SECURITY NOW!

Transcript of Episode #203

Boyer & Moore

Description: Leo and Steve explore the invention of the best, and very non-intuitive, means for "string searching" - finding a specific pattern of bytes within a larger buffer. This is crucial not only for searching documents but also for finding viruses hidden within a computer's file system.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-203.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-203-lq.mp3

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

Leo Laporte: Bandwidth for Security Now! is provided by AOL Radio at AOL.com/podcasting.

This is Security Now! with Steve Gibson, Episode 203 for July 2, 2009: Boyer & Moore. 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 your security, your privacy, protecting you online. And here's the guy, just the guy to do it, Steve Gibson of the Gibson Research Corporation, GRC.com, creator of SpinRite, discoverer of spyware. I feel like it's a Barnum & Bailey intro. Creator of SpinRite, discoverer of spyware, world traveler, the world renowned...

Steve Gibson: And you know, Leo, I've heard this before.

Leo: Once or twice.

Steve: Well, 203 times, actually.

Leo: Well, I think it's important, if somebody's listening - hard to believe - for the first time...

Steve: Yeah, you're right. You're absolutely...

Leo: ...that they know who I'm talking to. Maybe it's just an old, you know, TV thing. But I always feel like I have to introduce my guests. But...

Steve: Well, it certainly does let people know that if they've got complex iPods full of stuff, that they've found the channel that they were hopefully looking for. Or they've found the wrong one, depending upon what they're in the mood for. So that works.

Leo: Yes, yes, yes. So today we have one of two segments we're going to do. We'll have a break in between with a Q&A. But two segments we're going to do on, what, fundamental computer science topics.

Steve: Yeah. There are - the reason I thought of this was that I was actually implementing this particular amazing idea in the DNS benchmark the other day. And I just was thrilled again by how cool this is. It's a way of finding a substring in a buffer. And...

Leo: Now, before you say any more, why would you want to find a substring in a buffer?

Steve: Well, okay.

Leo: What are you talking about?

Steve: Has anyone ever searched a document for a string?

Leo: Sure.

Steve: That's what this is. This is string search. But it bears directly on our audience and the topic of Security Now! because how would you imagine you scour your system for viruses? You're searching memory. You're searching your files for known, you know, we talk about patterns or fingerprints or a snippet, a known snippet of a virus. You want to very quickly check your whole hard drive, essentially, your file system for something. Well, there's the way you would expect to do it, and we'll talk about that. And then these two guys named Boyer and Moore, Bob Boyer and J. Strother Moore, at the time they were at SRI and PARC, both up in Silicon Valley in Northern California. And in 1975 they said, we have a better way. And it's just, I mean, it just curls my toes, it is so cool. And I just, when I was thinking about this, I thought this is something that our audience would just get the biggest kick out of because I can explain it conceptually, and it's one of those things where people are going to go, oh, my god. I mean, you would have never thought of it. But here it is. And it's like, oh, it's so much better than, like, the dumb way.

Leo: You know, I love algorithms. Maybe that's a nerdy thing to admit. And there have been big, long books written on it, including the classic Donald Knuth books. Which are really, I guess, in a way logic puzzles. They're inventions. They're creations. And something like this, which can make a huge difference in something that every program does all the time, has a huge impact.

Steve: Well, yeah. And computer science has, like, it happened when we invented computers. And then over the years people have discovered things. They've invented really cool solutions to common problems. In two weeks we're going to talk about data compression as, like, there's two guys also, interestingly enough, whose initials are LZ.

LEO & STEVE: Lempel and Ziv.

Leo: I know them, yes, yes.

Steve: And so LZW is Lempel-Ziv-Welch; LZH, Lempel-Ziv/Huffman. Many people built on that. But there was again, just as with Boyer & Moore, these two guys had this really cool idea for, like, this is how we can compress data. And it works better than anything that had been done before. And people have improved on it. But there's still this fundamental concept. So we're going to have a couple episodes here of just really neat conceptual stuff. And as you say, Leo, this stuff is sort of fundamental computer science. It's actually why I like writing in Assembler is I'm constantly doing algorithms of various sorts. Not amazing ones all the time. But I've got my own Boyer-Moore string searching routine that is in the server at GRC, checking for various things in incoming data and also in the DNS benchmark that'll be happening soon.

Leo: So let's get this errata and updates out of the way before we get to Boyer & Moore.

Steve: We've got a bunch of security news. Nothing horrifying, although this first one is really annoying because of a real problem with terms and terminology. It turns out that Adobe Shockwave has a pretty significant remote code execution, remote people-take-over-your-machine problem. The problem is that this is not Flash, which is what is most common. This is sort of the original Shockwave that is authored by their Director tool, which of course they all - Adobe bought all of this from Macromedia. So there was Macromedia Director and Macromedia Shockwave.

Leo: Yeah. Right.

