
# How the Internet Works, Part 2

**Description:** During Part 2 of "How the Internet Works," Leo and Steve briefly review last week's discussion of the ICMP protocol, then discuss the operational details of the Internet's two main data-carrying protocols: UDP and TCP.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-026-2.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-026-lq.mp3

---

**Leo Laporte:** This is Security Now! with Steve Gibson, Episode 26 for February 9, 2006: How the 'Net Works, Part 2. Steve, it may seem like I'm here with you, but in fact I am basking somewhere in the Mexican Riviera.

**Steve Gibson:** Wow, these satellite uplinks really are good now, aren't they.

**Leo:** On a geek cruise, MacMania 4, reading "Fallen Dragon," thanks to you.

**Steve:** Ah, yes.

**Leo:** But, hey, the podcast must go on.

**Steve:** Yup. We'll make it come - I don't know what that expression is...

**Leo:** Heck or high water.

**Steve:** Wind or rain or sleet or snow.

**Leo:** Or snow, or..

**Steve:** I think I did it. Yeah, I think...

**Leo:** Yeah, or Oosterdam cruise ship.

**Steve:** Or Mexico Caribbean geek cruise for Leo.

**Leo:** Oh, yes. So we were - fascinating last week. We were talking about how the Internet works. And this is kind of laying the foundation for things we'll be talking about as the podcast continues, in security

terms. But you need to understand things like how routers work and what IP addresses mean and stuff like that to understand the rest.

**Steve:** Well, it's - if you'll pardon the pun, it's all interconnected.

**Leo:** Yes, it is.

**Steve:** And so, you know, I've been thinking back about, you know, last week's episode. I'm really glad we took the time to lay that out because there in, what, about 45 minutes or so is really the foundation of this technology that we're all using without really thinking about it because it works so well. You know, it's been said that any really effective interface becomes transparent. And of course the perfect example is the telephone, where, you know, when you're on the phone, you're not thinking about the fact that you're, you know, making sounds into a piece of hardware. Your mind has bridged that gap, and you're just thinking to the other person. And so it works.

**Leo:** Or to paraphrase Sir Arthur C. Clarke, any sufficiently advanced technology appears to be magic.

**Steve:** Indistinguishable from magic, exactly.

**Leo:** Indistinguishable from magic. I didn't mention this last time. But if, you know - Steve's talking about the technical aspects of this. But if you're interested in the actual history of James Licklider and all the people who invented the Internet...

**Steve:** He was the first guy at MIT, yeah.

**Leo:** Yeah. Licklider actually was at ARPA, I think, or DARPA, wasn't he? I can't remember. But he was - somebody was at DARPA who said, "Look, this is not working. I have two computers on my desk, and they don't talk to each other."

**Steve:** Right.

**Leo:** And so he said, "Guys, you have an assignment. Figure out how to get these two things to talk to each other." And of course Robert Metcalfe invented the Ethernet, and it's just really been amazing, the progress we've made. So there's a great book talking about all this. Katie Hafner wrote it; it's called "Where Wizards Stay Up Late." And it's light on the technical side. You're getting that from Steve. But if you're interested in the people behind it and what happened, the events - have you read that book?

**Steve:** I haven't.

**Leo:** I'm going to have to return the favor and send you a copy.

**Steve:** Licklider was actually at MIT...

**Leo:** Was he at MIT? Oh, okay.

**Steve:** ...in the early '60s. And what he did, he wrote the first, I mean, literally, it was science fiction at the time. He called it - I love this - he called it the "Galactic Network." That was his term.

**Leo:** He understood the scope of it ultimately. And at some time - I guess it will be the Galactic Network at some point, yeah.

**Steve:** Yup.

**Leo:** So last week we covered kind of the hardware layer. Yes? Would you call it that?

**Steve:** Well, it's like the lower layer interconnection stuff. You know, in terms of strict networking academic approach, networks are regarded as having layers where, like, layer one is the electrical...

**Leo:** The physical layer.

**Steve:** ...wiring, the physical layer. Layer two is the transport of sort of like whatever is going to be transported on top of that. And then successive layers are sort of encapsulated. And there are, like, higher level abstractions as we go. So...

**Leo:** It's a seven-layer model, as I remember.

**Steve:** So, yeah. And not all seven, I mean, some of them are sort of strange. It's like, you know, things that don't really map well onto what the engineers actually...

