Indirection: The Power of Pointers

Description: A feature present in the earliest commercial computers, known as "indirection," has proven to be necessary, powerful, beneficial - and amazingly dangerous and difficult for programmers to "get right." This week, Steve and Leo examine the Power of Pointers and why, even after all these years, they continue to bedevil programmers of all ages.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-237.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-237-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 237 for February 25, 2010: The Power of Pointers.

It's time for Security Now!, the show that covers everything you would like to know about security online and privacy. And of course no better person to do this than our security guru himself, Mr. Steve Gibson of the Gibson Research Corporation, that's GRC.com. Steve is here to save us from ourselves and the bad guys out there.

Steve Gibson: Oh, and those bad guys are industrious, Leo.

Leo: Oh, they are.

Steve: They really are.

Leo: I did a radio interview late last night on Coast to Coast, you know, the overnight show with George Noory.

Steve: Oh, yeah.

Leo: And, you know, they like to do conspiracy theory, alien abductions, that kind of thing. And they call - and they always call, and it's funny because they always kind of, like I think all mainstream media, grab onto kind of the wrong thing. Instead of,

like, the fact that somebody could hijack your browser, you know, they're "I hear that GPS has been hacked." And it's like, yeah, well, okay. If you have thousands of dollars and the right, you know, jamming equipment, I guess you could hack GPS. But apparently, theoretically they figured out a way to either, I mean, jamming's easy. But what they really are trying to do is grab GPS signals and reroute trucks or boats to where the bad guys are. Oh, you think you're heading to the depot. No, you're going to a dead end in the middle of nowhere.

Steve: Wow. Well, that would be some serious technology.

Leo: Exactly.

Steve: I mean, it's amazing to me that GPS works at all. And of course it's based on phenomenally precise timing of signals from multiple satellites. And so in order to deliberately mess with one particular target's belief of location, I mean, that's some serious voodoo.

Leo: That's what I said. I mean, you could jam it, that's easy. And there's easy ways to do that because it's a very weak radio signal. But to actually reprogram it, hmm.

Steve: Yeah.

Leo: Anyway, what are we talking about today?

Steve: Today the title of this podcast is "Indirection: The Power of Pointers."

Leo: [Laughing]

Steve: And we're going to talk about how they existed from the dawn of computing, and they have never stopped being a problem.

Leo: Wow.

Steve: Because they're really good and powerful and important; but, boy, they're just a constant source of trouble. And so it does tie in, I mean, this is basically sort of my continuing this thread of how computers work. And we're going to sort of start from there. But of course pointers being mishandled is a serious security problem. It's at the root of many of the problems that we talk about every day. And we've got, of course, some problems to talk about, new problems.

Leo: Isn't it interesting that, as we kind of develop from first principles computing,

that so early on we've got to the problems that can cause security flaws? I mean, it's only, you know, just a step up from machine language, and already we can see where the problems lie.

Steve: Yeah, it's not at all that computers were ever better at doing what they're doing. It's that when we started connecting them together, suddenly you no longer needed sneakernet in order for viruses to jump onto floppy disks and wait to be transported to some other machine. Now they've got the world, literally, is available to them. So...

Leo: The world is your oyster.

Steve: Yes. So last week we talked about Adobe and problems with Flash. Well, Adobe is still in the doghouse. They have just released an out-of-cycle patch. So how is that quarterly update going for you?

Leo: [Laughing] They haven't done a quarterly update yet.

Steve: No. They've got such problems. I don't know what they were thinking. Oh, we're going to hold this and wait, do it only every three months. It's like, good luck with that. So I wanted to let everyone know that there has been, since they last heard this podcast, a point update to Reader, both Reader and Acrobat. The 8-point series went from 8.2 to 8.2.1. The 9, we were at 9.3, now we're at 9.3.1. And so anyone using Reader ought to open up a PDF and then have it check for updates, and you'll find some.

There were two critical vulnerabilities. And turns out one is actually the same problem that they had in Flash. So there may have been some code sharing going on, and they realized, oh, shoot, we've got to fix Reader and Acrobat, too, because that's got the same problem. So what they fixed last week in Flash, they also have now fixed this week in Reader and Acrobat. So just time to get that updated.