Steve: There was also Flash. And the problem is that the way this stuff identifies itself in your computer is very confusing. For example, you'll see Shockwave Flash, and okay, wait a minute, is that the Flash or is that Shockwave? Is that the problem or not?

Leo: Right.

Steve: And so, for example, under Internet Explorer, IE has the facility for managing its add-ons. You click the gear and then select Manage Add-ons. And this is IE8, which is of course the current one and the one Microsoft is now pushing on people. So people who are at 7 or 6 are probably there deliberately, so I'll assume they're a little more expert. Anyway, you find the Manage Add-ons feature. And in IE8 it's got them listed, broken out by publisher. So then there's a heading, Adobe Systems Incorporated. And Shockwave Flash Object was what mine showed. And I thought, oh, there's Shockwave. But that's not the one that you need to worry about. You need to - normally IE only shows the loaded add-ons. You need look for, to select the list box for all the installed add-ons. Then it'll sit there and think for a while and give you a much bigger list. Scrolling down through all of the Microsoft debris which is not currently running in memory, you'll get down to, like, another Adobe section, likely. And so what you're looking for is just Shockwave, but not Shockwave Flash. What Adobe's site says is you need to uninstall this, then restart your computer before you install their update. So they have an update. Where we're trying to get is...

Leo: That's annoying. You mean, oh, you have to uninstall, then reinstall?

Steve: Yeah. That's what they say. And what I found was that I didn't have it installed. But apparently 60 percent of PCs and Macs do, and there's some 450 million - no, it's got to be 45 million, 45 million instances of this.

Leo: So not everybody who has Flash has Shockwave.

Steve: Correct. And so fundamentally, I mean, backing up, what I ended up doing was then I went to, under the Control Panel, after sort of getting myself confused about which one we were talking about because the nomenclature is very confusing, I went to the Control Panel in Windows to Add/Remove Programs, and there was a collection of little Adobe things. Normally it's sorted alphabetic, so they'll be near the top. And so I saw Adobe Flash and Adobe ActiveX, I think they were, and nothing else. When I installed the update from Adobe, then there was a new item in Add/Remove which was Adobe Shockwave. And I think under Firefox there was a reference to Director, like Shockwave for Director. Anyway, what I realized after all of this was I never had this installed in the first place.

Leo: [Laughing]

Steve: So then I uninstalled it, using Add/Remove Programs to uninstall it. And it's like I was back where I was. But at least now I know that I'm not one of these 45 million people who had installed this. So it may very well be that our listeners won't have it. But for anyone who does, apparently it will have been something that you installed manually. That is, it's not sort of where you go to a site and it says, well, you need Flash, click here to install it, and it just sort of does it for you. So that means that it has a presence in Add/Remove programs, at least in Windows, where you can remove it if you have it. And maybe you just ought to leave well enough alone after removing it and don't go install the update unless you know that you need it for something.

Leo: There are some pages that do say you need Shockwave.

Steve: Yes. And they're generally, I mean, they're...

Leo: They're older, I think.

Steve: They're older, and they're sort of - Shockwave and Director were sort of almost more like gaming. I mean, it's a high-end authoring package, not just like Flash, which is sort of like Silverlight for doing animated little simple lightweight things. It's like much more content is what the original Shockwave Director is used for authoring. So it's not something that I had installed, and now I'm sure. So I wanted to let our users know.

Anyway, where Adobe is now is at 11.5.0.600. And 11.5.0.596 and earlier are vulnerable. So if you've got Shockwave, which is again different from Flash or Shockwave Flash or however your system identifies it, then it is worth doing because this is one of those things where there's a large enough target size for the bad guys. If 60 percent of PCs and Macs have this installed, I mean, that seems like a high number given that it seems relatively uncommon and unnecessary. I've had my system running now for all year, and I have not run across a site that needed it. On the other hand, I'm not surfing as broadly as many people are. So I wanted to let our listeners, now that everyone's completely confused, know what's going on.

Leo: Yes. Well, that's good to know.

Steve: And, by the way, you go to get.adobe.com/shockwave, if you decide that you do need to update to the latest. So it's get.adobe.com/shockwave.

Leo: All right.

Steve: Google's Chrome browser has a new problem. It turns out that, if you have a deliberately maliciously formed reply to an HTTP request, so that is to say if you went to - if something got you to click on a link that went to a malicious server, and technical exploit details are available publicly, then the server can craft a response which will take over your machine. So that's not good. Chrome currently has about a 1.8 percent usage share on the 'Net. So less than one in 50 people are using it. Anything prior to Chrome 2.0.172.33 is vulnerable. So essentially that means if you're using Chrome, you just want to make sure you're using the latest one. Updates are available that fix this problem. So that's worth doing.

And we've also talked - we've talked recently about Foxit as the alternative to Adobe's increasingly large PDF reader. People, a lot of our listeners, I know, use the Foxit reader. Well, there's an optional add-on that has a new problem. It's the JPEG2000/JBIG2 add-on. Which is not...

