check to see if anything's new. It's like, okay, fine.

Leo: OEMs have done that, too. I mean, HP used to run BackWeb on the machine. It would automatically download and patch the machine.

Steve: Oh, thank goodness those days are gone.

Leo: Yeah. Well, that's maybe why the industry kind of went, unh. Because our memory of these unified patch solutions is not so hot.

Steve: Yeah, yeah. So anyway, I just thought it was an interesting statistic, that in general a little less than one every five days, they're...

Leo: That's about right.

Steve: Well, and the point is that, here we are, we're all geeks and nerds and, I mean, we love computers so we're listening to this podcast. But typical users, they just want to use the darn things.

Leo: Right.

Steve: They just, I mean, they're not living for this. This is their worst nightmare. And it creates a real tension in the consumer's mind that, boy, this thing's working against me. This computer's not my friend.

I wanted to note that RealNetworks lost definitively their two-year-running suit that the MPAA brought against them. In a settlement they finally agreed to give up their battle, to pay the MPAA $4.5 million to cover its legal costs, to permanently stop selling the product, and to reimburse the 2,700 owners of RealDVD $30 each. So RealNetworks' argument was that all their product, RealDVD, did was allow people who legally owned a DVD to copy it to their computer's hard drive in order to watch it there, rather than on a DVD player. And the MPAA argued that that was a breach of the DMCA and a breach of the agreement which Real had signed when they obtained the technology to decrypt DVD. So on whatever basis, Real said, okay, we're going to take the product off the market. So RealDVD no longer exists.

Leo: They gave in on that one. They could have fought on.

Steve: Yeah.

Leo: And I almost wish they had. There's a good guest's editorial in Boing Boing

saying Real kind of damaged us all by giving up on this one. And if they'd only gone on and maybe had won, which was possible in a trial, that maybe they could have protected this right to copy. Because they did everything right. They got - anyway.

Steve: Exactly. They obtained a license. They were protecting the intellectual property. It was a matter of the MPAA deciding, eh, we don't like that.

Leo: We don't like any kind of copying under any circumstance.

Steve: Yup.

Leo: Yeah. That's too bad.

Steve: Okay. Are you ready for this, Leo?

Leo: Yes.

Steve: The Energizer Bunny.

Leo: I saw this. Oy oy oy. And this actually harkens back to your spear phishing story, really. In a way; right? I don't know how they got infected, but...

Steve: Well, they don't know how they got infected [indiscernible]. So here's the story. There is a product called, from the Energizer battery people, called the Energizer Duo, which is a USB-powered, nickel-metal hydride battery recharger. And the USB interface allows your computer to monitor the charge status of the battery to see how far discharged it is and to watch it over time charge itself up. Since mid-2007, I think it's May 2007, that software that comes with this Energizer Duo product has been installing a trojan in the machines of everyone who used it. There's a DLL named arucer.dll which is in the Windows\system32 directory after you install this Energizer Bunny Duo software. It's a trojan which opens port 7777.

Leo: It keeps hacking and hacking and hacking...

Steve: Keeps hacking and hacking and hacking. And it lives in your machine whenever it's running. So they've taken it off the market, Energizer has. Anyone within the sound of this podcast who might own the Energizer Duo is advised to uninstall it. Uninstalling it removes it. You can also delete the arucer.dll from your Windows\system32 directory, and that will do the job, too. Then reboot your machine, and you'll be okay.

And presumably, if you opened a DOS box and did a netstat -an. That would show you the ports that your system currently has open and is listening on. And if you see 777 listed there, that's further confirmation that this little trojan has set up shop in your

machine. The good news is, if you're behind a router, it will be protecting you from incoming traffic on that port. So anyone who is out there scanning for the trojan, looking for connections, connectibility on port 7777, would not have been able to reach your machine. And as far as I know, this thing does not initiate outbound connections. It just sets up a listening connection so that if you were not behind a firewall, then you would have probably been discovered, and somebody would have taken over your machine.

Leo: Oh, yeah, well, there you go.

Steve: Now, I loved this story - it appeared in USA Today - because it gives our listeners a real good sense for the kind of technology and the kind of attacks which we're going to see more in the future. And essentially, in this case, how and why social networking is factoring into attacks increasingly. This is the result of an investigation late last year, in 2009, by a network infrastructure provider, Terremark. They researched a successful attack on an unnamed major financial U.S. firm. An employee of this unnamed major U.S. financial firm had a Facebook page which got hijacked through a means that was not disclosed. But we've talked about Facebook hijacking a lot in the past. It turns out it's all too easy to do.

So the bad guys got a hold of - got access to this employee's Facebook page. Looking at the page, they saw postings about a recent company picnic. And they saw that this person's Facebook friends were, not surprisingly, other employees in the company. So they sent email as if from this employee to the other employees of the company that were identified through this Facebook page. The email said something to the effect of, hey, check out the pictures I took at last weekend's company picnic.

