SECURITY NOW!

Transcript of Episode #233

Let's Design a Computer

Description: To understand the advances made during 50 years of computer evolution, we need to understand computers 50 years ago. In this first installment of a new Security Now! series, we design a 50-year-old computer. In future weeks, we will trace the factors that shaped their design during the four decades that followed.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-233.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-233-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 233 for January 28, 2010: Let's Design a Computer.

It's time for Security Now!, the show that covers everything you need to know about protecting yourself online. And here to get protected, our guru of security, Mr. Steve Gibson. He's the man in charge of the Gibson Research Corporation, GRC.com, the folks who created SpinRite, the world's best hard drive maintenance and recovery utility. But also Steve, in the process of running his own server, discovered the first spyware, coined the name "spyware," and in fact wrote the first antispyware tool. He still writes a lot of free, useful security products for us all. Hello, Steve. Great to see you.

Steve Gibson: Hey, Leo. Great to be with you again, as always every week.

Leo: We've got a fun one today.

Steve: Well, we have a fun series, actually. One of the things that I think this podcast excels at, if I may quote the feedback that we get from our own inbound mail from our listeners, is explaining stuff in a clear way that lets people come away from it thinking, wow, I understood that. I mean, we've tackled crypto in great depth, ciphers, and all aspects of security. And there's a ton of technology that we're all carrying around with us in our laptop or sitting on our desktop, the modern computer. And we got so much feedback from the old-time computing podcast that we did...

Leo: Oh, yeah.

Steve: ...that I thought, you know, we really do a good job of demystifying stuff. Let's tackle the big one, how a computer works.

Leo: Wow. That is the big one, isn't it.

Steve: Yeah, the big one. I mean, it's something, I mean, we're certainly going to keep everyone up to speed, as we always do, with security news. So we'll always have what's happening in the security news, and we'll interrupt this series if necessary for anything big. But there's - on modern machines we have this whole issue of, you know, people have heard the terms RISC and CISC, reduced instruction set and complex instruction set. There was the attempt of the PowerPC that sort of failed to Intel. We now have multiple cores. We've got superscalar execution and L2 and L1 caching. And all of these things have been done over time to solve problems.

And as I sort of sat down to map out how I wanted to explain why these things were done, I kept moving back further and further into, well, okay, to understand that, we need to understand this. And to really understand this, we need to understand that. And I worked myself all the way back to transistors.

Leo: First principles.

Steve: Yeah. And so what I want to do is I want to start with where this all began, back at the beginning, and take our listeners on a journey from literally where computing was 50 years ago, in 1960. One of the things that I really understood really clearly when I was reading the history of all this over the last year, sort of following my interest in the PDP-8s, was I developed a much greater sense for how expensive the simplest things were. And then I realized, in order to explain that, I needed to explain what it was that they were trying to do and what tools they had at the time.

And so what I want to do is I want to start at the beginning and create a foundation. And then, rather rapidly, because we don't want to take - we don't want to take a decade to explain five decades. But sort of like then zoom forward over the next several podcasts, looking at how things changed, what happened, what problems were encountered, and what solutions were created to solve them. And we'll end up, I think, some number of weeks from now, I'm not sure how many, but with our listeners understanding where this all came from to a depth they, well, that will surprise everyone, I think.

Leo: I think that's a fabulous idea. I think that that's really the only way to really understand what's going on. And we've gotten to a level of complexity now here 30 years into the computer revolution where people don't really - it's just, under the hood, it's just a black box. And it would be nice, I think. Remember we talked a while ago, I thought, if I wanted to teach a programming language class to my kids' high school - I was looking at books and stuff, and I found a really good book for that. And then I thought, well, what if I wanted to teach - and I still haven't done this. But I did teach a class in podcasting, though. And then I thought, what if I wanted to teach a computer science class? And I had the same thought that you did, which is start from first principles.

And somebody sent me a recommendation for a book called "The Elements of

Computing Systems." It was published by MIT Press in 2005 by Noam Nisan and Shimon Schocken, exactly the same idea, building a modern computer from first principles. And it's a really neat book. You can even buy the kit that goes with it, so that as you're teaching this class - and you start with AND gates and logic and stuff, the stuff you would need if you were going to have a multipurpose computer - they can actually start building this thing. And at the end they get a very simple but functional computer that they understand. And I think this is a great idea, Steve. I love this idea.

Steve: Well, that's what we're going to do. It's funny because this, I mean, exactly what you were talking about, sort of it gelled for me because a number of people have written and, you know, looking at my little PDP-8 kit that I put together because I have those videos on GRC now. And they said, yeah, but what can I do with it?

Leo: Right.

Steve: And what I realized was, when I looked at answering that question, and I have a new page on the site that is what you can do with one of these, I realized that there's this magical, I mean, and it almost is magical, I mean, even for someone such as myself who programs in machine language, instruction by instruction, there's this really magical sort of mystical amplification that occurs, by which I mean that such little simple steps can be strung together. And before you know it, you are having a basic compiler or a focal interpreter or, I mean, a functional operating system. It's just there's - I think that's where this all comes from is it's surprising what amplification factor there is in this notion of lots of little steps performed very quickly.

Leo: Yeah, yeah.

Steve: And everyone who's listening to this knows that's sort of the mystique of the computer is, oh, well, they're really dumb. They don't really do anything. But they...

Leo: But they don't do anything very fast [laughing].

Steve: ...don't do anything really fast [laughing].

Leo: I say that sometimes, it's a box of rocks, but it's just a really fast box of rocks.

Steve: Exactly.

Leo: And really it only does multiplication - doesn't even do that. It does addition, subtraction, comparisons, and branching; right? Well, I don't want to steal your thunder.

