
# Voting Machine Hacking

**Description:** This week Steve and Leo describe the inner workings of one of the best designed and apparently most secure electronic voting machines - currently in use in the United States - and how a group of university researchers hacked it without any outside information to create a 100% stealth vote stealing system.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-211.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-211-lq.mp3

---

INTRO: Netcasts you love, from people you trust. This is TWiT.

**Leo Laporte:** Bandwidth for Security Now! is provided by AOL Music and Spinner.com, where you can get free MP3s, exclusive interviews, and more.

This is Security Now! with Steve Gibson, Episode 211 for August 27, 2009: Hacking Electronic Voting Machines. This show is brought to you by listeners like you and your contributions. We couldn't do it without you. Thanks so much.

It's time for Security Now!, the show that covers all the things you need to know about keeping yourself secure on the Internet. Our guru of security, the man in charge, Mr. Steve Gibson of GRC.com. Steve is a security wiz, the man who discovered spyware, coined the term, has written many useful free security utilities, and has done this show for four years, on our fifth year now. Steve, good to see you.

**Steve Gibson:** Yes, and today it's the security of, or lack of security of, voting.

**Leo:** Oh, man, is that a topic.

**Steve:** Oh, and this is going to upset people, Leo. I mean, this is pretty scary stuff.

**Leo:** You know, we've been - nobody's proposing that we do electronic voting, are they?

**Steve:** Well, we have electronic voting. For the last...

**Leo:** Oh, this is the stuff we're using now, the Diebold stuff?

**Steve:** Yeah. In this case they're machines from Sequoia which have been reverse-engineered. And we're going to look this week in detail at what the vulnerability was and how it was leveraged into a demonstration that allows vote stealing in this class of machine. And so the problem is, I mean, everyone's concerned about, like, Internet voting. Eh, you know, whoa, slow down, we know we don't want to go there with viruses and malware and everything. But this is...

**Leo:** But we're not talking about that.

**Steve:** Yeah, this is the machines where you interact with them electronically - spinning a dial, pressing a button, whatever. Some people have paper printed sheets where they're like we used to do tests in college where you use a number whatever it is, No. 3 pencil and fill in the dot, and then they just use basically an optical mark reader in order to tally the vote.

**Leo:** Right, that's how we vote. That's how California votes. Right?

**Steve:** Well, not down here in Southern California. I'm looking at an LCD screen with a big wheel. And I spin the wheel and then punch the - and it's all electronic.

**Leo:** Oh, so you have these machines.

**Steve:** Well, yes. There are - and like the Diebold and Sequoia, there are all kinds of different machines. These particular machines are still in use. Well, currently in use. I don't think after this report gets a lot of air that anyone's going to feel comfortable using them. But I'll tell the story of how they were procured, and when, and where they're in use. And I really think this is going to be interesting for people because, using some good reverse-engineering techniques, but a lot of computer, fundamental computer technology that we're going to go into in this hour, it turns out it's possible for someone to sneak in the night before, play some games with each machine, and leave them in a state such that they no longer accurately record the votes that they are registered through their UI.

**Leo:** Oh, man.

**Steve:** It's bad.

**Leo:** So let's talk - do we have security updates? Security news?

**Steve:** Yeah, we have some, and a little bit of errata. And I have sort of a fun, different

SpinRite story than I've ever told before. It was a very quiet week in security. Thank goodness we have one every so often. I only have two items. One is just sort of strange. The U.S. Department of Agriculture has begun enforcing a policy which bans the use of any web browser other than Microsoft's Internet Explorer. Want me to say that again [laughing]?

**Leo:** I'm stunned.

**Steve:** One of their units, the Cooperative State Research, Education, and Extension Service, they say that they are following the so-called Federal Desktop Core Configuration Guide, which is a 2008 government-wide policy administered by the Office of Management and Budget, the OMB, which requires that agencies standardize operating system and browser settings to prevent security breaches. And so their logic is that Internet Explorer is the only browser which allows remote configuration and lockdown and management. So they have, I mean, this policy's been around, but it hasn't been enforced. They're now enforcing it, removing any non-IE browser. Specifically Firefox was mentioned in the news story.

**Leo:** What?

**Steve:** Because...

**Leo:** Well, I understand standardizing because then you can have standard policies and procedures. So if you have a bunch of different browsers, maybe they don't want to support them all. Yeah?

**Steve:** Yeah. I mean, now, this caused a lot of problems. For one thing, they maintain websites. And so their webmasters are no longer able to use non-IE browsers which visitors to their website use.

**Leo:** So they can't validate them.

**Steve:** Exactly. They were depending upon using these other browsers to make sure that their sites worked in non-IE browsers. So it's not clear how this is going to fall out. But, and people have said, wait a minute, you know, does the Federal Desktop Core Configuration specify that you can only use IE? And the answer is no, it doesn't. So it's feeling like there's been some overzealousness on the part of the IT people in this Cooperative State Research, Education, and Extension Service. But at the moment that's the policy, and people are no longer able to use anything but IE. Which is, you know, we know how feel about IE. I mean, I'd be really chagrinned if anyone told me I couldn't use Firefox because I'm, frankly, due to the add-ons and the additional features and controls I have, I can do much more with Firefox. I have more control over my Internet usage experience and, frankly, more security than if I'm using IE.

**Leo:** That's because you use NoScript and stuff. But, I mean, people use - you for a

long time used IE.