**Leo:** Lots of layers.

**Steve:** Yeah, exactly, what the engineers actually implemented. But, you know, from time to time in Security Now! I'll talk about Layer 2 or Layer 3 or Layer 4, which are sort of fundamental to what's going on, and always explain what I mean by that. And in fact this is a perfect segue into what I want to talk about this week because we talked about IP, that is, Internet Protocol packets, IP Version 4 [IPv4] and Version 6 [IPv6], the idea that IP packets all have a source IP and a destination IP and a Time To Live and a protocol number which is just a byte. So we're sort of - we're out of luck if we need more than 255 or 256 protocols. But it turns out that, you know, a lot of them have fallen into disuse. They were signed but never have amounted to much, unlike ICMP, which is Protocol 1; TCP, which is 6; and UDP, which is 17.

**Leo:** That's pretty much it right there. That's...

**Steve:** Yeah. I mean, those are the workhorses, although there's, like, you know, many other types of protocols, like, you know, we talked about BGP briefly. And, I mean, there's just - it's acronym soup if we got into it. But so packets are these fundamental workhorses of the Internet. Every Internet protocol packet begins with those fundamental bits of data - a source IP, a destination IP, that is, you know, who made the packet, where's it going, how long has it been trying to get there, which is that Time To Live, and what protocol sort of tells the system what the rest of the packet is. Is the rest of this an ICMP packet? Is it a TCP or a UDP packet?

So the major workhorse protocols, as we've talked about, is ICMP, which is, you know, people know it. We talked about how that is used, for example, in the traceroute command, and how that works to allow you to trace the route that packets take from going from you to some designated remote. And, for example, you know, so an ICMP time-exceeded message is sent back when a packet expires on the Internet, back to its apparent sender, to notify that sender that for whatever reason it didn't get there. Similarly, the ping command is another example of ICMP. The ping is sort of a human version of the ICMP called "echo request." And so what that does is, when you ping a remote IP, basically you're sending an ICMP echo-request packet to that IP. Any routers along the way just forward it on, as long as it hasn't expired yet. When it gets to the router that it's addressed to, or the computer, the computer, if it's configured normally, will send back in response to an echo request, sends an echo reply, which is just a ping reply. And this is used, again, by network engineers and just, you know, regular users who know about this, just to see whether the machine is up at the other end or connected. Is there a path between me and that other IP? So it's sort of another

sort of fundamental plumbing layer which is very useful for making sure that, you know, the Internet and networking stuff is working.

**Leo:** I had no idea that ICMP was on the same level as TCP and IP. I mean UDP.

**Steve:** Well, it's interesting. It's on the same level. But, for example, ICMP doesn't have the notion of ports. Ports is what we're just about to talk about. And so, for example, ICMP - I'm sorry, TCP and UDP do have this next sort of abstraction of a port number; whereas ICMP, just exactly as you said, Leo, is on the same level as TCP and UDP and the other protocols. But in the case of ICMP, it just deals with IP endpoints and not ports. You know, there's no notion of, like, pinging a certain port at a given IP, just pinging that IP.

So in the case of TCP and UDP, those are the workhorse protocols. And of course, you know, we've talked about often in our earlier episodes of Security Now!, it came up all the time when we were talking about VPN technologies and, for example, tunneling various protocols inside of a TCP connection or a UDP connection.

Well, let's talk about what UDP and TCP are, how they work, and how they're different. Basically, just like any IP packet, there's nothing magic about a UDP or a TCP packet except the protocol number that's up in that IP header that tells whoever is sending it and receiving it and passing it along the way what type of packet it is. And so, for example, a firewall that is receiving these IP packets would inspect that header, and it might be told, you know, to drop all UDP traffic because we don't want any in here. And so it would just look at that. It wouldn't care anything else about the rest of the packet. It would check to see if it was a protocol 17. If so, goodbye. Just, you know, drop the packet, do nothing further with it.

So UDP is sort of the basic payload carrier for Internet traffic. The UDP packet is - it's sort of, if you visualize the IP packet showing a source IP, destination IP, a Time To Live, and a protocol number, I ought to also mention that there is some error correcting. There is a checksum up in these packets, too, that make sure that when the packet gets to where it's going, or even as it's moving across the Internet, that there's been no inadvertent change in the packet's data. So after that, if this packet says I'm a type 17, meaning that I'm UDP, then immediately following is a source port and a destination port because, as we've said, UDP and TCP are port oriented.