Steve: Oh, there's no danger of that. I think we're going to take our listeners on a really fun journey.

Leo: I love this. I think this is - I can't wait. I'm very excited. Before we do, though, I imagine there are a few security - in fact, I know there are because I saw some security news this week that kind of shocked me.

Steve: Yeah. Yeah. The most disturbing bit of news is that it came out, through Microsoft's admission, that they knew about this flaw in Internet Explorer which had the out-of-cycle emergency fix last week that we talked about, the same one that was responsible for allowing bad guys into Google, Adobe, Rackspace, and many other companies. They knew about it last August.

Leo: Oh. Oh. Now, okay, okay. And in their defense, maybe it took them since August to fix it, like to figure out how to fix it safely. Yes? No.

Steve: Well, they apparently had no plan to fix it.

Leo: They sat on it.

Steve: I mean, yes. When it became a PR problem, when Germany and France told people not to use Internet Explorer anymore...

Leo: Then they fixed it.

Steve: ...until this was fixed, they had it fixed in a number of days, literally in a number of days, and pushed an out-of-cycle patch. By the way, I heard you and Paul talking about "out of cycle" versus "out of band."

Leo: Out of band. They say "out of band" for some reason.

Steve: You are so correct that "out of band" means, I mean, by definition, a different channel of communication.

Leo: Right.

Steve: And this is not a different channel, this was an out of cycle. But you're right, it's strange that Microsoft says "out of band." It's like, well, okay, well, somebody, I think, is trying to use a fancy term and doesn't quite have their finger on it.

Leo: Out of cycle makes sense, though.

Steve: For most of the world this was a zero-day flaw. But it certainly was not a zero-day flaw for Microsoft, who had this sitting in Internet Explorer for six months. And it wasn't until, I mean, in fact I have to say there's been a great deal of outrage in the security community that - basically, I've read security gurus saying that this is a real breach of trust to their customers that this kind of problem was allowed to sit there. And as long as Microsoft didn't know that it was being taken advantage of, then they were in no hurry to fix it. So that's just really disturbing.

A new problem has surfaced which is really sort of interesting. This one's been around for 17 years, not known about...

Leo: 17 years?

Steve: Since NT 3.1.

Leo: Jeez.

Steve: The very first version of NT had something called the NTVDM, the NT Virtual DOS Machine. This was the environment which Microsoft created to allow 16-bit applications, basically the so-called "DOS box," to function. It turns out that in order to do this, to pull this off, to allow a 16-bit DOS program to operate in a protected-mode environment, what you need to do is you create this virtualization, you create a container which intercepts the software's actions, like calling the so-called BIOS, that really, that no application software has access to when it's running in a protected environment, or even writing to the hardware.

The way DOS applications would put data on the screen is that the screens were what's called "memory mapped," where a chunk of memory was the screen data. And you would write into memory, and a character would appear on the screen. And so there was a dot matrix generator that converted the byte in memory into the physical dot pattern of the character. But the way the software operated was it just literally wrote into memory. Well, you can't - that's another thing you can't just wantonly do in a protected operating system like NT.

Leo: Yeah.

Steve: So there's a system of sort of wrapper around this program which is running, a 16-bit program, which is the so-called DOS box. Microsoft was once again told of a problem that existed in this last summer. It was found by some Google researchers who told Microsoft. And they gave Microsoft time to fix it. Microsoft didn't. And so the researchers recently went public with this problem. It's not a huge problem. In fact, probably only corporate IT needs to worry about this to a great degree because it's a - the consequence of this flaw, which is in all versions of Windows - this affects every version of Windows from the very beginning, 3.1 all the way up through Windows 7. It allows a local privilege escalation, essentially allowing a user who has no admin privileges on the system to acquire them instantly.

Proof-of-concept code exists. Security researchers have tried it; and they've, like, instantly had full kernel system-level privileges, no restrictions at all. Now, in our show notes is a link to Microsoft's security advisory; and also a OneClick, one of Microsoft's little OneClick Fix it buttons. So I don't know how home users, like those listeners of ours who are just using a computer by themselves, I don't think this is a problem for them. Microsoft will surely fix it, probably with February's standard second Tuesday of the month update. I would be surprised if they didn't because it looks like it's not going to be a big deal to fix.

Microsoft Security Advisory:

http://www.microsoft.com/technet/security/advisory/979682.mspx

Microsoft OneClick Fix it:

http://support.microsoft.com/kb/979682

But certainly they've acknowledged it, they're researching it, they understand now that maybe they should have gotten off the stick and fixed this sooner. Because it's not clear that something malicious that arrives in email might not be able to leverage this. So maybe individual users should be concerned. The problem is that the only fix for it is to completely disable the NTVDM, this NT Virtual DOS Machine. That can be disabled with a registry key, using system policies. When you do that, what you're doing is blocking the operation of 16-bit applications.

Now, frankly, I can't function without 16-bit apps. Even today, the editor I'm using, Brief, runs in a DOS box. I edit all my source code in Brief. And it's funny, I was talking to Mark Thompson recently, our friend at AnalogX. Mark has moved to Windows 7 and has really been inconvenienced by the fact that he didn't appreciate how many of the utilities he uses day in and day out are 16-bit apps. So he's been converting, he's been, like, literally converting some old trusted apps that other people wrote, that he had just been using forever. He's rewriting them in 32 bits so that he's able to be fully 32- and 64-bit only because he's still using 16 bits.

