SECURITY NOW!

Transcript of Episode #313

How The Internet Works: ICMP & UDP

Description: After catching up with a busy week of security updates, and some miscellaneous fun security news, Steve and Tom return for the second installment of "How The Internet Works" with a look at the ICMP and UDP protocols.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-313.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-313-lq.mp3

TOM MERRITT: This is Security Now! with Steve Gibson, Episode 313, recorded August 10th, 2011: How the Internet Works: ICMP and UDP.

It's time for Security Now!, not security in a few minutes, not security later, Security Now!, the show that keeps you safe online. And of course joining us the man, the myth, the legend, GRC.com's Steve Gibson. Welcome back, Steve. It's good to be with you again. I didn't think I'd be here. But Leo's got jury duty again.

Steve Gibson: Yeah, he actually, as we know, it's not that he's still waiting to see whether he has jury duty, as he was a week ago, but he actually did get impaneled, as it's called, and he's one of the 12 jurors. Or I guess he could be one of the auxiliaries or the extras. But he's on a really interesting case which he can't talk about because you're not supposed to when you're on a jury until I guess afterwards. Then I think you're released.

TOM: Yeah, right. Once the case is over, you're cleared. So we can't know anything more about it. But it sounds, from what he was able to say before he actually went, it sounds really interesting. So I'll be curious to hear about it. The judge thinks it'll take two weeks, so I'll be here on Security Now! this week and next week. And Leo will be back after that, barring some strange courtroom behavior.

Steve: And we're glad to have you. The number 13 figures oddly in this week's Security Now!. We're Episode #313; and we're also the second Tuesday, we've just passed the second Tuesday of the month, so we've got the standard Microsoft security updates, of which there were 13; and a mega patch from Adobe fixing 13 critical vulnerabilities in Flash and Air and Shockwave. So everything is 13 at this point.

TOM: Wow. That's interesting. We've also got #1 involved because not only is this Episode #1 of Year #7 of Security Now!, right, but we're also going to be talking about How the Internet Works: ICMP and UDP. And if I understand it right, ICMP is Protocol 1.

Steve: I think that's true.

TOM: Yeah. And it's also...

Steve: UDP is 17 or, no, 6, I think it is.

TOM: It's also 2011. There, I stretched another couple ones. And let's get into the security updates.

Steve: So we're seeing the same pattern that has been noted by a number of other security watchers, and that is that Microsoft is alternating the size of their security patches from large to small and large to small, month after month. Last month we had an almost-not-worth-mentioning little tiny patch month. This month we have a big one. And two months ago we had a big one.

TOM: Is that by design?

Steve: I don't know. But, I mean, it's holding. It's really odd that they'll just do a mega one and then a little one, and then a mega one and a little one. So it has been noted in the industry that Microsoft's following that pattern. And they do so again. So they've issued 13 updates which address 22 different vulnerabilities. We get the standard update to the MSRT, the Malicious Software Removal Tool, which they're continuing to refine and add signatures to month by month. And of course we know that that does a quick scan prior to applying patches because what Microsoft discovered the hard way was that patching was failing in instances where users' machines were infected with something which was interacting with their patches. So they had to add this preemptive MSRT to make sure that it was safe to change the DLLs that make up Windows because some of these the malware was written specifically to particular versions of the Microsoft prior patches. And so if anything was updated, it could cause, like, the system to break. It wouldn't be able to reboot, and users were blaming Microsoft when it was in fact the case that their system was already in bad shape, already had something that had crept into it.

In these 22 vulnerabilities, there were only two that Microsoft rated as really critical. We get sort of the standard cumulative patch for IE which sort of is a reissue of IE. And, interestingly, it was rated "critical" for versions 7, 8, and 9 of IE, but only "important" for number 6.

TOM: That's usually the opposite; isn't it?

Steve: Yeah, well, what we're seeing is, and this has all sort of been an ongoing theme for us, we often note that older software just has less problems because it's had more time to get pounded on. And so Microsoft is doing new things, introducing new code in their newer versions of IE, and some of it is going to have problems. But they're not messing with IE6 anymore, so it's sort of stabilized.

In fact, that's exactly the case with the second critical problem, which exists in the DNS server code that ships only with Windows 2003 and Windows 2008, the Server editions. It does not exist in their earlier Server editions because those servers didn't have support for something called NAPTR DNS records. This NAPTR is a new type of DNS record which, I don't know, when I look at it I just close my eyes, and I think, why don't we leave well enough alone, because it stands for Naming Authority Pointer. And it actually introduces something known as "regular expressions," or "regexes," into DNS.

Well, regexes are an amazingly powerful, but also nightmarish technology. And the idea of adding regexes into DNS queries just, I mean, it just makes my head spin. It's like, okay, do we have to just keep messing with this stuff that already works? This show is about how the Internet works, and we're moving through, taking a very, you know, take our time, slow, methodical deep look at the technologies. And DNS is one of these that has existed for decades and has been amazingly solid and resilient. Yet in pushing it forward, we're beginning to break it. We're beginning to create problems. And so...

TOM: Now, is this something Microsoft is implementing, or something that's being implemented that Microsoft is taking the lead from the folks who manage DNS?

Steve: It is an RFC, so it's a standard Internet protocol. But what happens is that several people reported to Microsoft privately that, if a malicious user were to register a domain where the domain server contained these NAPTR records, then arrange to query Microsoft's DNS server, asking it to resolve an NAPTR record, the Microsoft server would go query this other domain that contained these. And there was a standard coding error in Microsoft's implementation of this NAPTR processing, which doesn't surprise me because regular expressions are a real, I mean, you're just asking for trouble messing with regular expressions. So we've added those to DNS now, and Microsoft didn't do it completely right.

TOM: How does this fit in with DNSSEC? Because isn't DNS, aren't we supposed to be trying to lock down DNS and make it so secure? Why would we add regular expressions to it?

Steve: I know, I know. Actually, DNSSEC is essentially signing of records to prevent spoofing of them because DNS is a non-secure protocol, that is, it actually travels over UDP, which is one of the topics this week. We're going to talk about ICMP and UDP as the first two of the Internet protocols that we discover. And DNS is carried by UDP, which unlike HTTPS, which we also often talk about can be protected by SSL, also known as TLS security, there is no similar security for DNS. So it's very possible for bad guys to perform man-in-the-middle attacks on DNS, altering the DNS records as they're going out or back and forth to a client that's making a query. So DNSSEC is a means of adding that missing security to DNS. So it's different from the NAPTR records. And the good news is it has been around for a long time. And this stuff is just slow to get adopted. When you look at when these various standards are created, it's just inertia on the Internet. Well, I mean, and another example of that is IPv4 versus IPv6.

TOM: Oh, yeah, right.

Steve: 6 has been around forever. Well, not forever, but for...

TOM: Over a decade.

Steve: Yeah. We're just now saying, oh, well, I guess we really have to get around to doing this. Nobody wants to. So anyway, the point is that in something new which is adding bells and whistles to DNS, Microsoft made a mistake, and it created what they acknowledge is a critical error in Windows 2003 and 2008 DNS Server. People who are not running DNS on their Windows servers don't have a problem. So it's only if you do have the DNS service running, there's a mistake in it that can be exploited. So that gets patched.