**Steve:** I did.

**Leo:** And believed it could be made secure. And that was a long time ago.

**Steve:** Yes. I went - I had to jump through hoops, though. I used IE's zone system, and I locked down the Internet zone, and then I manually added sites that I trusted to my trusted zone. And those had less security associated with them. So it was possible to do. But Firefox just makes it much easier. So it's not only NoScript. But, for example, I use a little cookie manager that allows me to have all my cookies treated as session cookies, so they're forgotten constantly, unless I say, oh, this is one where when I come back I want to be remembered. Like I want to have eBay remember, and PayPal and so forth, so I don't have to go through all the log-in every time. So it's done on a selective basis. But again, it seems to me like a real step backwards for any government agency to mandate the use of any one browser, if for no other reason it's a lack of choice where choice enhances security instead of limiting it. But from a management standpoint, I can see that it's tough to support something as heterogeneous as our PC industry has become.

**Leo:** Yeah. It's interesting.

**Steve:** The other...

**Leo:** Just the Agriculture Department, though. It's not government wide.

**Steve:** Right, just USDA. And it's just - actually it's just even - it's just a unit of the USDA. So not even all the Department of Agriculture, who have a somewhat less than illustrious security history themselves. So it's just this one group. And maybe this is going to shine a bright light on it; and then the powers that be will say, wait a minute, this was a mistake. But at the moment it did float up to the top of the security news for the week.

The other thing just came across the radar from a report that I'm a little - I'm certainly interested, but I'm a little skeptical. The original story is in the German version of the Financial Times, which, you know, I can't read. But it was picked up and reported by The Register (theregister.co.uk). They've got a lot of good news. They're a little sketchy sometimes. They tend to be a little flamboyant and loose with the facts. So, and I haven't seen this anywhere else. I looked for it. I couldn't find anybody else picking up on it. But the news is that the Chaos Computer Club, which is an established, known hacking group, have said that within a month or two they will issue a public demo and release code so that anyone equipped with a laptop and an antenna can listen to GSM phone calls. And the story is that GSM has been cracked.

**Leo:** Wow.

**Steve:** Now, it's funny because we were talking about this, I think in a Q&A either last week or a couple weeks ago, where someone was asking, am I secure with using a cellular broadband connection? And I was saying that, yeah, probably, but that this is old encryption that was based on shift register technology to generate pseudorandom bitstreams which are XORed with the plaintext to create ciphertext, and that they were relying on trade secret information, not disclosing the lengths of these different, you know, they're like prime number length shift registers that spin around. Their outputs are XORed into the final bitstream. And that creates the encrypted text. And at the other end this same pattern is used to decrypt it.

You know, we're well versed now, our Security Now! listeners, in the notion of a pseudorandom data stream being XORed with, exclusive ORed, with the plaintext to create ciphertext. Well, this has apparently been cracked. So I just wanted to raise that note. I will keep track of this. And I'm sure, if this is the case, it's going to surface on a lot of other security sites and will make a lot more news. This just happened. And I went to the Financial Times article, but saw that it was all in German and said, well, okay, let's wait for other people to translate this. I would feel more comfortable knowing, seeing the details of what this Chaos Computer Club is claiming. And The Register tended to be a little more effervescent about it than I would like.

**Leo:** Yeah, yeah.

**Steve:** But it'll certainly be big news if, depending upon what flavor and level of GSM has been cracked, this is not good.

**Leo:** Yeah, because I'm GSM. I'm sure you are. I mean, GSM is T-Mobile and AT&T. It's everybody with an iPhone. It's not Verizon or Sprint.

**Steve:** Correct. That's CDMA.

**Leo:** Doesn't affect them, yeah.

**Steve:** Yup. And then my little bit of errata is I did want to just put on the radar for our sci-fi fans that Apple now has on their site, Apple.com, trailers for the upcoming James Cameron movie in December, "Avatar."

**Leo:** Yeah. People are really excited about this.

**Steve:** And oh, Leo. I mean, it looks a little bit too much like unicorns and fairies to me from the...

**Leo:** There is that element.

**Steve:** It's weird, yes. And so I was thinking, well, Mark Thompson, who likes those things, he'll probably get a kick out of that. But it also looked like it's good sci-fi, as well,

so. Have you seen the trailer?

**Leo:** No. I'm going right now to Apple to see it.

**Steve:** Yes, it's...

**Leo:** I've heard a little bit about it. One of the things he's kind of interested in doing is he believes all movies will be CGI soon; right? So a lot of this is fully computer-generated.

**Steve:** Yes. He's mixing live actors and CG, I mean, substantially. And that's my - I guess if I had a complaint, it's that from this preview that...

**Leo:** It looks like it's a cartoon in some areas.

**Steve:** It goes. It looks a little cartoon-y. It's not - it doesn't have the - just that real edge that we're used to seeing. But I think it's because he's had to bring in, I mean, like these are not - these sets that they're on don't exist.

**Leo:** They're all green screen.

**Steve:** They're all synthetic sets.

**Leo:** Yeah, yeah.

**Steve:** Yeah. So anyway...

**Leo:** Well, this is the future of moviemaking. I don't know if it's the - by the way, James Cameron also thinks all movies will be 3D soon. So he's kind of bought into this whole notion. I guess there's stuff you can do in CGI that you just can't do any other way.

**Steve:** Yeah, I mean, there's a scene in the trailer where a little helicopter is flying in this large space with sort of really neat sort of floating dirt mountains.

**Leo:** Right.

**Steve:** And I don't know how you do that...

**Leo:** You can't, yeah.

**Steve:** ...except to do it all in computer-generated imagery. So, yeah, I don't know. Have you seen any of the new 3D movies? I have not. And so I don't...

**Leo:** No. And I can't because I have mono vision. I don't have...

**Steve:** Your brain doesn't merge the two...

**Leo:** Yeah. So I can't really tell 3D, unfortunately. So I will never appreciate it. I hope it doesn't take off because then it means I'm not going to enjoy movies much anymore. They'd better provide a 1D or 2D version for oldsters like me.