So there's the possibility that people might be using 16-bit apps even today without their knowledge. I know I am. So I'm not going to worry about this because I'm making sure that I'm secure. And you'd have to have something evil running in your machine in order to somehow exploit this anyway. On the other hand, I wouldn't be surprised if clever people don't figure out some way to create a blended threat where something takes advantage of this in order to get more privileges than it might already have because, for example, one of the ways our most recent systems are more secure is they're taking advantage, for example, of not always running with admin privileges, running with reduced privileges. So it might be possible, if the NTVDM were enabled, to come up with a way of running some 16-bit code that would invoke the NTVDM, escalate privileges, and then do something behind your back.

So it's not clear that it's not a problem for individuals. But you can imagine that corporate IT, that depends upon having their systems locked down, this is a big master key to any locked down system that has its Virtual DOS Machine enabled. So it may just be that they'll use policies in order to disable this corporate-wide and deal with any specific needs, like for 16-bit code, that arise on the fly.

So we've got links to those things for our listeners. And the OneClick Fix it is probably the easiest thing to do. Microsoft's security advisory talks about - it's pretty complicated. Which is why I like the OneClick Fix it for people who just want to disable this and not worry about it in the meantime.

Microsoft Security Advisory:

http://www.microsoft.com/technet/security/advisory/979682.mspx

Microsoft OneClick Fix it:

http://support.microsoft.com/kb/979682

Leo: Those are just registry fixes, and that just disables it in the registry, I would guess.

Steve: Yeah. Although, as I was looking at it, it was like, okay, for XP you do this. For Windows 7 you do this. It's like, oh, okay. Let's just press a button...

Leo: Yeah.

Steve: ...and let it worry about it. Adobe Shockwave has released an update. And I want to remind people again, this is not Flash. This is not the Flash player but the separate Shockwave player. Anything up through - this is for both Windows and Mac - up through version 11.5.2.602 are vulnerable. And it's necessary to get the newer version. And I realize my notes are wrong here. It's not the same version. Oh, no, 11.5.6.606 is what you can now download. One of the problems is it does not have a simple update solution. Following Adobe's instructions, it's necessary to uninstall the old Shockwave player, reboot your system in order to get all of the old code out of being still in RAM, then download and install the new version.

My feeling is that, if people don't know they're using Shockwave, then they're probably not using Shockwave. This is different than the browser-embedded Flash player, which pretty much everybody is using. So I did want to make a note of that for those people that are using Shockwave player separately. There is a new version. And it's necessary to update because there are some - basically the arbitrary code injection exploits that we're always seeing, once again in an Adobe product.

And then just a brief note. We talked last week about the SSL/TLS secure socket layer renegotiation problem which the IETF has officially come up with a sort of unfortunately a kludge-y workaround for. I wanted to note that in Apple's security update, the first update of the year, which was last week, which they released on the 20th, they disabled SSL/TLS renegotiation to prevent any man-in-the-middle attacks until the final fix was ready. So I thought that was sort of nice. It's something you can live without. Given that you're at risk of exploitation, then, okay, why not just disable that in the interim? And so I imagine that next month or a month from now, whenever this update has been vetted sufficiently and we're, like, we're ready to deploy, then I imagine we'll see that Apple is updating their SSL/TLS renegotiation in order to secure it in a solid fashion.

Leo: What happens if it's disabled? Why don't we need it?

Steve: It's actually not used that often. The idea is that it's always been an option. And in theory, if you had an SSL connection that was really persistent, that is, at the beginning of an SSL connection you negotiate an agreed-upon symmetric key, the key length today is long enough that you could safely use the key for a huge number of packets' worth of traffic. But not forever. And the original designers understood that from, like, every month or so, I mean, we're talking about connections that long, that persistent, that over a long period of time it might be a good idea to be able to retire a symmetric key and renegotiate another one. Because the more data that you expose, the more opportunity there is to see patterns in the encrypted information. So the whole point of this renegotiation is to allow an existing connection to stay up, but for either side to request a renegotiation of the security context. And so the point is, with normal SSL connections having a life of a few minutes, you've just never seen this happen. And so...

Leo: And I suppose there's just some nice way that it'll failover without collapsing.

Steve: Good question. I don't know what it does. Probably one side says I want to renegotiate, and the other side says, I'm sorry...

Leo: Sorry, I don't do that anymore.

Steve: ...[indiscernible] disabled. And the other guy probably says, okay, fine.

Leo: We'll keep using the key.

Steve: Or maybe they drop the - you could also drop the connection.

Leo: Start a new one, yeah.

Steve: And just bring it back up again, exactly. So rather than renegotiating the existing one, you just drop it and reconnect. In which case you've got a first negotiation, rather than a renegotiation.

Leo: Of course. Simple. Simple solution.

Steve: And I did want to just mention, I had a little note here in my errata to remind everybody about my favorite mouse. I only mention it again because I've turned some other friends on to the Logitech Anywhere MX mouse.

Leo: I bought three after you talked about it.

Steve: And?

Leo: I love it.

Steve: Oh, isn't it just perfect?

Leo: And I'm a lefty. Remember we talked a little bit about, well, it's not completely left-right neutral. But it's close enough.

Steve: You're a left-handed mouser.

Leo: Oh, yeah.

Steve: Interesting. Because...

Leo: You're a lefty.

Steve: ...I'm as much a lefty as you are, but I mouse, I've always moused with my right hand.

Leo: Here's a weird one. On a trackpad on a laptop I use my right hand. I don't know why.

Steve: Huh.

Leo: I get kind of ambidextrous there. But, yeah, no, it's slightly righty focused, but only very slightly. And it's a really nice mouse. I do love it.

Steve: Well, my tech support guy, Greg, made a specific point of saying, hey, you know you told me about that mouse? I said, yeah? He said, I love it. And I said, okay. I've just got to remind everybody. I mean, literally, I'm batting a thousand at this point. Every single person I've told about it has said, hey, that's just like the best mouse I've ever used.