And then aside from that there were nine other important fixes, six of which potentially had a high exploitability rating, the way Microsoft now rates these things. There was some remote code execution in the data access components. There was a similar remote code execution in Visio. So if you had Visio installed, and someone sent you a maliciously crafted Visio file, and you opened it, it could run code on your machine and so forth. So basically standard advice is keep Windows patched all the time.

And then, similarly, it's Adobe's turn. Adobe's been really quiet now for a few months. We haven't had much happening with them. But now they're just letting loose the floodgates. All of their main components are being updated: Flash, Air, and Shockwave. I put a bunch of links in the show notes for users who can go to the show notes to get these links because, if you go to Adobe.com/software/flash/about, then that will show you, that will bring up a web page that shows you your current version, and also contains links to the latest.

I had been at 10.3.181.14 for Flash, and Flash has now been moved to 10.3.183.5. So anyone who's behind does need to get themselves made current. This fixes 13 critical vulnerabilities that affect Windows, Mac, Linux, Solaris, and Android platforms, so pretty much across the board. And of course not iOS because iOS doesn't have Flash and refuses to support Flash.

TOM: And not Chrome because Chrome automatically updates itself, so you don't have to take any action.

Steve: Exactly. And I think what we've discussed before is that apparently Google has a different relationship with Adobe where they've got, essentially, their own version of Flash that they're building in and may be responsible for themselves, or may get updates directly from Adobe which they then push out in Chrome. So as you said, it's just doing it automatically by itself. Air also got updated. And I'm reluctantly using Air because I use TweetDeck, which is hosted on Air.

TOM: Uh-huh. It's the only thing I use, too.

Steve: Yeah. And I wish I didn't have to, but I do. So if you are a - actually, if you just relaunch the thing you use Air with, it's very good. I know that TweetDeck is always telling me, oh, there's a new version of Air, so let's download it and then restart TweetDeck, and then you'll be good to go. So you can simply do that. Or you can go to get.adobe.com/air in order to update yourself. It just gives you a download.

And as regards Shockwave, I'm always reminding people that they probably don't need it, that it's the kind of thing that maybe five years ago, if you wanted to, like, do Elf Bowling or something, you might have needed Shockwave...

TOM: I forgot all about Elf Bowling, yeah.

Steve: But unless you really are addicted to Elf Bowling - I think you probably aren't anymore - it's worth removing it. Now, what you can do is you can just go to adobe.com/shockwave/welcome. And I did that in Firefox, and it said, oh, click here to get the plug-in. Don't. What that means is you don't now have it, and that's better than having a version that just needs to be maintained all the time and represents one more way that bad stuff can get into your computer. So what you would like, the outcome you would like is adobe.com/shockwave/welcome, and then to be offered the plug-in, and you just say, oh, step away, and close that tab, and just know that you don't have it, and you don't need it. If it is there, then I would seriously look at just removing it from whatever browser you've installed it in, unless you know you have to have it. If you know you need it, then, yes, you do want to make sure that you're current because it shares these vulnerabilities with the other tools. So it's certainly worth doing.

TOM: And if you're really that into Elf Bowling, you probably have more problems than just your security issues at this point.

Steve: Yeah, I would say try Angry Birds.

TOM: Yeah.

Steve: Because it's hosted on Flash, and you don't need Shockwave.

TOM: There you go. All right. Let's move into the security news. And I was really excited when I saw this xkcd cartoon yesterday. Randall Munroe, who does xkcd, is really smart, really funny, and it's absolutely worth reading this every day. But as soon as I saw his cartoon for today, Password Strength, I immediately thought of you, Steve, and the Haystacks, because he's talking about exactly what you were talking about, which is you make these really complicated passwords that you can't remember, and they're actually less secure than an easy-to-remember password.

Steve: Well, yes. So it's a great cartoon. I know that I'm involved in social networking when thousands of people are sending me this cartoon. It really filled up my Twitter feed. And I was glad for it because I appreciated knowing about it. So just for those who don't, you can just go to xkcd.com today. Or, if you're not listening to the podcast today, it's #936. So xkcd.com/936, which will get you to this fun cartoon.

I have to imagine, Tom, that this was inspired, in fact, by the Haystacks page because the second frame of the cartoon talks about how 2^28 bits of entropy, or 28 bits of entropy is 2^28 combinations, which takes three days - and he's correct about that, it's like 72 hours or something - at 1,000 guesses per second. Which is exactly the number I use on the Haystacks page. And then he says, "(Plausible attack on a weak remote web service. Yes, cracking a stolen hash is faster, but it's not what the average user should worry about.)" Which is exactly the language, or a version of exactly the language I have on that page. So I'm delighted that Randall Munroe picked up on that and probably knew about it.

The only problem I have is that his math is wrong. In the first frame he talks about - he uses little squares. I mean, I love how graphical and xkcd-ish this is. It's typical for the work he does. But he's not assigning bits for entropy correctly. And he's doing it in a way that benefits the point he's trying to make, so I'm not criticizing him. Just for the sake of our listeners, if you put his example into the Password Haystacks page, it shows that you've got - where he says his example has 28 bits of entropy, I calculate it at 72.3. And so rather than it being three days at a thousand guesses per second, it's actually 1.83 billion centuries at a thousand guesses per second.

But that really wasn't the point he was trying to make. He was trying to make the point, and the cartoon does beautifully, that what we've done in trying to create bizarre passwords that are impossible to memorize is we've actually, in some cases, not come up with something that has substantially more strength than, in his case, he suggests taking four easily memorized random words from the dictionary and concatenating them. So he computes them as each having 11 bits of entropy, so he's assuming that we had a dictionary of 2,048 words because that's 11 bits, and that we randomly chose them from the dictionary to assemble a four-word sentence which is easy to remember. So 11 bits of entropy each, times four words, is 44 bits of entropy. And then he says 2^44 is 550 years at a thousand guesses per second. So that's clearly long enough, since none of us are going to live that long.

TOM: And on Haystacks it says 78.3 billion trillion centuries. So it's an order of magnitude more secure than the first password still, even when you're comparing it through Haystacks.

Steve: Now, the problem with this is that he ends up with seven - so his example is "correct horse battery staple." And that's a total of 25 characters. It's surprising how many websites won't let you use a 25-character password.

TOM: Yeah. We've talked about this before. It's so frustrating.

Steve: Yeah, yeah. So some require, like, between eight and 16. So you'd have to drop a word or two or something. So really, if you end up with a website that has a ridiculously small or a worrisomely small maximum password length, then you're really forced to expand the size of the character set. He's using all lowercase. So I would say yes, that's a good password. But we also know there are other ways to create strength.

And in fact explainxkcd.com, which apparently follows xkcd.com's cartoons one for one, like daily, he explains this and has a link to the Haystacks page at GRC, explaining that this really comes from an understanding of what it takes to make passwords strong. And of course I take the point or the position that as soon as you're forced to do brute-force cracking, length matters more than entropy, which was the theme of the Haystacks page. So just sort of a fun little coincidence on the day that we're recording the podcast. Go ahead.

TOM: I was just going to ask, if I use "correct horse battery staple," I don't have to - my immediate reaction is, well, those are four dictionary words. Wouldn't a dictionary attack find them? But the fact that there are four concatenated random words makes it so that it's harder for that dictionary attack to work?