Google Chrome has been updated. Anything prior to 4.0.249.89 is a problem. So that's the current version, ending in 249.89. And it's important to update if you are a Chrome user. Now, we know that Google is not abandoning Chrome. Chrome I think has a little, somewhere between 5 and 6 percent of the market right now. So it's there. It's number four in line in terms of browser share. And Google certainly remains committed to it. The sort of the scary thing about vulnerability disclosures in Chrome is that, because it's open source, full details are publicly available via source code analysis. So one problem was a vulnerability created by errors in their handling of DNS and the way proxy lists are interpreted, which could lead to disclosure of sensitive data.

Another vulnerability resulted from an integer overflow in the V8 engine, which is their scripting engine. A third vulnerability is caused by an error in the way Ruby tags were being processed. There's a vulnerability number four involves the way href inside of iFrames, those are inline html frames, were processed that could lead to disclosure of the redirection targets. There's an error in the password manager which incorrectly prefills the HTTP authentication dialogue of one domain with the credentials for another, which could be exploited for phishing.

And finally there was an integer overflow in the way sandboxed messages are

deserialized that could lead to remote code execution. So a handful of your typical problems that they have addressed in a series of updates. And so anyone using Chrome should address that.

And we don't often talk about OpenOffice. But it has just been updated, so - and it's very popular. It's, like, the leading big suite over on non-Windows platforms, Linux, UNIX and so forth. And it is anything prior to 3.2 has a host of problems which have been identified. There's a series of security announcements over on OpenOffice.org. And so anyone who is an active OpenOffice user should just go to OpenOffice.org and get themselves updated to 3.2. Not all language variants are yet available in 3.2. English is, and they're beginning to get recompiled and brought online. So you'll want to make sure that yours, if you're a non-English speaker, or you're using one of those, is available.

And these are things we've actually talked about in other contexts. There's, like, GIF image exploits and remote execution kind of things. And it would be a mistake to think, oh, well, not that many people are using OpenOffice, so the bad guys are probably not focusing on it. What we're seeing increasingly, and we saw this with the Aurora, the so-called Aurora attacks against Google and the 30 other companies, is that sophisticated hackers are getting very smart about targeting their attacks. So, for example, in the case of Google, they were able to compromise one person's machine, and but that wasn't the person who had the kind of privileges they wanted or needed. So they had that person's machine send email to other people within Google who did have the access privileges on their machines. And that allowed the malware to jump from, you know, inside of Google from a less privileged user to a more privileged user, moving the malware toward their target.

So, I mean, that's a very sophisticated attack. And so you can imagine that, if somebody were, like if an enterprise were using OpenOffice.org throughout their enterprise, it would be possible to see that by looking at the nature of the documents that this organization produces. Well, that flags them as OpenOffice. Then the bad guys go look to see if there are any bad problems with OpenOffice that could be exploited. They'd say, oh, look, there's a problem with GIF images. So they would start sending people in that organization custom exploits for problems they know the software the organization is using, I mean, this is really happening these days.

So this is sort of the evolving nature of exploitation of vulnerabilities. It's no longer just being sprayed out onto the Internet. I mean, that's going on, too. But enterprises being targeted, as Google and these other companies we know now were, are looking closely at the software these enterprises are using and then turning around and looking for vulnerabilities. So the fact that you're not using something that's super popular really doesn't mean that you're not vulnerable to that kind of attack.

Leo: That's kind of surprising because, I mean, one of the things that Mac aficionados often say is, well, we're not a popular platform, so we're less likely to attack. So if they start attacking things like OpenOffice, that's a much smaller subset than the Mac universe.

Steve: Exactly. Now, my little bit of "I can't believe this happened but it's true" news is that irate parents claim and have sued and have proof that a Pennsylvania high school district has been spying on its students at home, using school-issued MacBooks with a webcam.

Leo: Yup.

Steve: Their security software, that was installed without disclosure in these machines. And so the idea was that, if the machines were lost or stolen, the security software would be used in order to help the district recover the lost or stolen machines. The problem is that, for reasons that aren't clear, the district got caught really misbehaving. According to the original complaint, there's a mom and dad whose last name is Robbins who accused a Harriton High School assistant principal of - oh, I'm sorry, was accused by a Harriton High School assistant principal of "improper behavior" in the student's home. The assistant principal showed the student a photograph taken of him at home through his laptop's webcam.

Leo: Oh, god. Oh, boy. That's not good.

