The Man in the Middle

Description: This week we have a really wonderful picture of the week in the form of a techie "what we say" and "what we mean" counterpoint. So we're going to start off spending a bit of time with that. Then we're going to see whether updating to that latest WinRAR version might be more important than was clear last week. And while HTTPS is important for the public Internet, do we need it for our local networks? What about using our own portable domain for email? Does Google's new Topics system unfairly favor monopolies? If uBlock Origin blacks ads, why does it also need to block Topics? Just how narrow (or wide) is Voyager 2's antenna beam, and what does 2 degrees off-axis really mean? Do end users need to worry about that wacky Windows time setting mess? And what's the whole story about Unix time in TLS handshakes? What can be done about fake mass storage drives flooding the market? And finally, let's look at man-in-the-middle attacks. How practical are they, and what's been their history?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-937.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-937-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. He's going to give you some more information about that WinRAR hack, how it works, and how to fix it. We'll also talk a little bit about the Unix time story and TLS handshakes. Do they really need it? And do we really need HTTPS? Sometimes it turns out we don't. And then Steve's going to blow the lid off of an explosive story, hard drives that lie. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 937, recorded Tuesday, August 29th, 2023: The Man in the Middle.

It's time for Security Now!, the show where we cover your security, your privacy, your online life with this guy right here, Mr. Steven Gibson, our man of the hour. Hi, Steve.

Steve Gibson: Yo, Leo. The last - watch out for that Super Glue. Can you believe that it is the end of August?

Leo: Yeah.

Steve: Wow.

Leo: You know, I kind of can believe it because we have a guy here, one of our employees who's been delegated to put up holiday decorations.

Steve: Oh.

Leo: And when I came in this morning, it said it right on the desk: It's fall. And I said, it's not fall, it's August.

Steve: Unfortunately, it's now fire season.

Leo: Yeah. It is a red - we got a red flag warning today in Northern California.

Steve: Yikes.

Leo: Yikes is right.

Steve: So we're at Security Now! Episode 937, and we're going to go until we have five digits. So nobody needs to worry.

Leo: Woohoo.

Steve: About running out after three.

Leo: Steve and I have committed to making it to the end of the Unix epoch.

Steve: That would be good, yeah.

Leo: I like that idea, yeah.

Steve: I do. So this week we have a really wonderful Picture of the Week in the form of a sort of a techie "what we say" and "what we mean" counterpoint. So we're going to start off by spending a bit of time having fun with that. Then we're going to see whether updating to the latest WinRAR version might be more important than was clear last week. And while HTTPS is important for the public Internet, do we need it for our local networks? What about using our own portable domain for email? That's an idea. Does Google's new Topics system unfairly favor monopolies? And if so, how? If uBlock Origin blocks ads, why does it also need to block Topics?

Leo: Oh, hmm.

Steve: Yeah. Just how narrow (or wide) is Voyager 2's antenna beam, and what does "2 degrees off-axis" really mean? We have the math now, thanks to one of our listeners.

Leo: Ooh.

Steve: Do end users need to worry about that wacky Windows time setting mess on their own desktops? And what's the whole story about Unix time in TLS handshakes anyway? What can be done about fake mass storage drives which are apparently now flooding the market? And finally, let's look at man-in-the-middle attacks, thus the name of today's podcast. How practical are they, and what's been their history?

Leo: Interesting. Interesting. You know, you're doing something now that all successful podcasts, TV shows, radio shows do, which is...

Steve: Not giving up.

Leo: Not giving up. Actually, you know what, that's number one, it really is. Longevity, especially in radio, and I imagine in podcasting, makes a difference. But calling back to previous episodes. So we have a few topics this week that call back to previous episodes. But I, you know, if you have missed those episodes, A, you should have listened to them. Why didn't you listen to them? But B, don't worry, we'll fill you in before we get too far.

Steve: That's right. And if not, you'll have FOMO.

Leo: FOMO. No missing out, unh-unh.

Steve: What out for the FOMO.

Leo: There are 937 episodes. If you have not listened to all 937 at this point, you must go back and listen to them. They're all available at TWiT.tv/sn or on Steve's site.

Steve: And if you don't know what we mean when we refer to the Portable Dog Killer...

Leo: Oh, lord.

Steve: Oh, well.

Leo: By the way, I forgot to ask you. You sent me a wonderful fidget spinner when they were big, about four years ago.

Steve: Yeah.

Leo: Do you still fidget?

Steve: Yeah, actually. I have a...

Leo: What do you use?

Steve: I have a little fidget ring right here.

Leo: Oh.

Steve: Which is kind of nice because...

Leo: Oh, it's got ball bearings in there, and you can spin it?

Steve: Yeah. And so the outer ring spins.

Leo: Oh.

Steve: It's kind of cool.

Leo: Do you keep it on your finger and spin it that way?

Steve: Yeah, I do, yes, exactly.

Leo: Oh, interesting.

Steve: Yeah.

Leo: I do not. But should I at some point have the urge to fidget, I have my spinner right here, just in case, in the drawer.

Steve: This is sort of related to the little tray at the top of my keyboard which is just full of little paraphernalia. I don't know, you know, I just - it collects things.

Leo: Oh. Oh. You have - the keyboard has a tray, or just the keyboard holder has a tray?

Steve: The tray itself is sort of - it was the top of that Northgate keyboard. It had a little tray [crosstalk]...

Leo: For paperclips, thumbtacks, mints.

Steve: Oh, and boy, stuff finds its way in there and then it never leaves.

Leo: Lint-covered mints. I am prepared now for a rather lengthy Picture of the Week.

Steve: And I think worthwhile. So, you know, it's referred to as "spin," right, where you sort of put the best face on something...

Leo: Yes, yes.

Steve: ...when you have a need to do so. So when we say a "horrible hack," what we mean is a horrible hack that I didn't write.

Leo: Somebody else's horrible hack.

Steve: That's right. But when it's a "temporary workaround," it's a horrible hack that I did write.

Leo: Yes, yes.

Steve: So when we say "It's broken," then that means there are bugs in your code. When we say, "Well, it has a few issues," then that means there are bugs in my code.

Leo: Wow.

Steve: "Obscure" is "Someone else's code doesn't have comments." "Self-documenting" is "My code doesn't need comments." We have "That's why it's an awesome language" says someone. Which actually means "It's my favorite language, and it's really easy to do something in it." Or "You're thinking in the wrong mindset" is "It's my favorite language, and it's really hard to do something in it." Like LISP.

Leo: Like LISP, exactly, yes, yes.

Steve: That's right. And then we have "I can read this Perl script," which actually means "I wrote this Perl script."

Leo: And even then sometimes not, but okay, yes.

Steve: Oh, boy.

Leo: I wrote this Perl script in the last week. Let's make it that.

Steve: Say I recently wrote it, exactly.

Leo: Recently, yes.

Steve: And then there's "I can't read this Perl script," which means "I didn't write this Perl script."

Leo: Or I wrote it last week.

Steve: It's impossible to read anybody else's.

Leo: Right.

Steve: Then we have, oh, that's "bad structure," which means "Someone else's code is badly organized." Or "complex structure" is "My code is badly organized." A "bug" is the absence of a feature I like. "Out of scope" is the absence of a feature I don't like.

Leo: Yes.

Steve: A "clean solution" is "It works, and I understand it." Then "We need to rewrite it" is "It works, but I don't understand it." Then "Emacs is better than Vi" means "It's too peaceful here, let's start a flame war."