**Steve:** It'll be interesting. The problem is, I know that there's a "Final Destination" movie coming out in 3D. I liked the "Final Destination" movies. Something about the plot lines just sort of appealed to me. But this one is - it's where the car tire is smacking you in the face. I mean, it's overdone. It's not just sort of subtle where you can forget that it's 3D and find yourself immersed in it. Instead they're taking advantage of, like, you know, poking sticks out of the screen at you...

**Leo:** Right. I don't like that. It's gimmicky when they do. It's gimmicky.

**Steve:** Yeah, exactly.

**Leo:** But I guess he feels, and he's probably right, that the future is more immersive, more realistic. And of course if you're using computers to generate it, it can be very, very authentic 3D.

**Steve:** Oh, it could be spectacular. The problem is that, as we know, in order to generate 3D you need to give each eye, each of your eyes its own image. And that's been challenging. One approach has been the blue-green, I'm sorry, the red-blue glasses, where you use tinting of the two different images and separate them that way so that one eye sees red that's filtered out by the blue, and the other sees blue that's filtered out by the red. But then you've got a problem with color. The alternative is to use high-speed shuttering where each eye is seeing alternative frames on the screen. And that's very expensive because now the whole audience has to have some sort of electronic gadgetry that they're wearing in order to separate the images. So, challenging.

**Leo:** By the way, somebody pointed out, absolutely, as usual, I'm U.S.-centric, that GSM may only be used by AT&T, T-Mobile in the U.S., but it's used worldwide on almost all phone systems.

**Steve:** Well, yeah, exactly. It is the global standard.

**Leo:** Yeah. So I apologize. I didn't mean to sound U.S.-centric there. Yes, everybody uses it worldwide.

**Steve:** And a note was forwarded to me about a month and a half ago that I had actually on my Starbucks machine and wanted to share because it was sort of an interesting story. It was forwarded to me by Sue, my office manager/bookkeeper, from someone named Sean. He said, "Hello. My name is Sean. I went to a flea market, and there was a man selling SpinRite for $80. I asked him how can he sell it. He told me he has a special deal worked out with you, so I bought a copy."

**Leo:** Oh, boy.

**Steve:** "It was a few days later when I got home to run the program. And when I did, I saw that it was registered to" - and then there's a person's name here - "with a serial number of" - and then he gives the serial number. And so he was reporting this to our office at our sales email account. He says, "I drove the 30 miles back to the flea market, only to find that the person selling what was clearly an illegal copy of SpinRite was gone. I have not used the product. I do not want to do that to you. And you never know if someone has stuffed something extra in it, like some bad guys do. The thing that bugs me is that I saved up for three months to get the $80, and I should have just gone to your site, but I did not."

**Leo:** Oh, I feel bad.

**Steve:** "I will destroy this copy and try to save the money again and get it from your site this time. You and Leo are doing a great job. You both provide very useful information that, when followed, will keep us very safe. Thank you to both of you." Well, Sue answered him and said she was sorry to hear what happened, and also blind copied me on it because she's known me for 22 years.

**Leo:** Yeah, I know you, too. And I know exactly what you did.

**Steve:** So I replied. I said, "I'm sorry to hear that you were sucked into that scam, and I'm certainly annoyed that some loser is making money from people - he has your $80 - by illegally selling our software that was licensed to someone else. But I'm also not happy with the idea of you needing to spend another $89 to purchase a legitimate copy from us. After all, although we do need to sell SpinRite to support everything we do, our cost to develop and support a copy of SpinRite is relatively low. Therefore, in a minute or two you'll receive a note directly from GRC's eCommerce system, which I'm able to securely access from Starbucks thanks to our Perfect Paper Passwords system, which you probably know about from the Security Now! podcast. The email you receive will be a receipt for a no-charge copy of SpinRite, licensed this time to you by name. And the links in the receipt can be used to download your own personal copy now and forever more. Thanks for your note and for your support of GRC. All the best."

**Leo:** Oh, thank you, Steve. I knew, I knew that was going to be the answer.

**Steve:** Oh, yeah, I mean, it sounds like he is a neat kid, and I appreciated him letting us know that this had happened. And but no way was I going to have him save up and buy another copy.

**Leo:** There's nothing you could do about that other copy, I guess. You can't...

**Steve:** Well, and I, you know, that's the nature of software. I mean, we know that it's high margin. We know that some is going to get stolen and pirated. What happened was, I mean, we know this copy. We know that person's name. When I first came out with the software distribution system, I was giving people an encoded URL that they could use forever, with the instructions "Make sure you never let loose of this." People were using download managers, which were behind their back spying on them and reporting the URLs they used for downloading software to a central database.

**Leo:** You're kidding.

**Steve:** No. This is going on. I mean, download managers are doing this.

**Leo:** Oh, my goodness. Do you know which manager it was?

**Steve:** I did at the time. This has been years ago now.

**Leo:** Oh, that's appalling. I had no idea.

**Steve:** Yeah. In fact, there was, like, the idea was it was supposed to be a service, that you could then go to this and

see what other people were downloading and use their URLs to download it yourself.

**Leo:** Oh, my god.

**Steve:** And so three or four copies, three or four of SpinRite 6's first users were using these download managers. We quickly learned that these copies had gotten loose. And so we marked them in the eCommerce system to kill them so that they could not be used, they couldn't be downloaded anymore, and they couldn't be used for upgrades. But they were loose anyway. We contacted these people, told them about the problem. They, you can imagine how horrified they were to know that everything they downloaded was being sent back to the mothership somewhere and being searchable in a third-party database.