Well, now, the idea of a port is an abstraction. What it really means is that the IP address gets the packet to a given machine. Let's for a minute ignore the complexity of having a NAT router stuck in there because that really does confuse things. So imagine that you've got a computer directly connected to the Internet, and your computer had a so-called "public IP," the one that your ISP gave you. You know, a cable modem, for example, is 70.181.whatever, you know, 2.1. So it's an actual public IP. The IP packet that has the destination IP got the packet to your computer. Your computer then receives it. Now what it wants to know is, of the various applications, that is, the software applications running in the computer, which one of the many ones that may be network aware should receive the packet? You know, and so there's a bit of a confusing terminology with servers and services because UNIX guys talk about servers as being processes running in the system. But it's easy to confuse that with, like, a physical server machine, where you'd have three servers that are physical boxes, as opposed to three services running in the system. So I'm going to try to say "service" when that's what I mean.

So, for example, if a packet came into the computer that had a destination port of 25, well, by sort of common agreement, that's for the SMTP service, that is, for sending mail. So packets are - okay, let me start again. Oops. So ports are sort of - are abstractions for services running in machines. If a packet arrived that had a destination port of 25, that would tell the computer that receives it to send this packet to the SMTP service, which is listening for incoming packets on port 25. If the packet were addressed with a destination port of 110, which is for the POP3, the receiving computer would hand that packet to this service, which is listening for incoming packets on that port.

So, you know, over the years people have sort of wondered what these ports are that we're talking about. You know, they're wanting to use, like, ShieldsUP! for example, at GRC, just to scan the ports of their machines to make sure that they're all closed. What ports are, as we've seen, is nothing but a 16-bit sort of address or abstraction in a UDP or TCP or potentially another protocol which is port oriented that tells the packet once it arrives at the machine, the IP address gets it to the machine, but to sort of, like, further decide within that machine who should receive the packet.

**Leo:** So there's no physical thing. There isn't a actual port.

**Steve:** Correct. Correct. It's not like, you know, serial ports 1, 2, 3, and 4...

**Leo:** That's why I think people get confused, because there used to be physical ports.

**Steve:** Exactly. Well, exactly, because of the notion of sort of like reusing the term "port" from the traditional use of, like, a serial or parallel port, to this notion of a 16-bit number. And all that really means is...

**Leo:** It's like a channel, would be maybe a better word.

**Steve:** That'd be a very good designation. It's too bad we didn't call them "channels," in fact, because it would be - I think that's a fantastic analogy for it. So with UDP, it has a source port and a destination port, the source port just being normally randomly assigned or algorithmically assigned by the computer that's generating UDP traffic and sending it across the Internet, aimed at a given destination. So, for example, if your computer wants to look up the IP address associated with Microsoft.com, it uses the UDP protocol and generates a packet that's aimed at port 53 of your designated DNS server. So the computer sends - it forms a DNS query. And by universal agreement, your DNS server machine will have a DNS service running in it, listening for any incoming UDP, or actually in the case of DNS it works both UDP and TCP, listening in this case for any incoming traffic onto that machine's port 53. And it will then receive the packet, look up the IP address for you of whoever you've asked for, like Microsoft.com, and then return a DNS response to your computer so your browser is able to go to Microsoft, knows what the IP address is for Microsoft.

So, literally, that's all that's going on is you're saying, my DNS server is this IP address. That gets the packet to the machine. Then you're saying. within this server - because the server could have, you know, it could have a web server, it could have a mail server, it could have many other types of - there again, I broke my own rule. I'm saying "server" when I meant "service." It could have a web service. It could have an email service. It could have DNS service, any kind of service processes that are there to answer queries or provide data or whatever the specific service does. The designation of port, when it's received by the computer, just specifies which service in the machine to sort of hand that packet over to for further processing.

So that's UDP, which is a very simple protocol. There's no real notion of a connection in the sense there is with TCP, which we'll talk about right now. It's just the idea of sending data from point to point. And in fact it's the simplicity of UDP that gives it its value. DNS is carried by default over UDP because there's no overhead associated with setting up a UDP connection. Literally, I send one packet to my DNS server asking for the IP address of Microsoft.com. It sends one packet back to me saying here it is. So again, these genius designers of the Internet recognized that there were many protocols or many applications that had no need for anything more fancy. If I sent my packet toward the DNS server IP, and, for example, it got lost along the way, or if the response got lost on its way back to me, I wouldn't get an answer to my query. So my end waits a few seconds for a response. If it doesn't get it, it sends it again. It waits a few more seconds. If it doesn't get it, it sends it again. It's responsible for retrying until it gets a response. Or, if it never gets one, it'll return an error to me or the application saying we couldn't get any answer from your DNS server. Something's broke somewhere.