Leo: Yes.

Steve: And "Vi is better than Emacs" means "It's too peaceful here, let's start a flame war." And then of course "IMHO" is the famous abbreviation which actually means "You're wrong."

Leo: Yes. In my humble opinion, you're full of it.

Steve: That's right.

Leo: Yup, yup, yup.

Steve: Okay. Then we have a couple that don't have two alternatives. We have "legacy code," which actually means "It works, but no one knows how." And then I love this, Leo. We have CTRL+X, CTRL+C, lowercase quit, CTRL+\, [ESC] [ESC] CTRL+C.

Leo: Oh, what's that?

Steve: Which actually means I don't know how to quit Vi.

Leo: Yes. Been there. Done that.

Steve: Oh, god, so have I.

Leo: Yes.

Steve: It's like, oh, how do I - what do I do here?

Leo: Colon Q. Oh, lord.

Steve: So what is a "suboptimal implementation" is actually "The worst errors ever inflicted on mankind." And then "These test environments are too brittle" actually means "Works on my machine. Have you tried rebooting?"

Leo: Oh, lord.

Steve: And then we have "Proof-of-Concept" is "What I wrote." "Perfect solution" is "How sales and marketing are promoting it."

Leo: Very nice.

Steve: So anyway...

Leo: Where did you get that? It's really the Table of the Week, John [crosstalk] pointed out.

Steve: That was the Table of the Week. That was one of our great listeners who ran across it. And he said: "Steve, I saw this, and I thought of you."

Leo: Nice.

Steve: And I said that immediately got promoted to the "must read" on the next episode of Security Now!. I actually did have something else in place. And it's like, no. That'll go next week. This one we have to play with.

Leo: Well, it's funny because it's true. I mean, it's all true.

Steve: Yes, yes. As I said, spin is a thing; right? And it's sort of the nature of politics in the tech world.

Okay. So last week I mentioned the recent release of WinRAR 6.23, and I noted then that it fixed a pair of critical vulnerabilities that could have been used to execute code on its users' systems. Well, then came the news that this pair of vulnerabilities had been discovered by bad guys and had been actively exploited at least as far back as last April. Maybe sooner, and that wasn't seen.

BleepingComputer's headline last Wednesday, after last Tuesday's podcast, was "WinRAR zero-days exploited since April to hack trading accounts." And since this really forms a cautionary tale which might catch any of us if we were to even briefly drop our guard, I want to share some of the details which BleepingComputer shared from Group-IB who were the ones who made the original discovery. I've edited what BleepingComputer wrote for a bit for the podcast.

But basically what they said was: "A WinRAR zero-day vulnerability tracked as CVE-2023-38831 was actively exploited to install malware when clicking on harmless files in an archive, allowing the hackers to breach online cryptocurrency trading accounts. The vulnerability has been under active exploitation since April 2023, being used to distribute various malware families, including DarkMe, GuLoader, and the Remcos RAT, Remote Access Trojan." And really that's got to be one of the best abbreviations or I guess acronyms that we've seen in a long time, calling it a RAT, which is, you know, it's perfect for Remote Access Trojan.

Anyway, "The WinRAR zero-day vulnerability," they said, "allowed the threat actors to create malicious .RAR and .ZIP archives containing innocuous files such as JPG images, text files, and PDF docs. However, when a user opens a file, the flaw will cause a script to be executed that installs malware on the device. BleepingComputer tested a malicious archive shared by Group-IB, who discovered the campaign, and simply double-clicking on a PDF caused a command script to be executed to install malware.

"The zero-day was fixed in WinRAR version 6.23, released on August 2nd, which also resolves several other security issues, including this flaw that can trigger command execution upon opening a specially crafted RAR file. In a report released last Wednesday, researchers from Group-IB said they discovered the WinRAR zero-day being used to target cryptocurrency and stock trading forums, where the hackers pretended to be other enthusiasts sharing their trading strategies.

"Their forum posts contained links to specially crafted WinRAR ZIP or RAR archives that pretended to include the shared trading strategy, consisting of PDFs, text files, and images. The fact that those archives target traders is demonstrated by the forum post titles, like 'Best Personal Strategy to Trade with Bitcoin.' The malicious archives were distributed on at least eight public trading forums, infecting at least a confirmed 130 traders' devices. The total number of victims and financial losses resulting from the campaign are unknown."

Anyway, so essentially there was a file that was innocent, innocuous, and a folder of the same name. And when you opened the file, for reasons of this bug, which had been figured out by bad guys, a command script in the folder was executed and would cause malware to be installed on this victim machine.

Leo: Oh. That explains it. Because, you know, we talk a lot about, you know, PDF, for instance. Because it's an interpreter, it's really writing - it's code, and the PDF is actually code, and then the PDF reader is an interpreter, interpreting the code. But it's been a long time since I wrote - I wrote a long time ago an ARC decoder. But I remember it was mostly lookup tables. It wasn't really doing interpretation. So this makes sense. It's not that the RAR file format is being interpreted, it's that you're including something hidden in the folder that then gets executed.

Steve: Yeah. And so - exactly. It was the thing in the folder that got executed, and that was unseen. And there was a bug in RAR that allowed a specially crafted archive to cause the folder's contents to be executed. And then what it did, even though you were clicking on the PDF, it didn't actually launch, it didn't open and launch the PDF, so the script did that so that the user thought that their action was actually successful, even though it was a script that ran that action for them in order to hide the fact of what it was doing in the background.

Leo: That's an unusually nice finesse because usually they don't care.

Steve: Right. They've already got you by the point.

Leo: They got you, yeah, right. That's nice.

Steve: Yeah. Although they're wanting to get in there in order to suck someone's cryptocurrency trading out, so they probably do want to be on the DL in order to get that done.

Leo: They don't want to make you suspicious, yeah. Well, it makes a lot of sense to me. I was wondering if WinRAR had some sort of weird interpreter built in, but no. Okay.

Steve: So, okay, think about this. WinRAR has been around forever. It's a trusted and robust archiving utility. I use it myself because it's highly configurable, much more so than ZIP, for example. And when configured to use large compression block sizes and to generate a so-called "solid" archive, which is one that's not easily editable after the fact, it could achieve unmatched compression levels. So naturally it's my choice. So I'm trying to imagine what would happen if someone who appeared to be trustworthy were to post a link to a .RAR archive in a public forum. You know, I guess my first thought would be to wonder why it wasn't a .ZIP, since that would be more common. But I might assume that the guy was more techie and, you know, knew about archiving since RAR does outperform ZIP.

I would hope that I would have the presence of mind to drop anything like that first into VirusTotal for its quick analysis and evaluation. And I'll just note for everyone here that it's possible to avoid even downloading a suspicious file first, since VirusTotal can be given a link to download and obtain and then analyze a file without us first needing to do so.

Leo: Oh, that's nice. I like that.

Steve: So that's an - yes. So you're able to just like right-click on the link in the forum, copy the URL, go over to VirusTotal, and then copy that URL into VirusTotal. It will download it and perform its analysis. And presumably it would have exploded in red alerts and everything.

Leo: Do we know that it would have discovered it? I mean...

Steve: You know, it's able to open all of these archives and self-decompressing files.

Leo: So it would see - it would see the malicious EXE.

Steve: Yeah. It would have, yeah, it would have absolutely seen this remote access trojan and gone, you know, "Warning, Will Robinson." Anyway, so in any event, I could imagine that I might open the RAR file since I trust WinRAR. You know? And that's really the crux of it; right? In this instance my years-long trust of WinRAR would have been misplaced. And if my guard was down, I might have been bitten.