So a number of employees received this email, which is on point, it's contextually relevant, it's from a friend, one of their Facebook friends who's an employee of the company. All of it makes sense. Clicking on the link in this one case installed a keystroke logger on the laptop of a female employee who received this email and had every reason to trust it, this so-called "weaponized" email. So she now had a keystroke logger on her laptop. Subsequently, when roaming outside of the company, she logged in through the company's VPN. And her login credentials and whatever else was required, like a client certificate and so forth, was captured and sent to the bad guys.

Leo: Oh, wow. Clever.

Steve: Yes. They now had the ability, and did, to log in through the corporate VPN, get into the Intranet. They spent two weeks roaming around inside the company before their presence was detected and had taken control of two of the company's internal servers by that time. So this really happened.

Leo: That's an amazing hack.

Steve: This is the way it happened. This is, you know, targeted. And I think what we're going to be seeing more in the future is this kind of vulnerability. I don't know whether - where the original impetus came from, whether it was this first employee whose Facebook page got hacked, then they looked to see who he worked for and said, oh, let's go after that company. Or maybe the initial intent was, let's go after this company. Let's

see if any of the company's employees have Facebook pages. That led them to a collection of Facebook pages. They were able to get - they were able to hijack one Facebook page, then leverage that in through a series of actions into remote VPN access into that corporate network.

Leo: And it wouldn't be enough to say, oh, we're not going to allow you to use Facebook at work because it was on her laptop at home.

Steve: Correct.

Leo: So you would have to say, as an employee of this company, you may not use Facebook anywhere, anytime. That's not going to happen.

Steve: And so what - yeah. And so the way the social networking is factoring into this is, for the first time ever, there's a sort of a formalized forum for the publication of interrelationships.

Leo: Right.

Steve: Here are all the people I'm Facebook friends with.

Leo: Right. You're not just giving away your birthday and your weight. You're giving somebody your social circle, your graph.

Steve: Your social network. And so one of the - and so we've seen already, and we've talked about this often, that it is the social engineering which is a continuing problem for security because that just slips under people's radar. It's like, oh, mail from Mom. Mom is not going to attack me. So that's what happened. And so we're seeing things like Twitter links being sent where it's like, "LOL, is this really you?" And then a link. And so it's like, oh, who's - if this comes from someone you know, and they apparently found something about you on the Internet, "Is this really you?," you're going to be moved to click that in order to find out what it is that they think they know about you.

Leo: You know, Dvorak's Twitter got hacked over the weekend. But, now, he says, he swears, oh, no, I didn't do anything. It was a security flaw at Twitter, and other people got hacked, as well. And I don't know what the truth is of that matter. But it does make you think, boy, if you really want to be secure, you should not be on these networks.

Steve: Well, I mean, with it comes some responsibility because social engineering hacks are able to be blended with these kinds of targeted attacks, like using weaponized email, to get underneath people's defenses. I mean, the problem is that 99.999 percent of the email you receive is fine. You have habits which generally don't hurt you. But it doesn't mean that that .001 percent can't really do some damage. Through no malicious intent on either of these employees' part, bad guys got access to her VPN login credentials and

used them to get access to inside a corporate Intranet. And it really happened just this way.

Leo: Last night I'm sitting watching a movie, and I hear my wife say, "Yes, no, we have a burglar alarm, but we don't use it. Yes, the stereo system's in three rooms of the house. Honey, how many square feet is our house?" I said, "Who are you talking to?" She said, "Oh, it's the insurance company." I said, "The insurance company called you, and they're asking these questions?" "Yeah." I said, "Hang up and get a number and call them back." She said, "No, no, it's Tanisha, she's fine." I said, "What do you mean?" "She's a very nice lady." I thought, this is not - even my son Henry said, "Mom, hang up immediately. What are you telling...." She told them all this stuff. Now, I think it was the insurance company.

Steve: Yes.

Leo: Which makes me think, what the hell are they thinking? But you just - and this is the kind of thing hackers do. I did an event on Thursday, you would have loved this event, called "Wired Families, Safe Kids." It was at our kids' school. And it ended up being the parents, you know, they came to learn how to protect their kids online. But really, the truth is, they want to know, okay, what's a password keeper? What should I do, you know, how do I - what do I do when the bank calls and they want - they wanted to protect themselves. Everybody's dying for this information, and everybody's terrified.

Steve: Yeah.

Leo: And I guess, when I hear stories like this, rightly so.

Steve: Yeah.

Leo: Get off Facebook. I told them I think Face- I'm really starting to think Facebook is not a safe platform. Twitter is clearly not.

Steve: Well, these things are, I mean, this is a refrain our listeners are probably getting tired of. But, I mean, they're so hard to do safely. It's just, it's so...