**Leo:** So that's a protocol to use where you can't afford to drop any packets.

**Steve:** Well, actually, that's the protocol to use where you're responsible for packets being dropped, rather than the protocol itself being responsible, which is a perfect segue into TCP.

**Leo:** Okay. So UDP does have kind of a response, but it's - see, I always thought UDP just didn't have any response at all.

**Steve:** Well, it doesn't. The protocol - and I'm glad you asked the question because I don't want to confuse people. UDP itself doesn't. It's the application using UDP that would then be responsible. And, for example, that's why VoIP, like what you and I are using right now, that's being carried by UDP because it's a lightweight protocol. It always sends the data that it's got right then. TCP, by comparison, is a much heavier weight protocol that's responsible for doing all kinds of extra work.

**Leo:** So with Skype, it's Skype's responsibility to say, oops, didn't get that packet, could you resend it?

**Steve:** Or maybe what...

**Leo:** Or maybe not.

**Steve:** ...Skype probably does is just ignore it.

**Leo:** Right, they don't care.

**Steve:** If you don't get it - see, Skype is more concerned - because we're doing voice over the Internet, Skype is more...

**Leo:** It's real-time.

**Steve:** Exactly, is more concerned with real-time. You know, so there's a little bit of an "er," you know, in the communication. It just figures, hey, these guys are talking, who cares if they hear a little blerch. We've just got to keep all this traffic moving between our endpoints.

**Leo:** You've just got to keep up. That's your only job.

**Steve:** Right.

**Leo:** Right.

**Steve:** And so what TCP does, by comparison, is - and we'll sort of back our way into this - is it does a number of things. If you send lots of small packets, TCP says, whoa, wait a minute, there's a lot of overhead per packet. Remember that we talked about there's an IP header and there's source destination, there's TTL, there's source port, destination port, there's a bunch of stuff that is overhead beyond just the actual data payload. So TCP does something where it coalesces packets into a larger packet so that, if you were trying to send a bunch of small TCP packets out of your computer, the lower networking layers would say, no no no, let's merge these together. But what that does is that delays the first packet until there's been some period of time you haven't sent one, or until you've got enough saved up that it's worth emitting a single larger packet from the system.

**Leo:** So it's concerned with overall network efficiency, not so much real-time.

**Steve:** Correct. So that's one of the things that happens immediately with TCP is you lose your real-time, you know, send what I've got right now, and let's not worry about anything else.

**Leo:** That made sense when network resources were scarce. You didn't want to waste bandwidth.

**Steve:** Correct. It was certainly more important there. The other thing that TCP does is it is an error-tolerant and sort of error-recovery, error-correcting protocol. That is, the receiving end maintains - well, actually both receiving and sending ends maintain a count of the bytes. When a TCP connection is established, the establisher sends a so-called "SYN" packet. And we've talked about SYN floods and, you know, zombies use SYN floods to attack servers. A SYN packet is sent. What the SYN packet contains - this is a TCP protocol SYN packet. It contains a 32-bit number from the connection originator, saying hi there, to the server, I want to establish a connection, and I'm going to number all the bytes that I'm now going to be sending to you,

starting with this number, with this serial number. The server sends its response, which is called a SYN/ACK. That's literally an ACK, which is acknowledging the receipt of the original SYN packet and its own SYN component, which says, fine, let's set up a communication. I'm going to be numbering my bytes, all the bytes I send to you, with this serial number. So both endpoints establish a serial number for their future dialogue. And upon receiving that SYN/ACK, the connection originator sends a final ACK to acknowledge the receipt of the server's SYN/ACK.

So this is called - I know, it's complicated. It's called a three-way handshake, a TCP three-way handshake. And it does a couple of important things. Not only does it allow each endpoint to sort of start a serialization of all the data that follows, but it also verifies that a roundtrip for each endpoint is possible. One of the weird things about the Internet, as we saw in our last episode, where you got all sort of this large conglomeration of routers, and all they know is when a packet comes in, they send it off towards its destination. Well, it's not necessarily the case that a packet going from point A to point B will flow backwards from point B to point A, that is, the responding packets will take the same path. Nothing says they have to.