So I'm not sure what lesson we take away from this. You know, "Never trust nothin' and live in a cave" isn't a practical strategy. But being unfailingly skeptical of anything being offered over the Internet by someone who you don't actually know probably is a practical strategy. And in fact it's increasingly a survival strategy on the Internet.

And this points to the other mistake those victims made. They didn't really know the person who was offering the download. You know? I mean, it was just someone - and that's the whole idea; right? They wanted what the guy was offering. They couldn't have really known them, or they would have never downloaded the file being offered, or at least they couldn't have had the requirement of really knowing them or they wouldn't have downloaded the file. You know, that old adage of "never taking candy from a stranger," you know, turns out to be as useful for adults as well as it is children. And here some guy who they didn't really know was saying, hey, here's my strategy, and boy, I'm getting rich on bitcoin. So check this out.

So anyway, I suppose our takeaway is just to refresh our healthy skepticism of the Internet and to remember to try to never drop our guard. All it takes, as we've often said, is one mistake to ruin our whole day.

So anyway, as it happens, I mentioned this update to WinRAR last week because I'm a WinRAR user, and there were two exploitable zero-days. And then it turns out, yep, not only were they exploitable, but they had been actively exploited and were biting people. So for what it's worth, you heard me mention that. If you're also a WinRAR user, I would increase the priority of updating your copy to 6.23.

Leo: It's such an old program. Actually, it's modern, but it's old. Does it automatically look for updates and update itself? I bet it doesn't.

Steve: No. No.

Leo: No. Okay.

Steve: And in fact, it's funny you mention that because when I saw that, I opened my copy of WinRAR and went poking around, you know, the Help and About and all that. No opportunity to update itself. There is a click to open the website, and that takes you then to...

Leo: It's on you, man. It's on you.

Steve: Yeah.

Leo: Check the version number.

Steve: And you know, there is a vulnerability. The opposite example of WinRAR is Notepad++. And boy, I don't know what the author of that is doing, but apparently he has too much free time.

Leo: Oh, a lot of updates?

Steve: Because, oh, my god, every time I use it, it's like, oh, there's a new version. It's like, no shit. Of course there is. It's been an hour.

Leo: My Xbox does that. I only play a game once every few weeks. And invariably it'll say, well, I have to download a 14GB file to do that. It's like, no, I don't - please. Thank you.

Steve: Yeah. Now, the good news is Notepad++ is elegant about updating. It'll say I have to close Notepad in order to update. It's like, oh, okay, fine. So it closes it, and then it updates it, and then it says "You want to reopen it?" It's like yes. And Notepad++ does reopen the things that it last had open. So it's not bad. But the point is the more you do these little incremental updates, the more opportunity there is to make a mistake.

Leo: True.

Steve: And, you know, if somebody were to infect his server and get a bad version in there, the whole world would suddenly have it.

Leo: Man-in-the-middle attack, something like that? Yeah.

Steve: So, yeah.

Leo: But this is why I like Linux, and I like the Package Manager idea, which is that if you install everything with a Package Manager, all you have to do is periodically run your Package Manager, and it will update everything that it's installed. That's a very nice - Windows is slowly moving in that direction with Winget. But that's the way to do it, I think.

Steve: Yup. Yup. Okay. So this is another of those episodes where I've recently received so many great feedback questions from our listeners that we basically have a listener-driven podcast. But lots to say about some of the things that they brought up.

John May, he sent me an - is it an X that he sent me, Leo? I hate that. Maybe. I got an X from John May. Anyway, it's actually a tweet, but don't tell Elon. He said: "Steve, I've been watching SN since I retired from my IT job in 2020. In this week's Episode 936" -

so that was last week - he said, "you talked about HTTP going the way of the dodo, if Google has their way. What about on private subnets that cannot be routed over the Internet? Why pop a message or hinder access if the traffic is staying local? I have many local devices I access via HTTP and don't want to add certificates. Hopefully these will be exempt." And he said: "Glad 999 is not the end."

Okay. So yes. The reason we need the encryption and authentication that TLS provides to HTTPS on the Internet is that the flow of packet traffic between the endpoints is out of our control on public networks and potentially exposed to the whims of bad guys. But that's not the case within a closed private network where the network components and all packet transit are local, typically kept within a single residence. As our local devices such as routers, network attached storage, webcams, home assistants, and other IoT gadgets have become more capable, the traditional control panels, you know, filled with lights and buttons and switches, have been all been replaced by web pages published by a simple web server running in the device.

And this is the point that John May is making. I'm sure many of us have had to fight with our web browsers when we wish to connect to a local router or other web interface over a simple HTTP connection. As John May worries, Google appears to be poised to make this somewhat more difficult in the future, maybe to the extent of saying no more HTTP. But I think that's probably going too far. We'll see.

Now, one solution would be to set policy based upon the distinction between publicly routable and private non-routable network IPs. To be clear, when we're talking about non-routable IPs, we're talking about the three networks defined in the early days of the Internet, much to the credit of its early designers because they had so much foresight, which were specified by RFC 1918, a low-numbered RFC from the beginning. The most common that we all see when we're behind a consumer router is the address range that begins with 192.168. Actually it's 192.168.0.0, extending through 192.168.255.255. And that's a group of 65,536 IPs, though for consumer use we usually keep the third octet fixed at, typically at a 0 or a 1, and use the 256 IPs by changing the last, rightmost lower octet.

In addition to the 192.168/16 network, which has been defined by that RFC, it also specifies a private network which is 16 times larger than that one, which goes from 172.16 through 172.31. And the third and final one is 16 times larger than that one, so it's 256 times larger than the 192.168 network, and that's the one that consists of all IPs beginning with 10-dot. And all three of the other octets fall within that 10-dot network.

So yes. Because none of those will be routed over the Internet, it would be tempting to have browsers willing to simply trust all connections to or from IPs within those three private ranges and allow HTTP connections there. But doing so would mean that we trust every IP within that range. In a small residential setting that probably makes sense. But many large corporations also use these same large private network ranges internally for their Intranets. And in such settings, access to raw network traffic is probably not well protected. You know, there's like networking closets around where it's easy to tap into the network flow, so eavesdropping over those networks becomes feasible, and widely allowing HTTP across large corporate networks would probably be extending trust too far.

We typically don't have such security concerns within a small local network, but adopting a general browser policy of not requiring TLS connections for private networks might be too permissive since not all private networks are, as we know, fully trusted. Fortunately, a middle ground that probably makes sense is the use of a self-signed certificate.

As we know, publicly-trusted certificates are signed by a certificate authority that the web browser already trusts. So the browser inherently trusts any certificates that any of its already trusted certificate authorities have signed. Which brings us to the question,

who has signed those certificate authority certificates which it already trusts? It turns out that in every case the certificate authority has signed its own certificate. Certificate authority root certificates are self-signed, and they're trusted simply because they've been placed into the browser's root certificate store.

And in turn, that means that nothing prevents an appliance like a router or a NAS from creating and signing its own certificate. When connecting over HTTPS with TLS, the first time a browser encounters a self-signed certificate, it will typically balk, complain, and say to its user, "Hey! This guy is trying to pawn off a self-signed certificate which, naturally, wasn't signed by anyone I already trust. What should I do? Do you want me to trust it?" To which the user can reply "Yes, trust this certificate from now on." The procedure varies from browser to browser and by version, but basically the self-signed certificate gets added to the browser's internal store of trusted root certificates, and from that point on it will be possible to establish regular TLS/HTTPS connections with encryption and a limited level of authentication.