Steve: No, it's really not good. So the school superintendent, a guy named Christopher McGinley, said, "There was no explicit notification that the laptop contained security software. This notice should have been given, and we regret that it was not done." So these guys are in hot water. It's interesting, there's a guy who bought a little, a subnotebook, a little Lenovo, in fact, who's a regular in the morning at Starbucks. And he bought it actually on my recommendation. And it has a webcam up at the top of the screen, you know, where they typically go in a laptop.

And the first thing he did was cover it up with a post-it note because he just, you know, I told him that it was very possible, and it had happened, that webcams were used by bad guys for spying. And so he just covered it over with a post-it note. He actually uses it as little crib notes for some of his function keys. And I got a kick out of the fact one of the articles that covered this story of the Pennsylvania high school talked about word had spread that this was going on, and the students were all covering up the webcams with post-it notes.

Leo: You know, it's really worse than even the story. They admit, for instance, 42 cases of doing this.

Steve: Yup.

Leo: The FBI is now involved. And then yesterday I read a blog post that pointed to a YouTube video where the IT guy from the school district was boasting about the software they use, which is called LANrev. And, frankly, the way he was talking, the thing that he liked most about it was that he could hide it, he could cloak it from the users that it was running.

Steve: It had rootkit technology so that it would be installed, and no one would know.

Leo: It's stunning. I mean, the implications of child porn, even, of, I mean, this is - the repercussions are going to be felt far and wide.

Steve: Yeah, exactly, the idea that there was some policy that had the school arbitrarily looking out of people's webcams. The parents of this student who was shown the photo of himself at home said that he was using a laptop that was duly and formally and properly authorized by the school. They took it home. It was never reported missing or stolen. There was absolutely no reason for someone at the school to activate that machine's camera and be looking out to see what he could see. Just creepy.

Leo: It couldn't be any worse. I mean, just couldn't be any worse.

Steve: It's creepy.

Leo: I'm giving a seminar in a couple of weeks on - it's called "Wired Family, Safe Kids." It's about keeping kids online, but keeping them protected and privacy and so forth. And all schools are doing - and by the way, this is for my kids' school - and all people like me are doing is saying, well, kids, you have to protect your privacy, don't reveal this stuff on Facebook. And then this comes along, and it's the same people who were saying don't put your stuff on Facebook were spying on you. It's, well, if there's any silver lining to this cloud, it's that people will now be much more thoughtful about this whole thing. I mean...

Steve: Well, yes. And I think, for example, that's the same upshot that will result from the trouble Google got in with the way they had Buzz originally configured.

Leo: Right.

Steve: I'm sorry for Google's, you know, the stain they've got, and I'm sure they'll recover from it. But the fact that it was Google and so high-profile and generated so much "buzz," it will help other people not make that mistake. I mean, these things have to be opt-in, not opt-out. And how many times in the last four and a half years have you and I talked about the whole - in fact, that was the name of the first antispyware utility written, which I wrote, was called OptOut because it was like, this is not okay to do, and tell people, okay, yes, I want to turn it off. It ought to be off, and you have to turn it on by default. I mean, on if it's what you want. And the reason people don't have things that way is they are trying to get some leverage. I mean, Google knew that if all this was turned off by default, then all this miraculous social networking glue that would all knit everything together wouldn't just happen.

Leo: Right.

Steve: It would be a much slower start than if it just - you added Buzz, and suddenly all your contacts were finding each other, and everything was just like, okay, wait, what are the consequences of this? People just weren't informed.

Leo: It's good. I mean, in the long run it's all good. It raises people's awareness of all of this stuff. And I love Buzz. I use Buzz all the time. And it didn't bother me. And

Buzz has changed how it works, so it's opt-in, not opt-out. But it is a black mark, it really is.

Steve: Yeah. And so again, I'm glad - I'm not glad that Google got hurt by it. But this is a perfect example for the world to see, just like this webcam in the laptops is like, whoa, I mean, I'm glad it's getting some attention because this is just not okay. And I'd like to see something like we have on cameras now where it's clear that the webcam is on, like little lights on either side of the camera. So they're turned on if the camera is on, just so you know that it's happening, so that you have some feedback.

Leo: Yeah.

Steve: Speaking of which, speaking of feedback, I have a fun note about SpinRite. The subject was "SpinRite Takes a Vacation." And I thought, SpinRite takes a vacation? Okay. Well, this is from a listener, Security Now! feedback. His name is Jared Shockley. He's in Bellevue, Washington. And he says, "Hi, Steve and Leo. I purchased a copy of SpinRite just over a year and a half ago when one of the hard drives in my web server started puking. I always heard you both talking about it, but finally got a copy. Ironically, it didn't fully repair the hard drive, but it found a huge error/failure in my RAID controller. I upgraded the hardware, and all was well.