Leo: They don't seem to care. I mean, truthfully.

Steve: It's so difficult.

Leo: It's difficult if you care, and they don't even care.

Steve: Well, and you read the fine print of the agreement, and they're completely

harmless. It's as the SANS editor said, who I read before. There's no downside for the companies. Companies are not held responsible for flaws in their software. Microsoft began this with the original license agreement a long time ago that said, "You use this at your own risk."

Leo: Right.

Steve: And back then there wasn't much risk. Unfortunately, the risk has gone exponential, and the agreements haven't changed. It's still the fact that the onus is on us.

In errata, I wanted to mention to our listeners, just because sci-fi is a topic that is near and dear to my heart, and you love sci-fi as well, Leo...

Leo: Oh, yeah. Oh, yeah.

Steve: I discovered just two days ago ABC's series "Flash Forward." And I'm enthralled.

Leo: Oh, good.

Steve: There was an ad that I saw briefly on TV. Someone, and I can't remember who, it might have been Mark Thompson, somebody said it's really good. But I hadn't started watching from the beginning. And it's like, oh, well, okay, too late, because I was - who knows how many weeks in I was. So I just thought, well, maybe I'll get it on disk someday or just - or I've got enough to watch. So I saw an ad that said, hey, the first - the series was in hiatus over the holidays. It's been in hiatus, I guess, for a long time. And it's getting ready to start up its second, the second half of the first season. And so the ad was for the first 10 episodes on DVD. So I said, okay, I've heard this is really good. That's enough.

I just watched the first one, and I was seriously hooked. The reason, for a sci-fi person, I recommend it. But it's not narrow in scope the way, for example, "Terminator" was when it was on Fox, or "Galactica." I mean, I think this will probably succeed in the market because there's a lot of human interest, it's got a lot of facets. Mostly, though, I think it owes its success to the fact that it's based on a novel by the same name by Canadian sci-fi author Robert Sawyer.

And the premise - this gives away nothing because everyone probably knows at least this much about the series if they haven't watched it, is that the entire world blacks out for two minutes and 17 seconds. I think that's the number. Maybe it was 213 seconds. Anyway, so for a little over two minutes. And during this time they see their life six months in the future. They see two and a half minutes, or whatever that length of period is. They're, like, moved, the entire world's consciousness moves forward six months for that period of time, and then they wake up.

Leo: So they know what's going to happen six months from now. Everybody. Not just one.

Steve: Yes, yes.

Leo: See, I thought maybe it was one person. They all know.

Steve: Everybody.

Leo: Oh, wow.

Steve: And so, first of all, it's really not good if everyone loses consciousness for two minutes.

Leo: Yeah. Planes fall out of the sky, cars run into trucks...

Steve: Yes. So we have mass devastation...

Leo: I'd fall off my ball.

Steve: ...globally. Now, this happened at 11:00 a.m. Pacific standard time, so they were asleep in China, so that was - there was less devastation there, which causes the CIA to wonder if China caused it. It's like, okay, folks, they were asleep. But what's really compelling is what is done with this concept. That concept has so much room for interesting things. For example, the FBI sets up a website called Mosaic to solicit people sharing their visions, their flash forwards.

Leo: Oh, interesting. Trying to piece it all together.

Steve: To piece it together. And, like, one FBI agent is, in his vision, he's with another FBI agent. Well, so in the first few minutes of the show, and again, this is not a spoiler, he, like, says, wait a minute, we can figure out if this is true. And he calls her across the planet, and she says, "I know why you're calling." Which meant that they had a shared vision. I mean, she had the same thing as - anyway.

So the point is, I recommend this highly. We're 10 weeks in. You have to see it in sequence. So it resumes next week. This is March 11th, the date of this podcast. The series resumes next week. You can watch it on the 'Net on ABC. You can watch all the episodes, all 10. You can get the DVDs. Who knows, you can probably find it elsewhere on...

Leo: It's on Netflix, I know.

Steve: ...on the 'Net. Anyway, it's just - I can't, of course, vouch for what's going to happen. This does feel to me like an arc which is fundamentally limited. I mean, they're going to run out of steam here at some point, or they're going to start stretching it out, I

mean, there are all kinds of ways they could screw things up. But, you know, sort of like "Galactica" did. "Galactica" was fantastic for a couple years. But it was a victim of its own success because they wanted to keep it going, but sort of they ran out of story. So anyway, this is just - I discovered it two days ago. I watched the entire 10 episodes in the last two nights because, I mean, I couldn't go to sleep. It's just like, what's going to happen next? It was really fun what they do with it. So I just wanted to recommend it. And the original book is available on Audible.

Leo: It is. And a number of people have told me that the book - read the book first.

Steve: Well, I...

Leo: Because they say the book is good.