Leo: It's a sweet mouse. It has the track, what is that, the scroll wheel with the clutch, which I really like. I love to spin that scroll wheel.

Steve: Yup. It's got like a zero-friction scroll wheel. And so, like, zooming up and down through pages is easy.

Leo: Right.

Steve: The Logitech software that they have available for the Mac and for the PC allows you to reassign the various buttons. What I've done is I use the tilt of the wheel as my browser forward and backward. And then I use the side buttons as top of page, bottom of page. Because...

Leo: Oh, I'll have to do that. That's good.

Steve: It's really nice because I'll often be way down a page, and I just want to jump to the top. So the upper side arrow takes me to the top. And also the bottom of the page is sometimes where I want to be. And so that jumps to either extreme. And then tilting the wheel left and right, you know, to the left takes me - is like hitting the browser's back button, and to the right is like the forward button. And so I just, when I'm - if I'm using a mouse other than that, because I've got some legacy mice around here, it's like, oh, I have just got, you know, to get another mouse and swap it in because it's just perfect, so...

Leo: And the reason they call it an Anywhere Mouse is because it uses a different kind of laser that works on glass and...

Steve: Yeah, actually the higher incidence of reflection laser. The laser points down more, rather than at as much of an angle. And that allows it to pick up texture, as you said, even from glass. I mean, you need no surface whatsoever. It'll mouse over like a window and work just fine. So it really does work anywhere.

Leo: We are going to get to our How to Design a Computer, our topic of the day. I love this so much. Before we do, though, I think you have a SpinRite note, and I have a note from our fine sponsors.

Steve: Well, in my effort to - which actually seems to not be that much effort - to always find something different that a SpinRite user has managed to pull off, I've got - Mihai G. is the name that I have because I can't even begin to pronounce the last name, G-h-e-o-r-g-h-e. So anyway, Mihai G. - and he helped me phonetically with his first name - says: "SpinRite Saves Xbox." And he said, "Steve, a long-time listener of Security Now! and other TWiT shows, my dad purchased SpinRite over three years ago due to a failing drive, which SpinRite repaired.

"Then last night, at a New Year's party, I was playing my Xbox and went to get a drink. I forgot that there were at least 10 kids between the ages of eight and 11 in that room. When I came back, the Xbox was upside down. I asked what happened. They told me they accidentally knocked it over. I panicked. I saw that the disk was scratched beyond any repair." I guess he must mean the CD or DVD that was - the removable disk. He said, "But fortunately, the Xbox itself still seemed to work. I powered it off and turned it back on. It took 15 minutes to boot up, rather than five seconds it usually takes. So I tried it again. I went to get a drink again, and the kids came and told me the Xbox fell over again." And then he says, "facepalm."

Leo: D'oh.

Steve: Oh. He says, "Now the Xbox would hardly even boot up, and would sometimes give me an E86 hard drive error." That's funny they used 86 because that's of course the old restaurant term for, like, when you 86 something, you're getting rid of it? Anyway, E86, hard drive error. "So I turned it off and put it away and said, 'Happy New Year.' This morning," and I notice he wrote on January 2nd, Saturday January 2nd, he said, "This morning," so, what, two days later, "I decided to see if I could salvage the hard drive. I then remembered I had SpinRite. I voided the Xbox warranty and took the laptop drive out of the case and hooked it up to my desktop via a SATA cable. I ran SpinRite on Level 2 and Level 4, and it would not even read data for the first 10 percent of the drive. But when I set SpinRite to start after 10 percent, it went blazingly fast through the rest of the drive. I decided to try the drive back in the Xbox again, and it worked. The Xbox booted up in less than five seconds. I unfortunately lost all my DLC," whatever that is, "but I have since redownloaded all of it. Thank you again, Steve, for a great product. It does fix everything."

Leo: It's probably downloadable content, I would guess.

Steve: Ah, okay. And probably what was happening was, even though he says SpinRite got stuck, well, that's what SpinRite looks like when it's in the middle of doing data recovery.

Leo: Right, right.

Steve: So it was probably busily fixing things, and he didn't, like, let it go long enough. But he let it go long enough that it fixed enough to get the Xbox...

Leo: Oh, that's a really good point. So let it go. When it's struggling, that's when it's getting the data that you want.

Steve: Yes.

Leo: That's when it's kind of recovering the data that's on the damaged sector.

Steve: Yeah, and there's not much feedback it can give you because it's just - it's working with the drive.

Leo: Right, going [imitating drive sound].

Steve: Exactly, and the drive's relocating sectors, and it's recovering things. And then afterwards it says, okay, look, we got some done, let's move on. So I would guess that it was that early phase where it didn't seem to be doing anything that actually that was it working.

Leo: That's kind of the secret sauce of SpinRite is it doesn't give up.

Steve: That's exactly right.

Leo: The operating system after, whatever, 20 tries it says, well, I can't read it. And that's why SpinRite can take sometimes - what was the longest one, months?

Steve: Oh, I get email from people who just have machines sitting on the side. And now it's just sort of like a matter of honor. They just want to see what it'll do. They go, okay.

Leo: But all it has to do is get it one in maybe a million times, but once be able to see that data, and it passes the CRC, and then it says, okay, I got it, and move it, and then mark that sector bad, and you've recovered that data.

Steve: Exactly. And SpinRite does a whole bunch of extra stuff, too. For example, if it finally does give up, there's a way for it to say, well, give me what you've got. And so SpinRite can even do a partial sector read, and nothing has ever done that before.

Leo: Yeah.

Steve: Sometimes that's enough.

