The Internet Dodged a Bullet

Description: What's the worst mistake that the provider of remotely accessible residential webcams could possibly make? What surprises did last week's Patch Tuesday bring? Why would any website put an upper limit on password length? And for that matter, what's up with no use of special characters? Will Canada's ban on importing the Flipper Zero hacking gadgets reduce car theft? Exactly why didn't the Internet build in security from the start? How could they miss that? Doesn't Facebook's notice of a previous password leak information? Why isn't TOTP just another password that's unknown to an attacker? Can exposing SNMP be dangerous? Why doesn't email's general lack of encryption and other security make email-only login very insecure? And, finally, what major cataclysm did the Internet just successfully dodge? And is it even possible to have a "minor cataclysm"? Today, we'll be taking a number of deep dives after we examine a potential solution to global warming and energy production as shown in our terrific Picture of the Week. Some things are so obvious in retrospect.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-962.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-962-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Coming up, we're going to talk about the webcams that accidentally put everybody's video in everybody else's house. Patch Tuesday is here. Well, it was last week. Steve has some notes on that. Why the Flipper Zero is being banned in Canada. And a nightmare scenario with DNSSEC that could have brought the whole Internet to its knees. Steve explains, next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 962, recorded Tuesday, February 20th, 2024: The Internet Dodged a Bullet.

It's time for Security Now!, the show you wait for all week long. I know it, I know it, one of the nine best security shows in the world, with Steve Gibson, the man in charge.

Steve Gibson: Not what I do now.

Leo: What number are we on that list, by the way? Number one, I hope.

Steve: No, no. I think maybe seven. But it wasn't ranked in, like, order of goodness. It might have been alphabetical. I think it was alphabetical, actually.

Leo: Oh, all right. Well, then we would come in towards the end, yeah.

Steve: I think "A" was like Appleby's something or other.

Leo: Security show, yes.

Steve: That's right.

Leo: Which is a wonderful, wonderful show, by the way.

Steve: Yeah. So, oh boy. No one is going to believe this, literally, because - well, you will once I'm done. But this has got to be the most underreported event that I can remember. And this sounds like hyperbole, and you know me, so it could well be. But the whole Internet was at risk for the last couple months.

Leo: What?

Steve: And a group of researchers silently fixed it.

Leo: What?

Steve: Because if it had been discovered, the Internet could have been shut down.

Leo: Wow.

Steve: It turns out there was a problem that has been in the DNS specification for 24 years. It's deliberate. And so it wasn't a bug, but it was a feature that a group of German researchers figured out how to exploit. And one receive - I'm stepping on my whole storyline here because I'm so excited by this. It's like, wow. Okay. So this is Episode 962 for February 20th, titled "The Internet Dodged a Bullet."

Leo: Not the first bullet, though; right? Didn't Dan Kaminsky save the Internet back in 2008?

Steve: No.

Leo: Oh.

Steve: Dan is a great publicist.

Leo: Oh. Was a great publicist, yeah.

Steve: I mean, and yes, the issue of DNS queries not having sufficient entropy was important. I mean, I wrote the, what the hell's it called, I forgot now, the DNS...

Leo: Cache poisoning.

Steve: The spoofability. No, the spoofability test. A lot of work in order to allow people to measure the entropy of their own - of the DNS servers that are issuing queries on their behalf. This blows that away. I mean, here I am doing it again. Anyway. Okay. So [crosstalk] questions.

Leo: Wow, I can't wait to hear this.

Steve: It is something. And Leo, be careful not to look at the Picture of the Week because this is a great one. So we're going to answer some questions. What's the worst mistake that the provider of - and I've not had too much coffee - of remotely accessible residential webcams could possibly make? Like when Lisa said, "We don't want any cameras in our house," what was she worried would happen?

Leo: Oh, yeah, yeah.

Steve: Yeah. What surprises did last week's Patch Tuesday bring? Why would any website put an upper limit on password length? And for that matter, what's up with the "no use of special characters" business? Will Canada's ban on importing the Flipper Zero hacking gadgets reduce their car theft? Exactly why didn't the Internet build in security from the start? Like what was wrong with them? How could they miss that? Doesn't Facebook's notice of a previous password being used leak information? Why isn't TOTP just another password that's unknown to an attacker? Can exposing SNMP be dangerous? Why doesn't email's general lack of encryption and other security make email-only login quite insecure? And, finally, what major cataclysm did the Internet just successfully dodge?

Leo: Yow.

Steve: And is it even possible to have a "minor cataclysm"?

Leo: All this and more from one of the best 17 cybersecurity podcasts in the world.

Steve: No, that's a different one. Mine was nine. We were in the top nine.

Leo: Oh. Well, SANS Institute says we're in the top 17.

Steve: Well, but where? And are they alphabetical?

Leo: They're in random order. But we're on there.

Steve: Or in order of love.

Leo: That's the most important thing. And, I mean, SANS Institute's pretty credible.

Steve: SANS is good. But I'd rather be in the top nine. So today we'll be taking a number of deep dives after, Leo, after we examine a potential solution to global warming and energy production - no, this is serious - as shown in our terrific Picture of the Week.

Leo: Oh, I can't wait.

Steve: And Leo, some things are just so obvious in retrospect.

Leo: Oh, wow. How could we not have known?

Steve: This is a podcast for the age. Thank god we got it in before 999.

Leo: Oh, my goodness.

Steve: Otherwise, you know, we'd have to wait into four digits.

Leo: Now let's see if Steve has overhyped his Picture of the Week. What is this going to do for us? It's going to save the world.

Steve: Solution to global warming and energy production.

Leo: Well, of course. You put light on the solar panels, and infinite electricity.

Steve: That's exactly right, Leo. So what we have for those who are not looking at the video is a picture of a rooftop with some solar panels mounted on it, as many are these days. What makes this one different is that it's surrounded by some high-intensity lights pointing down at the solar panels because, you know, why not? [Crosstalk] electricity.

Leo: This way it generates day and night, yeah. Perfect.

Steve: And my caption here, I said, "When you're nine years old, you wonder why no one ever thought of this before. Adults are so clueless!"

Leo: I bet you even knew better when you were nine.

Steve: Well, you know, it was interesting because this put me in mind of the quest for perpetual motion machines.

Leo: Exactly, yeah.

Steve: Remember those back in our youth?

Leo: Yeah, yeah, yeah.

Steve: I mean, and like even Da Vinci was a little bit obsessed with these things. There was one really cool design where you had an inner - you had a wheel with marbles riding tracks where the marbles could roll in and out, and the tracks were canted so that on one side of the wheel the marbles rolled to the outside. Therefore their weight was further away from the center so pulling down harder. And on the other side, the way the fixed tracks were oriented, the marbles would roll into the center, into the hub. So their weight was pulled - well, there it is. You just found the perpetual motion machine, exactly.

Leo: Never stops turning.

Steve: And, I mean, there were, again, I was, you know, five and interested in physics already because I was wiring things up with batteries and light bulbs and stuff when I was four. So I was, you know, I spent some length of time puzzling that out. The good news is now as an adult I don't give it a second thought.

Leo: You're saying you believe Newton's law of the conservation of energy. You just believe that to be true.

Steve: Well, the problem we have with our Picture of the Week is that lights are not 100% efficient in converting electricity into light. For example, they get warm, so you have some energy lost in heat. And the physics of the conversion are also not completely efficient.

Leo: Oh, yeah. Solar panels are no more than 5 or 10% efficient at the very most.

Steve: And there you go. So the idea would be you hook the output of a solar panel to the input of the lights. And then, you know, when the sun goes down, this just keeps going.

Leo: Keeps going.

Steve: Yeah.

Leo: Somebody has a good point, though. Maybe those lights are hooked up to the neighbor's electricity.

Steve: Well, and the only thing I could think when I was, like, trying to find a rationale, was that they might turn the lights on at night because for some wacky reason, whatever they're powering from the solar panels needs to be on all the time, or maybe they turn it on on cloudy days because, again, for the same reason. So it's sort of a sun substitute because of, you know, dumbness, because it is dumb. As you said, solar panels are way inefficient. So you're going to get much less juice out of the solar panels than you put into the array of lights which are busy lighting them up. But...

Leo: We're in a golden era for scammers. You're just going to see endless variations of this on YouTube and on Twitter and, you know, using water to power your car. And this stuff just never dies. It never dies.

Steve: Wait, does that work? I've got to try that.

Leo: No. No. No. No.

Steve: Okay. So in reporting the following story, Leo, I'm reminded of your wife Lisa's "wisely," so to speak, because this story's about Wyze, forbidding cameras of any kind inside your home. 9to5Mac's headline read: "Wyze camera breach let 13,000 customers view other people's homes."

Leo: Oh, boy.

Steve: Tom's Hardware: "Wyze security failure let 13,000 customers see into other users' homes." GeekWire: "Wyze security cam incident that exposed images to other users impacts more than 13,000 customers." And even our good old BleepingComputer: "Wyze camera glitch gave 13,000 users a peek into other homes." Now, one of our listeners, a user of Wyze monitoring cameras, was kind enough to share the entire email he received from Wyze. But BleepingComputer's coverage included all of those salient details and added a bit more color, as you might expect.