Steve: Well, okay. So here's one of the things that's hardest to get your head around, and this is the reason that first frame in the cartoon is a little misleading, is, for example, he appends a number "3" on the end, and he gives that three bits of entropy because that could be any one of 10 digits. But the key is the attacker doesn't know that you put a digit on the end. If you said to the attacker, oh, and by the way, while you're trying to guess my password, I ended it with a digit, well, then the attacker would go, oh, thank you very much. Now I don't have to try all the lowercase alpha, all the uppercase alpha, and special characters. I'll just try zero through nine. And so in that case he's right. That would be about, actually less than, three bits of entropy. No, it would be a little bit more, actually because three bits would be eight combinations.

But the point, the key of the concept is the bad guy has no idea what you've done. And if they did have an idea, if the bad guy knew that, for example, a password was four dictionary words, then, yes, then you've restricted the domain of experimentation. But the bad guy has no idea what you've done. So the fact is, it is much easier to make a much stronger password of a certain length by adding, changing the case, and salting it with some special characters. I mean, even, for example, if you took "correct horse battery staple," and you just stuck dashes in between, or your own special joiner character that you didn't tell anyone about, that makes it radically stronger because the bad guy doesn't know what you - if you make any change to it. Because, and that was the real insight that the Haystacks page tries to bring across is that anything that you do that is going to sort of take it off the map, all the feedback the attacker gets is it either matches or it doesn't. They don't get - it's not horseshoes and hand grenades. They don't get, ooh, that was a close one.

TOM: Yeah, yeah, yeah. I think that's what our minds do.

Steve: It's getting warmer. It's getting warmer.

TOM: Yeah, yeah. We think that. We're like, oh, well, we'll get close, and then we'll start to figure it out. But I think the only weakness that I can think of in this is if somebody cracks a password in some SQL injection attack at a site that was not properly salted, and they get your format, and they want to go after you. And so they go, okay, it looks like he takes the last two letters of the domain name, and then always has the word "d0g" spelled with a zero. If they get that little extra bit of information, it would undermine this.

But otherwise, this is definitely the way to go. Cory Doctorow on Boing Boing pointed out there's a study done at the University of London showing the cost of having these complicated passwords because people can't remember them. And then they rely on the less secure questions that allow you to recover passwords and all that stuff. So easy-to-remember secure passwords would save us time, money, hassle, and all kinds of things.

Steve: Or maybe passwords that you don't need to remember.

TOM: Yeah, now, that's - I'm really interested in that. Are you ready to...

Steve: Well, I'm not ready to talk about it yet. But I will tell our listeners, because you and I were talking about it before we began recording, the thing that I have alluded to a number of times, that I have said to Leo, I'm working on something, I think it's good, but I'm not ready, I don't know yet? Well, actually for quite a while we have known. A careful analysis of it has been done. And I have a way of doing non-software encryption that is paper-based encryption. Just using a piece of paper as sort of like a custom look-up chart of a certain kind, it is possible to do really strong encryption with no technology.

And that's what I wanted was, in fact, it's called "Off the Grid" because it uses a grid, but it's also not technologically based. It uses software to create the grid, but the actual use of it uses no technology. So there's nothing to store, nothing to remember. It turns domain names into a custom matching password for that domain. So each domain will just automatically get a different secure password, which you don't need to store, you don't need to remember. You can just use this thing again, and it will always give you the same password when it's given the same domain name. So I'll have that in a few weeks, and we'll talk about how it works.

TOM: That's thrilling. That's intriguing. I can't wait for that. The only weakness there is nobody has pen and paper anymore. We're going to have to start buying some pads of paper.

All right. Ever since we talked about Portable Sound Blaster last week, I just hear dogs barking everywhere now. It's one of those, you know, you learn a new word, and suddenly you notice it everywhere? My dogs are really good. But I hear them all around the neighborhood.

Steve: Yup. I've spent a Sunday afternoon with a friend with dogs barking in other people's backyards. And I think what happens is people must put their dogs out and then they leave, so they don't realize that their dog is just bored and just sitting there trying to, like, bark to be let back in the house. It's probably the fact that when the dog barks normally, then the dog's owners let him in the house, and he's happier being around. So inadvertently they're training the dog to bark when they let them out into the backyard, but they leave and don't know.

Anyway, I had a note here because I've continued to do some brainstorming and research in the background. And one thing really interesting happened that I wanted to share with our listeners because I know that so many of our listeners have been reading "Daemon," the book by Daniel Suarez, whose sequel is "FreedomTM." And one of the things that we encounter early in the book is Matthew Sobol's technology known as "hypersonic audio," where he's able to geospatially locate a speaking voice as if it's in some location right next to you, like speaking out of the air.

TOM: They used that in "Minority Report," too.

Steve: Yes. And it turns out that's true. It actually can be done. And what intrigues me about it is, and actually it's something that I want to experiment with, with this Portable Sound Blaster project, is essentially you're able to transmit ultrasonic frequency which cannot be heard by the target. That is, for example, bird hearing falls off around 8,500 cycles per second. So anything above that, birds won't hear, like 20 KHz.

And if you amplitude modulate that ultrasonic frequency, what happens is the air, it turns out, has a nonlinear response in the face of high pressure sound waves. The individual sound waves actually heat up the air a little bit and change its temperature on a wave-by-wave basis. The rate of propagation of sound through a physical medium is a function of temperature and pressure and humidity. And so that change in temperature changes the speed of sound through the air. And what happens is the individual waves that you're emitting interact with themselves. And so what that does is it creates nonlinear propagation.

Well, we know that a slide rule is nothing but adding. Essentially it's performing addition on a logarithmic scale, that is, on a nonlinear scale. So that's the way that addition becomes multiplication. Well, we also know from trigonometry there are a whole raft of trigonometric identities. One of them says that if you multiply two sine waves together, what you get is the sum and difference of the angles. So in terms of frequency, if we send out two different frequencies, we end up getting the sum and difference of the frequencies.

So, for example, if we sent out, instead of just a 20 KHz tone, which is inaudible, if we sent out a 19 KHz tone and a 21 KHz tone at the same time, that is, subtract a thousand and add a thousand, then the difference of that is 2 KHz, and the sum is 40 KHz. Well, we can't hear 40 KHz, either, but we can hear 2 KHz. And so what this does is this creates a means for producing audio, that is, audible sound, from inaudible sound. Yet it has the same directivity as ultrasound, which is highly beamable. You can beam ultrasonic frequencies very easily, whereas you cannot beam normal sonic frequencies. So anyway, the idea is that this thing would be able to whisper to birds and scare them out of the trees. You don't have to blast them. You just make the trees seem like they're haunted.

TOM: Because the birds are like, where's that coming from? I feel like I'm in the show "Lost."

Steve: Exactly, exactly.

TOM: So, now, is it because there's interference outside of the direction, and it cancels out? Is that why you'd only hear it in that one point that you're pointing it towards?

Steve: Yeah. There's some confusion about how this happens. But some of what I've read indicates that the sound is actually reconstructed when it arrives at its target. That is, the act of it hitting something, there's something called an "audio spotlight technology" where you can aim this at a wall. And when it hits the wall, that's when the ultrasonics is demodulated into sonic frequencies. And so to anyone standing around, the wall appears to be the source of the sound, not some transducer mounted up and back behind somewhere. So it's really cool technology.