Leo: Isn't the JBIG2 the same problem...

Steve: We've been having problems with that recently. It's not installed by default. You may not have it as part of your Foxit installation. But if you had installed it, you want to make sure that you're using something later than 2.0.2009.303. 303 and earlier are vulnerable of this JBIG2 or JPEG2000 add-on. And again, something tricks you to viewing a PDF document with one of these images embedded, and it can be a malicious image which will trip an overflow that exists in that add-on and execute code of the bad guys' design. So you want to make sure if you're using that add-on that you are later than 3.3 or earlier.

Leo: Okay.

Steve: And in errata, I did want to observe an interesting headline from CNN which was late last week, relating to Michael Jackson's death. CNN - this is the technology section of CNN.com. The headline was - it just said: "Jackson Dies, Almost Takes the Internet With Him."

Leo: Yeah, I saw that. I thought that was a great headline.

Steve: And I don't know if anyone knows this, but there was such a flurry of people jumping on the 'Net when they heard through word of mouth or through whatever channel that Michael Jackson had died, Google thought it was a Distributed Denial of Service attack.

Leo: I'm sure they didn't think that for too long. But initially that spike must have scared them.

Steve: Yeah, there was a huge spike on Google News. And apparently it's one of the largest mobile computing events. That is, it was people with their iPhones and cell phones and all their radio-connected...

Leo: Refresh, refresh, refresh.

Steve: ...cellular mobile devices, all wanting to find out what happened. The L.A. Times website crashed. And Twitter's servers suffered multiple crashes as everyone was twittering each other or tweeting or whatever.

Leo: That's how I learned about it. I learned about it on Twitter. And then my son texted me. And then my daughter called me. So, yeah. I'm sure - that was just me, I had a lot of traffic going on.

Steve: Exactly. And so you can imagine. So anyway, this is sort of an interesting commentary on the, I mean, the Internet obviously didn't die. But it definitely suffered a blow when suddenly something happened that caused a lot of people to all at once, in a relatively short timeframe, use the 'Net in a relatively focused way, people all tweeting or twittering or whatever the verb is you use. The L.A. Times being, of course, where this happened. At UCLA was where Jackson was taken, I guess. UCLA Medical, I think, is where they were piling in to find out what was going on. So the L.A. Times was where people naturally went. Maybe they were following links from Google News, who knows. But anybody in that chain suffered from the event because, as we've discussed before, the networks and even servers are generally sized to be able to handle the traffic they're normally receiving plus some amount of variation because you get daily variation. But not 100 times that much. And when that much hits, they collapse under that strain.

Leo: Well, because you don't design a site to always be able to handle that traffic. It would be just too expensive.

Steve: Oh, exactly. I mean, exactly. You would have servers sitting around doing nothing.

Leo: Excess capacity, yeah.

Steve: Exactly. And they'd have to be ready at a moment's notice. So they have to be on, burning power, heating up the air, so you've got to cool it off. And you'd have all this bandwidth that you're not using. So, I mean, it absolutely is the case, I mean, we know for example that the phone system will collapse if everyone goes off hook at the same time. It just doesn't have the capacity for dealing with that. When normal loads occur, and people go off hook, they get dial tone. If everyone does, nobody gets dial tone.

Leo: Now, there are, it's interesting, there are services you can buy now, kind of bandwidth-on-demand services that are designed to handle this. I don't know how well they handle this. But they kind of ramp you up. You don't pay for it all the time. But when you need it, it ramps up to scale to handle it.

Steve: Well, and the way they work is, again, they work on the law of numbers. They work on the idea that they'll have many customers who need the ability to have high, it's called "burstable," high burstable bandwidth. But it's unlikely that all of their customers would be needing to burst at the same time.

Leo: Right, right.

Steve: And so they say to people, okay, we're not going to charge you that much, but you're going to have this high burstable limit. They say to everybody that is their customer the same thing, presuming that no one is - that various customers have, for whatever their demographic profiles are, they're different from each other, so they're not going to be bursting at the same time. Once again, if by some weird coincidence many of their customers tried, they'd again hit the wall because no one can afford really, really high levels of bandwidth on an ongoing basis. It's just too expensive.

Leo: Isn't that interesting, yeah. And we had talked before about that's how people can insure themselves against DDoS attack, that they can buy bandwidth on demand so that if they get DDoSed they can - that's one way to handle this, to have so much bandwidth that the DDoS doesn't stop you.

Steve: Well, in fact that's a perfect model of what we were just saying. There are suppliers who say we will give you DDoS protection. So they've got all of their customers who are behind much bigger pipes, and they're sharing the daily cost among all those customers who want to make sure, if someone tries to attack them, they'll be prevented. And the goal or the hope is that not all of that company's customers would be attacked at once.