Leo: Right. Because often the sector has slack space or it's just a few bytes in there.

Steve: Or it's a text file; or, I mean, back in the day I remember dBase files, dBase databases would absolutely not mount if any part of it was bad. So, like, just one record out of a huge database could be unreadable, and the whole thing was lost. So SpinRite would come along and say, okay, look, here's the problem in this record. But guess what, you've got the rest of your database back.

Leo: Right.

Steve: People were like, oh.

Leo: Hallelujah.

Steve: They were quite happy.

Leo: You bet, yeah, really. Hey, we're going to take a break, come back. We're going to design a computer with you from first principles. I think this is such a great idea. This is a Security Now! that you might want to save for kids, for students, anybody who wants to understand. You know, I was reading the Jerry Pournelle/Larry Niven book, I think it was - I don't think it was "Mote in God's Eye," I think it was "Lucifer's Hammer." And one of the scientists in there observes, we don't understand how anything works in our modern life. And if this comet hits, and we lose the few people who do, and all the information, we can't rebuild it because - and think about it, how much of the technology you use do you not have any clue how it works? No idea. Even the internal combustion engine. We probably could only get the basics back from the stuff that we understand, let alone computers. So this is good. This is something we need to know. How does this stuff work? Where did it come from?

Steve: There's a series of sci-fi that I've been reading called "The Lost Fleet" series that I've been reading on my Kindle. And the premise is really interesting. A guy awakens from having been in hibernation for a hundred years. And it turns out that he went into hibernation because he jumped into an escape pod as his ship was being destroyed at the beginning of a war between two cultures that had been at war now during this entire intervening hundred years.

And unfortunately what happened was that the casualty rate was so high that fighting, the art of fighting coordinated fleets of starships had been lost. And so he comes back and is fully trained in essentially this, like, what it takes to fight fleets of starships at substantial fractions of the speed of light where you have to take into account the speed of light delay, you've got to realize that your own information is delayed. And so he's able to - he, like, takes command of what remains of the alliance fleet and has to instill discipline that they've lost, a whole different way of fighting, but then leads them on a series of successful engagements because he's able to - he has the training that none of the rest of them have. And it's just, it's really a fascinating series. I'm enjoying it. I'm in the fourth book now and having a ball with it.

Leo: Yeah, I have to say we live in the steady march of technology. I don't want to give anything away because "Mote in God's Eye" talks about this also. I guess it's something that Larry and Jerry think about a lot. I guess I can't, I can't tell you this without giving away a very crucial turn in this book.

Steve: I love the book. The book is so good.

Leo: Yeah. You have to just read the book and then - but he does talk about this notion of we live on this pyramid of technology, but we couldn't rebuild it overnight. We just - you know? And we'd have to start from scratch each time.

Steve: Yeah.

Leo: That's all I can say without giving away a very critical part of that book. What a fun book that is, too. All right, Steve. Let us talk about computers. How far back do we have to go to understand this?

Steve: Well...

Leo: When you say "first principles," are you talking silicon? What are we talking about?

Steve: Before that, actually. If we wind ourselves back in time, get into our Wayback Machine and want to understand the first successful computers - and I'm talking about, frankly, the PDP DEC machines. There was Honeywell and Burroughs, Digital Equipment Corporation, IBM. These early machines were pre-integrated circuit. So there wasn't this notion of multiple components that could be mass produced. That was an amazing breakthrough which is very easy to take for granted. I mean, lord knows everybody, well, everybody listening to this has integrated circuits surrounding them. And I would say throughout the day we're surrounded by integrated circuits doing different things.

But before that, before it was possible to integrate different types of components onto a single piece of silicon, which allowed then mass production, all of these components were separate. And it was the separateness of them and the need to interconnect them which put a tremendous limiting factor on the feasible complexity of these early machines. So what I want to do is really start at the beginning with, if you have resistors and transistors, how do you create logic from that?

We know that computers use ones and zeroes. There were computers, analog computers, which people tinkered with, which for a while could - they had the benefit of being able to, with a lower degree of accuracy, do things in the analog world where currents and voltages conveyed meaning. They were able to do things far more easily than digital computers of the era, within the same timeframe, except that what ended up happening was we needed more accuracy. Things like temperature could affect the accuracy of an analog computer.

And so it turned out that just treating everything as collections of binary information, ones and zeroes, even though you needed a lot of them in order to get the same resolution that you could get sort of for free with the voltage on a wire, you could represent that voltage with a sufficiently large number of ones and zeroes and get the resolution you needed, and also get absolute accuracy that would not drift over time. You didn't have to worry about calibration and temperature and super closely controlled power supply voltages. There was just all this forgiveness by taking the binary approach. So now analog computers are sort of long gone, and they've been completely supplanted by digital technology.

So one of the simplest, most basic components of digital logic is called an inverter. And I want to explain - here's where we wish we had GoToMeeting. But we're in a podcast, an audio format, so I'm going to need people to sort of...

Leo: To visualize here.

Steve: Yeah. If you're driving...

Leo: You're proving my point.

Steve: Exactly. If you're driving while you're listening to this, do not close your eyes. But anybody else, I'm going to draw a picture for you. We have to do a little bit of schematic work with electricity and early electronics to explain the principles. But when I'm done, I think everyone's going to get a kick out of what they understand. I'm going to simplify things a little bit here and there. But fundamentally this is the way all of this stuff began to work.

Imagine in this visual slate that there's a wire running along the top which carries a voltage, and another wire running along the bottom which is the ground. And this is the way most of these logic diagram schematics are drawn, is you'll have sort of a bus running across the top that has a voltage, which is just a pressure, essentially, created by a power supply. And anchored at the bottom is another wire, sort of a bus running horizontally that is the ground. You then - you interconnect things in between this positive power supply potential at the top and the ground at the bottom.