I say that it's a "limited level of authentication" since, as long as the device in question keeps the private key of the certificate it created to itself, no other device on the network, or anywhere else for that matter, can impersonate it. The user's web browser will have stored and been told to trust the web serving device's matching public key. That's what gets stored in the root store, the public key. So if the user's connection were to ever be intercepted by some other device, there's no way for that other device to reuse the trusted device's public key since it would never have access to its matching private key. So you do get a level of authentication even using a self-signed certificate, as long as the entity which created that self-signed certificate doesn't let go of its private key.

Many years ago I used OpenSSL to create a self-signed certificate with an expiration date 100 years in the future. And I did that because I didn't want to be hassled by the need to keep updating my own self-signed certificates...

Leo: Yeah, that's fair.

Steve: ...you know, as it would expire. And it worked great at the time. I've not tried to do that recently, and I'm wondering, maybe somebody knows and will let me know, whether today's nanny-browsers would complain that the certificate's expiration date is too far in the future. Remember that, you know - and maybe self-signed certificates are allowed to be exceptions. But we know that web certs now can only be valid for up to 13 months in the future. So maybe the day of creating a 100-year life self-signed cert is over.

In any event, for now we're able to tell our browsers to trust local HTTP connections. So, you know, they're not happy about it, but they'll do it. In Firefox, for example, I get a red slash across the padlock when I connect to my local ASUS router or my local Synology NAS. But when I connect to my local pfSense firewall router, the little padlock shows an exclamation point. And if I drill down, like click on the padlock in Firefox and then say, like, show me more, I get a popup. And I'm looking, I put it in the show notes for anyone who's curious, and what I see is that pfSense created a self-signed cert that I had previously asked Firefox to trust. And so it says the website is 192.168.0.1. And so that IP is in the cert and has to match, in the same way that a normal certificate has a domain name which has to match.

Leo: Is this screenshot from Windows XP? It looks a little dated.

Steve: Now, Leo.

Leo: Just a little bit dated. I may be wrong, but...

Steve: Yeah, it's Windows 7.

Leo: Okay. It's nice, I like it. I remember those days.

Steve: So anyway, and then it says "Verified by." And the verified by where it would normally say DigiCert, for example, it says "pfSense webConfigurator Self-Signed Certificate." Anyway, so - oh, and I clicked on it. That self-signed certificate is valid from January 3rd of 2020 through January 25th of 2025. So that's not bad. That's four and a half years. That's a nice range. And that does suggest that self-signed certs are allowed to have a longer life than current certificate authority signed certificates, which are limited to 13 months. Remember, that's that 397 days.

So anyway, I noted that Chrome was not happy with the pfSense self-signed certificate since I had not bothered to also tell Chrome to trust it. But I don't really care because I don't use Chrome that much. Anyway, there is a simple process for adding the certificate to Chrome's root trust store.

So anyway, I just kind of wanted to go over that for background. I can't see any of the browsers removing the ability to add our own trust roots in the form of self-signed certificates.

Leo: No, you have to. You have to.

Steve: And it's something that you only need - yeah, exactly. And it's something that you only need to do once for the life of that certificate. So it would be nice...

Leo: It is on you to go, whoa, this is self-signed because it's my device, as opposed to I'm out on the Internet and it's self-signed.

Steve: Right.

Leo: My Synology lets me do either that or use Let's Encrypt. And I actually would prefer to use Let's Encrypt, but I have to open a port for it and blah blah blah blah. It's kind of annoying.

Steve: Exactly. Exactly. You have to have a public domain name on the Internet in order to do that. So, yup.

Leo: Which I do. I do. But then the script that it runs has to get out through that port and blah blah blah.

Steve: Yup, yup.

Leo: So I just use the self-signed certificate.

Steve: Yeah. Or you just say, you know, in fact for me I just said, yeah, I don't care if the Synology NAS has a red slash through the padlock because I'm happy to talk to it on HTTP. I mean, I'm looking at it. It's sitting right here next to me. So no bad guys are in there.

Jack Skinner said: "Steve, I was listening" - oh, you know, Leo, it's 36 minutes in. Let's take a break.

Leo: Let's take a break. Good thinking. Nice time management. Well done. Much better than mine. I usually get to the ads three quarters of the way, and I go, "Oh, god, I've got a problem here." Okay. On we go. Xeets. You like xeets, by the way, X-E-E-T? Xeet? I xeeted it.

Steve: Yeah. Jack Skinner said: "Steve, I was listening to this week's podcast and had some feedback in regards to having third parties host your email versus running your own. There's a third option. Purchase your own domain name and use a third party that allows its use."

Leo: Oh, I do that. Right. Of course.

Steve: It's brilliant. That's like the...

Leo: So if Fastmail went away, it's still going to my email address. I just make it go somewhere else.

Steve: Exactly. Yeah, duh.

Leo: Oh, but wait. If we're using the obfuscated address, that might not work because they have to translate it. So what I do - which is why, by the way, I don't use those weird obfuscated addresses that 1Password and Bitwarden and others will let you use. I just have a domain, I won't say what it is, and then I can use anything with Fastmail, anything I want, at that domain.

Steve: Multiple accounts, yes.

Leo: Yeah. So every single, when I signed up for Verizon, it was Verizon at this, I'll say it, laporte.email. And so every time I sign up for something it's that person's name or that company's name at laporte.email. So I do control that. If Fastmail went away, it'd still work.

Steve: Exactly. So he says: "Then if the third-party host shuts down, you simply point your domain name to a new provider, and you have access to setup and use those email addresses again."

Anyway, yes, Jack, terrific idea. It moves a user's email away from an end user's ISP who we know is probably hostile to having an email server running out of their home, I mean, you virtually can't do it any longer. You know, but the user still maintains full control over their domain. So, and I did a quick Google search. There's a bunch of email providers. Zoho is one. Blue something is another. Anyway, there's a bunch that will allow you to bring your own domain to them. And then what you want is the ability to have multiple accounts. And so you just create email accounts.

Leo: Right.

Steve: And they also talk about, you know, having email routing services and so forth. So anyway, yes, that's [crosstalk].

Leo: And that's what Fastmail, our sponsor, does. They do the DNS for at least a dozen of my domains. So I can get email at any of those domains. Whenever I set up a new website, I don't want to handle the email at that website. I have Fastmail do it. And so I can use the MX record to point to Fastmail, and the other record, the A record, to point to the website.

Steve: And being able to use your own name, I mean, it's like the reverse of AOL; right?

Leo: Oh, yes.

Steve: It's a good domain instead of an embarrassing domain.

Leo: Right. No one at this point should ever be using a company's name as their domain for their email. Even if using Gmail you should be using your Gibson.com or something. You shouldn't be using Gmail.com.

Steve: And also, Jack said, he said: "Also, related to Google Topics. With the restriction that an advertiser will only get a topic if they have first seen that user on a different website related to that topic, do you think that this will give Google added power to be a monopoly in this space?" He said: "It seems like sites will have the incentive to use a single advertiser."

So that's sort of an interesting point. That actually had occurred to me before. The fact that an advertiser is only able to obtain topics for a visitor at a given site when that advertiser has previously seen them at a site that's associated with the topics that were going to be provided, favors larger advertisers with greater reach since the user's browser will have encountered that advertiser at many more different websites, thus they'll be more likely to qualify for receiving topic information about that user.