Here's what Bleeping just wrote yesterday. They said: "Wyze shared more details on a security incident that impacted thousands of users on Friday and said that at least 13,000 customers could get a peek into other users' homes. The company blames a third-party caching client library recently added to its systems, which had problems dealing with a large number of cameras that came online all at once after a widespread Friday outage. Multiple customers have been reporting seeing other users' video feeds under the Events tab" - I bet there were some events - "in the app since Friday, with some even advising other customers to turn off the cameras until these ongoing issues are fixed.

"Wyze wrote: 'The outage originated from our partner AWS and took down Wyze devices for several hours early Friday morning. If you tried to view live cameras or events during that time you likely weren't able to. We're very sorry for the frustration and confusion this caused. As we worked to bring cameras back online, we experienced a security issue. Some users reported seeing the wrong thumbnails and Event Videos'" - yeah. Whose hot tub is that?

Leo: That's not Mama walking around.

Steve: "'We immediately removed access to the Events tab and started an investigation.'" We bravely did that. Okay. "Wyze says this happened because of the sudden increased demand" - and I'll get to my skepticism on that in a minute - "and led to the mixing of device IDs and user ID mappings" - you don't ever want that to happen with your camera system - "causing the erroneous connection of certain data with incorrect user accounts. As a result, customers could see other people's video feed thumbnails and even video footage after tapping the camera thumbnails in the Wyze app's Events tab.

"In emails sent to affected users, Wyze confessed: 'We can now confirm that as cameras were coming back online, about 13,000 Wyze users received thumbnails from cameras that were not their own; and 1,504 users tapped on them. We've identified your Wyze account as one that was affected. This means that thumbnails from your Events were visible in another Wyze user's account and that a thumbnail was tapped.'" That is a confession. Somebody was looking at your video, baby. "'Most taps enlarged the thumbnail, but in some cases it could have caused an Event Video to be viewed.'"

Leo: Let me zoom in on that one.

Steve: What is that in the corner over there? Yeah, that's right. "Wyze has yet to share the exact number of users who had their video surveillance feeds exposed in the incident. The company has now added an extra layer of verification" - oh, you betcha - "for users who want to access video content via the Events tab to ensure that this issue will not happen in the future." Is that really your video you're about to look at? "Additionally, it adjusted systems to avoid caching during user-device relationship checks until it can switch to a new client library" - get rid of that old cache - "capable of working correctly, which would be convenient, during 'extreme events'" - they had in quotes - "like the Friday outage."

Okay, now, I like Wyze, and their cameras are adorable little beautifully designed cubes, you know, they look like something Apple would produce. But at this stage in the evolution of our understanding of how to do security on the Internet, I cannot imagine streaming to the cloud any content from cameras that are looking into the interior of my home. You know, maybe the backyard. But even then, you know, who knows? The cloud and real-time images of the contents of our homes do not mix. I understand that for most users the convenience of being able to log into a Wyze camera monitoring website to remotely view one's residential video cameras is difficult to pass up, you know, seductive, if you weren't listening to this podcast. And the whole thing operates with almost no setup.

You know, and Wyze's response to this incident appears to be everything one could want. You know, they've really - they've been honest, which could not have been easy. The letter that our listener shared with me, unlike the letter that BleepingComputer quoted, said that his account had not been affected. So you know, Wyze was on the ball. And they've clearly got logging working because they knew that 1,504 people did click on a thumbnail, like what is that? That doesn't look like my living room. Click. Oh, it's not my living room. Wow. Who's that?

Anyway, I should add, however, that I'm a bit skeptical about the idea that simply overloading a caching system could cause it to get its pointers scrambled and its wires crossed, thus causing it to get its users confused. If it really did happen that way, it was a crappy cache. And I'm glad they're thinking about, like, revisiting this whole thing

because there was a problem somewhere. Anyway, I do like the cameras. I'm just not going to let them be exposed to the outside world. That makes no sense. Okay.

Leo: By the way, Jason on MacBreak Weekly recommended a new Logitech camera that's HomeKit enabled, which means it doesn't send video outside your house.

Steve: Nice.

Leo: Or I guess it does probably to Apple encrypted iCloud, something like that.

Steve: And again, done right, probably.

Leo: Done right, of course.

Steve: So, yeah.

Leo: So HomeKit and security and Apple, that seems like a good choice.

Steve: Yeah.

Leo: It's too bad because the Wyze stuff is very inexpensive. I've been recommending it for years, and using it, too.

Steve: Oh, I think, in fact, you order one, they send you money.

Leo: Amazingly, yeah. It's so inexpensive.

Steve: And they're just beautiful little things.

Leo: Yeah, yeah. That's too bad.

Steve: Okay. We haven't checked in on a Microsoft Patch Tuesday for a while. Last week's update was notable for its quiet inclusion of a mitigation for what the industry's DNS server vendors have described as "the worst attack on DNS ever discovered." It seems to have slipped under much of the rest of the industry's radar, but not this podcast's. Since it could have been used to bring down the entire Internet, it is today's topic, which we will of course be getting back to later.

But first we're going to take a look at Patch Tuesday and then answer some questions and have some interesting deep dives. The DNS attack didn't happen, so the Internet is still here. Thus Microsoft doesn't get off the hook for fixing another round of problems with its products. Last week's patches included fixes for a pair of zero-days that were

being actively exploited, even though they were not the worst from a CVSS rating standpoint. That goes to a different pair which earned 9.8 CVSS ratings.

Overall, last Tuesday Microsoft released patches to repair a total of 73 flaws across its product lineup. Of those 73, five are Critical, 65 are Important, and the last three have Moderate severity. Fixed separately were 24 flaws in Edge which had been repaired in the intervening month between now and last month's updates.

The two zero-days are a Windows SmartScreen Security Bypass carrying a CVSS of only 7.6 and an Internet Shortcut Files Security Feature Bypass with a CVSS of 8.1. The first one of those two, the Windows SmartScreen Security Feature Bypass, allowed malicious actors to inject their own code into SmartScreen to gain code execution, which could then lead to data exposure. The lack of system availability could also happen, or both. Now, the reason it was only rated at 7.6 is that, for the attack to work, the bad guys needed to send the targeted user a malicious file and convince them to open it. So it wasn't like, you know, just receive a packet and it's the end of the world, which actually is what happened with this DNS business we'll get to.

The other zero-day permitted unauthenticated attackers to bypass displayed security checks by sending a specially crafted file to a targeted user. But once again, the user would somehow need to be induced to take the action of clicking on the link that they'd received. So not good. Was being actively exploited in the field. Both of these were zero-days being used. Still, this one rated an 8.1, and it's fixed as of last Tuesday.

The five flaws deemed Critical, in order of increasing criticality, were a Windows Hyper-V Denial of Service Vulnerability that got itself a score of 6.5, so Critical, but not a high CVSS; the Windows Pragmatic General Multicast, which is PGM, Remote Code Execution Vulnerability, scored a 7.5; Microsoft Dynamics Business Central/NAV Information Disclosure Vulnerability came in with an 8.0; and finally the two biggies, both getting a 9.8. To few people's surprise, we have another Microsoft Exchange Server Elevation of Privilege Vulnerability and a Microsoft Outlook Remote Code Execution Vulnerability, both of those 9.8s, both of those easy to do, both of those now resolved as of last week.

A senior staff research engineer at Tenable, the security firm, said in a statement that this was very likely to be exploited, and that exploiting the vulnerability could result in the disclosure of a targeted user's NTLM (NT LAN Man) v2 hash, which could be relayed back to a vulnerable Exchange Server in an NTLM relay or pass-the-hash attack to allow the attacker to authenticate as the targeted user. So it was a way of getting into Microsoft Exchange Server through this 9.8, basically backdoor vulnerability.

And believe it or not, last Tuesday's updates also fixed 15 - I had to, like, what, really? Yes, 15 remote code execution flaws in Microsoft's WDAC OLE DB provider for SQL Server that an attacker could exploit by tricking an authenticated user into attempting to connect to a malicious SQL server via Windows OLEDB. So 15 remote code execution flaws. It must be that someone found one and said, wow, let's just keep looking. And the more they looked, the more they found. So Microsoft fixed all of those.

And rounding off the patch batch, as I mentioned, is a mitigation, not a total fix, for a 24-year-old fundamental design flaw - not an implementation flaw, a design flaw - in the DNSSEC spec which, had they known of it, bad guys could have used to exhaust the CPU resources of DNS servers to lock them up for up to 16 hours after receiving just a single DNS query packet. We'll be talking about this bullet that the Internet dodged as we wrap up today's podcast.