So anyway, so we fixed the problem. And I changed the way our eCommerce system works so that the download links are good for literally one time. The act of clicking on it

kills that code, so it cannot be used again. And if you don't use it within about five minutes, it expires. And it's easy. You just get another one from us. You can get them anytime you want, as many as you want. But so that solved the problem, but not before a few got away from us. And those are the ones that are floating around the 'Net, and we see them from time to time. So it's like, oh, well, that's the way things go.

**Leo:** That's shocking. You know, I remember we did this - this is how long this has been going on. On The Screensavers we had Avi Rubin, security researcher, come on and show us - this must have been six years ago - how electronic voting machines could be compromised. He's been telling Diebold and the other companies about this for six years, and it's still a mess.

**Steve:** Yeah. The thing that appealed to me about this particular story is that this machine was very well designed. We've talked often about the mispurposing of operating systems, the idea of using Windows in a checkout stand in a supermarket or in an ATM machine, because it's a huge, complex, massive operating system with all kinds of bells and whistles. And every bell and whistle you have has to be exactly correctly designed, or there's a potential for exploitation.

So when I hear that a voting machine is using Windows, or even Linux, I mean, people say it needs to be open source so that it can all be perused. Well, I completely agree that where voting is concerned, you ought to really have people taking a look at the code and understanding it. But I object to the idea of building basically a simple application, I mean, electronic voting is trivial. There's nothing to it. It's display some names and select one of them and record that. The idea, frankly, that you would use any operating system is offensive to me because it's overkill, and it's overkill in a way that can definitely hurt you. I mean, look at the security problems, for example, that Apple has had as a consequence of the third-party open source software that they've got bundled with the Mac. Many times it's not code that Apple wrote, it's just - it's public libraries that are out there, they're open source, but we know that doesn't mean that they're secure. It means that there's a better opportunity for analyzing them, seeing what's going on.

But for me the idea solution would be a very small, very lightweight solution for voting where the code is also open source, that is, many researchers have looked at it and tried to find ways around it, yet there just isn't anything there you don't need. So, I mean, there's just so many opportunities when you have a huge operating system. So what I'm going to talk about today is a - is exactly that. It is a very beautifully designed, tight little voting machine which is the kind of thing, exactly, that I would recommend people come up with for this kind of solution. It uses an 8-bit Z80 processor.

**Leo:** Wow. Blast from the past.

**Steve:** Yes, the Zilog Z80, which was a sort of a superset clone of the original Intel 8080 processor. It has a 16-bit address bus, so we know that 16 bits gives us access to 64K of addressing space. What happened is - oh, and this machine is called the Sequoia AVC Advantage. It is still in use in New Jersey, Louisiana, and other places. The machines specifically are version 5.00D. And they were originally purchased in 1997 by Buncombe County, North Carolina. They were purchased for $5,200 each. In January of '07, so after a 10-year life, they were retired, these particular ones, the ones that are no longer in service, were retired from use and auctioned off on a government auction website where a researcher at Princeton, Andrew Appel, purchased a lot of five of these machines for a

grand total of $82.

**Leo:** Just for fun. He just wanted them for fun.

**Steve:** So what is that, $13 or $14 apiece, he gets these used voting machines. The problem is, they're in use elsewhere in the country. So now he has them and says, okay, how secure are these things, which I've got now five copies of, which are in use elsewhere in the country? So a bunch of guys from UC San Diego, Princeton, and University of Michigan - and their names, Ed Felten is among them, they're well-known security researchers, academic types - they decide they're going to analyze this machine.

Now, what's interesting about the machine is it really was designed by people who did, I'm convinced, the best they could to come up with a secure solution. The total machine is this Zilog Z80, which can address 64K of memory. But they needed more than that much code to do all the different things that the machine had to do. So it contains three 64K ROMs, and each of these ROMs is divided into quarters. So there's four 16K pieces per ROM. And instructions that the Zilog chip can execute cause various quarters, these various quarters of these ROMs, to be mapped into the address space of the Z80. This is something that's been done for years whenever systems outgrow their address space.

For example, people may remember, anyone who was using DOS in the old days remembers EMM, the extended expanded memory system. Actually I think it was Lotus that - I think it was LIM, wasn't that the acronym, Lotus, Intel, and Microsoft. What happened was spreadsheets got bigger than the 640K that the PC could handle. The PC had a 640K, actually it was a megabyte, but the top chunk of it was used by the BIOS and video and I/O space. So you're able to put 640K of RAM in the original PC and XT in order to fill it out to its full size because remember that Bill Gates famously told us that, oh...

**Leo:** Nobody'll ever need more than...

**Steve:** Oh, that's 10 times more than an Apple II has, so obviously that's plenty. So what happened was spreadsheets were very popular. They were getting big. So it was necessary to somehow add more data space to the PCs. So you could then buy basically a RAM add-in card, which used a technology exactly like this little voting machine used, called bank switching, where there's like a huge amount of RAM, and you could cause pieces of it, a bank at a time, to be accessible in a certain address range that was within this 640K space. So you'd basically swap in and out chunks of a much larger memory. You couldn't, there just weren't enough addressing bits to uniquely address as much RAM as you now had in your machine. So you could sort of do them a bank at a time, a piece at a time. And so the Lotus 1-2-3 spreadsheet, an API was created to allow this to be done, to be accessed in a standard way. And this allowed your data to occupy much more space than the system was able to access at one time by sort of swapping these things in and out.