TOM: Yeah, that is. That's fantastic. All right. Well, I like hearing the updates on the Portable Sound Blaster. We're going to get to how the Internet works, but we have a SpinRite testimonial first.

Steve: Actually, yeah. We have a listener, Bob Thibodeau.

TOM: Thibodeau. I'm 97 percent certain it's Thibodeau because I knew someone who spelled it the same way and pronounced it that way.

Steve: I think you got it, Bob Thibodeau. His subject was YAT: Yet Another Testimonial. And he said, "Steve, ho hum, just another testimonial about a business saved by SpinRite." And he said, "Being an avid TWiT-er - T-W-i-T-e-r, not Twitterer, TWiT-er - I've been addicted to Security Now! since Episode #1." Oh, he says, "Long about Security Now! #50, I decided that I should invest in SpinRite for the four computers in my business since I don't have a dedicated IT department, and I'm the resident geek, and I never made time for routine tests." Actually, this is the moral of the story, but we'll get to that in a second.

So he says, "It was Microsoft's Tuesday update on May 8th when it began. I had updated three of the computers when I was in the shop updating number four. All went well until the post-update reboot. Then, nothing. I tried booting off a boot CD, but dir c:\ reported nothing. I had not backed up this computer in a little too long, and on it were critical customer files for some impending jobs. Yikes. What to do?

"After much fretting and wringing of hands, I remembered I had invested in SpinRite after hearing you telling Leo about some of the letters you'd received. I got out the CD storage case and found my copy of SpinRite which I had burned to CD, popped it in, and ran the recovery. After a few hours I came back, and it was finished. It had recovered some sectors and marked several others as unrecoverable. I held my breath and rebooted. The C:\ drive was back, and Windows seemed to start okay. But then it choked partway through with the error NTOSKRNL, the ntoskrnl.exe, missing or damaged. I'm no IT pro, but I knew that a missing NTOSKRNL was not good.

"A Windows repair or reinstall was probably in order, but I didn't want to risk losing the critical customer files. So I decided to pull the hard drive and pop it into a USB case. I plugged it into another system which was working and, voila, thanks to SpinRite, the C: directory appeared. I copied the necessary customer files and directories and reinstalled the hard drive in the system. All seemed to be in order until I went to the CD storage case to get the Windows install disk, but it was not to be found. I had the CDs for the other systems, but they were XP Pro systems, not XP Home, as this one was.

"I tried running the Windows repair, but it reported missing file ntfs.sys. I tried copying it from another system, but got the same error. Clearly, a reinstall was in order. Now if only SpinRite could recover the missing Windows install CD," which he says as a joke because of course it was missing. He says, "The rest of the story, I was unable to locate the missing CD, so I bought a Windows Home upgrade CD, and the system is up and running once again. Jobs got done and products delivered. Thanks for the great product and for all your whitehat products like ShieldsUP! and the like. Keep up the good work and the fascinating netcasts with Leo. Regards, Bob at Imprinted Specialty Products

Company."

And of course the moral here is, if he had only run the SpinRite which he had purchased from time to time, it would have prevented this from getting to the point where SpinRite was able to get the drive back, which was good, because then he was able to get his critical files off of it. But he'd pushed it a little bit past the point where it was able to bring those particular OS files back to the point that they would be able to boot SpinRite. We've heard many stories where SpinRite did make the system bootable again. In this case, it had gone too far.

So I'll just remind people, I know that many people have purchased SpinRite to support the podcast and me and my efforts, for which I'm eternally grateful. But do use it. Every so often take it out and run it because it'll keep your drives from getting into a condition where you'll wish you'd used it sooner.

TOM: An ounce of prevention is worth a pound of kernel headaches.

Steve: Especially when you've already bought it.

TOM: Yeah, exactly. Let's get to the main topic today: Part 2 of How the Internet Works. We're talking about ICMP and UDP.

Steve: Right. Four weeks ago we started in on an updated, we're going to take our time, do a real thorough look at the fundamental underlying sort of core technologies of the Internet. So last time, when we did our first episode of this ongoing series, I explained how there was this fundamental conceptual breakthrough that the pioneers of the Internet made where, instead of having a physical connection from one point to another the way we did at the time, we had leased lines where there was a telephone line permanently anchoring two endpoints that they could use for communicating. Or we would do a dial-up with a modem in order to dial into a modem pool to hook up to CompuServe or the Source or all the BBSes that existed on a much smaller scale than the big providers. But in every instance there was essentially an unbroken connection from these two points.

So the concept of changing that and going to a so-called packet-based approach was a huge breakthrough, the idea that, as we discussed four weeks ago, you would have routers which were linked to each other, and you would send your individual packets which were addressed to a destination IP address and sort of just trust or hope, really, that they would get there. Each router's job was to receive the packets, treating them all pretty much the same, and just look at the IP header in the packet, which enclosed whatever payload this was carrying. And all it would see was that it was typically version 4, which is what we've always had before, and we know that we're all in the process slowly of moving to version 6. But traditionally it's been version 4.

And really the only information, there were a few pieces of information that we talked about four weeks ago, primarily the destination IP, which was a 32-bit number composed of four bytes. The router would look in its so-called routing table and essentially just decide, when the packet came in, which one of the connections, sort of the outgoing connections that connected it to the next router should this packet be put out on, and it would send it on its way.

So I want to talk a little bit about some of the fine points today of what it took to really make that work, and two of the simplest protocols carried by the IP protocol. I used the example four weeks ago of nested Russian dolls, where one of the key concepts is a hierarchy of protocols. The beauty of that is that it made it future-proof. That is, all the routers on the Internet had to understand was the IP protocol, which was as simple as it could possibly be. It carried the version number which told it what the - and the version number was the first four bits of the first byte of the packet. So it immediately identified the format of the balance of the packet.

For example, an IPv6 IP packet has a different format because, for example, it's got a 128-bit source IP and destination IP, not 32. So the header in the packet is different for IPv6 versus IPv4. So it's the very first four bits that come in tells the router, oh, here's the form that you can expect to find in all the bits that follow. But one of the key insights that the developers of the Internet had was we are not going to worry about anything but the absolute minimum information that we need to get the job done, meaning that - and this is where this nested dolls visualization comes in is the IP packet itself is the IP header with an undefined payload. The packet doesn't care what it's carrying. The router doesn't care or even know what it's carrying. Its job is simply a little bit of housekeeping, and then forwarding the packet on.

TOM: And this is why Net Neutrality and Deep Packet Inspection really drives some people nutty, because it messes with that.

Steve: Exactly. It's beginning to break these rules. And it's the integrity of these rules which is so responsible for the Internet surviving as well as it has, and for the Internet being as apolitical, like in the true sense of politics. It doesn't like or dislike any particular traffic. It doesn't know or care what this traffic or that traffic is. It just gets it, and it sends it towards its destination.

Now, one of the problems that the designers realized they would have is the question of what's called "router loops." That is, we have this - imagine just a complex network of interconnected routers. And each router has a routing table which, when it receives a packet, an IP packet, it looks at the destination IP, and it looks in this routing table to determine a direction that the packet should be sent, that is, which of its outgoing connections to other routers take this packet towards its destination. And that's all it does. It essentially puts it in the output queue. And when there's bandwidth available, out it goes on the next hop towards its journey.