If we had two resistors - a resistor is a component with two wires coming out of each end which, as the name sounds, resists the flow of current through it. Essentially what it does is you run current through it, and it gets hot. It dissipates current in the form of heat. So imagine in this circuit diagram that we have two resistors connected, the first one at the top, coming down to the second one, which then connects to the ground at the bottom. So that we have a circuit formed just with two resistors in series. And for the sake of simplicity we'll assume that they have the same amount of resistance. Well, this forms something called a "voltage divider" because essentially, when we make this circuit with just two resistors in a series, voltage will flow through this circuit.

And the direction of voltage flow is sort of controversial. I can't remember now, I was trying to remember which direction I learned in high school. Some people think of voltage flowing from the negative to the positive. Some people think of it from the positive to the negative. It really doesn't matter. Technically one direction is current flow, the other is the flow of the electrons, which sort of goes, being negative, goes in the other direction. So either way, all you have to have is a consistent system, since it's really sort of an arbitrary designation which way the current is flowing.

So we have this what's called a "voltage divider." So at the very top is our power supply voltage. What happens is the resistors share this voltage drop, as it's called, between the positive power supply voltage and ground, so that the junction where they're connected in the middle will be at half of that power supply voltage because they evenly divide it. And so that's sort of the first thing to see is you have two resistors connected together. They form what's called a voltage divider. And the voltage in the middle, or the voltage at their junction, where they're connected, is half of the total voltage.

So now we take out the bottom resistor, and we replace it with a switch, just a standard mechanical switch. It's got two wires; and, depending upon whether the switch is open or closed - "open" means they're not connected, "closed" means they are. If we close the switch, then the switch is essentially a short circuit. So now that resistor that's still on the upper half of this little circuit, its lower lead is connected through the closed switch to ground. So its lower lead is now at zero voltage, at ground, when this switch is closed. If we open the switch, then we've disconnected the circuit, and the lower lead now has the same voltage as the power supply because there's no current flowing through this resistor. There's no voltage drop across the resistor.

So now we go to the next step, and we replace the switch with a transistor. A transistor is a three-lead device, a three-terminal electronic device. We've all heard of transistors, of course. The way it works is it's like a - it works like an electronic switch. We put this transistor in the circuit. And so the transistor has an input on what's called the base lead of the transistor such that, when we put a positive voltage on that base lead, on the input of the transistor, the switch closes. That is, the transistor sort of works like the switch that we just took out. But it's controlled with the voltage on its base.

Actually voltage and current get complicated here, and I want to keep this sort of simple so we can stay to what's important. But the idea is that, if we put a positive voltage on the base of the transistor, that is, the input of the transistor, some current will flow through the base, which turns the transistor on. But remember that when the transistor is on, it pulls the lower end of that resistor that's coming down from the supply voltage, it pulls it down to ground, that is, down to zero. So what we have is an inverter because, when we put a positive voltage on the input of the transistor, it turns on, which pulls that junction between the resistor and the transistor down to zero. So a one goes in, and a zero comes out. And if we move the voltage on the base of this transistor, the input of the transistor down to ground, then the transistor turns off. And with the transistor off, then that junction between the resistor and the transistor goes up to the power supply voltage. In other words, a one.

So what we have is this, with just two components, this resistor that goes up to the positive power supply with a transistor hooked to it going down to ground. We have an input into the transistor, and the output is that junction between the resistor and the transistor. And that creates an inverter. So we have with these two components probably the most basic logic system that you can have.

So that's an inverter. It doesn't, I mean, it's certainly useful by itself. But we can do something, make one additional change to it to begin to create some logic gates. And that is, we take another transistor and hook it to the same place. That is, we put these two transistors in parallel with each other. Another transistor hooked to the same place so that either of them are able to be turned on and pull this output down to ground, that is, hook the bottom of the resistor down to ground. So now look what we have. If we turn either transistor on by putting a one, binary one into either of the inputs, then that transistor will turn on and pull the output down to ground. And they can both be turned on. We get the same result. So what we have is a, in logical terms, is called a NOR gate. Which NOR stands for "not or." So if either input is a one, the output is a zero. So we have the beginning of logic.

Now, we know how an inverter works. The inverter was just the transistor and the resistor. So we could take one of those and hook it to the output of this little NOR gate, and that would invert its output, turning it into an OR gate. So now if either of the inputs is high, the output of the first part is low. And then that's inverted so that the output of the final thing is high. So if either input is high, the output of our little OR gate, composed of this NOR followed by an inverter, is high. We have an OR gate. Or, conversely, we go back to this NOR gate, where either input is high, and the output is low. If we put inverters on the inputs, on each input, then look what we have. If we just, with the NOR itself, if either input is high, the output is low.

The other way of saying it is, only if both inputs are low is the output high. So if we invert the inputs, then only if both inputs are high will the output be high. Which is an AND gate. So we could have two inverters that feed into the NOR gate, and we end up with an AND gate. So it's possible just with this, just using simple components of transistors and resistors - and this is actually a family of logic called RTL. RTL stood for Resistor Transistor Logic. And circuits that were exactly like this were very popular. And this is the way digital logic was originally created. So it's clear that, by assembling these little building blocks in different configurations, we're able to create sort of fundamental logical connections.

Now, one of the core things that a computer has is registers. It needs to have, it needs to somehow hold data that it's working on. We need to be able to, for example, load data from memory into something called a register. You know, an accumulator. Well, what is that? How do we go from having AND and OR things, which we know now how to build, how do we have memory? How do we store something?