But first, Ben wrote: "Hey, Steve. Love the show. I hated the security courses I took in school, but you make it way more interesting. I haven't missed a podcast from you in three years. My question was: I was recently going through my password vault and

changing duplicate passwords. I encountered a lot of sites with length and symbol restrictions on passwords, for example, no passwords longer than 20 characters, or disallowing certain symbols. My understanding of passwords is that they all get hashed to a standard length regardless of the input, so it can't be for storage space reasons. Why the restrictions? Even if I input an eight-character password, the hash will end up being 128 bits, or whatever. I would love some insight because it makes no sense to me. Thanks, Yuri." So that's his real name.

Okay. So, Yuri, you're not alone. When you look closely at it, none of this really makes any sense to anyone. I think the biggest problem we have today is that there are no standards, no oversight, and no regulation; and everything that is going on behind the scenes is opaque to the user. You know, we only know in detail, for example, what LastPass was doing, and the entire architecture of it, because it really mattered to us, and back then we drilled down and got questions from the guy that wrote that stuff. And so, you know, we really understood what the algorithms were. But on any random given website we have no idea. There's just no visibility.

Leo: You can also kind of feel how antiquated that is when you say the words "the guy who wrote it." Besides your software, SpinRite and all of that, nothing's written by one person. That's nuts. Right?

Steve: You're right. You're right. You're right.

Leo: Yeah. The fact that Joe Siegrist wrote LastPass 30 years ago by himself is amazing.

Steve: Yeah. So everything is opaque to the user. We hope that passwords are stored on a site's server in a hashed fashion. But we don't know that. You know? And assuming that they are hashed, we don't know whether they're being hashed on the client side in the user's browser or on the server side after they arrive in the clear over an encrypted connection. That much we do know because we can see that the connection is encrypted.

A very, very long time ago, back at the dawn of Internet identity authentication, someone might have been asked by a customer support person over the phone to prove their identity by reading their password aloud. I'm sure that happened to a few of us. I'm sure I was asked to do it like at the very beginning of all this. The person at the other end of the line would be looking at it on their screen which they had pulled up for our account to see whether what we read to them over the phone matched what they had on record. That's the way it used to be. Of course, as we know, this could only be done if passwords were stored as they actually were given to the server, in the clear, as indeed they originally were.

And in that instance, you can see why the use of special characters with names like "circumflex," "tilde," "pound," "ampersand," "back apostrophe," and "left curly brace" might be difficult for some people at each end of the conversation to understand. Tilde? What's a circumflex? What's a circumflex? What? So restricting the character set to a smaller common set of characters made sense back then. And we also know that in those very early days a web server and a web presence was often just a fancy graphical front end to an existing monstrous old school mainframe computer, you know, up on an elevated floor with lots of air conditioning, and that computer's old account password policies predated the Internet. So even after a web presence was placed in front of the old mainframe, its ancient password policies were still being pushed through to the user.

Today we all hope, all, that none of that remains. But if we've learned anything on this podcast, it's to never discount the power and the strength of inertia. Even if there is no longer any reason to impose password restrictions of any kind - well, other than minimum length, that would be good, you know, because the password "a" is not a good one - restrictions may still be in place today simply because they once were.

And hopefully all consumers have learned the lesson to never disclose a password to anyone at any time for any reason. We see reminders from companies which are working to minimize their own fraud, explicitly stating that none of their employees will ever ask for a customer's password under any circumstances. And we're hoping that's because they don't have the passwords under any circumstances. They were hashed a long time ago, and they're just being stored.

The gold standard for password processing is for JavaScript or WASM running on the client, that is, the user's browser, to perform the initial password qualification and then its hashing. Some minimum length should be enforced. All characters should be allowed because why not, and requirements of needing different character families - upper and lower case, numbers and special symbols - also make sense. That will protect people from using "123456" or "password" as their password. And those minimal standards should be clearly posted whenever a new password is being provided. It's really annoying, right, when you're asked to change or to create an account or change a password, and then after you do, it comes up and says, "Oh, we're sorry, that's too long." Or, "Oh, you didn't put a special character in." It's like, why didn't you tell me first?

Anyway, ideally there should be a list of password requirements with a checkbox appearing in front of each requirement, dynamically checked as its stated requirement is met. And the "Submit" button should be grayed-out and disabled until all requirements have checkboxes in front of them, thus those requirements have been met. And a password strength meter would be another nice touch.

Once something that has been submitted from the user arrives at the server, then high-power systems on the server side can hash the living daylights out of whatever arrives before it's finally stored. But since we also now live in an era where mass data storage has become incredibly inexpensive, and where there's very good reason to believe that major world powers are already recording pretty much everything on the Internet, all Internet transactions, in the hope of eventually being able to decrypt today's quantum-unsafe communications once the use of quantum computers becomes practical, there's a strong case to be made for having the user's client hash the qualifying password before it ever leaves their machine to traverse the Internet.

Once upon a time we coined the term PIE, P-I-E, Pre-Internet Encryption. So this is like that. This would be I guess PIH, Pre-Internet Hashing. But, you know, JavaScript or preferably WASM should be used to hash the user's qualifying input.

Much of that gold standard that I just described is user-facing, and its presence or absence is obvious to the user. You know, unfortunately, we rarely see that going on today. It would be necessary to reverse engineer individual web applications if we wished to learn exactly how each one operates. Since avoiding the embarrassment of breaches and disclosures is in each company's best interest, and since none of the things I've described is at all difficult to deploy today, we can hope that the need to modernize the user's experience while improving their security will gradually overcome the inertia that will, you know, always be present to some degree. So we'll always be dragging forward some of the past. But at some point, you know, everything should be catching up.

Leo: I like it.

Steve: Yup. Felipe Mafra said: "Hello, Steve. First of all, I'd like to thank you and Leo for the great show. I'd like to bring you something very interesting that recently happened on this side of the border. The Canadian Industry Minister Franois-Philippe Champagne proudly tweeted on February 8th that they are banning the importation, sale, and use of hacking devices, such as Flipper Zero, that are being widely used for auto theft. This is an attempt to respond to the huge increase in auto thefts here in Canada. Even if I believe it's good that the government is trying to address this issue, I found myself, rather than blocking the usage of such devices, it would be better if the industry was required to make things right by design.

"This pretty much aligns with last week's Security Now! Episode 960 regarding security on PLCs" - you know, Programmable Logic Controllers - "as we see no commitment from those industries to make their products safe by design. Anyways, I wanted to share this with you and get your perspectives. Thank you again for the show. Looking forward to 999 and beyond. Best regards, Felipe."

Okay. In past years we've spent some time looking closely at the automotive remote key unlock problem. What we learned is that it is actually a very difficult problem to solve fully, and that the degree of success that has been obtained by automakers varies widely. Some systems are lame, and others are just about as good as can be. And we've seen even very cleverly designed systems, like as good as they could be, fall to ingenious attacks. Remember the one where a system was based on a forward rolling code that was created by a counter in the key fob being encrypted under a secret key, and the result of that encryption was transmitted to the car.

This would create a completely unpredictable sequence of codes. Every time the unlock button was pressed, the counter would advance, and the next code would be generated and transmitted. And no code ever received by the auto would be honored a second time. So anyone snooping and sniffing the radio could only obtain code that had just been used and would no longer thus be useful again.

So what did the super-clever hackers do? They created an active attack. When the user pressed the unlock button, the active attack device would itself receive the code while simultaneously emitting a jamming burst to prevent the automobile from receiving it. So the car would not unlock. Since that happens when we're too far away from the car, and it's not that uncommon, the user would just shrug and press the unlock button again. This time, the active attacking device would receive the second code, emit another jamming burst to prevent the car from receiving the second code, then itself send the first code it had received to unlock the car. So the user would have to press the button twice. But they just figured the first one didn't make it. The second one unlocked the car.

By implementing this bit of subterfuge, the attacker is now in possession of a code that the key fob has issued, thus it will be valid, but the car has never seen it. And it's the next key in the sequence from the last code that the car did receive. It is diabolically brilliant, and I think it provides some sense for what automakers are up against.

From a theoretical security standpoint, the problem is that all of the communication is one-way, key fob to auto. The key fob is issuing a one-way assertion instead of a response to a challenge. What's needed to create a fully secure system would be for the key fob's unlock button to send a challenge request to the car. Upon receiving that request, the car transmits a challenge in the form of an unpredictable value resulting from encrypting a counter. Again, the counter is monotonic, upward counting 128 bits, and it will never repeat during the lifetime of the universe, let alone the lifetime of the car or its owner.

So upon receiving that unique challenge code sent by the car, the key fob encrypts that 128-bit challenge with its own secret key and sends the result back to the car. The car, knowing the secret kept by its key fobs, performs the same encryption on the code it sent and verifies that what the key fob has sent it was correct. Now, I cannot see any way for that system to be defeated. The car will never send the same challenge, and the key will never return the same response. And no amount of recording that challenge and response dialogue will inform an attacker of the proper responses to future challenges. If some attacker device blocks the reception, the car will send a different challenge. The key will respond with a different reply. And once that reply is used to unlock the car, the car will no longer accept it again.