Well, the designers realized, if you had big network of these routers, it was possible for a router to make a mistake if its routing table weren't configured correctly, so that a packet might bounce in the wrong direction, that is, it might be sent out the wrong interface. And it was possible that it could come back around to an earlier router in just a network of interconnected links. So if that happened, you'd get a loop because...

TOM: Get stuck in a circle, just going to the same routers over and over again; right?

Steve: Exactly. And so the problem is, what would potentially happen is you'd have packets that would never die. I mean, very much like we have, like, malware and spyware and viruses and worms that are still out there from a decade ago, trying to reproduce, they never die.

TOM: Zombie packets.

Steve: Yeah, they're on some server in a closet somewhere that got infected with Code Red or Nimda or something. And it's just out there randomly probing the Internet, the way it has been for 10 years, and it's never going to go away. So the designers said, okay, we need expiration of packets. We want the packet to be able to get to its destination. But we need it not to live forever because that would be bad. The entire Internet would end up getting clogged up potentially with packets that never die, that just go around in circles forever and bog the whole system down.

So what they added to this fundamental outer layer, the IP layer, the outer wrapper, no matter how deep this wrapper goes, the outer layer that's handled by the routers has something called TTL, the Time To Live. And it's a byte, which we know can have up to 256 different values. And again, here's where we get brilliance on the part of these guys. Any router that receives an incoming packet, and that's what all routers do, every router that receives this packet decrements the TTL value that the packet currently has. From whatever value it comes in at, it subtracts one. If that number ever goes to zero, that is, if after subtracting one it's now zero, so the incoming packet had a TTL of one, which the router subtracts one from and goes to zero, the router simply drops the packet. It will not forward it on.

And that simple, just something that simple, that measure solves the problem of packets living forever. And in fact what the router will do is it reports - this is one problem that it reports. We talked last week about how, if routers got congested, they would not generate a report. That is, if a router was trying to forward a packet, and the buffer on the outgoing link couldn't hold any more packets waiting for transmission, it had permission, formal permission from the original designers to simply discard the packet. Well, that was one of the things that freaked out the original designers because this meant that sending traffic across the Internet was unreliable. You couldn't count on it getting there. But they said, hey, that's a consequence of packet routing. We'll worry about that later. We're just going to do a best-effort forwarding of packets across routers. And we do not want to generate more traffic in the case of congestion because that would be bad. So we're just going to drop it.

In the case of a router's packet expiring, though, the router will send back a message to the packet's originator, since the IP packet coming in has both a destination IP address where it's going to, where it would like to go to or was trying to get to, and a source IP that tells the router the IP address that generated that packet, all other things being equal. We know that there are, like, spoofing of source IPs and so forth. We'll be talking about that at length in the future. But the router will send back sort of a maintenance level packet, and that's where this first protocol that lives on top of or inside of the IP protocol comes in, that's ICMP. The router sends back a message saying "time exceeded," essentially. It encodes in the ICMP packet - there's like a type of ICMP packet and then a subtype. And so the router sends back a message saying "time exceeded," meaning that your packet didn't make it to its destination. For whatever reason, it timed out, it died. Its Time To Live expired before it got to its destination.

So this does a number of things. First of all, historically, this was really interesting because the Internet didn't used to be very large. In the beginning it wasn't global in scope. It started off just being a bunch of universities and a few government entities interconnected experimentally to see if this whole thing worked. So packets never had to jump very many times. Remember that Time To Live doesn't mean seconds. It isn't, even though it's called "Time To Live," it's not the flow of time, it's the number of routers. It's the number of hops that...

TOM: It's like a counter. It's like balls and strikes.

Steve: Exactly. Or in this - yeah, exactly. So the original operating systems were setting the TTL to a relatively low number, like 16, or maybe 32, because that was enough. The Internet didn't have - it wasn't that big. It didn't have that many routers. But as ISPs came onboard, as ISPs had their own tiers of routers, and as ISPs were connected to

ISPs, the Internet grew, and there was this notion of the Internet diameter, which is a cool concept. In the same way that a circle's diameter is the distance between the furthest points on a circle, the Internet diameter is the largest number of hops between the furthest two points anywhere on the Internet. I mean, we probably never - you never thought about the internet having a diameter. But the analogy applies.

So the idea would be someone at that location trying to send a packet to the machine at the other farthest away point on the Internet. Well, even with everything working correctly, no router loops, no routing table problems, if the operating system generating the source, the original IP packet, were setting its TTL too low, it couldn't reach the destination. And that happened. There was a period of time - and this wasn't long ago, this was maybe 10 years ago, we had Windows, and we had UNIX, I mean, the Internet was maturing - some people in some locations were unable ever to get to other websites.

TOM: They were too far away from the website.

Steve: They were too far away.

TOM: It's weird to think about that.

Steve: Isn't that neat? Yeah. And so when that was recognized as a problem, there was a quick flurry. And again, it's like the guys who originally designed these, there's, like, sort of this approach of conservatism. It's the TTL with eight bits. They didn't give it 32 because the Internet could never be four billion hops in diameter. They gave it eight, and they thought, well, that's, you know. And they initially set it to 16, so just counted down from that. So if anything was more than 16 routers away, and no one was in the beginning, then there would be a problem.

But what they realized was, oh, crap, the Internet got bigger all of a sudden, and we weren't paying attention, and our operating systems are still setting the TTL too low. So operating systems quickly changed that. And, in fact, there are some that are at 128. Many are now setting the TTL to 255, which is the maximum value it can have.

TOM: Are we ever going to run out of TTL?

Steve: It's actually a problem. I mean, if we ever had that many hops. You'd have to be probably in a very deep hole somewhere, trying to reach somebody else in a very deep hole somewhere else.

TOM: Mars.

Steve: So that you had to go many routers out to the top, then many routers along the so-called Internet backbone, and then many routers back down into another hole somewhere, in order not to be able to get there because that's an awful lot of hops. But so that's where and why this whole TTL, this Time To Live occurred. Now, one of the things which this creates, which has been a mixed blessing for ISPs, is the ability to trace the route that packets take.

TOM: Hence traceroute.

Steve: Exactly. So the way that works is normally you emit an IP packet of whatever sort with a TTL deliberately large enough to get to the other end, wherever it's going. And these days we set them to 128 or sometimes 255, and off they go. And that's all you hear about it. But we do know that any router that is responsible for expiring a packet by decrementing that TTL value to zero, it has a responsibility to send back a notice that, sorry, this thing died on the vine. We couldn't, I'm not allowed to send it any further, and I'm not going to. So I'm going to send you back a notice letting you know. And the ICMP packet that it sends back has its IP, that is, the IP address of its own interface that it uses for originating that packet back to you. So when you, the sender of a packet that died out there on the Internet somewhere, receive this ICMP time exceeded message, you get the source IP of that message is the router IP where the packet died.