Now, the answer to the question Jack asks at the end is unclear to me. That is, he says, it seems like sites will have an incentive to use a single advertiser. If websites are reimbursed by advertisers based on how well targeted their visitors are - which is to say,

how many topics are being returned to the advertisers on their site - then larger advertisers having greater reach would have superior targeting. If they were to reimburse more, then that site might choose to place more of their ads with them than with some advertiser who's paying less. But that logic would appear to apply just as well to today's advertising climate as much as in tomorrow's Topics-based system, if that's what ends up surviving.

So anyway, it's unclear to me how Topics furthers a monopoly. But it certainly does further the larger advertisers because the way Google has designed this, Topics are being filtered essentially based on the reach of an advertiser. So it does sort of suggest we're going to see further consolidation and fewer larger advertisers on the Internet. And I don't really have any basic sense for what the spread of advertiser size and reach and so forth is. Are advertisers hugely geographically oriented, or oriented by country, or language, or what?

So anyway, it certainly is interesting that we have topics that we do. And I forgot to even bring up the issue of language. Obviously Topics would be language-independent because it's actually an index number into a list of topics. And so they would be, you know, index numbers are language neutral. And then the lists get translated into whatever language.

So anyway, Donn Edwards said: "Hi, Steve. If uBlock Origin is blocking all the ads on a website anyway, what difference does it make if Google Topics is on or off?" He says: "As far as I can tell, a typical website doesn't use the Topics API, only the sites that actually serve the ads. So if no ads are served, no topics are registered with the browser. Am I wrong?"

And anyway, so that's a good point. I said, but - this gave me an opportunity to say at least the way I use uBlock Origin is to allow "Acceptable Ads." And I've got that in quotes because that's like a term. You know, I'm not, and we talked about this a long time ago, Leo, back in the Adblock Plus days, I'm not against seeing ads on websites if it helps them to generate revenue. And as I've said, I'm not even against them knowing a little something about me, as long as it's not invasive of my privacy, if, again, it helps them to generate revenue. I'm against having things flashing at me and screaming at me and jumping up and down, you know, in a futile attempt to get my attention. That's annoying.

So there is this Acceptable Ads initiative which identifies advertisers who agree NOT to do any of that. So it's not the case that as a uBlock Origin user I'm seeing no ads, I'm just not being driven into an epileptic seizure by the tame and considerate ads that I do see. And for anyone who doesn't know about this, it's AcceptableAds.com is the site that sort of runs all of this.

And there was some controversy about larger advertisers being able to buy themselves into that list of acceptable ads. But all I can tell you is that, when I look at somebody's web experience who's not doing this, it's like I can't believe they survive. And whereas mine, I'm like unconscious of there being any ads. They're just not, you know, in any way annoying to me. So again, I'm happy to provide some support. If putting the ad in front of me generates some revenue for the website, great. It feels a bit like a cheat because I'm not clicking on any ads. But fine. I'm happy to pay the price.

Leo: What brand of coffee do you buy? What kind of laundry detergent do you use?

Steve: Yeah. Fortunately...

Leo: Brands work. I hate to tell you.

Steve: Yeah. I do, yeah, I...

Leo: You may not click on the ads.

Steve: So you mean just sort of seeing it in front of you.

Leo: Ads work.

Steve: And then you...

Leo: Yeah.

Steve: Yeah.

Leo: Yeah. I mean, you don't buy a no-name laundry detergent. You buy Tide. Right?

Steve: Right.

Leo: You don't go to Joe's Coffee Shop, you go to Starbucks. So the ad doesn't have to work by getting you to click it. I'm not a fan of ads either. But in fact I think one of the reasons Netflix and others have done so well without ads is people are thrilled to watch a TV show, I loved watching "The Diplomat" without any ads. Right? And you can even see when you're watching these shows, it originally was on, what, FX? You could see where the denouement happens, and they pause because it's time to sell Tide, and then they come back because there's no ad.

Steve: And in fact the ad is so disruptive that they sometimes go back a little bit in the show to remind you...

Leo: Right, right, what you might have forgotten.

Steve: ...what you were watching, exactly.

Leo: Yes.

Steve: If you were the first purchaser of a TiVo, I was second in line.

Leo: Yeah, yeah, yeah.

Steve: Because I cannot watch live broadcast television. It's just, it's impossible.

Leo: All I will say, though, is you can't both not watch ads and complain about the subscription. You've got to, somehow, this stuff has to be paid for. It ain't free. And, you know, I know that because TWiT doesn't exist in a vacuum. We have to either join the Club for 7 bucks a month or watch ads. I don't care. You do whichever one you prefer. But you've got to pay, I mean, Steve's not cheap, folks. He's not that expensive, either, but, you know. We've got to pay people. We've got to pay the lights.

Steve: Okay. So a bit of math here, which I loved. Philip Le Riche, he said: "Voyager 2's antenna does not produce a pencil beam. The 3.7 meter dish at S-band, which is around 10cm wavelength, has a limited resolution of around the arctan of 0.1/3.7, or 1.5 degrees. So who knows, that might be around 10dB down at 2 degrees off axis. So not so incredible that they were able to boost the power to shout and get a command through. To say its beam was off the earth by 5AU is therefore somewhat misleading."

So Philip, I love the math, thank you. And that does explain why they even bothered to shout at Voyager 2. I'm sure they knew exactly how attenuated their signal would be and thought that there was a chance that the message might get through anyway. Anyway, very cool. We have the best listeners, Leo.

Peter Chase said: "Re the Windows time problem. I use an ancient program called Dimension 4 which polls various time servers."

Leo: Oh, wow. There's a memory. Wow.

Steve: Do you know Dimension 4?

Leo: Yeah, oh, yeah. Long time ago.

Steve: Didn't ring a bell for me. He said: "It polls various time servers," he says, "in my case every 30 minutes, to keep my computers on time. After listening, I went in and turned OFF 'Set Time Automatically' in Windows settings. Was this a relevant thing to do, or does this issue just apply to servers?"

Leo: Oh, interesting question, yeah. Dimension 4 comes from a day when you didn't have NNTP, reliable NNTP servers you could get the time from. Remember you had to set time manually in the old days, yeah.

Steve: Yeah. Yeah, yeah, yeah. And I remember like listening to WWV go tick tock, tick tock, tick tock.

Leo: The time at the tone. Yeah, yeah.

Steve: So, okay. So it appears to be relevant, that is, this whole STS thing that we talked about last week, and we'll talk again about it in a second, to Windows 10 and on, and Server 2016 and on. In other words, yes, I know that my own Win10 workstation did have the "UtilizeSslTimeData" value in its registry. So Windows 10 Pro has this STS time setting. However, this is different from "Set Time Automatically," which I believe should be left on. It's right there in the Control Panel, says Set Time Automatically. Windows normally references an Internet time server at time.windows.com. So I think it makes sense to leave that feature enabled.

And when I opened my Control Panel under Windows 10, it showed when the last time that server was polled, and Windows made sure that it was still set correctly. So I would not turn off Set Time Automatically. I would, if you're concerned about this, I would turn off that UtilizeSslTimeData in the registry. It's a DWORD normally set to 1, set it to 0.