Steve: I would imagine the book is good. It looked to me - the book has a Wikipedia page. And the book's plot already diverges significantly...

Leo: Good.

Steve: ...from the TV series.

Leo: That's good because you could read the book, and it won't affect your enjoyment of the series at all.

Steve: That's correct. Yeah. Anyway, I just - I recommend this series. It is science fiction. It is really clever. And as I'm watching it I'm thinking, oh, this is good because somebody who was really a sci-fi author laid down the foundation. And I have to say that the writers, who have - just the clever things that happen from this premise of the world sharing two and a half minutes of its future is really fun.

Leo: Great, great science fiction, I think, is like that, that you say, okay, here's the thing. Now, what are the implications? And you could tell the writer just goes - really has fun saying, well, it would mean this, and it would mean this, and it would mean this...

Steve: Exploring the implications.

Leo: Yeah.

Steve: Yes.

Leo: If just one thing were twisted, you know? What if Germany won World War II or something like that? And you just go with it, and it's really fun. I really enjoy it. But I haven't seen the show or read the book, so now you've given me something, a new assignment. I'm excited.

Steve: Well, I know how busy you are, Leo. But it's really good.

Leo: I'm always looking for one - I want one show. I can't do more than one show. One show a week I allow myself. Always looking for that show. And I don't have one right now, so.

Steve: It's good. It's not frustrating the way "24" was.

Leo: Or "Lost." I think - I haven't watched "Lost." But people I hear just go crazy over "Lost."

Steve: And I think probably, once it's all said and done, I've asked people, okay, now that you - because, like, Mark Thompson is a total Lost-aholic. And I said, so is it worth getting it and watching it? He's like, oh, yes, yes, yes, yes. I guess it kind of lost its way in the middle, but then it kind of came back toward the end. So...

Leo: Oh, [indiscernible] loving it, so...

Steve: Yeah.

Leo: And this is the last season, so...

Steve: Yes, yes, yes.

Leo: I'm saving - there are certain things I'm saving for the nursing home. That's one of them.

Steve: So I have an interesting SpinRite story which involves Astaro, of all things, one of our sponsors.

Leo: Oh, cool.

Steve: From someone named Mark. He didn't share his last name. And I'll keep his email address private because he didn't give me permission to share it. He said, "SpinRite Fixes Astaro Firewall and Wife's Payroll Submission." He said, "I enjoy listening to Security Now! every week. Because of Security Now!'s Astaro ads, I built an Astaro gateway for

my home using an old Pentium II. I love it. So thanks also to Astaro for sponsoring the podcast and offering free home licenses. It just works, 24-by-7-by-forever." And he said, "(See the attached graphic that shows the unit working from June through September when we had a long power outage that outlasted the battery backup and an extended period it was off in December when I unplugged it when I was on vacation.)"

He continues, "It amazes me, since it runs on a 900MHz processor with a 20GB hard drive. I think I've been using this hard drive for at least 15 years." Now, he must mean 20MB hard drive. Maybe not. 15 years? Yeah, I guess 20GB - "for at least 15 years. Until this week. One night this week I got home, and my wife was very upset because 'The Internet was down.'" Don't you hate when the Internet goes down?

Leo: I hate it when that happens.

Steve: Yeah. "She needed to submit her payroll form, otherwise she would not get paid for two weeks. This was a serious problem. So I started doing network troubleshooting, and I noticed the PCs in my home network had a 169.254.x.x IP address. Well..."

Leo: Self-assigned.

Steve: Yes, that's Microsoft's block that is for - what machines get if there's no DHCP server responding. They sort of assign themselves an address. So he continues, "That pointed immediately to a DHCP problem. The Astaro is the DHCP server. So next I checked on it. Strangely, it didn't respond to an administrative panel request from a web browser, nor did it respond to a ping to its fixed IP address, either. I rebooted the Astaro gateway to see if it would fix things, and noticed as it rebooted that the Astaro boot sequence said the hard drive was write-protected. That's not good. Whenever I run into a hard drive problem, whether it's on my TiVo or my Windows systems, the first thing I do is run SpinRite at Level 2.

"After a short period running at Level 2 in this instance, or at least much shorter than waiting for SpinRite to maintain a 1TB drive at Level 4" - he's saying that because it was only a 20GB drive - "I see a recovered sector on SpinRite's display. I was expecting many by the time the scan was completed. But it was time to try the drive since SpinRite was done. I rebooted the PC, and the Astaro gateway started running as it always had. SpinRite fixed the Astaro gateway's drive; repaired the sector, just as I have seen it do with my other hard drives; and my wife was able to submit her payroll on time. Thanks, Steve, for an excellent product. Thanks to Leo for choosing good advertisers."

Leo: Yes.

Steve: "I love products that just work without you even thinking about them, like Astaro, and products that do exactly what you expect from them, like SpinRite."