Well, now, clever UNIX guys who were putting this all together in the beginning said, hey, it would be cool to be able to trace the route - and actually more than cool, it might be very necessary in some cases to trace the route that a packet takes. So let's come up with a command - initially this was in UNIX - where we will deliberately set the TTL to one and launch the packet. Well, we know what happens. The first router it hits decrements that one to a zero and goes, oh, crap, this packet died. And so it sends back an ICMP time exceeded message with its IP, which we, the entity trying to traceroute this, we print that out on the screen or record it.

Then we send a packet to the same goal, the same destination, but this time with TTL set to two. So it goes to the first router, which decrements it to one. Then it goes to the second router, which decrements it to zero, and that router now has the dilemma of being unable to forward it. So it sends back its ICMP time exceeded message with a source IP of its source IP back to us. And so clearly, by simply sending out packets successively with an incrementing TTL, we're able to get back the IP address of every router along the way that this particular packet addressed to this particular destination would take.

TOM: And not only do you get the map, but you know by the TTL number how many hops it was, independent of counting up the number of IP addresses you get.

Steve: Yep. And then you can do one other thing which is a little bit flaky, but it can be useful, and this is what software does, you can measure the length of time for that roundtrip. The reason I say it's a little flaky is that you never really can know when a packet goes a few hops out and then comes a few hops back, there's no way to individually know, like, which link might have been slow. But if you do it often, then because you are getting sort of a total loop time for a packet that expires and comes back, if there were one router that were, like, really bogged down and having a problem, and if the roundtrip time suddenly increased when you went one router further than that, that would be a way of sort of nailing the responsibility of a slow router at a specific location.

TOM: Now, is that how we get ping?

Steve: Well, ping is a little different. Ping is actually, whereas the destination unreachable message is type code 3 in the ICMP packet, the so-called "echo reply" is zero. It's like the original message. I'm not sure why echo request is 8. But the idea is, ping is a little different, but it's the same sort of, like, underlying Internet plumbing. Ping is another command that probably all Internet-savvy users know about, where you just say ping, space, and then you're able to put, for example, www.microsoft.com, and your computer will look up the IP address of Microsoft.com in the same way that your browser does and then send off a packet in that direction. What it's doing is it takes a standard IP packet, gives it the normal TTL, that is, we don't want this one to expire, we want it to actually get to its destination. And the payload of that IP packet is an ICMP packet.

So here again we get this nesting of the protocols. The IP packet contains an ICMP packet of type 8, which is echo request. And so this is the originator, just asking to verify connectivity. And thus the word "ping," which sort of comes from sonar radar, where you ping something and get back an echo, a sonar echo from the burst of sound that you sent out. This is sort of exactly the same thing in the Internet world, the idea being that, by agreement, universal agreement, all machines connected to the Internet should, when the machine itself, not programs running in it, not servers running in it or services or applications or anything else, nothing running in the machine, the operating system itself, which is hosting the so-called IP stack, when that IP stack receives a packet and looks at it to decide what to do with it, it sees that that IP packet contains an ICMP echo request right there and then, requiring no other processing, it's supposed to send back an echo reply.

And that is sort of fundamental low-level plumbing of the Internet that allows engineers the convenience of making sure that routers are routing, that links are up, that things are working, that you're able to ping the destination IP. What that says is I gave an IP - or a domain name, but oftentimes an IP if you're, like, working with the actual plumbing of the Internet, you're not looking at web domain names, you're actually looking at IP addresses - I was able to ping that IP address and got a response back. The beauty of that is it relies upon nothing else. Maybe the web server's not up. Maybe it's not answering email. Maybe, you know, all these other problems can happen. But you want to go to the lowest common denominator and determine whether your traffic, any traffic, is making it there and back. Because if it doesn't respond to ping, then we're talking a little bit about the original days because unfortunately these rules have been broken. But if it doesn't respond to ping, then that's where you start. It's like you've got to get that working first. Then you know your traffic is getting there and back, and you can then start working your way sort of up into more sophisticated levels.

The problem is both ping and traceroute have security problems. I'm guilty of popularizing the notion of computers being stealthful, of them not revealing themselves at all. And one of the things that ping does is it says, ah, there's somebody at that IP address. Well, if everybody were wearing white hats, and we were all being good guys, then this wouldn't be a problem. But it's sometimes the case, unfortunately, that bad guys are using these protocols against us, and ping can create a security vulnerability just verifying that that machine is there. And in fact you can flood a user with pings, and that's what some of the early botnets did is all they did was just ping people like crazy because all operating systems are able to use it. And so it's a simple way of just flooding a given IP with traffic that will just bury it.

TOM: Poor man's DDoS.

Steve: Exactly. Exactly. The other problem is that traceroute, because by deliberately expiring packets en route towards a destination it's possible for bad guys to map the topology, that is, the interconnectedness, through ISPs or into corporations, if every link along the way responds with its IP address back to the sender, then again people with malicious intent can use traceroute in order to get the IP addresses of intermediate routers inside of corporations which corporations may not want outsiders to have. Or even ISPs may not want outsiders to have.

So unfortunately, due to abuse of these fundamental protocols over time, rules have been broken. For example, many consumer routers now have an option of whether or not to respond to a ping because it's the router itself, the router at that public IP prior to it doing its NAT translation into a private network, it's the router there that is receiving the ping because it's the destination IP address. If the protocols were all going to be obeyed, the little IP stack in that router would respond to an echo request with an echo reply.

TOM: But unless I'm running the server, I don't want that. I don't want anybody to know my router's there. There's no reason for that.

Steve: Exactly. And, I mean, on one hand it's unfortunate that it's been abused, but it really has been. There were many, you know, years ago when script kiddies were running little botnets, and it hadn't gone sort of big-time as it has now, they would use the ping responses of people they wanted to, like, blast out of IRC chatrooms in order to see whether they'd taken someone down and, like, overloaded their router by pinging their router. So exactly as you say, Tom, it seems to me, it's unfortunate, but if the end user wants more security, being stealthful, looking like there's nobody there and not responding to a ping is the way to do that. And on a much greater scale, ISPs are now often blocking traceroute. They will suppress by configuration, they're suppressing their own routers' response to time exceeded messages. They will, if a packet expires inside an ISP, the router drops it, not sending back a time exceeded.

So people may have noticed in some cases, if any of our listeners have done traceroutes, you'll sometimes notice that you'll get back a few, like first a few hops, and then there's like a dead zone of some number of hops. And then suddenly it comes alive again. What that dead zone is, is a range of routers that have been administratively configured not to send back time exceeded messages when they expire packets en route. They won't reveal their presence. So there's just a blacked out area in a traceroute. Then it'll come alive again because they're willing to pass those time exceeded packets through their network, just not to originate them. And so then you'll continue tracerouting out till you finally reach your destination.

And the other thing I should mention is that some ISPs will block ICMP traceroutes, which is why many users who have state-of-the-art Internet probing utilities will notice that traceroutes can use other protocols. They can use UDP protocol, or TCP, that we'll be talking about in the future, because all of those are encapsulated in the IP, the outer wrapping, the IP protocol packet which is where the destination IP lives and this TTL, the Time To Live, lives.