And SKYNET, who tweets from @fairlane32, he said: "Steve, I'm curious. Is the STS 'feature,'" he has in quotes, "only in Domain Controllers?" And we know now that it's not, it's in all Win10 and both desktop and server variants. He says: "I have two servers running Server 2019 Standard, and they've never exhibited different clocks. They're not Domain Controllers. That said, I do have a bunch of Dell Optiplexes," and he says, "as you know." He's participating over in the SpinRite development work. And he said: "...that communicate with one of those servers because of our time and print management software for the public access computers. I have had in the past few weeks a few workstations with an error message during boot up that says 'WARNING, clock not set!' and giving me the option to hit F1 to try again or F2 to go into the BIOS setup to configure the clock.

"I go to the BIOS, and the clock is off. Not by 137 years, mind you, but it's still off. After resetting it and rebooting, it's fine. Sometimes I don't even bother. I just reboot the machine, and Windows boots fine, and the clock shows the correct time. Is this behavior because of STS" - no - "or is the local CMOS chip getting, shall we say, tired and sleepy?" Which is the case. And he says: "Thanks for the info."

Okay. So motherboards generally have a very long-life battery, which keeps a very low power oscillator running 24/7 to keep track of the time and date. But the batteries do die over time, and sometimes the clock is not read properly. Actually, I recently struggled with this with SpinRite, since some motherboards disable hardware interrupts and cause time to be lost while SpinRite is running because SpinRite's calling into the BIOS when it's talking to USB devices, and they literally - time freezes from the perspective of all programs running outside of the BIOS. Hardware interrupts are just off. You know, the keyboard won't work. Nothing happens.

So I needed to be reading the motherboard's real-time clock on the fly, and it turns out many are surprisingly flaky. Part of the problem is that this is ancient technology, and the real-time clock uses what's known as a "ripple counter" instead of what's known as a synchronous counter. With a ripple counter, that means that each digit overflows into the next higher digit so that when a digit wraps around from 9 back to 0, its carry ripples to the next digit to the left in the counter. So it's possible to sample the count while it's in the middle of updating and "rippling," and as a result you'll get a bad time reading. So for SpinRite I designed another heuristic algorithm to figure out what time it really was without being fooled by inaccurate values. And so far it appears that mine was designed somewhat more carefully than Microsoft's, you know, that Microsoft did for Windows 10 and later, and for their server.

And Superevr is his tweet handle. He said: "Strangely, the TLS spec going back to the very first SSLv1, has a client hello that includes" - and he's talking about the handshake - "that includes the Unix time stamp, but acknowledges" - this is SSLv1. "The spec acknowledges that: 'Clocks are not required to be set correctly.' In 2013, someone

drafted an RFC titled 'Deprecating gmt_unix_time in TLS.'" And he includes a link. He says: "But it doesn't look like it ever left the draft phase. The acknowledgement of its futility raises the question of why it's there at all."

Okay. So this brings us back to last week's discussion of heuristics. I left last week's podcast curious myself. So I did a bit of googling and digging around. Here's what I learned: We've spoken of cryptographic protocols which require a nonce, a nonce standing for a number used once. Typically, nonces do not need to be secret, but they do need to be unique. If nonces are not unique, then in the case of TLS, this opens the opportunity for replay attacks. This brings us to the question of how to produce a nonce that's guaranteed to be unique.

One way, which was suggested in that earliest SSL document, SSLv1, was to concatenate the Unix time, which as we know is a 32-bit binary count that changes once per second, to concatenate that 32-bit binary count, the Unix time, with another 32 bits of random noise. Together they would produce a 64-bit nonce. Each endpoint's 64-bit nonce would be concatenated with the other's, that is, that's what the handshakes are sharing, among other things they are each sharing their 64-bit nonce. They each then concatenate them to produce a final shared 128-bit nonce, which is then used to drive the TLS crypto.

Now, when you think about it, if the requirement - and it's a firm requirement - is for an endpoint to never make the mistake of reusing a nonce, one way to do this would be to start with that 32-bit Unix time which is going to change once per second, then have a second 32-bit count which just increments once every new connection. The only way for such a system to issue a duplicate nonce would be for that second 32-bit connection attempt counter to wrap around and return to a previous count before the 32-bit Unix time, which changes every second, had changed, which of course means that you'd have a 32-bit count counting from zero to 4.3 billion before it wraps back around to zero, do that within a second. 4.3 billion TLS handshakes for one machine in a second? No. That's not going to happen.

So that's a useful solution. You're not going to get a repeated nonce just by taking Unix time, changing it once per second, and concatenating it with a connection attempt counter, which is 32 bits. So it's going to have 4.3 values before it repeats. A second will have elapsed before that 4.3 billion count has had a chance to wrap around.

Now, okay. Somebody might suggest that using a simple counter to form the second half of that 64-bit nonce is a bad idea since it would make the system's nonce easily predictable. Now, that's not a known problem for TLS; but, you know, known problems have a tendency to become - or unknown problems have a tendency to become known problems. So there's a simple and secure solution. Take a 32-bit cipher, using a randomly chosen key, and use that to encrypt a 32-bit counter to produce an unpredictable sequence of 32 bits that would therefore never repeat in less than 4.3 billion uses.

Okay. Anyway. But here's the point. The only requirement for that 64-bit handshake nonce is that it never repeats. That's it. The use of Unix time has always merely been a suggestion in the TLS specification, not a requirement.

Leo: Interesting.

Steve: It was just one way that, you know, this could be done. Which is why the OpenSSL folks decided to never do it.

Leo: Yeah.

Steve: Back in December of 2013, so nearly a decade ago, right, because here we're in '23, two guys, one with the Tor project and the other with Google, produced that TLS Working Group Internet-Draft standard with the title "Deprecating gmt_unix_time in TLS." And the brief abstract, it's one line, or it's actually two lines, reads: "This memo deprecates the use of gmt_unix_time field for sending the current time in all versions of the TLS protocol's handshake. A rationale is provided for this decision, and alternatives are discussed."

Okay. I'm just going to share the introduction that's three paragraphs, and it's interesting. They wrote a decade ago: "Current versions of the TLS protocol, dating back to the SSL 3.0, describe a gmt_unix_time field, sent in the clear, as part of the TLS handshake." And we know that it actually goes way back to SSLv1, but they're only talking about what was then current. "While the exact format of this field is not strictly specified, typical implementations fill it with the time since the Unix epoch (January 1, 1970) in seconds. This practice," they write, "is neither necessary nor safe."

Leo: Wow.

Steve: Yeah. "According to RFC 2246, the TLS Protocol Version 1.0, gmt_unix_time holds 'The current time and date in standard Unix 32-bit format (seconds since the midnight starting January 1st, 1970, GMT) according to the sender's internal clock.'" And they're still quoting from the spec. "Clocks are not required to be set correctly by the basic TLS Protocol; higher level or application protocols may define additional requirements." So they said: "This text is retained unchanged in RFC 4346 and RFC 5246," meaning subsequent iterations of the TLS spec.

They said: "The gmt_unix_time field was first introduced in SSL 3.0, the predecessor to TLS 1.0. The field was meant to preserve the protocol's robustness in the presence of unreliable random number generators that might generate the same random values more than once. If this happened, then SSL would be vulnerable to various attacks based on the non-uniqueness of the Random fields in the ClientHello and ServerHello messages. Using the time value in this way was meant to prevent the same Random value from being sent more than once, even in the presence of misbehaved random number generators."

So, you know, we spent a lot of time in the early days of this podcast talking about bad random number generators and how easy and common, unfortunately, it was to have them. So the idea was pair up a 32-bit random value that may not be that random with something that is incrementing once per second because that way, even if the random value comes back around before too long, at least the time will have changed so that concatenation, the 64-bit concatenation of those two 32-bit values will itself be unique.