Leo: In this case, the traffic was huge everywhere.

Steve: Yeah. And it was legitimate traffic. It was just too much of it.

Leo: Right, right.

Steve: Yeah. I did have an interesting and sort of short little SpinRite anecdote from a Michael Barber, who wrote saying "SpinRite Saves My TiVo Without Finding Errors." He said, "For the past few weeks, when I was watching TV, it would cut out and go all pixilated. For months I assumed that it was just the digital cable since that does happen from time to time. But lately it's been getting worse, much worse.

"One day I tried to go back to the recorded list, and the screen was frozen. No amount of button-pushing would work. I rebooted the TiVo - it's a Series II - and it hung on rebooting. I reset it again, and it came up that time. At this point I knew it was the drive and not the cable causing the pixelation, and my drive was dying. So I took the 160GB drive and took it out of the TiVo and put it in my PC and ran SpinRite on level 3. It said level 4 was going to take 25 hours. And three hours and 30 minutes later at level 3, SpinRite was done. I was a bit confused because SpinRite didn't identify any bad spots on the drive. Regardless, I put the drive back in my TiVo, and all has been well ever since. No more pixelation. Thanks for writing SpinRite and hosting Security Now!, two truly invaluable tools."

Leo: Isn't that interesting. Does that make sense?

Steve: Yeah, it's often the case. It's one of the things that is a constant annoyance, frankly, for me is that SpinRite's working with the drive, and there's really nothing that I can report because the drive's relocation of defective sectors is deliberately invisible.

Leo: Oh.

Steve: I mean, it's even invisible to SpinRite. SpinRite is able to cause the drive to see that it's got a problem reading a sector, which is still correctable but getting almost to the point of not being correctable. That's what causes the drive to say, oh, shoot, we've got to fix this now. So the drive fixes it. But there's no error at any point. This is normally supposed to be automatic, but some drives need SpinRite's help. Or the sector is coming back unreadable often.

Well, SpinRite keeps asking for the same sector. It does a whole bunch of different things like moves the head in different, in either direction random distances and then comes back at that sector. That causes slight different alignment of the head during each attempt. And so the idea is that SpinRite will sit there working with the drive, continually asking for this sector. And it comes back bad, bad, bad, bad, bad. Just once it'll be correctable. And then the drive says, ah, we got the data. It uses that opportunity to swap in a spare, and everything's fine.

So there's really nothing for SpinRite to report except, hey, your drive's all fine now. Put it back in your TiVo, and everybody'll be happy. So there weren't errors caused. There weren't bad sectors because they were fixed. So I know it's difficult to explain; and people say, well, I don't know, it didn't do anything, but it works now.

Leo: But it fixed it. It didn't do anything, but it fixed it.

Steve: Exactly. It's like, well, yes, it did, but there was really nothing I could tell you.

Leo: Right.

Steve: Except, you know, be glad you have SpinRite.

Leo: Sometimes just poke - in other words, sometimes just poking the drive and checking every sector gets the drive to fix the problem itself.

Steve: Well, yes. In the TiVo the drive was returning errors. So TiVo was saying, oh, I can't boot.

Leo: TiVo remapped it.

Steve: Well, no.

Leo: No. The firmware did.

Steve: It was seeing that the sector was bad. SpinRite doesn't give up.

Leo: I see.

Steve: And so - and SpinRite has a bunch of strategies for helping the drive get one last good read. It just needs one last correctable read. And then the drive at that point is so freaked out that it says, oh, my god, look how - I had a burst error that was the maximum I'm able to correct. Thank goodness I can correct it. So it did. But that stimulates it then to spare the sector out and bring in a new replacement where it puts the data that it corrected back in. So there's no - at no point is there anything that is like a bad sector because we fixed it. But we really did fix it. I mean, work got done even though SpinRite said well, you know, this took three hours and a half, but look, you've got something for your time and trouble.

Leo: And that's another point is that there are bad sectors all the time on hard drives that they fix you don't even know about. I mean, that's just part of the process.

Steve: Yes. Well, in fact, exactly. So the degree of badness is always being tolerated. Once upon a time in the old 20MB drives they would list the known defects on a label on the front, and you hopefully put those in when you were low-level formatting the drive. Many people didn't. So one of SpinRite's features back then was it would find the defects and mark them out for people. You know, many OEMs were just slapping the drives in and sending them out the door because they didn't have time to do all that. Or they didn't have people trained to do that. So SpinRite would find the defects and mark the sectors bad and protect you from yourself.

Now what's happening is the densities are so high that almost all sectors have some degree of correction. There's some problem with them, but it's not bad. And the drives are designed to tolerate that. It's when they get bad that you have a problem. And so what can happen is the sector can go from not very bad to really bad, that is, like, unreadable. That's when you need SpinRite to come in and sort of back it away from unreadable, get one last good read, which allows the drive to say, whew, thank goodness we got that. And then it's able to put a spare in.