And I should finally talk about the other purposes that the ICMP packet has. We talked about how it's - because it's sort of like your lowest level Internet engineering plumbing protocol. It is the packet where you do a ping by sending out a type 8 echo request and, all other things being equal, receiving a type 0, which is echo reply. The other thing it has is I talked about this time exceeded message. Well, that's contained within a ping type 3. But ping type 3 is sort of a generic destination, destination unreachable message. And then there's a subtype in the ICMP packet where you could have, like, the reason for its unreachable. So type 3 means there was a problem. We couldn't - something was unreachable. It might be that the network is unreachable, which is a subtype 0. The host is unreachable, so subtype 1. The protocol is unreachable, the port is unreachable, or fragmentation was needed along the way.

And then, finally, the other major type is time exceeded. Fragmentation comes up because, as we talked about four weeks ago, it is possible for a router to receive a packet of a certain size on an incoming link. But routers are sort of heterogeneously connected to a collection of other routers. It might be that the router needs to forward the packet across a network that, for whatever reason, can only handle smaller packets. That used to be the case on telephone lines with modems. You might have, for example, a chunk of a network that was bridged or interconnected using a high-speed modem or some other protocol than IP. So it would carry the packet, but it wasn't able to, by virtue of the protocol it was using, it wasn't able to carry a large packet. So the interface, the outgoing interface would be configured to know what the maximum packet size is that it's able to send.

So, again, the bright people who designed all this from day one, they recognized this could be a problem. So they designed into the outer wrapper, that original IP wrapper, the ability to packets to become fragmented, that is, where not all of the packet might be forwardable across the next link, that is, the next hop toward its destination. If that happened, the router had the ability and the permission by default to chop that packet into one or two or more pieces and then send them on in sort of bite-size pieces.

No router reassembles packets that have been fragmented. It simply forwards them on. So if a link is encountered where a packet needs to be fragmented, it will chop the packet up into however many number of pieces are necessary and send them each on toward their destination. And the router that receives it, now it sees them just like any other smaller IP packets and sends them on their way. The problem is that this does create problems for some protocols, like audio protocols, where we care about performance. Suddenly we've got, like, lots of little packets that are having to be broken up and forwarded.

It would be nice, the original engineers decided, if there was some way to probe the network to have it tell us what the maximum size packet it's able to send is. So there is a bit in the header, there's an eight-bit field of flags, just sort of general purpose flag bits, in the original IP header. And one of them is - actually I think it's four bits, come to think of it. I'm just doing this from memory. But I think it's actually just four bits. One of them instructs the router not to fragment. It's called a "DF bit" for Don't Fragment. And if that bit is set, in the same way that the TTL going to zero expires the packet and prevents the router from forwarding it, if the do-not-fragment bit is set in an IP packet that must be fragmented in order for it to move outbound from the router, the router will instead send back another one of these ICMP low-level plumbing packets saying "destination unreachable," and the reason is "fragmented needed." And the router that generates that will include in that message the size of the maximum size packet that the link it was trying to send the packet out of can handle.

So that's called the Path MTU. MTU is Maximum Transmission Unit, which is to say what is the maximum size that we're able to use from where we are to where we're trying to get to. And so what'll happen is, if we are sending a packet out that is too large for any link on its way towards its destination, we can set - and if we want to proactively discover the maximum size packet that we're able to send without causing its fragmentation along the way, we're able to do that by setting the do-not-fragment bit, which says to whatever router receives it and is unable to forward it because the link it's trying to forward it across can't handle a packet of that size because for whatever reason, the protocol that it's trying to use, then it sends an error back to us with the maximum size that link can handle.

So we receive that and go, ooh, okay. Good to know. We then send out packets no larger than that. And again, we may leave the do-not-fragment bit on until we're sure that we're able to get to the destination. That would tell us if there's any other link even smaller than what we have now. So it's a means of discovering how large a packet we're able to send from where we are to where we're trying to get to along the way, and again by receiving these ICMP sort of low-level plumbing packets for any trouble that we have of various sorts. And there actually have been instances where routers or even sometimes inexpensive consumer products have not properly handled these critical Internet plumbing problems and have messed up traffic as a consequence. So, I mean, there are some of these protocols that - some we can ignore. It's like, okay, you could say that not responding to a ping is okay, except even that has caused problems.

TOM: Am I remembering this right? Is this what took YouTube off the Internet, a problem with the routers in some country? Maybe it wasn't YouTube. But I remember something with the MTU being set by a country, and it caused huge problems, like you're saying.

Steve: Yes, it can because - and it would be a streaming media company because they really do need to establish, they can't have all of their - I was going to say their pagments being fragmented, but their packets being fragmented. They need to determine what that is. And so I think you're right, Tom. I don't remember if it was YouTube. But I'm sure that I remember that there were - if that is not handled correctly, you can end up with some serious problems that nothing will get around.

Also there was one type of server, I want to say it was an IRC server, where - but I don't know why it would have been - oh, yeah. I think IRC, well, I don't remember now. There was some server where you would make a connection to it, and maybe it was just FTP. But I thought it was a little more exotic than that. You would make a connection to it, and it would ping you back to sort of like verify that you were there. And if you didn't respond to that, it would not finish negotiating the protocol.

And so there were some instances, I remember that's one of the reasons that I thought that the original ZoneAlarm firewall years ago was clever was that they had adaptive stealthing. If somebody you were not connecting to tried to ping you, the ZoneAlarm firewall would drop the packet. But if you had an outbound dialogue with a given remote IP, and you got an ICMP echo request from them, then it would respond. And that ZoneAlarm firewall at the time was the only one that had this smart, adaptive ping response which allowed it to do a better job at creating whatever it was. I want to say IRC for some reason. I don't know why the IRC server would have been doing that. But anyway, it's been a while since I've been thinking about that.

And we have time, so I want to discuss the one additional protocol, aside from ICMP, which is another simple perfect example of nesting protocols, and that's the so-called UDP protocol. It stands for User Datagram Protocol.

TOM: Although some people call it Unreliable Datagram Protocol sometimes.

Steve: Yes. And it's actually - it's got both designations. And that's actually sort of a play on the fact that it is in fact unreliable.

TOM: On purpose.

Steve: But like the geniuses - yes, exactly. Like the geniuses who created all this in the beginning, they designed the system so that it would still work in the face of designed-in unreliability. As we talked about four weeks ago, and I reminded us earlier, if a router gets congested, it has permission to just discard packets that it's unable to route because its outbound buffers are full and there's too many packets trying to get out on a link that is congested. It's able to just drop it and say, oops, sorry, wasn't able to do anything with that.

So we have the outer layer IP packet, which contains the version number of the IP protocol, typically 4, someday more typically 6, we hope. We know that it contains some flags, like lack of fragmentation permission. It contains the overall length of the entire packet so that the router knows as data's coming in where the packet ends. We know that it contains the source IP and destination IP. We know it contains the TTL, the Time To Live, for that packet. And it contains a checksum which allows the router to verify that there has been no communication error so far as this packet is moving from hop to hop across the Internet.

There's no notion of ports. We've all heard about and talk about ports a lot. That's not in the IP packet. And again, this is one of the brilliant innovations of the originators. At the lowest level, all we care about is individual IP addresses. What happens after the packet gets there is where we begin to add a next layer of complexity. We know that the IP packet can carry this ICMP payload, which is used for low-level plumbing. Well, the next level up in complexity is it can also carry a UDP packet. And true to form, this UDP packet is the minimum necessary to add just one more little layer of complexity.