So anyway, they go on to explain that, as I suspected and mentioned last week, one of the problems with using Unix time was that it could make clients identifiable by serving as a form of fingerprint. But mostly they noted that the only reason for using Unix time was that it was a convenient means for changing the bits in half of the 64-bit nonce every second to protect against a repetition in the other 32 bits from a poor random number generator.

Okay. So this brings us back to Microsoft. The bottom line is that Microsoft incorrectly assumed that every incoming TLS connection was supposed to contain the 32-bit Unix time of the other machine that it was connecting with. And now we know that's not true.

It's at best a suggestion and a convenience for the implementation. The original spec even explicitly states that connecting machines are not required to have the correct time, and that's still in today's spec. It's just meant to be an incrementing value. I'm certain that Microsoft built in some sanity filtering of their own. That's why it mostly works. And then they have that "confidence level" heuristic that we talked about last week.

But the underlying assumption that handshakes will contain GMT time, especially when the massively popular OpenSSL doesn't, that's flawed. And then, as we've seen, they're apparently shockingly stubborn about fixing this issue. It's hurting their users, and they apparently refuse to care. I wanted to close the loop on the question of TLS handshakes and Unix time, and now we know what's going on.

Okay, now, Leo, last week I mentioned that I had a tantalizing something else that I wanted to share.

Leo: Yes.

Steve: About SpinRite.

Leo: I've been on tenterhooks ever since.

Steve: So let's take our last break, and then I'm going to talk about something that we discovered which is really quite terrifying.

Leo: Oh, no.

Steve: Oh, yes.

Leo: That's a good tease. Now, I'm on tenterhooks, Steven. Tell me.

Steve: Okay. So I mentioned last week that I had something else I wanted to share. One of our very prolific testers, whose handle in our newsgroup and in GitLab is "millQ," that's just how we've always known him, he reported a month ago, on July 27th, that one of his oldest, actually his oldest, 256MB flash memory devices - megabyte flash memory devices - was showing up RED in SpinRite. RED means that there's something SpinRite doesn't like about the drive. None of his other USB flash drives had any trouble, just that one.

Now, one of the great things about GitLab for me is its issue tracking. I'm desperate not to let any problem fall through any cracks, but I cannot be everywhere all the time. As it happened, when millQ filed this issue, the gang was testing the latest SpinRite build while I was working on the Windows app which installs SpinRite onto USB drives, diskettes, and CDs to make them bootable. So the DOS side wasn't where I was focused at the time. But once the Windows utility was ready for its first testing, I turned the gang loose on it and switched back to SpinRite.

The error that SpinRite was reporting was that millQ's flash drive had failed SpinRite's transfer confidence testing. Although SpinRite would allow the user to retry the test or

even to push past the failure if they want, SpinRite couldn't vouch for how it was working with that drive.

Now, remember that I mentioned last week that SpinRite now has an extremely flexible driver that learns about the environment it's in. In order to make absolutely positively certain that the driver has settled upon the right parameters for a drive and controller, it performs a series of confidence tests on each drive it encounters.

It first fills a nine-sector buffer with pseudorandom noise and then copies it to a secondary buffer. It then goes to the end of the drive to read the drive's last nine sectors into that first buffer, which should overwrite all nine sectors of that pseudorandom data that was preloaded there. It then compares the two buffers to verify that none of the sectors match. That assures us, and actually it assures SpinRite, that reading from the media into RAM is actually happening because the read will have changed all nine sectors of the read buffer, which will then no longer match the noise that was preloaded there.

It then takes the nine sectors that it read and copies them into the secondary buffer. It inverts the primary buffer and writes the nine sectors of inverted data back to the drive. It then refills the buffer with new pseudorandom data, then rereads the data which it just wrote, which should now be inverted, if it was actually written, over the buffer. So it then reinverts the buffer then compares all nine sectors with the original drive data that was first read and saved. And it verifies now that all nine sectors match between the two buffers. And, finally, it rewrites the originally read data back to the drive, restoring it to its original contents.

Okay. Now, SpinRite goes through those gymnastics to absolutely and positively confirm that it's able to properly both read and write to the drive across multiple sectors. Those tests and comparisons will fail unless everything is working. There are a number of other things that SpinRite does after that, but it was that confidence testing that millQ's flash drive was failing. And what's significant is the drive wasn't complaining. The drive was saying everything was fine.

Now, several years ago I might have doubted SpinRite more than the drive. But we now have 764 registered users in GitLab, so SpinRite has been quite well pounded on by this point. And we don't see drives failing this. This stood out. It was completely unusual. I should note that read and write errors reported by the drive are fine if they occur at the time. If the drive says that it's unable to successfully read or write, that's no problem since that's not what we're testing for here. So SpinRite will simply move nine sectors over toward the front of the drive and try again. And this entire process is tolerant of storage errors. Again, that's not what we're looking for.

So here's the "gotcha" of millQ's drive: It was failing this test and was not reporting any errors. SpinRite told it to read nine sectors, and it said it did. And SpinRite confirmed that nine sectors had indeed been read. Then SpinRite told it to write nine sectors, and it said it did. But when SpinRite asked it to read those nine sectors back, they were not the nine sectors that had just been written. This drive was accepting read commands and write commands and apparently ignoring the write commands.

Since SpinRite now has the ability to check an entire drive this way, millQ started SpinRite at its new Level 5 which reads, inverts, writes, reads and verifies, then reinverts, writes, then rereads and verifies. What millQ discovered was that at exactly the halfway point, the drive silently stopped writing. It also sped up considerably, which was interesting.

Leo: Well, it wasn't doing anything else.

Steve: Exactly. Since writing to flash is much slower than reading. So millQ's drive was labeled 256MB. And a query for the drive's size, which is what SpinRite did, declared itself to be 256MB, but it was only storing 128MB. And here's the real gotcha: No operating system would detect this. Operating systems assume that, unless a drive reports an error when writing data, that data was properly written. And modern drives take on a lot of responsibility for making sure that that statement is true. What was diabolical was that the drive appeared to be functioning perfectly. It was formatted with a FAT file system, and millQ had used it to successfully store and retrieve a great deal of data through the years. But perhaps he'd never stored more than 128MB of data there.

Now, when this was discussed with GRC's spinrite.dev newsgroup, many people there, some who have a deep background in data recovery, were not the least bit surprised because it turns out that - and this is why I'm bringing this up - there is an epidemic of fake drives flooding the retail market. I suppose in retrospect we should not be surprised. But I wanted to make sure all of our listeners were aware.

Since the FAT file system, which is the default file system for all such drives because it's universal, the FAT file system places all of its file system metadata at the front of the drive. You know, the famous FAT, you know, stands for File Allocation Table, which is what gives the file system its name, and the root directory and so forth are all written at the front of the drive.

A 16MB compact flash card might have its firmware tweaked to lie about its size, then be relabeled and sold as a 512MB compact flash card. Such a card will format without any trouble, and it will appear to be working perfectly. Then a wedding photographer sticks this brand new card into their camera and spends Saturday taking photos before, during, and after a wedding, only to later discover that although a directory listing shows that all of the photos taken are there, the actual content of those files is not. And those precious memories are irretrievably lost forever.