Well, there's an interesting trick. It would be fun to know who actually was the first person to come up with this because it's clever. And that is, we've seen how we create an inverter, where we just have a resistor coming down from the power supply to a transistor such that, if we put a one in, we get a zero out. Well, if we connect another inverter to the output of the first one, and then connect the output of that second inverter back into the first one, so essentially we have two inverters that are connected in a chain, or in a ring. Look what happens. If we have a one input going into the first inverter, we know that it gives us a zero out. Well, that zero goes into the second inverter, giving us a one out, which we feed back into the first inverter. So it's stable. That is, it just sits there like that. And it'll sit there forever like that.

But if something were to briefly, for example, pull the input of that first inverter, which is a one, pull it down to ground, to zero, well, then, its output would go to one. So the input to the second inverter, which would now be one, it turns into a zero, which it feeds back into the beginning, and that will be stable. So whatever state these two inverters are in, they stay in. And that's the basis for another fundamental logical thing called a flip-flop because it flips or flops in either direction, and it stays that way.

Now, when I talked about like if something pulled that input down, the way this is actually implemented is with something like a NOR gate. So if - and this circuit gets a little bit more complicated, and I'm about to sort of start waving my arms around and not going into the same level of detail as we pull back from this. But if we, instead of hooking these inverters to each other, we hooked our NOR gates to each other, then imagine that both - so the circuit is we have a NOR gate - we have two NOR gates. The output of the first one goes to one of the inputs of the second one. The output of that second NOR gate goes to one of the two inputs of the first one.

So we still have the notion of these things being connected to each other in a ring. But now we have each of those NOR gates has the other input, which is not participating in this circular interconnection. And that's actually where we're able to briefly change one, briefly, like, put a pulse on one to flip this flip-flop from one state to the other. And that's the basis for a register bit.

Now, we want to do other things like add bits together, add numbers. It turns out that addition of a binary number is essentially synthesizable just from these fundamental logic blocks. And we've sort of talked about this a few weeks ago where, if you look at adding two bits together, if both are zero, the result is zero. If either one is a one, then the result is one. If they're both one, then the result is zero with a carry. And so binary math works exactly the same as, for example, the decimal base 10 math that we're used to where, for example, if you had five, you were adding five and zero, you'd get five. If you add five and five, you get 10, meaning that the units is zero, and we've carried a one into the tens position. Well, binary works the same way, where if we have two ones, the result is zero, and we carry the one into the next position. So it's possible to express that logic with just the gates that we've seen designed here.

What I just described is known in logical terms as a "half adder" because it's able to take two binary bits and produce a result with a carry. Now, the tricky part is, the next bit over, the next highest bit, it needs to be able to add its two bits, but also accept the carry from the position to the right, the one less significant bit. That's a little more complex nest of logic which is called a "full adder" because it's able to take two inputs and the possibility of the carry from the prior result and incorporate that into its output and also send a carry to the next one.

So essentially, by stacking enough of these full adders together and interconnecting them so the carry-out of one goes into the carry-in of the next, and the carry-out from that goes into the carry-in of the next, you're able to take two binary numbers; and, after this thing settles down, there's like a ripple effect. If you imagine that you put the two numbers in, well, the result of the first two will produce a carry that goes into the second two. And that may produce a carry that goes into the third two and so forth. So this carry ripples down through these full adders to produce a result.

And then the final stage of this full adder, it produces a carry which in computers is often called the "overflow bit." Because the idea is, if we were adding 16-bit numbers together, the result of adding 16-bit numbers could be a 17-bit result, that is, a number that would not fit in 16 bits. And that's exactly the same as in decimal. If we're adding, like, single decimal digits, well, we know that a single digit can hold up to nine. So we could be adding any two decimal digits that sum up to nine. But if we try to add seven and seven, well, we know that that's 14. Well, that 14 won't fit in a single digit. It needs two.

Similarly, it's possible to add two binary numbers where the result won't fit in the same size as the binary numbers, so you have that final - that's what happens with that final carry bit that overflows outwards. So that's sort of the fundamental architecture for the way bits are managed in a computer.

Leo: Do you think that people figured that out a priori? I guess, you know, Alan Turing did all this, long before hardware was available to do it, in his head. And maybe even going back to Ada Lovelace; right? I mean...

Steve: Well, I mean...

Leo: But we didn't have this binary thing until we knew it was going to be a switch.

Steve: Right. That's a very good point. And all of this can be expressed mathematically rather than electrically.

Leo: It's Boolean logic; right?

Steve: Exactly. Boolean algebra, Boolean logic, allows you to do all of this kind of stuff and work out these problems. I mean...

Leo: And that's well known. I remember studying that in college, before there were personal computers. And it's fun. You do it all on paper. And you have AND, OR, NOT. I can't remember if you have things like NAND and NOR. But you learn all those. And there's even symbols for all of that.

Steve: Right.

Leo: So it makes sense that then, if you give somebody some hardware, and you say, well, okay, you have a switch and you have inverters and all that stuff, now, how do you duplicate those functions in this hardware? And that's really what you're talking about.

Steve: Exactly. And at the time, now, if we think about the cross-connected inverters with some additional logic around them, one of the things which basically forms a storage register. And then you want the ability to load them with a value that's coming in on a set of signal lines for however many bits. Well, that's going to take, oh, maybe, call it 20 transistors and some resistors. So that's for, like, one bit of NOR gates cross-connected with some other gates to gate their inputs. So that's maybe 20 transistors.