Leo: Very interesting.

Steve: So it's all just science.

Leo: Now, let's get Boyer and Moore in here.

Steve: Okay. So this is time for everyone to sort of take a deep breath.

Leo: [Taking deep breaths]

Steve: I'm going to describe something as clearly as I can, that I think will be clear. But it's going to require some visualization.

Leo: Okay.

Steve: So maybe set your propellers to half speed. It's not too hairy, but it's just so conceptually beautiful that I needed to share this with our listeners. I think our listeners will get a kick out of this.

So here's the task. We have a big block of data we'll call the "buffer." So I'm going to make sure I use my terminology correctly so I don't get people confused. So we have a buffer of data which are characters, we'll call it English characters, in ASCII, which is the standard encoding for characters where each character is a byte long. And we have what we'll call the "pattern." The pattern is another much shorter string of characters, and we're trying to find any locations where that pattern occurs in the buffer. And as we said at the top of the show, an example of this is any time you search for something in a document. The document would be the buffer of this larger collection of a string of characters. And even though on the page we see them as going across and then down and over and across, in normal reading order, internally it's really just one long stream of characters.

And so we're looking, when you do a Find in a word processor document you're saying, okay, find this string, that is to say the pattern, within - somewhere within this buffer. And of course then the other very important use is this is how you search your file system for malware. You have the so-called - we know that we're getting updates of so-called patterns from our antivirus vendor constantly, and that we're having our system checked for these known malicious patterns. So it's very important for all kinds of reasons to be able to do this. And we would like to be able to do it as quickly as possible.

So anybody sitting down, sort of learning programming for the first time, and is given the task, okay, how do I find this short pattern anywhere within this much bigger buffer? And so intuitively you say, okay, well, let's see. I look at the first character of the pattern. And a good place to start would be to just sort of scan forward, looking at each character in the buffer, for an instance of the first character of the pattern. Because of course if you're going to have the whole thing match, then that means that all the characters have to match, and it makes sense to start with the first one. So you just sort of - you would scan forward through the buffer looking for the first instance of the first character of the pattern. When you find it, this is a - it's like, okay, hey, I found the first character of the pattern. This is a possible match at this point.

So what you would do is you then compare successive characters in the pattern and in the buffer, one for one, looking to see if they all match. And if you get to the end of the pattern, that is, all the characters have matched the pattern and the buffer until you've got to the end of the pattern, then you've obviously found a match of the pattern in the buffer. If you get, like, partway through the pattern, and the pattern's character corresponding doesn't match in the buffer, it's like, whoops, well, we thought we had something. We got a few characters in, but we didn't make it all the way through the pattern. So we've got to keep looking. So you just keep moving forward till you find another instance of the first character of the pattern in the buffer and then compare successive characters until you see whether you've got them all. And so that's, I mean, that's sort of the straightforward, this is how you would think you'd find a substring, a pattern in a buffer.

Leo: That's, you know, if you sat me down and said how would you do it, that's the intuitive way to do it, the obvious way to do it.

Steve: Yeah. I mean, and many, many, many, many people...

Leo: I've done it. I've written routines to do that.

Steve: Yes, exactly. That's the way you would think to do it. So in '95 these two guys, Bob Boyer and J. Moore, his middle name is Strother, J. Strother Moore, were collaborating in Northern California. And they're both computer scientists. Boy Boyer was at SRI, Stanford Research Institute; and Moore was at PARC, the famous Xerox Palo Alto Research Center. And they, being algorithm guys, came up with a substantially better way to do something this simple, I mean, this obvious. And it's massively cool. What they said was...

Leo: Should I interrupt you here and say, "Save this," while I mention Nerds On Site?

Steve: I think you should do that. We'll come right back.

[Commercial break]

Leo: So we understand kind of the problem. And I think it's pretty obvious...

Steve: Obvious solution.

Leo: Yeah, I mean, that's how I would do it. But clearly this uses a lot of cycles. And I think there's some magic coming to get this down into fewer cycles. I'll tell you, in a way this was the discipline in the early days of computing when you had little memory, slow CPUs, small hard drives. You had to optimize. And in some ways I think it was better then. I mean, Bill Gates constantly was - every pioneer I've talked to - you, Gates, Steve Wozniak - in Woz's case it was parts. They're always trying to pare it down to the most efficient. And there was a joy in getting the fewest lines of code or the fewest parts.

Steve: Or the design of the Apple II. I mean, Woz is a friend. And I was just, when I looked at the design of the Apple II, it was so elegant. I mean, what they did with so few parts was just amazing.

Leo: And we've lost a little bit of that now because there's so much RAM and such big hard drives.

Steve: Oh, we haven't lost a little of it, Leo. It's just gone.

Leo: I think probably - I would bet in hardware design you have it because you still have to keep those costs down.