So the only problem with this system is that now both endpoints need to contain transceivers capable of receiving and transmitting. Previously, the fob only had to transmit, and the car only had to receive. So transceivers add some additional cost, though not much in production since both already contained radios anyway. But what this does mean is that a simple software upgrade to the existing hardware install base will not, and cannot, solve this problem. I doubt it's possible to create a secure one-way system that's safe against an active attacker while still reliably unlocking the vehicle without unduly burdening its user.

The system I've just described is not rocket science, it's what any crypto-savvy engineer would design. And since this problem is also now well understood, I would be surprised if next-generation systems which fix this in this way once and for all were not already on and off the drawing board and headed into production. But that doesn't solve the problem which exists, and will continue to exist, for all of today's automobiles.

So now let's come back to Felipe's point about Canada's decision to ban the importation of devices such as the Flipper Zero. We know that doesn't solve the problem. But will it reduce the severity of the problem? Yeah, probably somewhat. Kits will spring up to allow people to build their own. Canada is a big place. There's nothing to prevent someone from firing up manufacturing and creating homegrown Flipper Zeros or their like. It's an open-source device. I mean, like the design is all there. What we keep seeing, however, is that low-hanging fruit is the fruit that gets picked and eaten. And many people won't take the time or trouble to exert themselves to climb a tree to obtain the higher hanging fruit. Hand 'em a piece? Sure. Work for it? Perhaps later. So I would argue that making car theft even somewhat more difficult will likely be a good thing. And the Flipper Zero is, at best, a hacker's gadget. It's not as if it has huge non-hacker applications.

Leo: No, but it's a lot of fun.

Steve: It is a lot of fun.

Leo: And I was going to use it on my car, and then Russell said don't because you could actually lock yourself out of the car because the car's security features will see that you're doing it and will prevent you from using your regular fob after that. So I declined. But I was able to get into the office. I was able to clone our key fob and use it.

Steve: Oh, yeah. It is a - I did a little bit of brushing up on it yesterday. It is a very cool device.

Leo: I gave mine to Father Robert. So it's now in Italy.

Steve: You gave it a good home.

Leo: I think he's really going to get a lot of use out of it.

Steve: And actually, you know, from a hardware-hacking standpoint, all of the little GPIO pins along the back...

Leo: Oh, it's really cool.

Steve: It's very, very cool.

Leo: It's a great device. I think you could duplicate it with an Arduino or any - a variety of other devices, as well.

Steve: But as we've seen, packaging counts.

Leo: Sure.

Steve: Like remember the picture of that TPM buster that we talked about last week, where it had the little row of pogo pins along one side.

Leo: Right.

Steve: It just looked adorable. And it's like, wow, that's very cool.

Leo: It's cute.

Steve: So Leo, let's take a break, and then what are we going to talk about next? We've got, oh, we're going to talk about why the Internet didn't start off having security in the first place.

Leo: Oh. You know, I interviewed, I guess it was Vint Cerf, the Father of the Internet, back in the day. And I asked him, you know, why didn't you think of putting crypto in? And he said no one knew. We just didn't - he said we would now. We now know how people use the Internet. But at the time - anyway, I'm very curious to what you have to say about this one. All right, Steve. On we go.

Steve: So M-Scott tweets: "Steve, I'm wondering about your thoughts. The cybersecurity community seems to bemoan the lack of security baked into the original Internet design and ceaselessly encourages designers of new technology to bake in security from the get-go." Well, we certainly agree with the second half of that. "Several books I'm reading for a cyber and information warfare class suggest that government

regulation to require security is the answer and should have been placed on the Internet in the initial design. However, I suspect, if security had been a mandate on day one, the robust cyber community we have today would not exist. I see the Internet as more of a wicked problem where solutions and problems emerge together, but cannot be solved upfront. Your thoughts? Thank you for your continued service to the community."

Okay. I suppose that my first thought is that those writing such things may be too young to recall the way the world was when the Internet was being created. I'm not too young. I was there, looking up at the IMP, the Interface Message Processor...

Leo: Oh, wow, yeah.

Steve: ...a big imposing white cabinet sitting at Stanford's AI lab in 1972, as the thing known at the time as DARPANET was first coming to life. The problems these authors are lamenting not being designed-in from the start didn't exist before the Internet was created. It's the success of the Internet and its applications that created the problems, and thus the needs, we have today. Also, back then we didn't really even have crypto yet. It's nice to say, "Well, that should have been designed in from the start."

But it wasn't until four years later, in 1976, that Whit Diffie, Marty Hellman, and Ralph Merkle invented public key crypto. And a huge gulf exists between writing the first academic paper to outline a curious and possibly useful theoretical operation in cryptography, and designing any robust implementation into network protocols. No one knew how to do that back then, and we're still fumbling around finding mistakes in TLS decades later. And one other thing these authors may have missed in their wishful timeline is that the applications of cryptography were classified by the U.S. Federal government as munitions.

Leo: Oh, yeah.

Steve: In 1991, 19 years after the first IMPs were interconnected, Phil Zimmermann, PGP's author, had trouble with the legality of distributing PGP over the Internet. Today, no one would argue that security should always be designed in from the start, and we're still not even doing that very well. We're still exposing insecurable web interfaces to the public-facing Internet, and that's not the Internet's fault. So anyone who suggests that the Internet should have been designed with security built in from the start would have to be unaware of the way the world was back when the Internet was actually first being built. Some of us were there.

Leo: Yeah. And, you know, Vint Cerf did say the one thing he would have added had they been able to was encryption. That's exactly right.

Steve: Yeah.

Leo: Yeah. But they just couldn't at the time.

Steve: No, I mean, it didn't exist. It literally, like, you know, sure, I mean, we knew about the Enigma machine.

Leo: Right.

Steve: But, you know, you're not going to have gears and wires on your computer.

Leo: You don't have time to do that, yeah, yeah.

Steve: You know, it's like, what? No.

Leo: Yeah, yeah.

Steve: So, I mean, so we were playing around with these ideas. And Leo, remember also when HTTPS happened, when Navigator came out, and SSL was created, there was a 40-bit limit on any cipher that would leave the U.S.

Leo: Right.

Steve: So, you know, we had 128 bits, as long as it didn't try to make a connection outside the U.S. I mean, it was a mess back then.

Leo: It was. It was.

Steve: Yeah. Jonathan Haddock said: "Hi, Steve. Thanks for the podcast, having been listening from very early on. Was just listening to Episode 961 [last week] and the section about Facebook telling users they entered an old password. A downside of this practice is that an attacker knows the incorrect password was one that the person has used. Given the prevalence of password reuse, the attacker is then in possession of a confirmed previous password that could still work for that user elsewhere. A middle ground would be to simply say when the password was last changed. That way, the user experience is maintained, but the previous validity of the password is not exposed. Thanks again. Jonathan."

And I think he makes a very good point. It's not as useful to tell a user that their password was recently changed, but the fact is that it could be useful to an attacker. That is, you know, just telling the user your password was changed last Tuesday, well, okay, that's definitely more useful. But the difference that Jonathan is pointing out is that, if a bad guy is guessing passwords, and is told by Facebook, well, close, you got close, this is the password from last week, well, now the attacker knows he can't get into Facebook with that password, but he may be able to get into something else this user logs into with that password if they haven't, you know, if they were reusing that password elsewhere.

So it's weird because something was bothering me a little bit when we talked about this last week. Like why isn't this a problem? It turns out there is some information leakage from Facebook telling their users that. Probably it's worth doing still, but I'm glad that Jonathan brought this up.

Leo: Yeah.

Steve: Shawn Milochik said, and while we're sharing - oh, I said: "And while we're sharing brilliant observations from our listeners, here's one from Shawn Milochik." He said: "On the topic of passwordless email-only login, I think the largest benefit to the companies is that this is" - that is, using email links for login, which we talked about last week. He said: "The largest benefit to the companies is that this practically eliminates password sharing."

Leo: Oh, you're right.

Steve: Isn't that cool? Yes. He says: "It's one thing to give someone your password for a news or a streaming site. It's quite another to give them access to your e-mail, and probably the ability to reset your banking password, among other things." So Shawn, brilliant point. Thank you.

Lars Exeler said: "Hi, Steve. Love the show and just became a Club TWiT member." He said: "I'm wondering about TOTP as a second factor. So a password is the 'something you know' factor. A TOTP is the 'something you own' factor. My question is, isn't the TOTP just based on a lengthy secret in the QR code? So in my mind it's just like a second password, with the convenience of presenting itself as six digits because it's combined with the actual time. What am I missing here? Regards from Germany, Lars."

Okay. A password, right, is something you know. But the way to think of the TOTP, which is based upon a secret key, is that it's "transient and time-limited proof of something else you own or know." But transient and time-limited proof. And as we know, a password can be used over and over. But in any properly designed TOTP-based authentication, the six-digit token not only expires every 30 seconds, but each such token is invalidated after its first use. This means that even an immediate replay attack which still fits inside that 60-second expiration window will be denied. It's those two distinctions that make the TOTP a very powerful and different form of second factor.