Leo: That's neat. I hardly need to do an Astaro ad now, do I. That's nice. That's really nice. And I'm with you, you know, when software works, or when hardware works, when something is well designed, it is such a nice feeling. And unfortunately

it does seem kind of rare.

Steve: Yeah. Not getting better.

Leo: Not getting better. So, Steve, we're ready for more in our Computer 101 series.

Steve: Yeah, we've done an hour of security news and updates and stuff, so let's talk about how computers work. This is the fourth installment. And what people will eventually recognize is that I've been very carefully and deliberately laying a foundation which I believe will end up in everyone really getting how computers function in a fundamental way, that is, demystifying this really by showing that, in my opinion, there's not that much to it. I mean, the complexity comes from refinement over time, not from the fundamentals.

So in the first installment we talked - we went to the very beginning and talked about resistors and transistors and how they could be connected to create simple logic gates and inverters, and how, for example, when you connect two inverters to each other in sort of a ring, that's an inherently stable device called a flip-flop which forms a bit of a register which is able to be loaded and remember a binary value, one or zero. And you concatenate a bunch of those in order to be able to store more complex, larger values.

We then, in the second installment, looked at machine language, the idea that machine language is nothing other than some form of storage which is addressable, where the address is stored in a program counter, which is just the address of the current instruction we're executing, and that the instruction itself is a collection of bits. We read a word out of memory, and the individual bits have specific assignments. Like, for example, in the instances I've been using, the top four might be the opcode, the operation code that says what this instruction instructs the computer to do. For example, add the contents of another location in memory, specified by the rest of the bits in the instruction, to an accumulator, called the "accumulator" because it accumulates things. Or store the value of the accumulator into a location in memory, or add or subtract, you know, whatever. So it's just that simple.

And then in our third installment we looked at indirection, which is a very powerful concept. We looked at the power of pointers because in the machine language, at the machine language level, we might sacrifice one of those bits, one of the precious bits in our machine language word, to be an indirection flag. That is to say, if this bit is off, then the address specified by the rest of the bits is the address of the data that we want to load or store or add. But if that indirection bit is a one, then that says that that location contains the address of the location to work from. That is, it's indirect. It's not a direct access, an indirect access. And the other way of saying that is that that location is a pointer to the actual data. So two weeks ago we looked at what that means to have pointers.

So this week I want to introduce the next major advance in the notion of how sort of the fundamentals of how computers work at the hardware level with the introduction of the concept of a stack which uses this notion of indirection in order for it to function. We need to back up a little bit, though, and talk about multiple register machines because that sort of factors into the value of the stack. So far we've talked about a simple machine which had a single accumulator. And many of the early machines did because

the point I was early making was that all of this hardware was so expensive in the beginning. Transistors and resistors and the space they required, the power they consumed, the problem of interconnecting them all meant that the designers had to be extremely lean with the design of their systems. So a register was a luxury to have.

At the same time, it turns out that it's really convenient for a programmer to have more registers, that is, more than one accumulator. Now, remember, though, that we don't get these things for free. That is to say that bits in the instruction word, if we're going to have more than one register, bits have to be consumed in order to specify which register. So the advantage of having one was that you didn't need to specify which one because there was only one. So it was implied, there was this implication that if you're loading, well, there's only one place to load it into, and that's the accumulator. And if you're storing, there's only one place to store it from, which is the accumulator.

So when machines began to have multiple accumulators, which was extremely handy to programmers, some bits of that instruction word needed to be dedicated to specifying which accumulator. So, for example, one of them, one of the early machines at the time was the family of machines from a company called Data General that was an early competitor to DEC, Digital Equipment Corporation, called the Nova minicomputers. The Nova was also one that I programmed back in the day when computers had control panels, you know, front panels with lights and switches. And it had four accumulators, four accumulator registers, which were designated AC0 through AC3.

And so the instructions in the Nova instruction set had two bits to specify which one of the four accumulators the instruction would act on. So there was load accumulator and store accumulator. But now you had four accumulators, not just one. So two bits in the instruction needed to be sort of taken away in order, well, needed to be dedicated to that purpose for specifying which accumulator you meant. Typically that meant that you were taking bits from the rest of the word, like from the addressing range. So you were able then to address less memory with the instruction, but that was made up for by the fact that having multiple registers was so convenient.

So the way the early machines handled subroutines was originally very awkward. The notion of a subroutine, probably our listeners are familiar with the concept, is that you would have some piece of code located somewhere in the computer which performs some useful function. For example, the original PDP-8 didn't have a multiply instruction. You could "add," and you could "and," and you could "shift," but there was no multiply.