So this little voting machine, this Sequoia AVC Advantage, uses that. What it did was it took its 64K, and one quarter of one of those 64K ROMs was its so-called BIOS, which was always mapped into the address space starting at the beginning. So zero to 16K. Then any other one of the 16K chunks in these three 64K ROMs could be mapped into the second 16K space. That left 32K for RAM. And so the first 32K was ROM. And the second 32K was RAM, thus making up the full 64K, 16 bits' worth, that this little Z80 chip

could access. I mean, everything about this is cool so far. Then they went one step further. Because they knew this was a voting machine, security was paramount. They made it so that it was impossible to execute code from RAM. They thought, there's no way, there's no reason that anyone has a legitimate reason for executing code from RAM. So...

**Leo:** That's correct. That's right.

**Steve:** Absolutely right. And we know what a problem that is for our computers. I mean, all the buffer overflows and running code on the stack and running code in data buffers, this is the problem we have today. And 20 years ago, when this thing was first designed, the engineers said, let's just prohibit that with hardware, something no one can get around. So they added hardware which any attempt to execute out of that lower 32K, any attempt to fetch an instruction from the RAM, that upper RAM half of the instruction area of the addressing region, immediately causes a halt of the system. It just locks up. It actually, there's something called a "non-maskable interrupt," an NMI, which chips at the time had, that I think we still have them in our current hardware, that is an interrupt, a hardware interrupt that nothing can block. It can't be masked off by software. And it caused a hard jump into the BIOS to put an error code on the LCD display, and then it did a halt. So it's just there was - these guys understood that this kind of security was important.

So the researchers took one of these machines and held it up, held the circuit board up to a bright light. It was just a two-layer circuit board. So if you shine a bright light through the circuit board - because it's just made out of, it's not plastic, it's fiberglass - you're able to see the traces on both sides of the circuit board. Typically one side traces run horizontally, the other side they run vertically. And so, and they knew what the chips were. So they came up with a schematic for this. They also knew what the instruction set of the Z80 was. And so they dumped out the three ROMs so that they could see what the code was. Basically they reverse-engineered the machine. But reverse-engineering something which is secure shouldn't be a problem. You know, we've talked about security by obscurity. So the idea would be that this could be in the public domain, and it shouldn't make it any less secure because you don't want to rely on what people don't know for your security.

**Leo:** Right.

**Steve:** We've talked about that often.

**Leo:** Plus it's great to have other eyes looking at it. That can be a real value.

**Steve:** Oh, believe me. What these guys did, I'm sure if the engineers are still around who designed this machine and paid so much, put so much evident care and concern into security, they're just shaking their heads because here we have hardware that will not execute any code other than what was provided. So no kind of code injection can be used.

Okay. Now we need to talk a little bit about the way stack machines work. The notion of a stack was an innovation which occurred to someone, I'm not sure where in the

development of computer evolution. But, for example, my old favorite little dinky 12-bit PDP-8 did not have this notion originally of a stack. So when you wanted to execute a subroutine - a subroutine just sort of being a piece of code that many different places in the program might want to run. So the idea is, instead of repeating that code throughout the program, you only have it in one location. And different places in the program are able to execute that subroutine, the idea being that the subroutine does whatever it does. And when it's done, it returns to the instruction after the one that called it, that invoked it.

And by being able to go back to where it was called from, instead of, for example, always going back to the same place, if it goes back to where it was called from, then you can call it from anywhere you want, knowing that once it's done it'll come back to you, and you can continue doing what you were doing. So you need some way of knowing where you were called from, that is, the subroutine needs a way of knowing where it was called from in order to go back there.

Well, before stacks were created, and the solution that the PDP-8 used, was the first word of the subroutine was always left blank. And when the subroutine was called, the computer would put the address of the instruction after the one that called it into the top of the subroutine. So it would sort of just be stuck there. Then the subroutine would do whatever it does. And when it's all done, it would jump to the location stored at the front of it. And that would take it back to where it was called from.

Well, that was an elegant solution, and it was the only one we had at the time. But there was a problem with that. And that is, you couldn't have reentrant code. That is to say that, for example, not only could the subroutine not call itself, because if it called itself it would overwrite the return address at the top of it with another return address, but it couldn't call any other code that might have some reason for calling it. That is, you could never nest subroutines. Which ended up being a real problem as programs got more complicated because it just - you had to really understand what your program's flow was and make sure that there was no way that a subroutine could ever execute code where anywhere downstream it could get called again prior to it returning.

So the innovation of a stack was tremendous for computer science. The idea was that, instead of storing the return instruction in the code of the subroutine, instead we would have sort of a separate scratchpad which would automatically, I want to say, grow. I'm trying to think of how to describe it. It would automatically accept values and return values in a last-in, first-out mode, a so-called LIFO, meaning that if you put a value in, that's the value you get out. And as you take values out, they come out in the reverse order that you put them in, in very much like a stack. If you could imagine, if you imagine something that's called a stack, for example, like a stack of plates, where if you put plates on the stack, the stack grows. And as you take plates off, you're getting them in the reverse order.

So what happens is, with a stack-oriented machine, which is what everything is using now, it's such a successful and popular concept that when a subroutine was called, the return address was placed on the stack, and the subroutine would do what it does, and then a special instruction, a return instruction, would always take the value that's on the top of the stack, which would have been the last one placed on the stack, and return to there. So the beauty of that is that, if that subroutine called some other code that ended up calling back to the subroutine, well, this all just gets stuck on the stack. So as the return instructions are executed at the end of each subroutine, the values are popped off the stack in the reverse order, and everything works. Everything sort of comes back just exactly the way it's supposed to.