"Now onto the story at hand. I got a new 500-gig hard drive for Christmas last year for both my girlfriend's and my laptops. I did not have a spare machine I could install it in to run a preinstall check of the drive with SpinRite. My laptop was so happy to have the spare space and faster drive. However, I noticed that Windows Home Server's backup was failing due to an error. I checked into the errors, and it was a drive error. Checking into all of my logs, there was a problem with my new drive. I was crushed.

"On top of this, I found it the day before my girlfriend and I were supposed to go to the Washington Coast for a mini vacation, including lots of digital photography. I hadn't installed my girlfriend's drive. So I thought I could run SpinRite on the current drive and then clone to her new drive. So I started SpinRite on Level 4 at 6:00 p.m. the night before we were supposed to leave. The next day we both were getting everything packed to go at around 10:00 a.m., and I looked at my laptop. SpinRite was only 42 percent of the way through and had been there since we woke up at 7:00," which meant it had hit some bad spots on the drive and of course was in deep recovery mode.

He says, "I was crushed. I didn't want to stop SpinRite, but I needed the machine to go with us. Suddenly I got a wild hair. I have a universal power adapter for my laptop with a car plug. The only thing I was concerned about was keeping the system cool. I figured out how to arrange the laptop in the back seat of the car with the power supply and a USB fan blowing on it. We drove all the way out to the coast..."

Leo: What?

Steve: "...and checked into our lodging. The whole time the laptop was running on the car power, so it wasn't on its own battery. It was about 50 percent done when we got there, and I then relocated it into the room. One more night of work, and SpinRite was done at 10:00 a.m. the next day. After 40 hours of Level 4, with lots of data recovery, it

had found nine bad sectors that it could not recover, but 15 that it could. I performed the clone to the other 500-gig drive without any issue. Immediately, after the second clone, I ran SpinRite on Level 2, and it loved the drive because it had fixed all the bad sectors."

Leo: Of course.

Steve: "Since I still have the original drive, I can recover any damaged files. The vacation was a wonderful time. The little I needed to do, I was able to do on my Triple E PC while SpinRite saved the day. As I just took a new job as director of IT for a company, one of my first purchases is enough licensing for SpinRite to have a site license. Thank you for a great product, Greg for awesome tech support, Sue for incredible customer purchasing support, and Leo for putting everything on the Internet. Sincerely, Jared Shockley.

Leo: That's nice. That's really nice, yeah.

Steve: Great note, yeah.

Leo: And deserved, Steve.

Steve: Well, I've paid my dues.

Leo: So there's a word in programming, "indirection," that is one of the fundamental programming concepts. And I remember very well when I first started studying C. I mean, I remember PEEK and POKE from Basic, which is essentially the same idea. But when I first started studying C, that was the hardest thing to get was indirection.

Steve: Yes.

Leo: And that's what you're going to teach us.

Steve: We're going to talk about it. It is - it's something I didn't want to get into week before last when we laid out the fundamental architecture for how simple a computer is because it begins to add a little more complexity. But it existed back then on a PDP-8, on the PDP-11, the early Data General Nova machines. It's always been there. And it's a very powerful concept, but the power is the problem. So we're going to cover the whole issue of indirection and pointers.

Okay, so if we turn back two weeks to where we talked about machine language, we'll remember and recall from that that we have a bunch of memory, which is organized in words, each word containing a set of bits; and that a program counter is used to address a particular word in main memory, which it reads from memory. The bits then determine what the machine does in that step. And everything is done sort of step by step.

So, for example, in the machine we sort of - the virtual machine we designed two weeks

ago, the upper four bits are opcode, and that would give us one of 16 different possible operations. And so, for example, if it was 0000, if those first, the left-most four bits were all zeroes, that might be the add instruction. And the balance of the bits in the word would be sort of, where the opcode is the verb, the balance of the bits would be the noun, that is, add what? That is to say, add the contents of a certain location, where those bits in the word would specify the address. Or we might load from a location, or store to a location, or AND the contents of the accumulator, which is sort of our scratch pad storage, with a certain location.