So the multiplication process, which is very useful to a computer, you had to simulate a multiplication out of the much smaller instructions. So typically what a programmer would do, and in fact I did this on the code that I wrote for my little PDP-8 front panel toys, I wrote a multiply because the PDP-8 chip that I was using had no multiply built in, but I needed one. So I created a multiply subroutine where it was a series of the available instructions, which had the effect of performing multiplication. Well, if I was only doing that one place in my program, then the program would just - it would execute into this multiplication operation and then continue.

But say that I wanted to be able to multiply from various locations in the program. Well, were it not for having this concept of a subroutine, I would have to duplicate that multiplication code everywhere I wanted to perform a multiplication. But the cool thing about this notion of a subroutine is you could have one instance of it in memory, and you could jump to it from all the places in your program where you want to perform a multiply.

The problem is getting back because, if you're going to jump to this routine from various

locations in memory, different locations in your program, then once the multiply routine is done executing, it needs to know where to go back to. You can't hardwire that back to one location because you're jumping to it from different locations, and you want to return to the instruction right after you have jumped to the subroutine so that you can continue executing where you were.

Leo: Too bad there's no way to kind of store that place that we came from, just temporarily keep it somewhere.

Steve: Well, what the original designers of the PDP-8 did was they kind of came up with what we would now regard as a real kludge. They stored the location of the instruction after the subroutine call in the first word of the subroutine and then started executing from the second word of the subroutine. That meant that the subroutine always sort of knew who had jumped to it, where it had been jumped to from, because that address was in its first word. So there's a perfect example of where indirection comes in. It would, when it was all through doing its work, or in this case this multiply, it would jump indirectly through the first word of the subroutine. Since that first word contained the address that it had come from, that took it back to the place, the instruction underneath the one that had called it. So it was very, I mean, it was elegant from that standpoint.

But it turns out that this approach has a problem. And that is, if the subroutine were doing something more complicated, like calling other code somewhere else, what if that code needed a multiply? Well, then the other code would call the multiply, and its return address, the location from which it called, would overwrite the first call of the multiply. In other words, well, the way to say this is that this was not recursive. You could not nest these calls.

Leo: Yeah, because you'd overwrite the first call when you did the second call.

Steve: Exactly.

Leo: You could never get back to the beginning.

Steve: Exactly. So programmers, if they knew that their code needed to be recursive, that is, if there was a chance that the work they were doing might cause, through any means, might cause them to be executed again, before they returned they would have to go to all kinds of extra work, for example, save that return address at the beginning of the subroutine somewhere else so that, if they got called again, it would be safe against replacement.

Well, what eventually was developed is a brilliant concept called a "stack." A stack is a complete abstraction, that is, we can think of it, and people have, in describing how it worked, have talked about, for example, a stack of plates in a cafeteria. There's a buffering acronym that people may have heard, "last in, first out," or "first in, first out," or "first in, last out" or various combinations of that. What it really is, the way to think of it in concrete terms is there's a register called the "stack pointer."

So remember we have the program counter, which is a register that points to the current instruction that we're executing. Imagine now that this computer has another register

called a stack pointer which is, at the beginning of the program, say that it just points at the very end of memory, like way, way up high at the end of the computer's memory, where nothing is going on. And so essentially, we'll call that all ones, you know, 111111111 is the highest value of the computer's storage. So that value is put into this stack pointer register.

Now, a few instructions are used. For example, in common terminology they would be called "push" and "pop," where the conceptual notion is you push something onto the stack, or you pop something from the stack. What pushing on the stack does is it stores the value of what you're pushing in the location pointed to by the stack pointer, and then subtracts one from the stack pointer. So again, we're using this powerful concept of indirection so that the stack pointer - it's called a stack pointer because it's pointing at a location in memory.

So say that we had something in an accumulator that we just needed to put somewhere safe for a while. We do what's called "pushing it on the stack," which merely says we ask the computer's hardware, the processor hardware, to sort of deal with it. Here, take this. And...

Leo: I like that. Deal with it.

Steve: Deal with it, exactly. So the idea is, the stack pointer, which is pointing at the end of memory in the beginning, we say, here, store this on the stack. Push this on the stack. So that value is copied to where the stack pointer is pointing, at the very last word of memory. And then the stack pointer is decremented by one. Well, that decrementing by one is this concept, the concept of pushing it on the stack because, notice if I say, oh, now deal with this, something new has come along, deal with this. So, again, we push this new thing on the stack. Same stack, but this time the stack pointer that was decremented by one last time, well, it's now pointing to the second-to-the-last word of memory. So we store this second thing that came along in the second-to-the-last word of memory, and so on.

So we're able to say, deal with it, deal with it, deal with it. And we don't really care where these things are being stored because that housekeeping is being done by the hardware. Eventually, though, we say, hey, I need that back. And so that's called "popping the value" from the stack. Popping the value does the reverse. It copies the contents of where the stack pointer is pointing back to wherever we're telling it to go, like we want to load that, we want to - if we're popping the stack into a register, it copies where the stack pointer is pointing, back to our register.