**Leo:** Right. They're just another kind of packet traffic; right? Yeah.

**Steve:** Yeah, well, it might be, for example, that a certain link between routers is high speed in one direction and low speed in the other. So that, if the routers had figured that out, they might choose not to route over a lower speed direction, but route in a different direction that would get the packet to its destination sooner. So the point is that you don't know, necessarily, that the fact that you can get your traffic to a destination necessarily means you're able to get it back from the destination. So the TCP three-way handshake, because the first guy sends a packet to the second guy, who sends a packet back to the first guy, who sends his final packet back to the second guy, that three-way handshake verifies to each endpoint that, one way or another, they can exchange data. So here's an example of why the designers thought, well, what if we don't want to go through all that? Well, that's what the UDP protocol is for because there's none of that nonsense with UDP. With UDP it's like, here's my query, here's your answer. Just two packets, for example, instead of this whole connection initiation setup process that TCP has.

**Leo:** It's really kind of surprising that so few things use UDP. It seems like UDP is a good idea.

**Steve:** Yes, it's a great idea.

**Leo:** But almost everything uses TCP.

**Steve:** Well, and the reason is - God, you ask great questions, Leo. The reason is, so much work is done for programmers. All you have to do as a programmer is say, I want a connection with the server. Then down in the networking underbelly of the computer, all of this SYN, SYN/ACK stuff, and the serialization of all of that is done completely transparent to the programmer.

**Leo:** Ah. So we have lazy programmers to blame.

**Steve:** Yeah, I didn't want to use the L word.

**Leo:** You're a programmer.

**Steve:** And I use actually UDP wherever I can because it is so efficient. I have a system that I wrote that allows me to monitor my network, the complete health of my network, that is remotely located at Level 3. And I built my own UDP client-server system. Yes, it would have been easier if I'd used TCP. But I didn't need TCP because I could just send a query every couple of seconds and ask for an update from the server. So for many things it's really more efficient.

**Leo:** Is there any quantification of how much more efficient? Are we talking 5 percent, 10 percent? How much overhead is there in TCP?

**Steve:** You really couldn't - you couldn't estimate because, for example, there is a burden on the UDP system to, like, retransmit if it needs to.

**Leo:** Right.

**Steve:** And, for example, a long-lived connection won't see much overhead. That is, sure, you need to establish a connection with TCP. And so you send a couple of very small packets back and forth. Once that's done, though, then there really isn't much more. There's some per-packet overhead, but not lots. So in our TCP connection, we've got both endpoints that have given each other their serial numbers. Now they have a full duplex connection, meaning that either of them can send data to the other, anytime they want. And the beauty, from the - I'll say it - the lazy programmer's standpoint, is all you have to do is just give the operating system some stuff to send. It chops it up in packets, which by the way UDP doesn't do for you, so you have to be concerned about the size of the packets you're sending over UDP. Not with TCP. TCP chops it up in nice-size packets that are optimally tuned for your network and ships them off. It also puts the serial number of the byte which is the first byte of the packet in that packet for you.

Now, what's cool is the receiver is keeping track of this. It knows from the SYN packet that it received where the serial numbering is supposed to start. As the packets come in, it makes sure that they come in in sequence. Because one of the other weird things that can happen in this loosey-goosey router Internet is there's no guarantee that packets are not going to arrive out of order. If you're sending a bunch of packets towards a destination, they could, you know, routers have the option of, like, of getting overloading and, for example - or a link between routers might get overloaded, so that a different link would get used. Well, that might be a faster link. So packets sent later could arrive sooner at the receiving end.

**Leo:** Packets sent later could arrive sooner. All right.

**Steve:** And UDP won't detect that for you. The TCP protocol does. All hidden transparently down in the network, the receiving end is keeping track of serial numbers. If it receives a packet that is not sequentially next after the prior one it received, it'll hold onto it and wait for, like, the missing packets to get filled in. The program that's running at that end, receiving data, doesn't see any of this. If it were caring, it would notice that it hadn't seen anything for a while and might sense a delay. But TCP itself, the protocol, is doing all this for the programmer. And it's acknowledging to the sender the, like, how far along I've received your data so far. What that means is, if a packet got lost along the way, after waiting, like, enough, and there's all kinds of very complex adaptive algorithms for how long it ought to be waiting, it will be acknowledging the data that it's received. If the sender doesn't get acknowledgment of the stuff it's been sending, it will automatically resend it. So even the sender is, like, holding onto data that it has sent until it's been acknowledged as received by the receiving end. I mean, it's just phenomenal how well this system works. And again, this was designed in the early days of the Internet as part of the fundamental plumbing that we can all now just take for granted.