Well, back in 1960 a transistor cost a couple dollars. I mean, like, one transistor was a couple dollars. And it was a little, sort of a little silver can, smaller than a dime, sort of like a pencil eraser. Think of it like the end of a regular pencil eraser, sort of like that, with three leads. So it's a couple dollars. Well, say that the resistors that are also going to be scattered all over the place are free. We'll just toss them in because they were always pretty inexpensive. But 20 transistors at $2 each is $40 for the logic to implement one bit of a register. So there's - and that's not - that's just, like, raw cost. That's not even burdened with all the other costs.

The other thing then you have is the need to interconnect all of these because you've got 20 of these little eraser head things with three wires and a bunch of resistors. Now they have to physically live somewhere, on a circuit board. And that's got to have interconnections, which are traces on a circuit board. But now you've got to interconnect them into a family of these. So you need connectors to allow this little circuit board that represents, remember, just one bit of storage forming one bit of a register. It's got to be - you've got to be able to plug that into some sort of a back plane where wires then go from one to the other to connect them into a multi-bit conglomeration.

So maybe this is $50 by the time you're done for this one-bit register. And you want to do a, what, a 20 - you want 20 bits of binary resolution. So now you've got $1,000 that you've spent for 20 binary bits of storage. That's all this is, is just, you know, it can store a 20-bit value. But you haven't been able to do anything else with it yet, and you've spent $1,000. So, and that's not profit. I mean, that's not $1,000 of sale price, that's $1,000 of cost, including something for the interconnection.

So from the beginning the engineers who were trying to design a computer that could do something useful, they were designing them with these individual switches called transistors, and resistors that sort of go along with them, at a cost that meant that literally every single bit that they used was expensive. And they were trying to bring the costs down as quickly as they could, as far as they could, to make these computers much more accessible to people.

What I want to do next week, since we sort of understand this, is take a look then at the next stage, which is what do you do when you've got memory, and you've got a register like this, how do you turn this thing into something that can get some work done? And that's what we'll do in two weeks.

Leo: I love it. I love it. You know, it's so interesting to think, what, you said a thousand bucks for 20 transistors, something like that...

Steve: 20 bits.

Leo: 20 bits.

Steve: One 20-bit register.

Leo: And now we've got, somebody pointed out in the chatroom, we've got NVIDIA GT200 cards which cost about $100, $200 for, get ready, 1.3, what is it, 1.4 billion transistors. Billion. Billion. It's amazing, isn't it. But it was a huge insight to say we can do this. And then that began, with Moore's Law behind it, that began the amazing revolution that we've seen today.

Steve: I read a really interesting book last summer about the invention of the integrated circuit. And the breakthrough, it was actually there were some people in Texas, at Texas Instruments, and also at Fairchild in California. And there was some argument about who actually got it first. But at about the same time there was parallel invention because what happened was everybody was running up against what they called the "interconnection problem." It's like, we are dreaming big. We're ready to, like, do more.

But what happened was, just the physical need to interconnect the pieces, that became the limiting factor. They just, you know, the individual components were so big, and that they physically took up so much room, that you just - you needed to lay out the space. And there was - before long it got too big to run wires all around it. And so the interconnection problem was what the integrated circuit solved when it was able to say, hey, you know, I mean, they even knew they could make resistors out of silicon, they could make diodes, they could make transistors, the fundamental pieces they knew how to synthesize. But they didn't know how to, even on silicon, how to interconnect them. That breakthrough then opened the floodgates.

Leo: It's amazing. Eden in our chatroom sent me a link to the Wikipedia article on transistor count, which has a remarkable graph that shows how rock-solid Moore's Law is. It's a logarithmic chart that starts in 1971 with the 4004 which had 2,300 transistors. Essentially a transistor's a switch; right, Steve?

Steve: Yes. Exactly as we just covered, it is just like - it replaces a switch.

Leo: So it's 2,000 switches. Going up to 2008, where a 2 billion-switch quad-core Itanium - and but look how straight that line is, if you go to this curve, and because it's a logarithmic scale that means doubling. Incredible. I mean, it's such a - it's one of the most remarkable predictions of all time because it's held true for almost 40 years. And, I mean, true. I mean, right-on true.

Steve: Yeah.

Leo: Almost self, maybe self, I don't know, self-inflicting because in fact Gordon

Moore was the chairman of Intel, so maybe they said, well, we have to do this. I don't know. But amazing. Just amazing.

Steve: Yeah.

Leo: Really remarkable. What an age we live in. And it all starts where we just started.

Steve: Well, and what is so remarkable, and this is what we'll look at in two weeks, is I'm going to - I hope to be able to demonstrate with such crude tools, with something so simple as a really basic computer, it is possible to do an amazing amount. Again, as we said at the beginning, like a dumb box of rocks. But really fast rocks.

Leo: Really fast rocks. Hey, we're going to next week do questions and answers, as we always do on the modulo 2 shows. So send your comments, your thoughts, your suggestions, not just about this topic, but also anything having to do with security, to Steve via his website, GRC.com/feedback. That's where the form is, GRC.com/feedback. And we'll pick 10 or so questions, and Steve will answer them next week.

Don't forget you can go to GRC.com for many other reasons. Of course SpinRite, the world's best hard drive recovery and maintenance utility. It's just a must-have if you've got a hard drive. But also all the free stuff Steve does. And he does so much great stuff like ShieldsUP! and Perfect Paper Passwords, his DNS Benchmark tool. It's all there, including show notes for this show, all of our 233 episodes, transcriptions and audio, too. In fact, 16KB versions for the bandwidth-impaired. It's at GRC.com.

Steve: So we will pick up where we left off in two weeks...

Leo: I can't wait.

Steve: ...with the design of a computer.

Leo: Great, great subject. We'll see you next week on Security Now!.

Steve: Thanks, Leo.