Leo: Yeah, you're not - if you were sharing the secret every time, then it would just be another password. But you're not.

Steve: Right.

Leo: Right? That's not floating back and forth.

Steve: You are proving that you know the secret.

Leo: Exactly, yeah.

Steve: Without sharing it. And that's a key.

Leo: That's one step better. A little bit better.

Steve: Yes. Gavin Lanoe wrote - and this is a long one, but this needs to be shared. "Dear Steve. I first came across GRC in the early 2000s for recovering hard disks and your ShieldsUP! scanner, just as I was at the start of a career in IT and IT Security. About a year ago, I rediscovered you on the Security Now! podcast and also discovered the wider TWiT network. The Security Now! podcast over the last year is now the highlight of my week and even goes so far as to have a diary note in my calendar..."

Leo: Nice.

Steve: "...so I can try to listen to the creation live at what is late night where I live." Gavin hails from Germany. He said: "This week I discovered an issue with routers provided by one of our local ISPs, and I thought if there was ever a reason to drop you a message worthy of your time, this was it." And I would add, even worthy of our listeners' time.

He said: "Over my 30 years or so in IT, I, as I'm sure we all have, gathered a number of people who we look after for home user/family and friends IT support. I noticed over the last week or so some of my 400 or so 'private clients' were reporting strange issues. They reported to be getting warnings when browsing that sites were dangerous and were getting a lot of 404s from streaming services which weren't working, and connections to email services via IMAP secured with SSL were not working, connection and certificate errors and such.

"When I sat back and thought about it, all the users were on the same ISP and were all using the ISP-provided router." He said: "Side note: I usually recommend replacing the ISP's router with something more substantial. But when it's working fine, or doesn't need anything better, like a single elderly home user with an iPad and nothing else, it's hard to justify the spend."

So he says: "I got access to one of the users' routers, and after some investigation I found that the DNS servers listed just didn't look right. A trace route showed the server was in the Netherlands, and we're in Guernsey, Germany. The listed owner of the IP had nothing to do with the local ISP. Reverting the DNS Servers to the ISP's addresses resolved all the issues." And he says: "Later I reset them to Quad9." And here it is. "I also found that the ISP had left SNMP enabled on the WAN interface with no restrictions, and the username and password to gain access to the router were both 'admin.'"

Leo: No. That's for their convenience, not yours. Oh, boy.

Steve: That's exactly right. He said: "I have rotated through all of my other customers that have the same router and am locking them down, checking the DNS servers, disabling SNMP and resetting the password. I may even go so far as replacing the routers in their entirety because they may have also been compromised in some other, yet undiscovered way." Yes, I would say that he's been listening to this podcast.

He said: "I wrote to the ISP yesterday to explain the issue and received a call back today. It did take me some time to explain that any suitably skilled bad actor can connect to the routers via SNMP" - which, you know, Simple Network Management Protocol. And boy, is it dangerous - "with the default credentials, admin/admin."

Leo: Unbelievable. Oh, my god.

Steve: "And reconfigure" - I know. "And reconfigure the router settings. The engineer I was speaking to had trouble comprehending how this was possible without the web management enabled on the WAN side, but he eventually understood. We contacted another customer while we were talking, and he got to see the issue firsthand and is taking this up with their security team internally."

Leo: Good.

Steve: Lord, I hope so. He said: "My understanding for their reason" - here it is, Leo, get this. "My understanding for their reason for this configuration is" - believe it or not, it was on purpose - "they want to be able to remotely connect to a customer's router to assist in fault finding."

Leo: Yeah.

Steve: And, hah. They won't have to look very far.

Leo: Yeah.

Steve: "They have a script that connects to the routers via SNMP and enables web management on the WAN interface..."

Leo: Yup, there you have it right there. Jesus.

Steve: "...for a period of time, and then later it removes the web-based access again."

Leo: Uh-huh.

Steve: "They didn't consider that the open SNMP management interface with default credentials and an easy way to guess community string could be exploited in this way."

Leo: We never thought of that.

Steve: Yeah, wow.

Leo: What?

Steve: Yeah, we're accessing them remotely.

Leo: Yeah, but that's us.

Steve: You mean bad guys can, too?

Leo: Geez.

Steve: Wow. "The engineer," he says, "the engineer did seem quite disgruntled" - get this - "disgruntled that I often replace the ISP-provided routers."

Leo: How dare you?

Steve: Yes. "But as I explained to him, if it has a vulnerability on it, I'm going to replace it without any consideration for their convenience of remote diagnosis."

Leo: Absolutely. Geez.

Steve: He said: "Thank you, Steve. I hope this may get a mention on the show." And how. "But regardless, I really do look forward every week to your deep dives into detail behind the week's security-related news. Warm regards, Gavin from Guernsey."

So wow, Gavin. Thank you for sharing that. Apparently this ISP believes that SNMP stands for "Simply Not My Problem." But they would be wrong.

Leo: They would be wrong.

Steve: And this deliberate unprotected SNMP exposure with default credentials is breathtaking. It is a horrific breach of security.

Leo: Yeah.

Steve: SNMP, for those who don't know, is a funky, non-user-friendly, UDP-based protocol. It was originally and is still present in contemporary devices. But, like, very old original networking gear which allows for both the remote over-the-wire monitoring and control of that gear. Unfortunately, it also comes from an era before security was being given the due that it deserves. On the one hand, it's a good thing that this ISP has the wisdom to keep their customers' web management interfaces disabled by default. But that good intention is completely undone by their persistent exposure of the router's SNMP service, which I'm still shaking my head over.

So one remaining mystery is what was going on with the DNS rerouting. Given that most of today's DNS is still unencrypted in-the-clear UDP, this would be part of the requirement for site spoofing, to be able to spoof DNS. But with browsers now having become insistent upon the use of HTTPS, it's no longer possible to spoof a site over HTTP. So someone would need to be able to create web server certificates for the sites they wish to spoof. The fact that it's so widespread across an ISP's many customers tells us that it's probably not a targeted attack. That means it's somehow about making money, or intercepting a lot of something.

Having thought about this further, though, Leo, he mentioned lots of email problems when people are using secure IMAP. Well, not all email is secure. And so you could spoof email MX records in DNS in order to route email somewhere else, in which case clients would have a problem connecting if the DNS was spoofed over a TLS connection. So basically it's really only HTTP in the form of HTTPS where we've really got our security buttoned down tight with web server certificates. Other things that use DNS, and there are lots of other things that use DNS, they're still not running as securely as the web is. So that could be an issue.

Jeff Zellen said: "Steve, listening to SN-961 and thinking about the discussion on passwordless logins and using email as a single factor. Is my recollection correct that email is generally transmitted in the clear? Can't some unscrupulous actor sniff that communication? I'm especially concerned if the email contains a link to the site. This provides all the necessary info to anyone who can view the email. A six-digit code without any reference to the site where it should be used would be more secure as it is both out of band, but also obfuscated from the site. Of course this is all based on my possibly antiquated recollection of the lack of privacy in email."

Okay. Now, the email transit encryption numbers vary. I saw one recent statistic that said that only about half of email is encrypted in transit using TLS. But Google's most recent stats for Gmail state that 98 to 99% of their email transit, both incoming and outgoing, is encrypted. And it's a good feeling to have that. GRC's server fully supports all of the transit encryption standards, so it will transact using TLS with the remote end whenever it's available. And it's a good feeling knowing that when Sue and Greg and I exchange anything through email, since our email clients are all connecting directly to GRC's email server, everything is always encrypted end to end.

But it's not encrypted at rest. Jeff's point about the possible lack of email encryption in transit is still a good one, since email security is definitely lagging behind web security now that virtually everything has switched over to HTTPS. On the web, we would never even consider logging into a site that was not encrypted and authenticated using HTTPS. For one thing, we would wonder why the site was doing that; and, like, alarm bells would be going off. And even back when almost everything was still using HTTP, websites would switch their connections over to HTTPS just long enough to send the login page and receive the user's credentials in an encrypted channel, and then drop the user back to HTTP.

So by comparison to the standard set by today's web login with HTTPS, email security is sorely lacking. If we revisit the question then of using email only to login to a site, where that email carries what is effectively a login link, or even a six-digit authentication token, what we've done is reduce the login security of the site to that of email, which today we would have to consider to be good, but not great, and certainly far from perfect.

Another difference worth highlighting is that a web browser's connection to the website it's logging into is strictly and directly end-to-end encrypted. The browser has a domain name, and it must receive an identity asserting certificate that is valid for that domain. But this has never been true for email. Although modern email does tend to be point-to-point, with a server at the sender's domain directly connecting to the server at the recipient's domain, that's not always, nor is it necessarily true. Email has always been a store-and-forward transport. Depending upon the configuration of the sender and receiver's domains, email might make several hops from start to finish. And since the body of the email doesn't contain any of its own encryption, that email at rest is subject to eavesdropping.