So the Z80 was a stack-oriented machine. And so the designers took advantage of its stack orientation in order to write their code. So the problem was that from a hacking standpoint it's not possible for us to provide any code because the only code we can provide would be in RAM. But we could provide pointers to code in ROM. So what these hackers cleverly realized is that there was, spread throughout the code for this voting machine, were subroutines, all ending in a return instruction. And they didn't want necessarily to do what these subroutines did. But they looked at the last few instructions prior to the return instruction and said, okay, is that useful for something? Is the little bit at the end of the subroutine useful? What does that do?

Well, it turns out that they wrote some code to look for all the return bytes. In the Zilog Z80 instruction set, a return instruction is a C9 in hex. These are all 8-bit instructions. A C9 is a return instruction. So they found all the C9s that were in ROM. And then they looked at the instructions just in front of those C9s to see what those do. And what they were trying to do was come up with a corpus, come up with a collection of useful things where they could jump to the near, just near the end of the subroutine and get a little bit of work done. Not a whole bunch. Just add something to the accumulator. Maybe subtract two values. Or put something somewhere in memory. Just little bits of work which happened to be at the end of all the subroutines, all the various subroutines that existed in this code.

One of the cool things from their standpoint about the Z80 is that every single byte is an instruction. It was back in the day where we had 8-bit bytes. So you had 256 possible instructions. And in the Z80 it was completely dense. That is to say, this map of 256 possible instructions was completely full. There were no invalid instructions. There were no privileged instructions. On more advanced machines like the Pentiums and PowerPCs and so forth, they have much larger instruction sets that won't fit in a single byte anymore. So many times there's areas of sort of ranges of instructions which are illegal. They're just not defined. They're for the future. They just didn't need all of the instruction space. And in other cases there are so-called privileged instructions, for example, which are powerful, which only the operating system running in the kernel, but not the user, is able to execute.

So for example in a Windows environment or in a Linux or UNIX environment, the user processes cannot actually do I/O. You can't access the physical hardware port because if you allowed that to happen, then users would have too much power. Instead you have to ask the operating system to do those things on your behalf, and it manages conflicts between programs that way. So this is the way things have evolved. But back in the days of the Zilog Z80, there was no notion of privileged instructions. What that means is that any data in this ROM that they had access to could be instructions. Even if they weren't meant to be, if they were never - if they were just like regions of data that happened to have C9, for example, in them, then the things in front of these return instructions could be executed. Nothing would generate an error. Nothing would blow up. Nothing would go wrong.

So what the researchers did was they found all these little return instructions, and they analyzed the work being done just before the return. Because what they realized they could do is come up with a stack which doesn't have code on it because the stack is in RAM, and we know that we can't execute out of RAM. But we can have pointers in the stack into subroutines. And when the subroutine returns, it'll come back and get the next pointer from the stack. Because that's how stack machines work.

So what they did was they aggregated sets of little tiny bits of work at the end of all these different subroutines into what they called "gadgets." And a gadget would do a defined thing, like it would add two values together. It would subtract one from another.

It would perform a nonconditional jump or a conditional, a branching jump. Or it would do all the various things that programmers want to do. They were able to come up with little tiny fragments of work which when aggregated together created a complete pseudo instruction set. And it was what's called...

**Leo:** That's quite clever.

**Steve:** Oh, it's so clever. And it's what's called Turing complete. Remember we've talked about Alan Turing, the cryptographer, and the guy who gave us the Turing machine. This whole notion of computability, Alan Turing really researched and brought to the world. The definition of a Turing complete computation engine is one that can compute anything else that a Turing complete machine can compute. So what these guys did was, they defined their own instruction set, which was Turing complete, meaning they could do anything they wanted. And this is what...

**Leo:** That's amazing.

**Steve:** I love that these guys were academicians because they really pushed this probably further than they had to. But they just wanted to say, we can do this. So in looking at the disassembled code, there were very few buffer overruns. But they found one. They found one in one part of the file system that had been defined. And they realized that there was a way they could get about 12 bytes onto the stack. They could gain a foothold. Now, that wasn't enough to do much, but they didn't need much because they were able - there was one error that they found in the file system where, if they made a funky file, then they could get the thing, the code that was interpreting the file, to put some data on the stack.

This would normally never be a problem. These machines weren't networked. They were standalone. They had two cartridge slots on them. One was the so-called "results" cartridge, which was - they're battery backed-up static RAM cartridges. And the other one was an auxiliary cartridge. Which could be - and these could also be mapped into the address space of the processor. So they realized that they could put in a specially designed cartridge where the file system of the cartridge - and they reverse-engineered all of this. I think they said it took - they're guessing it took about 16 man months.

**Leo:** Oh, man.

**Steve:** So if you had 16 people, that's one month. If you had eight people, that's two months. So, but remember, this is a voting machine in use today. So they figured how they could - basically the protocol was somebody the night before the election, where these machines are going to be used, goes into the voting place, pulls out the results cartridge just a little bit to dislodge it from its mounting. There's a plastic security loop which runs through holes in the cartridge to prevent it from falling out and, frankly, to prevent it from being removed. But it turns out that the holes are big enough you could pull the cartridge just out so that it's no longer making electrical contact.

The auxiliary cartridge was empty because there was none necessary. It was sort of an add-on, upgradeable feature, but wasn't used. They were able to plug their cartridge in, turn the machine on, and when the machine saw that the auxiliary cartridge was in place,

it would load one of the files in the file system which they had prepared. That, because their file was deliberately made incorrectly, but from looking at the reverse, from looking at the disassembly of the code, they figured out where there was a fault in the processing of the file system. And again, this is so easy to do because even the developers, if you showed them that there was this problem, the developers could say, well, but yeah, but we're making the files. So we know how to interpret them, and we're not going to make a bad file. And they could also say, and even if someone did, they can't execute their code. Because these are RAM cartridges that are in the RAM space, and they won't execute code because we made sure only our ROMs could execute code. In the hardware. No way around it.