So once doing that, the program counter would increment to the next word in memory and do whatever that said, whatever the opcode in that word specified. And so, if you really think about it, it's a script. This program is a script of step-by-step instructions which the computer executes. And it gets a little more complicated because it's able to step out of sequence using skip and jump instructions to go somewhere else. So there's our computer.

Now, imagine a problem, as the designers of this early computer and all early computers did, where for example we have a document that's living in the computer's memory, and we want to search it for a certain word, which, you know, and we use FIND in our word processors all the time, the idea being that the computer needs to scan down through memory, through this document, to find what we've told it we want it to locate. So with this simple computer that we've got, how do we specify a succession of different addresses in memory? That is, the word contains the address we want to load, but it just contains that one address. It doesn't, like, how do we scan through memory?

Well, if we only had what we've described so far, there would be two ways to do this. You could have individual instructions, one after the other, that loaded successive locations in memory into the accumulator to see whether it had what we were looking for, that is, an instruction would be "load location 100," and then it would check to see, then it would be "load location 101" and would check to see, and "load location 102." Well, obviously that's a hugely long program because you're needing several instructions in order to check each location in memory. So that's arduous.

Now, another approach, the other approach would be something that is generally frowned on, and that is self-modifying code. That is to say, since the instruction itself is in memory, and for example it said "load location 100," then the program could change the actual data for that instruction from 100 to 101 and then load it, see if we found it. If not, then increment that location, the actual specified in the program, to 102. So the problem is that it requires that the program is modifying itself, which becomes messy pretty quickly. So what the original architects of these early machines decided is instead of the instruction, like said load 100, instead of that instruction specifying what to load, the instruction would have an option of specifying the location that contains the address of what to load.

Okay, so now - and we have to be careful, even like the way we talk about this, because it's amazing how easy it is to get tangled up. But in these early instruction sets, as I talked about it so far, we had for example a four-bit opcode, and the rest of the bits specified what the opcode acted on, what address in memory was loaded or stored or added or whatever. These early computers used one more bit, that is, so there was an opcode of four bits, and then for example another bit right next to it called the "indirection bit." And then the rest of the bits that were remaining specified the location. That is to say that the designers of these machines took one more bit for this purpose from the instruction.

So what this meant was, if it was a so-called, for example, an indirect load, if it said

"load indirect 100," what that meant was the computer would get the contents of location 100 and treat the contents as the address to load the data. In other words, that the location, the contents of location 100 was a pointer to the data that should be loaded. And that is an incredibly powerful concept. That is...

Leo: It seems so simple.

Steve: Well, yes. And the reason it, I mean, it is simple, and it was even simple to do in hardware. I mean, all they had to do was they were going to load the contents of 100 anyway, so they did. They loaded the contents of location 100, for example. So the question is, do you use what you just loaded, or do you treat it as the pointer to what you want to load? And that's - so the logic in the computer was, I mean, it was inexpensive for them to implement this. And they got something so powerful as a result.

So if we return to our simple example of searching memory, all we need to do now is the program refers to location 100, but we're using the value of that as the address of the data that we're going to load. So we simply increment that location. And in fact the early machines, like the PDP-8 and the PDP-11 and even the Data General Novas that was another machine of that time, they had what was called "auto-incrementing memory," that is, auto-incrementing locations, literally a reserved block of memory, typically down at the beginning of memory. In the case of the Data General Nova it was location, I think it was 78 - I'm trying to think of which location. I don't remember now. I think it might have been octal 10 through octal 17.

Leo: It's so funny that you remember it.

Steve: So it was the eighth through the 15th locations. And the way that worked was, if ever you referred to those locations indirectly, the computer itself would increment the value for you. And what was really neat - remember we talked two weeks ago about this notion of memory which destroys its contents when you read it, like core memory, which is what was used back then - in order to read the contents of the memory, you needed to destroy what was there. Essentially you wrote all zeroes into the memory word. And the inductive pulse caused by the core's switching from one to zero is what let the computer know what had been stored there.

But in the process you wrote zeroes. So it was necessary to have a second memory cycle to write back what you had just destroyed. Ah, but in the case of auto-incrementing, you wanted to write back one greater. So what was so clever is that you sort of got this auto increment, or auto decrement, for free. That is, it sort of folded it right into the recovery of the prior contents of the core memory so that, again, very simple logic to just increment the value by one. We talked about that last week in one of our Q&A questions about how do you increment something. And it's very simple logic to do.