Last week I coined the term "login accelerator" to more clearly delimit that value added by a password. And the whole point of the email loop authentication model is the elimination of the "something you need to know" factor. Yet without the "something you

need to know" factor, the security of email loop authentication can be no better than the security of email, which we've agreed falls short of the standard set by the web's direct end-to-end certificate identity verification with encryption.

As a result of this analysis, we might be inclined to discount the notion of email loop authentication as being inferior to web-based authentication with all of its fancy certificates and encryption, except for one thing. The ubiquitous "I forgot my password" get out of jail free link is still the party spoiler. It is everywhere, and its presence immediately reduces all of our much-ballyhooed web security to the security of email, imperfect as it may be. Even without email loop authentication, if a bad guy can arrange to gain access to a user's email, they can go to any site where the user maintains an account, click the "I forgot my password" link, receive the password recovery link by intercepting their email, and do exactly the same thing as if that site was only using email loop authentication.

What this finally means is that all of our identity authentication login, either using all of the fancy web technology, or just a simple email loop, is not actually any more secure than email, and that simple email loop authentication without any password is just as secure as web authentication, so long as web authentication includes an "I forgot my password" email loop bypass. So the notion of a password being nothing more than a login accelerator holds, and the world should be working to increase the security of email since it winds up being the linchpin for everything. So, really interesting thought experiment there.

Two last quickies. Alex Neihaus, a long-time friend of the podcast and early advertiser, I think, what, our first advertiser...

Leo: I think our first, yeah, with Astaro, yeah.

Steve: Yeah. He said: "Regarding 961" - last week's episode - "and overlay networks, pfSense has a killer Tailscale integration. With this running, you can set up pfSense to be an exit node, combining safe local access and VPN-like move-my-apparent-IP-address. That's useful for services that are tied to your location. Second, some cable providers' apps require UPnP, which you can, at your own risk, enable in pfSense-Tailscale."

So Alex, thank you for that. Given that I'm myself a pfSense shop, as Alex knows because he and I sometimes talk about pfSense, and that Tailscale integrates well with pfSense's FreeBSD Unix, Tailscale will certainly be a contender for a roaming overlay network. So thank you, Alex.

And finally, just a reminder from a listener. He said: "Hi, Steve. Given your discussion of throwaway email addresses during last Security Now! podcast episode, I'd like to bring your attention to DuckDuckGo's Email Protection Service. This free service is designed to safeguard users' email privacy by removing hidden trackers and enabling the creation of unlimited unique private email addresses on the fly. This feature allows users to maintain a distinct email address for each website they interact with.

"What makes this even more compelling is DuckDuckGo Email Protection's integration with Bitwarden. Bitwarden has included DuckDuckGo among its supported services, alongside other email forwarding services. This integration allows Bitwarden users to easily generate unique email addresses for each website they interact with. Take a look at" - and he provided the links which I've got in the show notes. So I thank our listener very much for that.

Leo: Okay. This I've been waiting all day for, Steve. What the heck? Are we in trouble?

Steve: No.

Leo: Oh, good.

Steve: The bullet was dodged. But the story is really interesting. The vulnerability has been codenamed "KeyTrap," and if ever there was a need for responsible disclosure of a problem, this was that time. Fortunately, responsible disclosure is what it received. What the German researchers who discovered this last year realized was that an aspect of the design of DNS, specifically the design of the secure capabilities of DNS, known as DNSSEC, could be used against any DNSSEC-capable DNS resolver to bring that resolver to its knees. The receipt of a single UDP DNS query packet could effectively take the DNS resolver offline for as many as 16 hours, pinning its processor at 100%, and effectively denying anyone else that server's DNS services.

It would have therefore been possible to spray the Internet with these single-packet DNS queries to effectively shut down all DNS services across the entire globe. Servers could be rebooted once it was noticed that they had effectively hung; but if they again received another innocuous-looking DNS query packet, which is their job after all, they would have gone down again. Eventually, of course, the world would have discovered what was bringing down all of its DNS servers. But the outage could have been protracted, and the damage to the world's economies could have been horrendous. So now you know why this podcast is titled "The Internet Dodged a Bullet." We should never underestimate how utterly dependent the world has become on the functioning of the Internet. I mean, it's like, what don't we do with the Internet now? Like all of our entertainment, all of our news, you know, do I have an FM radio? I'm not sure.

Okay. The detailed research paper describing this was just publicly released yesterday, though the problem, as I said, has been known since late last year. Here's how the paper's Abstract describes what these German researchers discovered, to their horror. They wrote: "Availability is a major concern in the design of DNSSEC. To ensure availability, DNSSEC follows Postel's Law, which reads: 'Be liberal in what you accept, and conservative in what you send.' Hence, nameservers should send not just one matching key for a record set, but all the relevant cryptographic material, in other words, all the keys for all the ciphers that they support and all the corresponding signatures. This ensures that validation will succeed; and hence availability, even if some of the DNSSEC keys are misconfigured, incorrect, or correspond to unsupported ciphers, will be maintained."

They write: "We show that this design of DNSSEC is flawed. By exploiting vulnerable recommendations in the DNSSEC standards, we develop a new class of DNSSEC-based algorithmic complexity attacks on DNS, which we dub KeyTrap attacks. All popular DNS implementations and services are vulnerable." This is them writing. "With just a single DNS packet, the KeyTrap attacks lead to a two million times spike in CPU instruction count in vulnerable DNS servers" - remember, that's all DNS servers - "stalling some for as long as 16 hours. This devastating effect prompted major DNS vendors to refer to KeyTrap as 'the worst attack on DNS ever discovered.' Exploiting KeyTrap, an attacker could effectively disable Internet access in any system utilizing a DNSSEC-validating resolver.

"We disclosed KeyTrap to vendors and operators on November 2, 2023. We confidentially reported the vulnerabilities to a closed group of DNS experts, operators and developers

from the industry. Since then, we have been working with all major vendors to mitigate KeyTrap, repeatedly discovering and assisting in closing weaknesses in proposed patches. Following our disclosure, an umbrella CVE was assigned."

Okay. So believe it or not, all that just actually happened, as they say, behind closed doors. Google's Public DNS and Cloudflare were both vulnerable, as was the very popular, the most popular and widely deployed BIND 9 DNS implementation, and it was the one - we'll see why later - that could be stalled for as long as 16 hours after receiving one packet.

So what happened? As the researchers wrote, months before all this came to light publicly, all major implementations of DNS had already been working on quietly updating because had this gotten out, it could have been used to bring the Internet down. With Microsoft's release of patches which included mitigations for this last week, and after waiting a week for them to be deployed, the problem is mostly resolved, if you'll pardon the pun.

I say "largely" because this is not a bug in an implementation. And apparently even now, as we'll see, some DNS servers will still have their processors pinned, but they will still at least be able to answer other DNS query functions. It sounds as though thread or process priorities have been changed to prevent the starvation of competing queries. And we'll actually look in a minute at the strategies that have been deployed. Characterizing this as a "big mess" would not be an exaggeration.

KeyTrap exploits a fundamental flaw in the design of DNSSEC which makes it possible to deliberately create a set of legal, but ultimately malicious, DNSSEC response records which the receiving DNS server will be quite hard pressed to untangle. Once the researchers had realized that they were onto something big, they began exploring all of the many various ways DNS servers could be stressed. So they created a number of different attack scenarios.

I want to avoid getting too far into the weeds of the design and operation of DNSSEC, but at the same time I suspect that this podcast's audience will appreciate seeing a bit more of the detail so that the nature of the problem can be better appreciated. The problem is rooted in DNSSEC's provisions for the resolution of "key tag" collisions, the handling of multiple keys when they're present, and multiple signatures when they're offered.

I'm going to quote three paragraphs from their research paper. But, you know, just sort of let it wash over you so that you'll get some sense for what's going on without worrying about understanding it in any detail. You will not be tested on your understanding of this. Okay. So they explain: "We find that the flaws in the DNSSEC specification are rooted in the interaction of a number of recommendations that in combination can be exploited as a powerful attack vector."

Okay. So first, key tag collisions. They write: "DNSSEC allows for multiple cryptographic keys in a given DNS zone." Zone is the technical term for a domain, essentially. So when they say "zone," they mean a DNS domain. "For example," they say, "during key-rollover or for multi-algorithm support." Meaning you might need multiple cryptographic keys. If you're retiring one key, you want to bring the new key online, right, before the old key goes away. So for a while you've got two or more keys. Or multi-algorithm support. Might need different keys for different algorithms if you want to be more comprehensive.

So they said: "Consequently, when validating DNSSEC, DNS resolvers are required to identify a suitable cryptographic key to use for signature verification." Because, you know, the zone is signed, so you want to verify the signature of the zone to verify it's not been changed. That's what DNSSEC is all about is preventing any kind of spoofing. So

they said: "DNSSEC uses key tag values to differentiate between the keys, even if they are of the same zone and use the same cryptographic algorithm." So they just could be redundant keys for some reason.