So they'd turn the machine on with their special funky cartridge plugged into the empty auxiliary slot, go to the main menu, tell it to load from the auxiliary cartridge. That allows them to get their special file in, which trips up the file system interpreter in a way that lets 12 bytes end up on the stack. And that gives them a foothold. Those 12 bytes are pointers into existing ROM code, the end of subroutines, just little fragments, a few instructions at the end that they've figured out how to knit together. And that then loads additional code, which gets them into the machine.

And once they get this corpus collected, they have the ability to do anything. They then - the attacker pushes the power switch. It turns out that it's a soft power switch. It's a power switch which is read by the software, just like all of our consumer electronics. I mean, an iPhone, anything you've got in your pocket running on batteries, these are soft power switches. You're typically not actually disconnecting anything. I don't know if people have noticed, but if you do take the battery out, it takes much longer for it to run down than it does if it's just, quote, "off," because they're not actually ever off anymore. So this is like that, a soft power switch, which then their code fakes this machine being turned off. It blanks the displays, turns off the LEDs, blanks out the LCD.

The one thing they say they didn't succeed in doing, and it's like, well, we decided we did a good enough job, was they did not and could not turn off the LCD backlight. So there was one little bit of hint was that the LCD was still backlit. But otherwise this thing pretended to be powered off and asleep. And the machines have a 16-hour backup battery which was part of the spec for the voting machine, so that you could still vote even in the event of a power outage. So even if the machines were then unplugged, or if they hadn't been yet deployed for use the next morning, in the next morning's election, they would survive the night prior to being put into use. So the code was there. When the machines were turned back on they exactly emulated the normal startup sequence.

Now, the beauty is that they had access to all the ROM that actually does the work. So if the machine - you know, the ROM that would do the normal good work. So they were able to call all of the ROM routines for doing normal election operating appearance. And in their demo what they did was, once the election was over, and the menu was used to end the voting, they took half of the number of votes given to the second person on the ballot and gave them to the first person. So they just moved them over. I mean, they could do anything, any logic you wanted about what percentage of what votes go to who and so forth. But just for their demo purpose they said, okay, just to show that we can actually do vote stealing, we'll take half the number of votes that the second person on the ballot got and subtract those and put them in the first person. And then all the other accounting that was in the machine they balanced out so that everything worked.

So from standing way back from this, this is a matter of you approach the machine sometime before the election. You slip your cart- you pull the cartridge, you dislodge the good election results cartridge. You slip your hacker cartridge into the unoccupied adjacent slot, turn the power on, do a couple things on the menu, turn the power off, pull

your cartridge out, go to the next machine, do the same thing. Go to the next machine, same thing. You have corrupted the results from these voting machines which are currently in use in the United States.

**Leo:** Wow. Now, a couple of points. One, they had the source code.

**Steve:** No.

**Leo:** They didn't have the source code.

**Steve:** That's the beauty. And they make a point of saying prior efforts at subverting voting machines have had the advantage of information that was not publicly available.

**Leo:** Right, right.

**Steve:** Like the source. They had no source.

**Leo:** Oh, man.

**Steve:** They had nothing but purchasing five of these for $82.

**Leo:** Wow. And this is one, the Sequoia machine is just one. There are other machines which also have been shown to be hacked, hackable.

**Steve:** Yes, in fact, I don't know if any have been shown not to be.

**Leo:** Yeah, I think it's the other way around, you're right.

**Steve:** I mean, it's - the problem is that, I mean, I loved this particular machine because it's the way it should be done.

**Leo:** It was smart.

**Steve:** It's a tiny little 8-bit processor. It doesn't take anything to count votes, to turn some LEDs on and show something. I mean, we had that back on our Atari videogames. I mean, it takes nothing to be a voting machine. Yet traditionally, or I should say contemporarily, we've got Linux and UNIX and Windows, god help us, and compilers and networking and so much opportunity for something that I regard as about as mission-critical as anything could ever be. And certainly the stakes are high. I mean, there are people, certainly not people we know, hopefully, but who would actually do this.

**Leo:** Oh, yeah.

**Steve:** Who would, if they could, they would actually feel that they had a moral right, or they just maybe don't even have morals. I don't know. But they would think, hey, if I can, I want my man to win, no matter what. And unfortunately...

**Leo:** Yeah. Or foreign governments or, I mean, there are people with sufficient resources to do this.

**Steve:** Yes.

**Leo:** Absolutely.

**Steve:** And that was the point that was made here, is 16 man months is nothing. And buying five of these for 82 bucks from a government auction site, I mean, these are available to private citizens. And it was from having no information about these machines, they were able to completely take them over, build their own Turing complete pseudo instruction set, then use the existing code as their own subroutines to save them from having to recode the whole thing, I mean, the machine knew how to be a voting machine. Yet they just - they shimmed themselves in and did it in such a clever way that they got around this fundamental limitation that only code in ROM could be executed. They just used...

**Leo:** That's the amazing thing.

**Steve:** ...little fragments that were already there and knit them all together.

**Leo:** That's the truly remarkable and slick thing.

**Steve:** Isn't that cool? That's, like, wonderful.

**Leo:** Yeah. And that's so slick. Wow.

**Steve:** Oh, and the other thing, they had one more thing, they had a routine in there called "pet" because there was a watchdog timer as part of this. The idea being a watchdog timer is something in many embedded computer systems. The idea is it's a hardware timer, and it makes sure that the system isn't hung or isn't misbehaving. And so what happens is, in the regular software loop that the machine is in, every so often it sort of just - it resets this timer before the timer has had a chance to time out. And when the timer times out, it also yanks on the so-called NMI, the non-maskable interrupt, to say hey, something is wrong, wake up. Because the idea would be that the machine would have frozen so that the software wasn't running. Well, while their code was in control, there was nothing ticking and preventing this watchdog, the watchdog timer

from expiring. So they had a little bit of subroutine they called the "pet" to pet the watchdog, to keep it happy.