So now, with this very bare-bones instruction set, we're able to easily scan through as much memory as we want to. We simply say, instead of using location 100, for example, on a PDP-8 or even in the early Data General Nova, the Nova also had auto-decrement, a block of memory. And when you referred to it indirectly, the computer would decrement so that you're able to sort of scan from high memory down, as opposed to low memory up.

And so in the case of our little project here to locate something in memory, we would

establish the beginning of the buffer that we want to scan. We would put its address into, say, location octal 10. Then we would say "load indirect 10." So we're not loading the contents of 10. The computer reads the contents of location 10 and increments it and puts one more than that back in 10. Then it uses the value that it read from location 10 as the address containing the data to be loaded. And so our program can be very small, very efficient. And every time it does this load indirect octal 10, it gets - what actually is loaded is a word somewhere else in memory, and it's the successively next word every time we do this. So again, very simple, tiny steps, but very powerful.

Now, what we've done is to empower the programmer with a concept which is needed for the sake of programming efficiency. But it's tricky because even today we're talking about security problems all the time that contemporary software has. And Leo, you were talking about you've done programming.

Leo: Yeah.

Steve: And, for example, in C, pointers are used...

Leo: It's built into the language. It's...

Steve: Well, exactly, it's intrinsic property of the language. And in fact pointers have historically caused so much problem that there are languages that boast they don't have them. Because it's like, oh, if you don't have this, you can't get yourself in trouble. And what happens is, programmers can easily make the mistake of whether they are referring to something or they're referring to where that something points to. And it's funny, I think the problem is there isn't a good analogy in life, that is, we're used to seeing something, and you reach out and grab it. And there's, you know, there's no indirection most of the time.

And so I don't think mentally we humans model something as abstract as a pointer. I mean, we understand intellectually what it is. But in the years I've been programming, I'm always having to be very careful. And programmers who have used pointers extensively know they have to be very careful to make sure that there isn't a gap between what they mean and what they tell the computer. Because the computer, as we know, is very literal. It'll do exactly what you tell it. So one of the, for example, in C or any of these pointer-based languages, you need to be able to get the address of an object as opposed to the contents of the object. And if you think about it, if you had a language, say like Basic, the Basic language, until you had, for example, PEEK and POKE, as you were referring to, Leo...

Leo: Yeah, yeah. Which is - that's indirection in a way, right, because you can...

Steve: Oh, it absolutely is. Yeah. If you just have the Basic language, where you say A equals 1, B equals 2, C equals A plus B, you cannot get yourself in trouble. I mean, there's no notion of pointing to something else. You know, A and B are variables. The language takes care of that for you. If you say C equals A plus B, then, again, the compiler is completely hiding that.

Leo: Right.

Steve: But as soon as you say A equals where B is pointing to, now, I mean, you have let the genie out of the bottle because, as a pointer, that B, where B is pointing to, it can point to anything. I mean, it could point to outer space. It can point to the operating system. It can point, I mean, to data structures inside the program. I mean, suddenly there is an awesome amount of responsibility that comes with that power. And frankly, it's one of the things that makes C, that allows people to regard C as a relatively low-level language. It was designed from the beginning to be a language close to the machine in which you could implement system-level software. You know, UNIX was written in C.

Leo: It's huge, yeah. Huge.

Steve: Yeah. And so it is a - it's an intrinsic of machine language. It's always been there. One of the variations as we evolved is the notion of what was called "index registers." You could - or "indexing," which is just another way of saying the same thing, where you could, in some of the early machines that had, for example, like the Data General Nova had four accumulators, AC0, 1, 2, and 3. And the last two accumulators, AC2 and 3, could be treated as so-called index registers, which is exactly what we're talking about. We're saying that they contain the address of the target location rather than their contents being used directly. And index registers are a component, and indirection is a component of all contemporary machines today. They come in different shapes and sizes and additional complexity. But this basic structure has been around from the beginning and is really powerful.

Leo: Indirection. Next, recursion. I mean, I tell you, pointers was hard for me, even though I'd had, as I said, had experience with PEEK and POKE. The little caret and little ampersand in C, it was like, I use what, when? But once you get it, it is so powerful. And it's really not that hard. You just did a great job in 15 minutes of explaining it.

Steve: Well, it's not hard. What I think, though, is it is mistake prone.

Leo: Ah, well, okay. Now the security issues arise, yeah.