They said: "The triple of zone, algorithm, and key tag is added to each respective signature to ensure efficiency in key-signature matching." Again, don't worry about the details. "When validating a signature, resolvers check the signature header and select the key with the matching triple for validation. However, the triple is not necessarily unique."

And that's the problem. "Multiple different DNS keys can have an identical triple." That is to say an identical tag. "This can be explained by the calculation of the values in the triple. The algorithm identifier results directly from the cipher used to create the signature and is identical for all keys generated with a given algorithm. DNSSEC mandates all keys used for validating signatures in a zone to be identified by the zone name. Consequently, all DNSSEC keys that may be considered for validation trivially share the same name. Since the collisions in algorithm-id and key name pairs are common, the key tag is calculated with a pseudorandom arithmetic function over the key bits to provide a means to distinguish same-algorithm, same-name keys." Again, just let this glaze over.

"Using an arithmetic function instead of a manually chosen identifier eases distributed key management for multiple parties in the same DNS zone. Instead of coordinating key tags to ensure their uniqueness, the key tag is automatically calculated. However" - here it comes - "the space of potential tags is limited by the 16 bits in the key tag field. Key tag collisions, while unlikely, can thus naturally occur in DNSSEC. This is explicitly stated in RFC-4034, emphasizing that key tags are not unique identifiers. As we show, colliding key tags can be exploited to cause a resolver not to be able to uniquely identify a suitable key efficiently, but to have to perform validations with all the available keys, inflicting computational effort during signature validation."

Okay, now, just to interrupt this for a second, cryptographic keys are identified by tags, and those tags are automatically assigned to those keys. Work on DNSSEC began way back in the 1990s, when the Internet's designers were still counting bits and were assigning only as many bits to any given field as would conceivably be necessary. Consequently, the tags being assigned to these keys were, and are still today, only 16 bits long. Since these very economical tags only have 16 bits, thus one of 64K possible values, inter-tag collisions, while unlikely, you know, one of 65536, and we've got the birthday paradox which makes collisions more, you know, happen more often than you'd expect.

If DNSSEC were being designed today, tags would be the output of a collision-free cryptographic hash function, and there would be no provision for resolving tag collisions because there would be none. The paragraph I just read said "the key tag is calculated with a pseudorandom arithmetic function," in other words, something simple from the '90s that scrambles and mixes the bits around, but doesn't do much more.

Leo: Like a salad spinner or something.

Steve: Right, exactly. It's still salad. It's just better now. Consequently, servers need to consider that key tags are not unique. And so what the attack is, is it deliberately makes all the key tags identical, forcing the server to check them all.

Leo: Oh, brilliant.

Steve: Yes, yes.

Leo: How many key tags can you put in there?

Steve: We're getting there. We're getting there. But you make them all the same, and the server can't use it to select the key. It's got to try them all.

Leo: Right.

Steve: So the first attack is key tag collisions. Literally, they all collide. Okay. On to the next problem, multiple keys. "The DNSSEC specification mandates that a resolver must try all colliding keys until it finds a key that successfully validates the signature, or all keys have been tried."

Leo: Of course.

Steve: Right.

Leo: What could possibly go wrong?

Steve: That's right. "The requirement is meant to ensure availability, meaning DNSSEC will try as hard as it can to find a successful signature. Even if colliding keys occur, such that some keys may result in failed validation, the resolver must try validating with all the keys until a key is found that results in a successful validation. This ensures that the signed record remains valid, and the corresponding resource therefore remains available."

However, what they call eager - they say: "However, this 'eager validation' can lead to heavy computational effort for the validating resolver, since the number of validations grows linearly with the number of colliding keys. So, for example, if a signature has 10 colliding keys, all with identical algorithm identifiers, the resolver must conduct 10 signature validations before concluding that the signature is invalid. While colliding keys are rare in real-world operation, we show that records created to deliberately contain multiple colliding keys" - meaning all the keys are colliding - "can be efficiently crafted by an adversary imposing heavy computation upon a victim resolver."

Okay. And the third and final problematic is multiple signatures: "The philosophy," they said, "of trying all of the cryptographic material available to ensure that the validation succeeds also applies to the validation of signatures. Creating multiple signatures for a given DNS record can happen, for example, during a key-rollover. The DNS server adds a signature with the new key, while retaining the old signature to ensure that some signature remains valid for all resolvers until the new key has been propagated. Thus, parallel to the case of colliding keys, the RFCs specify that in the case of multiple signatures on the same record, a resolver should try all the signatures it received until it finds a valid signature or until it's used up all the signatures."

Okay. So we have the essential design features which were put into the DNSSEC specification in a sane way, I mean, all this makes sense, with the purpose of never failing to find a valid key and signature for a zone record. Their term for this, of course,

is "eager validation." They write: "We combine these requirements for the eager validation of signatures and of keys, along with the colliding key tags to develop powerful DNSSEC-based algorithmic complexity attacks on validating DNS resolvers. Our attacks allow a low-resource adversary to fully DoS a DNS resolver for up to 16 hours with a single DNS request."

Leo: Holy cow.

Steve: Yeah.

Leo: Wow.

Steve: One request, and the server goes down for 16 hours. They wrote: "Members from the 31-participant task force of major operators, vendors, and developers of DNS and DNSSEC, to which we disclosed our research, dubbed our attack 'the most devastating vulnerability ever found in DNSSEC.'"

Okay. Now, the researchers devised a total of four different server-side resource exhaustion attacks. And I have to say, Leo, I was a little bit tempted to title today's podcast after three of them. Had I done so, today's podcast would have been titled "SigJam, LockCram, and HashTrap."

Leo: I would have done that.

Steve: I know.

Leo: That's good. That's good.

Steve: I know. SigJam, LockCram, and HashTrap.

Leo: It's a little like a law firm.

Steve: And while I certainly acknowledge that would have been fun, I really didn't want to pass up, you know, I didn't want to lose sight of the fact that the entire global Internet really did just dodge a bullet. And we don't know which foreign or domestic cyber intelligence services may today be silently saying "Darn it. They found it. That was one we were keeping in our back pocket for a rainy day, while keeping all of our foreign competitor DNS server targeting packages updated." Because this would have made one hell of a weapon.

Okay. So what are the four different attacks? SigJam utilizes an attack with one key and many signatures. They write: "The RFC advises that a resolver should try all signatures until a signature is found that can be validated with the DNSKEY. This can be exploited to construct an attack" - we're going to answer your question, Leo, about how many - "using many signatures that all point to the same DNSSEC key. Using the most impactful algorithm" - meaning the most time-consuming to verify - "an attacker can fit 340 signatures into a single DNS response, thereby causing 340 expensive cryptographic

signature validation operations in the resolver until the resolution finally fails by returning SERVFAIL response to the client."

Leo: That shouldn't take 16 hours.

Steve: Ah, it gets better.

Leo: Oh. Because it's more.

Steve: Because that's linear. We're going to go quadratic in a minute.

Leo: Oh, good.

Steve: "The SigJam attack is thus constructed by leading the resolver to validate many invalid signatures on a DNS record using one DNS key."

Okay. The LockCram attack does the reverse, using many keys and a single signature. They write: "Following the design of SigJam, we develop an attack vector we dub LockCram. It exploits the fact that resolvers are mandated to try all keys available for a signature until one validates or all have been tried. The LockCram attack is thus constructed by leading a resolver to validate one signature over a DNS record having many keys. For this, the attacker places multiple DNS keys in the zone which are all referenced by signature records having the same triple - name, algorithm, key tag. This is not trivial, as resolvers can de-duplicate identical DNSKEY records, and their key tags need to be equal.

"A resolver that tries to authenticate a DNS record from the zone attempts to validate its signature. To achieve that, the resolver identifies all the DNS keys for validating the signature, which, if correctly constructed, conform to the same key tag. An RFC-compliant resolver must try all the keys referred to by the invalid signature before concluding the signature is invalid for all keys, leading to numerous expensive public key cryptography operations in the resolver."

And the next attack, the KeySigTrap, combines the two previous attacks by using multiple keys and multiple signatures. They say: "The KeySigTrap attack combines the many signatures of SigJam with the many colliding DNSKEYs of LockCram, creating an attack that leads to a quadratic increase in the number of validations compared to the previous two attacks."

Leo: SigLock JamCram is so much worse. Wow.

Steve: Yeah, you don't want one of those.

Leo: No.

Steve: "The attacker creates a zone with many colliding keys and many signatures matching those keys. When the attacker now triggers resolution of the DNS record with

the many signatures, the resolver will first try the first key to validate all the signatures. After all the signatures have been tried, the resolver will move to the next key and again attempt validation of all the signatures. This continues until all pairs of keys and signatures have been tried. Only after attempting validation on all possible combinations does the resolver conclude that the record cannot be validated and returns a SERVFAIL to the client."

Okay, now, elsewhere in their report, when going into more detail about this KeySigTrap, they explain that they were able to set up a zone file containing 582 colliding DNSSEC keys and the same 340 signatures that we saw in SigJam. Since the poor DNS resolver that receives this response will be forced to test every key against every signature, that's 582 times 340, or 197,880 expensive and slow public key cryptographic signature tests. And that was caused by sending that DNS server a single DNS query packet for that domain.