The UDP packet contains the source port, the destination port, the length of its own payload, and a checksum, followed by whatever it contains. And that's the point, anything. So what UDP adds to what we already have, which is really not much, it's just enough to get us there and handle specifying where we want to go, and handle dying on the vine if we can't get there, and handle fragmentation, just enough. What the UDP adds is some abstraction of what we want to do once we get there. And that's port numbers. Ports are nothing but 16-bit values carried in the packet. That's all they are. I mean, we talk about them like they're magic, like port 80 and port 443 and...

TOM: Port 13, 113.

Steve: Exactly. And we get email from IMAP on 143 or from POP on 110. Or we send it to SMTP on port 25. And so ports, ports, ports. DNS is on 53. So everything is about ports. But all it is, is just a number. It's a destination port where we came from which is mostly used for the sake of sending something back to us. I'm sorry, the source port. I got that wrong. The source port, where we came from, which is the source of this traffic, which is used for the sake of getting something back to us.

And then the destination port is like the destination IP. The destination IP contained in the outer IP wrapper, in the IP header, that gets us to the machine. Then, if the protocol is UDP, that says, oh, UDP packets contain port numbers. The IP doesn't. The IP header doesn't, but the UDP packet does. So what all that does is, because it contains a destination port, that tells the software running in the computer which service to send this to. So when services start up, like an SMTP, Simple Mail Transfer Protocol, service starts up in a UNIX machine or whatever server hardware it's running in, it registers itself to listen on port 25, that is, listen for incoming traffic on that port number, which essentially says, when traffic comes in - actually, technically, this is TCP protocol, which we'll be talking about next. But, for example, a DNS server listens for UDP traffic on port 53.

And so all these ports are is agreement. They're just abstractions that have been sort of universally agreed to. Servers, mail servers will listen on port 110, 143, and 25. DNS servers listen on port 53. Web servers listen on port 80 and for secure traffic on 443. And so there is this array of port numbers, the idea being that that allows a sender to identify the class of traffic, the type of traffic that it's sending, simply by saying I want to send traffic to the following IP, that is, the machine at this IP, and to the service listening for traffic on this port number.

So the port number, which is a 16-bit value, so it can have any value - actually port 0 is sort of reserved. So it can have any value from 1 up to 65535. And by convention the first 1K ports, the first 1023, since we're not counting zero, or it'd be 1024, the first 1023 ports are reserved as service ports, or server ports. And again by convention, services typically set themselves up and listen for connections on those ports. And within systems like UNIX, the user processes that are running are unable to listen on those service ports. Only services that are registered with the proper permissions are able to set up shop and listen on those lower numbered, from ports 1 to port 1023, those are reserved for that. Other user processes are able to listen on higher numbered ports. And so, for example, you'll often find, like, a system will start up an IRQ server, and it runs on port 6666, for example, and also sometimes 6667.

TOM: The port of the beast.

Steve: Exactly. And sometimes people will run an alternative web server on port 8080, which being above that 1023 boundary is up in user space, so it's not where you normally run a web server. Which is why, in order to reach it on the URL that you're using, you've got to put a colon after the URL and then manually override your web browser's normal use of port 80 and put a :8080 to tell your web browser, ah, we're going to connect to this location, but we want you to connect on port 8080 rather than on port 80, which is what normal web browser traffic would be using.

TOM: We're going to have to wrap it up, Steve. But we want to get to the important point about UDP at the end here, which is what it's good for; right?

Steve: Well, yeah. It is a general purpose, traffic-carrying protocol. So it's used for UDP. UDP is probably the thing most often it's used for. But because it doesn't have - we mentioned it's unreliable, meaning you can't guarantee it gets there. There's no mechanism for the application that generates UDP traffic for knowing that it gets there. So, for example, when DNS, which uses UDP, sends a DNS query in a UDP packet to some server's port 53, which is by agreement just the 16-bit number where the DNS server is living, if it doesn't hear a response, that can happen. So it'll send it again. And then it'll wait a little bit longer, it backs off a little bit, and then sends it again. So it'll retry until it gets a response because nothing that we've talked about guarantees delivery of UDP traffic.

That's why it's also known, as you mentioned at the beginning of this, as the Unreliable Datagram Protocol, because it's up to the application itself to deal with that delivery. One of the things that it's used for often, we're using it for right now, Tom, and that is real-time communications, audio and video, that is to say media streaming on the Internet. I'm sending traffic to you. As I'm speaking it's streaming to you over the UDP protocol, which was chosen because it is so simple. If something gets lost on the way, the audio reconstruction at the other end will try to, like, make up for that lost gap.

TOM: Or one little pixel gets dropped, it can be interpolated. And our eyes even can fix things when - if the frame rate drops a little low. So that's why it's so great for things that are not like we have to have every little piece of it. We just want it to get there fast.

Steve: Well, and kind of that boing-boing sound that our listeners will hear from time to time in this podcast, that is a lost packet. That is a packet that was either lost or delayed too long. And the codec which is reconstructing that couldn't wait any longer. It had to guess. So it tried to fill in using the audio that it already had, so that there isn't an actual dead spot. It figures that that funny little kind of that springy sound is better than just having nothing right there.

TOM: The cry of the lost packet.

Steve: So we don't have congestion control. We don't know if things are too busy. We don't know if things are missing. Oh, we also don't know if they even arrive out of order. That's one of the things that the next protocol we talk about, TCP, handles all of these problems for us transparently. The problem is it introduces overhead that can cause a problem. So that's useful for downloading files where we have to have the packets arriving and reassembled in the proper sequence or our file would get broken. When it's not important, when we really care about minimal overhead, UDP is the protocol we want. When we want the convenience of making sure that something gets to its destination exactly right, then we use TCP.

TOM: That's why sometimes you'll hear me say of sentence before end of.

Steve: Okay.

TOM: So anything else for UDP? Because we have to, I'm getting the wrap sign before we go to TWiG.

Steve: Nope, we've got it. And we will continue when we continue the series with talking about the TCP protocol, which is so brilliantly conceived, it's equal to all the brilliance that we've talked about so far. It is just a spectacular protocol.

TOM: TCP's the one people most likely have heard of because they hear about TCP/IP. Probably that and HTTP are the famous ones.

Steve: I would say it is probably demonstratable that TCP is the most used protocol in the history of Earth, as protocols go. It is THE protocol. I mean, you could argue that IP is always there because if TCP is there, then IP is encapsulating it. So, okay, technically, yes. But, I mean, in terms of a high-level protocol, TCP is it because it's all of our downloads, all of our web browsing, I mean, most of what the Internet is used for is over TCP. And we will discuss it in detail next time.

TOM: All right. Look forward to it. Thank you, Steve. I'll be back with you next week. Leo will be on jury duty for one more week. And we'll be covering another Q&A session next week. Don't forget, you can find all the things Steve does, and he does some great stuff, over at GRC.com: ShieldsUP, SpinRite, the Haystack protocol we were talking about earlier in the show with xkcd. You can get that there. Good place to test out your passwords, make some more secure passwords. Steve, always great to talk to you. Thanks for letting me sit in for Leo again.

Steve: It's a pleasure, Tom. Talk to you next week.

TOM: All right. That's it for Security Now!. We'll see you next time.