Well, I stepped a little bit on this. I'm sorry. Since we store something, in order to push the stack, we store something and then increment the pointer. If we pop the stack, we first decrement the pointer. So it's now pointing at the top of the stack, and then we copy where that is pointing back to our register. That sequence is important because it's got to be at the exact reverse to pop as what we do when we push. But the advantage is that, again, we told the computer to accept some data in a certain sequence of pushes. When we tell it we want that data back through a series of pops, it returns them in reverse order because the stack is being popped, and the pointer is incrementing back towards what's called the bottom of the - back toward the bottom of the stack, back to the beginning. So we get this data in reverse sequence.

So, okay. There's sort of the fundamentals. Well, now, look at this in terms of subroutines. Because if the computer is set up, as all computers today are, to store that

return address on the stack, then this problem of recursion completely goes away. If our jump-to subroutine instruction, instead of storing the address of the instruction which follows it, and as the PDP-8 did, at the beginning of the subroutine, it simply pushes it on the stack. That's the phrase, "pushes it on the stack." Then we don't have to worry about it. The return address has been "stacked," as again is the way it's referred to. We execute the subroutine.

And the last instruction of the subroutine is called a "return," a "return from subroutine" instruction. And it does the reverse, essentially, of this jump-to subroutine. It removes the last thing that was put on the stack and uses that as the address to return to. So the value of this, the reason this is recursive, is that if this subroutine called some other subroutine and executed some other code, which came back and called the original subroutine, that is, it would stack that address also, then now two return addresses are on the stack, one for each time the subroutine was called. And in fact other subroutines could be called in any order, the only requirement being that they are returned from in the reverse order that they're called, which is normally the way a programmer would design things. You always - you finish up what you're doing, and then you return from the subroutine.

So the beauty of this is that the series of return instructions properly remove the addresses from the stack in the reverse order that they were stored, which is exactly what you want in order to sort of unwind yourself and get back to where you came from. So that technology is now intrinsic to the design of our computers. One of the other things that the stack is useful for is saving the state of the computer. Say that this multiply subroutine needs to use the registers, which were probably being used by the person that called the subroutine. Well, it can't mess them up. It needs to leave them alone.

So one of the things that this multiply subroutine could do is save those registers which it will be altering, but which it wants to be able to restore, on the stack. So at the beginning of the subroutine it'll say, push register zero, push register one, push register two, which stores their contents on the stack. The subroutine doesn't care where the stack is, what the stack pointer is currently pointing to. It just sort of said "deal with it" to the computer. Then it's able to use those registers any way it wants to as scratchpads for its own work. And once it's done, it needs to put things back the way they were so that the routine which called the subroutine doesn't have things messed up.

So it says pop register two, pop register one, pop register zero, remember, in the reverse order that it pushed them. And so, successively, the values that were stored are restored into these registers. Then it says return from the subroutine, and the stack will be pointing at that point at the proper return address, so long as the same number of registers were popped as were pushed because, again, you need to keep these things balanced, which is some of the problems programmers get into if they're working at a low level and are not careful. You do have to be careful because with this power comes, not surprisingly, responsibility.

Leo: A compiler will do it automatically. It's only if you're managing the stack yourself.

Steve: Well, yes. Except that many of the security flaws that we have talked about are a consequence of stack overflow.

Leo: Right.

Steve: We've talked about that often. So it is possible, in a powerful language like C, to allocate buffers. One of the other uses of the stack, and this is where this directly impinges on security, notice that we've been talking about pushing and popping things. Well, say that a subroutine needed some, like a buffer, it needed 256 bytes of buffer space for its own purposes, just sort of like to use for a while, while it's doing stuff. One of the ways a programmer could get buffer space is by manually subtracting, like, 256 bytes from the stack pointer.

So look what that does. Essentially, it's almost like - it's like if you had said push push push push push push push push, 256 times, you would have moved the stack pointer down in memory, coming down from the top. And you end up with this block of memory. So a programmer is able to say - is able to subtract a value from the stack pointer and use that area which is above the stack pointer as their own scratchpad because, if other things, for example, if this subroutine then pushed other values or called other subroutines, those would all be down lower in the stack. So the act of moving the stack pointer down creates a buffer. Sort of it's a nice, very fast and efficient way of allocating some memory that could be used.

The problem is, look what happens if you were to ever write too much data in that buffer. Well, that would overwrite values on the stack which were above the buffer that you allocated. So that if you then popped values, like the register values off the stack, and even more significantly, if you did a subroutine return - remember the subroutine return's address is stored on the stack. So if a bad guy were able to cause a buffer overrun and write over any of the stack, they could put their own return address of the malicious code into the computer, and it would jump there. So that instead of returning where it was supposed to go from the subroutine, it would go into the malware. And that's exactly how much of this malware gains control over the system, just like that, by taking advantage of the stack. Powerful as it is, it does need to be treated with extreme care because it controls the execution path of the processor.