**Leo:** And perhaps over-designed in light of today's Internet, or no?

**Steve:** No. I don't think so. There have been, especially with TCP, there has been some evolution of the protocol. For example, no one expected that the Internet would be as big as it became. Well, bigness equates to delays. So, for example, TCP didn't used to be patient enough.

**Leo:** Right.

**Steve:** And so it had to be made to be more patient. The other thing is that these links are also very high speed, meaning that a huge amount of data can be sent in a blob to the receiver. TCP also, in the beginning, wasn't originally designed to allow so much outstanding data not to be acknowledged. Remember we were saying that the receiver is acknowledging the data that it receives as it gets it back. So as link speeds have gone up, TCP has had to evolve, again, sort of to become more adaptive, to figure out, wow, I can allow much larger chunks of data to be unacknowledged by the receiver just because I'm sending it at such a fast rate that the delay across the Internet lets me have a whole bunch of data in transit.

**Leo:** Right.

**Steve:** So, I mean, it's really - it's been further developed over time. But the fundamental concepts that were laid down from the beginning have really stood the test of time beautifully.

**Leo:** It's a remarkable system. So that's UDP and TCP.

**Steve:** And I think that really gives us a strong - with this episode and last, we have a strong understanding now of what IP addresses are, what ICMP with, like, ping and traceroute, and how that's used. And then our two real workhorse protocols: UDP for just sort of like quick short messages; and TCP, which really takes advantage of major networking technology. The so-called "TCP/IP stack," as it's called, a stack because it can be viewed as a layer of protocols, this TCP work does just an incredible job for the programmers so they don't have to worry about packets being lost, about packets arriving out of order. They set up a connection, and then the operating systems at each end deal with all of the Internet dynamics so that they know they're going to send data, and it's going to be received reliably, even if things arrive out of order or are lost in transit. It's just a beautiful system.

**Leo:** TCP/IP is what?

**Steve:** Well, the acronym, of course, is Transmission Control Protocol/Internet Protocol.

**Leo:** How does that relate to TCP?

**Steve:** Well, it's sort of the same thing. TCP is carried over the IP protocol, or within an IP packet.

**Leo:** So when we're talking about TCP, we're talking about TCP/IP.

**Steve:** Right.

**Leo:** Really. Same thing. Okay.

**Steve:** Same thing.

**Leo:** Got it. Great, Steve. Once again, you've done it again, my friend. Explained it all in very clear ways.

**Steve:** Yeah, I'm glad to have this technology in place because we'll be able now to refer to these things a little more casually, with the presumption that the people who have been following along and listening understand what we're talking about and have this background.

**Leo:** Go back and listen to 25 and 26, if you don't.

**Steve:** They're going to be core episodes, How the Internet Works.

**Leo:** Great. Of course, more information about this, and maybe even this time some illustrations, available at GRC.com/securitynow.htm.

**Steve:** Check the show notes.

**Leo:** His show notes will be there. So will a 16-kilobit version for the bandwidth impaired.

**Steve:** And of course our transcripts that people love to read.

**Leo:** Yes. Thanks, Elaine.

**Steve:** Especially, I think, for these two episodes.

**Leo:** Yeah. This is one where you might want to...

**Steve:** You probably want to read along, yup.

**Leo:** Follow along in the home version. We do thank the folks at AOL Radio for providing us with the bandwidth and for broadcasting the show on their podcast channel at AOLmusic.com. I'll be back from my cruise next week. And we will be back next Thursday with yet another fabulous episode of Security Now!.

Oh, let's not forget, folks. I want to give you a little plug, Steve. SpinRite, which is the ultimate in disk, file, and recovery tools, and a must-have for everybody, should be in your toolkit, and of course makes it possible for Steve to do these because that's his day job.

**Steve:** It pays the bills, exactly, it's my day job. And, you know...

**Leo:** Although this has become your day job, actually. I know the truth.

**Steve:** Yup. Just to plant the idea in people's minds that, you know, if they end up with a problem, SpinRite is probably the best thing ever written that can recover from hard disk trouble.

**Leo:** Yeah. If you've got hard drive troubles, SpinRite's the cure. All right, Steve. We'll see you next week.

**Steve:** Always a pleasure, Leo.

**Leo:** I'll bring you back a huarache or something.

**Steve:** I just want to hear how the book went.

**Leo:** Oh, I'll tell you.