Steve: True.

Leo: So you're still trying to reduce the chip count. Not to the degree that Woz had to. I mean, they had hocked Steve Jobs's VW bus and sold Woz's HP calculator to fund Apple. So they didn't have a lot of money. But I still think that some of that in hardware design; but you're right, in coding, you know, save a few cycles on a substring search? Bah. Fortunately this stuff is around, and people use - I presume people use it in their libraries. So, all right. I love this. This is the - I think we're coming to the aha moment; right? Where you did the obvious. You did the basic intuitive thing. But as is often the case in computer science and algorithms, the first thing that comes to mind is not the best thing that comes to mind.

Steve: Well, and this happened in 1975. You know, computers had been doing string searches for 20 years before then.

Leo: Wow. Wow. So everybody had been doing it that way.

Steve: A lot of people had been writing the normal kind of string search.

Leo: Character-by-character string searches, yeah.

Steve: Start at the beginning and march down until you see that they don't match.

Leo: And start over.

Steve: And then start again. And so...

Leo: But you could see how that would kind of algorithmically increase in CPU usage because...

Steve: Well, yes, it does mean you're examining every single character in the buffer looking for potential matches of the string.

Leo: Right.

Steve: So, okay. To visualize this a little more clearly, sort of think of the buffer as Scrabble tiles, like a long, long, linear string of Scrabble tiles. And the pattern is a shorter, similar sort of little string of Scrabble tiles. And so what we've been talking about is we've been talking about sliding this shorter pattern of Scrabble tiles along the long buffer of Scrabble tiles. And what we're trying to do is we're looking for anywhere as we slide along that they all match up, that the adjacent tiles of the pattern exactly match those in the long buffer. And so intuitively, to save on - in terms of like an algorithm, as we were saying before, you just slide along till the first tile matches up with the one adjacent, the pattern and the buffer, and then you check along to see if they all matched up, in which case, bingo, we've found a match. If not, you just keep sliding along, checking the first one, until it lines up, and then you check to see whether they all do. What Boyer & Moore hit on was don't look at the first tile, look at the last one. And...

Leo: Okay, now, I'm trying to - okay. All right. So this is a huge insight.

Steve: Oh, well.

Leo: Start at the end.

Steve: Start at the end of the pattern rather than at the beginning. Why does it matter? Because if you...

Leo: You know, this is interesting because I'm looking in Volume 3 of Donald Knuth's "Sorting and Searching." And I think this came out before Boyer & Moore. So it's not in here. This is 1973, and you say Boyer & Moore was right at 1973?

Steve: '75, and they published in '77.

Leo: Ah, so it's interesting. So this is the classic algorithm book.

Steve: Yes, and Knuth does have his own search, which is a forward-oriented search.

Leo: Right.

Steve: So you're right, this was after that.

Leo: Isn't that interesting, wow.

Steve: So, okay. So...

Leo: Start at the end.

Steve: Start at the end. So we've got our first little pattern of tiles lined up at the beginning of the buffer. We look to see whether the last tile matches the buffer. And presumably, I mean, chances are it's not going to. Well, here's the key. So the last tile didn't match. But the question is, what did it not match against? That is, what was the tile in the buffer? And here's the insight, is if that tile that did not match doesn't occur anywhere in the pattern you're searching for, you can jump that pattern all the way down by its length. Because think about it. If the tile at the end doesn't match, and it doesn't occur anywhere in the pattern, then as you slid the pattern along one tile at a time, you couldn't ever get a match in any of the intermediate positions because we already know that the tile that was in the buffer where the last tile of the pattern was doesn't occur anywhere in the pattern. So there's no point in sliding it along one tile at a time. Just jump the entire pattern down by its length.

Leo: Oh, of course. So that, if the pattern's long, could save you a huge amount of time.

Steve: You've got it, Leo. This is a huge win. The longer the pattern is you're looking for, the faster this goes, rather than the slower it goes. Which is what you would think, you know, which is what you would end up with starting at the beginning and moving down. So you like longer patterns. Now, okay. So what happens if the tile in the buffer does occur somewhere in there? Well, the way this is implemented in code is a table is created that is where every entry in the table corresponds to a character. So, for example, if we're matching bytes, then we've got - we know bytes are eight bits. And so we would have eight bits' worth of entries in this table, which is to say 256 entries. And most of those entries would probably represent characters that did not occur in the pattern we're searching for. So we fill them all with the length of the pattern. That is because - so the idea is, if we find that at the end of the pattern, we look that character up in the table, and it says jump the whole length of the pattern. So the entry in the table tells us - it's called a "jump table," not surprisingly - how far we can jump the pattern forward.