Steve: Exactly. Because this is what we see. And in fact if you - remember, one of the things that the bad guys do is they are able to confuse data and instructions. The bad guys, when we talk about remote code execution exploits, the idea is that data in a GIF image or in a buffer that is moved across the Internet, it is not supposed to be executable. But because of the power of pointers, literally for this reason, because of the power of pointers, it is possible for data to get confused with code and for the bad guys to leverage this power to get the data that they provided to be executed as code. And at that point all bets are off.

Leo: Yeah. That's why you have this feature in Windows where you can't execute code out of the data stack.

Steve: Right, DEP, Data Execution Protection.

Leo: Right.

Steve: The idea is that there are regions of memory which a programmer can, when they're setting things up, they're able to say, okay, I intend for this buffer to be data, not executable. And that was a feature added relatively recently to, in the case of Intel, to the Intel architecture so that blocks of memory that were being allocated by the operating system could be marked as read-only, writeable, or executable. Or not, in the case of leaving this bit off. So that literally, if your program attempted to jump, that is, again, we have a program counter in today's processors, just like we did back then.

So if your program counter attempted to be set to the address of an address inside this block of memory, there are gates in the chip which check the privilege bits associated with this allocation of memory and say, oh, wait a minute, the execute bit is off on this block of memory. We cannot execute from this memory. Therefore the program counter is not allowed to fetch from that. And what that does is it pulls an exception, essentially, a violation deliberately that returns control to the operating system, saying whoops, this program just tried to do something it should not try to do. We don't know why. We don't know that it's a bad guy. It could just be a mistake in the code. But...

Leo: Or it could be intentional. Programmers do this all the time. They stick program code on the stack which is, as we now know, bad.

Steve: Yes. And in fact Windows depended upon that in the old days. Back before hardware graphics acceleration, where you were wanting to move rectangles of data around from one place to another on the screen, it was too slow if you had a general purpose chunk of code that would say move this many bits, and then is counting down the bits as you move them, and then goes back over and does another line, and so it's like that does line by line in a raster scan order. The problem was that was just too slow.

So what Windows did was, when you said I want to move a rectangle of data from one place on the screen somewhere else, it actually wrote custom code on the stack in order to do just that one operation one time, much faster than you could execute a much more general piece of code to do that. And then it would just discard it. And in fact we're going to be talking about what is a stack week after next because it's one of the next...

Leo: Oh, good. Oh, yay, yay, yay.

Steve: ...the next evolution of fundamental technology that - and actually the early machines did not have a stack. The machine we've been talking about, our little hypothetical machine, there was no stack. But the introduction of that concept was another sort of crucial, critical addition to the way computers work that it was so good, no one would do one without them these days.

Leo: Yeah, yeah. It's these little abstractions that advance computer science in big leaps. And it's wonderful when you get it because your brain and your understanding of how this stuff works advances in a big leap, too. You really feel it. You go, I get it, pointers. Or, I get it, stacks.

Steve: And the abstraction is fun.

Leo: Oh, I love it.

Steve: I mean, it's fun to, well, in fact it is, I think that's one of the hooks for people who love to program is that they just - they get off on this kind of true abstract thinking. It's just great.

Leo: Absolutely. That's where the art and the joy of programming comes in. Steve, you're the best. Steve Gibson is the man in charge at the Gibson Research Corporation, GRC.com. You can find SpinRite there. That's his bread and butter, his day job, the world's best hard drive maintenance and recovery utility, a must-have if you've got a hard drive. GRC.com.

You'll also find the show there, including 16KB versions which Steve creates himself, edits with his own little razor blade and tape, and puts online so that those of you with low bandwidth, even you can hear the show. And transcriptions, which Steve pays for himself. He's a very generous guy. We don't do any of this stuff for him. He does it all on his own. Show notes and more. GRC.com. If you've got feedback, next week is a feedback, a Q&A episode. GRC.com/feedback is a great place to go to leave a question or a comment or a suggestion.

Steve: Yup, please do.

Leo: Yeah. And I think that - oh, oh, I forgot, there's so much great free stuff there, too. Don't worry, even if you don't have a dime to your name you can get great free stuff at GRC.com. Steve, thank you for explaining pointers so succinctly and wonderfully. I wish I had this when I was starting out.

Steve: They are very, very powerful. They come from indirection. And like these other core technologies, they'll always be with us. And so we'll do stacks in two weeks.

Leo: Thank you, Steve. We'll see you next time on Security Now!.

Steve: Thanks, Leo.