Leo: If we turn - that's one of the reasons to turn on Data Execution Prevention; right? Because if you can't execute code in the data area, doesn't that prevent that kind of attack?

Steve: Yes, exactly. One of the...

Leo: Is the stack considered - the stack's considered data area.

Steve: Yeah. Now, it is the case that some programs will deliberately put executable code on the stack, or they will allocate other data areas and, for example, load some of themselves, like they may be a modular program where only part of the program is loaded in memory at a time. And so at some point it says, oh, I need to go get another chunk of myself. So they'll load themselves another piece of themselves off the hard drive into memory, which technically is data memory, and then execute that. There's nothing preventing that, except it's regarded as poor programming to do that. And so it's the kind of thing that Data Execution Prevention would flag as an exception, which is why some benign software will not run with DEP enabled because it'll cause that problem.

Leo: Good to know. Good to know. And some benign software wants to use the stack as a code area because it gives them some flexibility.

Steve: Exactly.

Leo: It's a kind of programmer's trick.

Steve: Exactly.

Leo: Push code on the stack.

Steve: It's very, very handy and very efficient.

Leo: And very bad. And stop doing it. Well, now, that's stacks and registers. Do you want to get into recursion today, too? So I guess in a way we talked about recursion because recursion required the stack.

Steve: Yes. I'm a little self-conscious about having made everyone's eyes cross here with this stuff.

Leo: No, that was great.

Steve: But it's a crucial concept because what we're going to do in two weeks is discuss how computers are able to do everything at once. That is, one thing we've never talked about, and it's...

Leo: Selecting, yeah.

Steve: ...crucial, is interrupts, this notion of interrupts and the way that sounds can play, the mouse can move, the pointer can move, the hard drive is reading and writing, I mean, there's so much happening at once. And this concept of being able to interrupt code, which we're going to cover in two weeks, absolutely requires the stack in order for no matter what's going on to have its state saved. That is, we talked about one instance, for example, where the subroutine, in my example of multiply, needed to use some of the registers, that there aren't an infinite number of registers. So if it was going to be modifying them, it needed to be able to save and restore them. Well, so does a system where anything could happen at any time.

So I wanted to explain, both from a security standpoint, but also sort of from this fundamental architecture, how vital this concept of a stack is, how cool it is that you're able just to say stack this, stack this, stack this, stack this, and not have any specific concern with where exactly it's been stored.

One of the reasons this is so efficient, remember we talked about the notion of a register being implicit in an instruction, when we're not having to specify the stack pointer because that's implied by the push and pop instructions. They know how to do the work. We just sort of say, we say to the computer, take the data, I don't care where you put it, just as long as you can give it back to me when I tell you that I want it. And the key is that it's this notion of it being in the - we get it in the reverse direction than we gave it allows all of this power and essentially allows this recursion because, as long as we sort of unwind our self in the reverse direction that we wound up, we end up right back where we're supposed to be. So it works.

Leo: That's very cool. Love it. I love talking about the deep roots of computer science. From a guy who was there at the beginning, frankly. Pretty close.

Steve: Well, there are just a couple more key concepts. And we will have really pretty much everything we need in order to understand where we are today. Because I want to quickly, probably the episode after we talk about interrupts, talk about today's microprocessors, the whole CISC versus RISC controversy, the issue of how many registers is enough, how Intel got itself tangled up as power continued to be consumed with the design of its processors, and the insane things that designers have done in order to squeeze every last bit of processing power out of these chips. Now that people, now that our listeners have a sense for exactly how these things run, there's enough understanding to understand very cool technology like out-of-order execution and superscalar and pipelining and the stuff that we've all got in today's processors, not just the way PDP-8s used to work.

Leo: Well, thank you, Steve. It's always fascinating. And we'll get more into the details of the nitty-gritty. Now, next week, though, we answer your questions. It's our Q&A episode. So if you've got a question, about this or anything in the security world, go to GRC.com/feedback, and that's where you can ask questions of Steve. And we'll pick 10 or so for next week - he will.

By the way, while you're at GRC.com, don't forget to check out SpinRite, the world's best hard drive recovery and maintenance utility, a must-have. If you've got a hard drive, you need SpinRite. He also has a lot of good free stuff there - Shoot The Messenger, DCOMbobulator, Wizmo, ShieldsUP!, I could go on and on. GRC.com. And you will find 64KB and 16KB versions of this file, the podcast that you're listening to there, as well as the show notes and transcripts, too. It's all at GRC.com.

Steve, thank you for joining us. We'll see you next time on Security Now!.

Steve: Thanks, Leo.