But say that we had a pattern we were searching for that had "E" a couple times. Well, if when we're checking a match the last character in the pattern doesn't match up with what's in the buffer, but what's in the buffer is an "E," well, what we want to know is, what's the minimum distance that we slide the pattern to cause the last "E" in the pattern to line up with the "E" in the buffer. And so the idea is with Boyer-Moore you build this table before you start. Before you do any string matching, you build this table. You first fill it all with the length of the pattern. That is, you fill all the entries in the table the length of the pattern. Then you go through, and you quickly scan the pattern from the back to the front. And as long as you haven't changed that entry in the table, you put in the distance that that character occurs from the front of the pattern.

So, for example, if "E" were four bytes from the end, that says then you're able to slide the pattern forward four characters that will cause the "E" in the buffer to line up with the "E" in the pattern. The point is, that's the first chance that the pattern might all match in the buffer is if you slid it forward four. So what this means is that, from an algorithmic standpoint, searching for a string is as simple and fast as you check the last character in the pattern and the buffer to see if they're the same. If not, you use the character that you found in the buffer to look in this table for how far you can jump. Most of the time you're going to be able to jump the whole distance of the pattern because the chances are that character doesn't occur in the string. Sometimes you can't jump the whole distance because the character you found at the end does occur one or more times in the pattern. But you can still instantly compute by looking it up in this table how far to jump to line up that character with the last instance of the character that occurs in the pattern.

So everything is this table lookup, which is extremely fast with computers. They've got instructions just for doing table lookup. And so the idea is you then jump forward by however much, and you start at the end again. As soon as you have a mismatch, you look in the table how far you can jump. Most of the time you can jump the entire distance of the pattern. And that's this amazing insight that these guys came up with, which everybody uses. It is today like the most efficient way of finding a substring in a longer buffer. And exactly as you instantly got, Leo, the longer the string you're looking for, the faster the algorithm runs.

Leo: That's the big surprise of this.

Steve: Yeah.

Leo: Yeah. In fact, now I think whenever I do grep searches on text, I'm going to have to make a longer string. I always thought, oh, I'll make it shorter. But make it longer, and it'll be faster.

Steve: And so what happens is, before you start searching, that jump...

Leo: The table is built, yeah.

Steve: ...table is built that tells you how, for every character that you find, how far you can jump. The other thing that's interesting about this is that the larger the alphabet, that is, like when we're searching for, say, some random binary data in malware, which is not - or actual executable code that is what malware is written in, uses the whole alphabet. All eight bits' worth are generally going to be occurring in binary files. Whereas in ASCII files we're using a small subset. For example, the alphabet is 26. With upper and lowercase we're at 52. With alphanumerics we're at 62, and special characters. Still we're many fewer symbols that are common than the 256 in a byte. So the other nice thing about this is the more dense the use of the alphabet, the less likely it is that the character you hit in the buffer is going to be in the string. So in general...

Leo: Right, it's more efficient.

Steve: ...you're just flying through the buffer doing your search. Anyway, it's just - it's one of those things where you can - it can be described visually. You can sort of think about it. But again, searching from the end is not what you would ever sort of just automatically come up with when someone said, hey, you know, write me a little routine to do a string search. Yet it is so powerful to start at the end because, if you get a character, you hit one that doesn't occur anywhere in the string you're looking for, there's no point in sliding all the way along all those intermediate positions. You know you can't get any matches because you already know that the character at the end doesn't occur anywhere in between. So just jump over the whole distance and then check again. And then, by using this table, you're able to quickly do intermediate jumps. When you do get a character that occurs somewhere, you jump to where it might match. But then you start again at the end. And if it's wrong you just say, oh, forget it, and jump the whole distance. It's just way cool.

Leo: Very, very interesting. Not intuitive at all. Now, sometimes with algorithms like this, I know this happens with search, that it's better at some kinds of material than other kinds of material. And there may even be places where it's counter-indicated. It makes sense that, if it's completely random data, it's better; right? Because you're going to have - you now have fewer hits.

Steve: Yes. You would certainly have - if you had, well, for example, if the string you were looking - if the pattern had many duplicate characters in its body, then you would tend to be checking...

Leo: More, right.

Steve: ...those possible alternatives. Instead of doing whole jumps, you would have a more likely chance of doing little, less-than-full-length jumps.

Leo: So is it less efficient if it's looking through ones and zeroes, then?

Steve: It's more efficient when you're taking larger chunks at a time. That is, when...

Leo: Right, bigger alphabets.

Steve: Right.

Leo: More randomly distributed would be good.

Steve: Right.

Leo: So if it were ones and zeroes, a small alphabet might not be as good in binary data is what I'm wondering.

Steve: Well, but the beauty is you're taking the binary data at a byte at a time.

Leo: Oh, okay. You're right, of course. So you are doing an alphabet, then.

Steve: Right. So aggregating in larger chunks increases the size of the alphabet. And as I was saying, the larger the alphabet, the more efficient the end result.

Leo: It's really, really cool.