**Leo:** [Laughing] It just shows you how smart people are. I like it that these guys are good guys, not the bad guys.

**Steve:** Well, yes. And this kind of news helps to keep our politicians aware, and certainly our citizenry aware, that we really need to decide how we're going to move forward. Are we going to allow voting machines to be insanely complex so that no person on earth can testify to their security? The new ones are that way. I mean, no one can testify to Windows or Linux or UNIX's security. We know on this podcast we're constantly bombarded with serious problems in these systems because they're so complex. Complexity is the enemy of security.

So I say something super simple like this, which you then open up to everybody and let guys like this use this level of cleverness and say, oh, we found a way around this. And we'll end up with something bulletproof in no time. The problem is we're still a profit-based system, and we've got independent companies saying, oh, no, our voting machines are completely secure, and you have to pay us this much money to get them. And we're not going to tell you what's inside.

**Leo:** Yeah. Well, I just have the feeling that we shouldn't be doing electronic voting of any kind.

**Steve:** I really do like the idea of just filling in the bubbles with your No. 3 pencil and running it through a scanner. And then you've got - and you keep all of those. And then you do - you take them somewhere else and run them through different machines to make sure that the numbers come out the same.

**Leo:** Although I suppose any counting machine can be subverted. So aren't we at risk no matter what?

**Steve:** Yeah. I mean, it's...

**Leo:** I mean, anything can be subverted. So maybe the key is just to really do this, which is do these kind of studies and get the word out and work harder on making this stuff good. The code seems like it shouldn't be so complicated that you could write it without a buffer overrun. I mean...

**Steve:** I agree.

**Leo:** ...it can't be that complicated.

**Steve:** I agree.

**Leo:** So maybe we just need to concentrate on coding it better.

**Steve:** It is the case that it was our old friend the buffer overflow in one routine where it would have normally never caused a problem, where they said, ah, we can get a foothold. And now we know by using little fragments of existing code that we can knit ourselves into existence. That's just wonderful.

**Leo:** And failing that, they would have no way in. Right?

**Steve:** Correct. They could have said - they would have sat there frustrated, saying, well, we've developed a complete pseudo machine language using - by the way, this notion of, like, a whole list of subroutines, that's called "threaded code." Threading is what, like, for example, the language Forth was a threaded compiled code system where the actual program was just a series of subroutine calls. And in this case these were little micro subroutines, a few little instructions at the end of existing subroutines that they were able to figure out how to build a complete machine from.

But you're right. Without a foothold, they would not have been able to get that. Although the system was clearly designed without this notion in mind. In the paper that these guys wrote, they made the point that 20 years ago, in '88, when this was first designed, I mean, nothing wrong with it being 20 years old. I'm here to tell you that sometimes good stuff is the secure stuff.

**Leo:** Yeah.

**Steve:** Back then, no one had heard of the so-called "return-oriented programming." That's the - there's this notion, it's called "return to libc" is sometimes how it's known, or return-oriented programming. No one had ever thought about that. No one had come up with this idea. So 20 years later this concept arose. And so the researchers made the point that something like a voting machine, with a useful lifetime of X decades, its security has to stretch throughout its entire lifetime. They made the point, for example, that cartridges containing RAM would have had to be a certain physical size back then. But now we could put a whole supercomputer system in what was a RAM cartridge back then. And so they had to be robust against that kind of evolution in the fundamental computing technology, too.

**Leo:** Wow.

**Steve:** Just a cool story.

**Leo:** Yeah, really great, really great. Fascinating stuff. Well, thank you for bringing it to our attention; you know? Wow. And I guess there really isn't much moral to be taken from this except that we have to test these things very, very thoroughly before we even consider using them.

**Steve:** Yeah, exactly. I would say skepticism is, you know, healthy skepticism is what you want to have. Again, the idea that these guys are performing this kind of reverse engineering, creating a demonstrable vote-stealing result, using machines which are currently in service in the United States for elections, that's got to hit the radar screens of politicians and the committees that decide what's allowable conduct, what's allowable for the technology we're going to be using for voting.

**Leo:** Great stuff. What a good show. What an interesting story. Next week we answer questions.

**Steve:** Yup.

**Leo:** So if you've got questions about this or other subjects or want to raise some issues, I know a number of people in the chatroom were saying, well, is IE8 really less secure than Firefox? So I'm sure we'll get some questions about that. Here's how you get back to Steve. You just go to GRC.com/feedback and fill out the feedback form.

**Steve:** Please do.

**Leo:** Steve will collect your questions, and we'll have some answers next week. You can also, when you're visiting GRC, of course, get some great stuff. There's all sorts of wonderful useful security utilities - ShieldsUP!, Wizmo, DCOMbobulator, Shoot The Messenger, and of course Steve's bread and butter, don't forget the great SpinRite, the world's best disk maintenance utility, available right there, and only there. Right?

**Steve:** Hopefully not at your local flea market.

**Leo:** Not sold in any store. GRC.com. We probably should have been saying this all along. Do not buy it from a flea market.

**Steve:** Well, you may get more than fleas.

**Leo:** Yeah. Steve, thanks so much. Great to talk to you. We'll talk again next week.

**Steve:** Thanks, Leo.