Now, interestingly, the researchers discovered that not all DNS servers were as badly affected. For some reason, several took the request much harder. Upon investigating they discovered why. For example, the Unbound DNS server is DoSed approximately six times longer than some of the other resolvers. The reason is the default re-query behavior of Unbound. In its default configuration, Unbound attempts to re-query the nameserver that gave it this malicious zone five times after failed validation of all signatures. Therefore, Unbound validates all attacker signatures six times before returning a SERVFAIL to the client. Essentially, Unbound is being penalized for being a good citizen. Disabling default re-queries brings Unbound back to parity with the other servers.

But BIND, the famous, I mean, it's the most-used server on the Internet, BIND - in fact, that's where Unbound got its name, right, it's a play on BIND. BIND was the worst. That's the one with the 16-hour DoS from a single packet. And they explained why: "Investigating the cause for this observation, we identified an inefficiency in the code, triggered by a large number of colliding DNSSEC keys. The routine responsible for identifying the next DNSSEC key to try against a signature does not implement an efficient algorithm to select the next key from the remaining keys. Instead, it reparses all keys again until it finds a key that has not been tried yet. This algorithm does not lead to inefficiencies in normal operation where there might be a small number of colliding keys. But when many keys collide, the resolver spends a large amount of time parsing and reparsing the keys to select the next key, which extends the total response duration of the DoS to 16 hours."

So what all of this should make clear is that these potential problems arise due to DNSSEC's deliberate "eager to validate" design. You know, the servers are trying as hard as they can to find a valid key and signature match. DNS servers really want to attempt all variations, which is exactly, unfortunately, what gets them into trouble. The only solution will be something heuristic. We've talked about heuristics in the past. They can be thought of as a rule of thumb, and they usually appear when exact solutions are not available, which certainly is the case here.

As we might expect, the researchers have an extensive section of their paper devoted to "what to do about this mess," and they worked very closely for months with all the major DNS system maintainers to best answer that question. I'll skip most of the blow-by-blow, but here's a bit that gives you sort of a sense and a feeling for it.

Under the subhead "Limiting all validations," they wrote: "The first working patch capable of protecting against all variants of our attack was implemented by Akamai. In addition to limiting key collisions to four, and limiting cryptographic failures to 16, the patch also limits total validations in any requests to eight." In other words, they basically just said it is Akamai patched their DNS to say, okay, it's unreasonable for anyone to expect our

server to work this hard. There are not really going to be that many key collisions in a DNSSEC zone. There aren't, you know, it's just not going to happen in real life. And we're not going to constantly be failing.

"So let's just stop after we've got four key collisions. We're just not - we're just going to say, sorry, times up, you know, we're not going to go any further. And let's just stop after we've had 16 cryptographic failures. No matter what kind and what nature, that's all we're going to do because it's unreasonable for any valid DNSSEC zone to have more. And we're also going to cap the total limit of validation requests to eight."

So then they wrote: "Evaluating the efficacy of the patch, we find the patched resolver does not lose any benign request" - meaning DoS is avoided - "even under attack with greater than 10 attacking requests per second." In other words, it doesn't fix the problem, it just it mitigates it. It gets it under control. "The load on the resolver does not increase to problematic levels under any type of attack at 10 requests, 10 malicious requests per second, and the resolver does not lose any benign traffic. It thus appears that the patch successfully protects against all variations of KeyTrap attacks.

"Nevertheless," they said, "although these patches prevent packet loss, they still do not fully mitigate the increase in CPU instruction load during the attack." You know, it's still an attack. "The reason that the mitigations do not fully prevent the effects of the KeyTrap attacks is rooted in the design philosophy of DNSSEC. Notice, however, that we are still closely working with the developers on testing the patches and their performance during attack and normal operation." "Still" as in today still.

Okay. So the disclosure timeline, you know, the responsible disclosure timeline for this is extra interesting since it provides a good sense for the participants and the nature of their efforts and interactions over time. So in the following, they wrote: "We describe the timeline of disclosure to indicate how the vulnerability was reported and how we worked with the experts from industries to find solutions for the problems we discovered."

Okay. So November 2nd of 2023, the initial disclosure to key figures in the DNS community. They said: "Both confirm that KeyTrap is a severe vulnerability, requiring a group of experts from industry to handle."

Five days go by. Now we're at November 7th. "Confidential disclosure to representatives of the largest DNS deployments and resolver implementations, including Quad9, Google Public DNS, BIND9, Unbound, PowerDNS, Knot, and Akamai. The group of experts agree that this is a severe vulnerability that has the potential to cut off Internet access to large parts of the Internet in case of malicious exploitation. A confidential chat-group is established with stakeholders from the DNS community, including developers, deployment specialists, and the authors." That is, you know, the researchers here. "The group is continuously expanded with additional experts from the industry to ensure every relevant party is included in the disclosure. Potential mitigations are discussed within the group."

Two days later, November 9th: "We share KeyTrap zone files to enable developers to reproduce the attacks locally, facilitating the development of mitigations."

November 13th, after four days: "Akamai presents the first potential mitigation of KeyTrap by limiting the total number of validation failures to 32." That doesn't work.

November 23rd, 10 days later: "Unbound presents its first patch, limiting cryptographic failures to a maximum of 16, without limiting collisions." Also a non-starter.

Next day: "BIND9 presents the first iteration of a patch that forbids any validation failures."

Okay, now we jump to December 8th, so a couple weeks go by. The CVE is assigned to the KeyTrap attacks, although nothing is disclosed. It's just an umbrella CVE to encompass them all.

Now we move to January 2nd this year, 2024, beginning of the year: "After discussions with the developers, we find some have problems recreating the attack in a local setup. We thus provide them an updated environment with a DNS server to ease local setup and further facilitate testing of patches. In other words, the researchers actually put this live on the Internet so that the DNS developers could DoS their own servers."

March 1st. A month goes by, or two months go - no, I'm sorry, the next day. I'm getting the dates are in an odd, well, you know, day/month/year. Anyway, so January 3rd: "BIND9 presents the second iteration of a patch, limiting validation failures."

Same day, they said: "Our newly implemented DS-hashing attack proves successful against all mitigations which do not limit key collisions, including BIND9 and Unbound, and is disclosed to the group." So, whoops, patch again, everybody.

January 16th: "Our Anytype attack circumvents the protection from limiting colliding keys and limiting cryptographic failures."

And on January 24th: "The first working patch is presented by Akamai. Other resolvers are implementing derivatives of the countermeasures to protect against the attacks."

And so now we get to the reports and the German discoverers of this final conclusions. They write: "Our work revealed a fundamental design problem with DNS and DNSSEC. Strictly applying Postel's Law to the design of DNSSEC introduced a major and devastating vulnerability in virtually all DNS implementations. With just one maliciously crafted DNS packet, an attacker could stall almost any resolver, for example the most popular one, BIND9, for as long as 16 hours.

"The impact of KeyTrap is far reaching. DNS evolved into a fundamental system in the Internet that underlies a wide range of applications and facilitates new and emerging technologies. Measurements by APNIC show that in December of 2023, 31.47% of web clients worldwide were using DNSSEC-validating resolvers. Therefore, our KeyTrap attacks have effects not only on DNS itself, but also on any application using it. An unavailability of DNS may not only prevent access to content, but risks also disabling other security mechanisms, like anti-spam defenses, Public Key Infrastructure, or even inter-domain routing security like RPKI or ROVER.

"Since the initial disclosure of the vulnerabilities, we've been working with all major vendors on mitigating the problems in their implementations, but it seems that completely preventing the attacks will require fundamentally reconsidering the underlying design philosophy of DNSSEC, in other words, to revise the DNS standards."

Leo: Wow.

Steve: So as we can see, as I titled this podcast, the Internet really did dodge a bullet. And I've got to say it's also terrific to see that the technical and operational level of all of this, at that level we have the ability to quickly gather and work together to resolve any serious trouble that may arise. Fortunately, that doesn't happen very often. It just did, and it's like, whew, exactly, everybody is breathing a deep sigh.

Leo: It was never exploited by anybody. Do we know?

Steve: Nope.

Leo: No.

Steve: No. Far as we know. I mean, had it been, it would have been discovered. These guys...

Leo: You would know, yeah, yeah.

Steve: Basically they reverse engineered this from the spec, kind of going, well, what if we were to do this? And what if we were to do that? And when the servers went offline, they thought, uh, whoops.

Leo: Yeah. Yeah. What if we did the - oh. Hello.

Steve: Yeah. And how can we make it worse? And how can we make it worse? And how can we make it worse?

Leo: Well, good research. And let's hope everybody fixes their - it's ironic because there was a huge push to get everybody to move to DNSSEC for a while. We talked about it a lot.

Steve: Yup.

Leo: Oh, well.

Steve: Yup.

Leo: Steve, another fabulous show in the can, as they say.