Steve: It is just - it's one of those conceptual little just gems. And you can imagine Knuth saying, ooh.

Leo: Ooh, I missed that. I missed that one. Well, there's a lot of algorithms in his - you know, he's only published two volumes. I think that he's working on the third.

Steve: Actually he has three. I own the first three. But I think there's, like, there's a bunch more. And you're right, he is working on them now.

Leo: Yeah. He stopped because they were stalled out. But that's the point. I mean, if you've got thousands of algorithms in a book like this, I have - oh, I see them behind you, yeah. Is that it? There's one, two, three, four it looks like you've got there.

Steve: Yeah, but I've got two copies. And maybe you're right. I was trying to look just to see...

Leo: I think there's only two. I think there's Volume 1 and 3.

Steve: I've got two of each.

Leo: Yeah, Volume 1 and 3. I know that's what I've got. And I heard he was working on Volume 2. Donald Knuth is a professor of computer science at Stanford, and these are...

Steve: No, I've got volumes 1, 2, and 3.

Leo: There are three.

Steve: So there are three, but he's definitely working on finishing them, which those of us who are into this stuff, it's like, oh, come on, come on. I mean, those are the bibles. Just he's done such a beautiful treatment of these fundamental approaches to solving these problems.

Leo: He'd better hurry. He's in his 70s now, so...

Steve: Yeah, but he's good.

Leo: He's really good. And he also...

Steve: He can spit this stuff out.

Leo: He also created his own...

Steve: Typesetting system.

Leo: Yeah, he created TeX to do this.

Steve: Because, yeah, because there was nothing that was able to adequately - he wasn't able to express himself on paper using the computer. So he said, okay, well, I'll just have to do a sophisticated typesetting system.

Leo: And it uses kind of a pseudo language, a pseudo - it's almost Assembler language to demonstrate this. Because he didn't want to do it in any specific language.

Steve: Well, he has - it's called MIX, M-I-X, which is his own little Assembly language. And so he shows you code for these things in this little MIX language. And in fact a MIX emulator is available under GPL on the 'Net.

Leo: Oh, really. Oh, that's neat.

Steve: You're able to actually write, and it'll compile and run these little MIX programs for you.

Leo: Oh, how neat. In January - I'm reading in the Wikipedia article. "January 1990 Knuth announced to his colleagues he would no longer have an email address so he could concentrate on his work." He knew in 1990 that what we now know...

Steve: This was going to be a problem.

Leo: 20 years later, oh, yeah, email's a problem.

Steve: I don't think he's probably doing any tweeting or twittering.

Leo: He's not twittering, either, I can guarantee that. Yeah, three volumes so far, and he's working on Volume 4. And apparently updates are released to the website. So that's really cool. That's really cool.

Steve, I loved this. This is really fun. And I know next week we're going to do a Q&A segment. But the week after you want to do another one of these computer algorithm shows?

Steve: I've got one lined up, Leo. That's what we're going to do.

Leo: Good. All right. Do you want to tell us, or is it a surprise?

Steve: Well, we've got two guys once again. This time we had Boyer...

Leo: Oh, you mentioned at the beginning, yeah, yeah.

Steve: Bob Boyer & Moore; right. And next time we're going to do Lempel and Ziv.

Leo: I can't wait. I know a little bit about this one because I wrote a very early compression program for the Mac when there were no compression programs out there, using LZW.

Steve: Cool.

Leo: We will return next week. Please listen every, let's see, the show comes out Thursday. We record on Wednesdays on live.twit.tv, Wednesdays at 2:00 p.m. Eastern, 11:00 a.m. Pacific. But because I'm going to China, the next recording will not be for a couple of weeks. But we will be back on July, let's see, 18th, 19th, 20th, 21st - 22nd. I think that's a Wednesday.

Steve: Yes, it is. And the 23rd is the Podcast 206, and I've got it slated here as a mega security news update.

Leo: There'll be a lot to talk about. Let's see, while I'm in China...

Steve: We're going to have a lot to talk about, so we're just going to do that.

Leo: ...the Chinese hackers will be busy. Then we'll have something to report. Steve, always a pleasure, especially when you do this kind of stuff. Go to GRC.com. That's his website. You can get a copy of SpinRite there, the world's best hard drive maintenance and recovery utility bar none. I can say that unequivocally. Also great free stuff like ShieldsUP!, Shoot The Messenger, DCOMbobulator, Wizmo, soon some really neat new stuff coming out. Of course Perfect Paper Passwords are there, as well. And his forums, some great security forums. And that's where you can also leave questions for our Q&A sections. That's at GRC.com/feedback.

Steve: Yes, please do. The questions are definitely appreciated.

Leo: 16KB versions of these shows, transcripts, and show notes also there. And we have additional show notes thanks to our intrepid audience at wiki.twit.tv. Steve, we'll see you next time on Security Now!.

Steve: Thanks, Leo.