This, it turns out, is a surprisingly common occurrence. Just google "fake ssd" or "fake thumb drive" and be prepared to get bored scrolling the screen. It is endless. When this came up last week I jumped onto Amazon and purchased the cheapest 2TB thumb drive they had. It's silver and beautiful. I have a picture of it in the show notes. It's got that blue USB tongue to indicate USB 3.0. And that 2TB cost me $26.88.

Leo: And that's when your troubles began.

Steve: Exactly. Exactly.

Leo: By the way, you said in retail, but I think Amazon is probably the source of 99% of this crap.

Steve: And eBay. Amazon and eBay.

Leo: Oh, yeah, eBay. And I guess, you know, Alibaba, if you've bought stuff there.

Steve: Yup, Alibaba and AliExpress, exactly. So this thing arrived last Friday. I plugged it into SpinRite, and the screen immediately turned red. That drive immediately failed SpinRite's end-of-drive data storage test. I was curious about what it was doing; so I stepped through the writing and reading and writing process at the end of the drive, and

its behavior was really bizarre. It seemed to sometimes write, but not reliably. And as I kept poking at it, it suddenly stopped failing, and it began working correctly - at least for those nine sectors at the end. You know, were some bad spots remapped so they're now good? Maybe.

But remember we're talking about, if you want to believe this, 16 trillion bits of storage, that's right, you know 2TB, right, bytes. So eight bits in a byte, 16 trillion bits of storage in a tiny sliver of metal-enclosed plastic. I'm skeptical that such a drive contains much wear-leveling technology, or much technology at all. I'm actually skeptical that it contains 16 trillion bits, but we'll get to that in a moment.

What I know is that it was not initially working, and that a bit of exercising it, you know, it sort of began to appear to work, at least at the end of the storage region. Who knows if other "unexercised" areas of the drive function similarly. Relying upon this brand new drive from Amazon would be a disaster. We know that. And remember, writing to it and reading from it produced no errors. That is, it just said, yeah, fine, got the data, move along, everything's working.

Leo: That's what's scary, yeah.

Steve: Yes. So there's no indication that it's not storing everything, when in fact it may not be storing anything. Now, one possibility is that not all flash storage is created equal. Chips are manufactured in large wafers, then cut into individual pieces and tested. So there's some manufacturing testing that's done to qualify each chip for sale. What happens to chips that fail to make the grade? You know, say that they don't get an F, but perhaps they get a D-. You know, you have to imagine that there's a market for chips that fail to make the grade. Someone will buy them and package them for sale on Amazon or eBay or AliExpress or Alibaba. It looks like memory. It just isn't very good or reliable. And maybe it isn't memory at all.

But I think this is a relatively new phenomenon that has arisen in the last few years. I'm inclined to believe that millQ's drive, since it is so very old, may have just failed. It probably once was a 256MB drive, and the chip that was providing the second half of its storage died. But there is absolutely no doubt that fake mass storage is a very real problem today. And Leo, I have another picture in the show notes, if you scroll down a bit. This was sold as a high-end SSD. For those who can't see the picture...

Leo: Are those matchsticks in there? What the hell? What is this?

Steve: So that is an SSD case. It was sold as a high-end SSD. That's ballast to give it some weight.

Leo: Oh, my god. And then it's all held together with glue stick.

Steve: With hot glue.

Leo: Oh, hot glue.

Steve: Yup. And so that's a thumb drive that was stuck into a little - to a USB-to-USB adapter. They even ran wires from the light of the back of the thumb drive over around to an LED...

Leo: Well, at least they took the extra time.

Steve: Exactly. So that this thing would flash.

Leo: Oh, my god.

Steve: And if you didn't know, I mean, so somebody purchased this, believing that it was an SSD with all of the endurance and longevity that you would...

Leo: I love it. To make it feel heavy they've got a couple little nuts in there, and I don't know what those copper things are, just some crap lying around the shop, just to give it some weight, some heft.

Steve: Yep, yep.

Leo: Oh, my god.

Steve: So it doesn't feel empty.

Leo: And then they glued it down so it wouldn't rattle.

Steve: Exactly.

Leo: Wow. This is awful.

Steve: This is what is out there, Leo.

Leo: So this could be a big moneymaker for you if SpinRite could somehow - because, you know, obviously the drive controllers are trusting the drives, and the drives are lying.

Steve: Yes.

Leo: It would be really valuable if SpinRite could actually check and see what the capacity of a drive was.

Steve: So the first question is, how can we tell if a solid-state memory is fake? Depending upon the cleverness of the memory controller, which is to say, to which lengths its perpetrator goes, there's actually only one way to know for sure, which is to simultaneously have every sector of the memory committed and in use, and then verified. You'd fill the entire memory with deterministic pseudorandom data.

Leo: That's not ideal. That's going to take time and wear it out.

Steve: Yes. Well, yes, one pass of writing, but you're right, it's not ideal. This is done to prevent data compression which is apparently another trick...

Leo: Oh, god.

Steve: ...which is being employed by some cheaters. So the amount of data you can store is a function of what you store because they're compressing the data. That's also been seen. So once it's filled in this way, the entire memory would then be read to verify that it still contains the deterministic pseudorandom data. This is the capability and the capacity that the memory claims it offers, and of course it's what you believe you've purchased. Anything less is a cheat. And there's actually no way to fake-out that test.

Now, if you really want to know for sure, it's necessary to do this because a much smaller memory could be arranged as an MRU, a Most Recently Used cache, so that if you moved through the memory, reading and writing it piecemeal, that would be fooled, even though the entire memory could not be used at once because it didn't exist.

Okay, now, Leo, I'm right where you were. Performing this true full test is beyond the scope of SpinRite 6.1. But now that I know this is happening out in the world, SpinRite 7 is going to incorporate this full pseudorandom fill-and-verify technology because this is the sort of ultimate assurance that SpinRite should be able to offer.

Now, having said that, in today's reality, it seems unlikely that scammers are going to go to the trouble of engineering an MRU cache to fake out lesser tests. As it is today, SpinRite 6.1 will instantly disqualify any fake storage that places its reduced memory at the front of the drive. Now, it could place some at the front and some at the end. And then SpinRite, you know, wouldn't detect that; 7 will. So - I forgot what I was going to say. Oh. So 6.1 would instantly detect it if there's no storage at the end, just as it did with millQ's drive. And since this has been a recognized problem for a while, a handful of freeware utilities have been created to detect fake solid state storage. I've not looked at them, so I can't vouch for their operation or behavior. But they do exist, and they're easy to find.

So I don't know what I purchased from Amazon for $26.88. It looks all shiny and new, but no way can that be trusted to store data. I have a feeling that I'm going to run a little test of my own in the background to fill that drive with serialized data and see if it, like, actually can be read back after it's been written. We'll see. But we do know that it was not generating an error when I was telling it to write something, and it said it did, when in fact it didn't.

Leo: This is really problematic because there's really no way to tell what you're getting.

Steve: Nope.

Leo: Unless you open - I guess you could open it up. But even then, it wouldn't necessarily tell you anything.

Steve: Nope.

Leo: I bought a 10TB drive a while ago, and for the longest time I couldn't figure out why my Synology wasn't working. And I realized, well, this drive is not doing 10TB, not doing what it said it was going to do. And I wonder if that was also a scam drive. Brand name. I think it was probably Western Digital Red. But who knows, it could have been, you know, it was bought on Amazon. Some guy could have taken a label and stuck it on.

Steve: Well, and remember we once talked about and showed on the show a Seagate drive where the word "Seagate" was misspelled.

Leo: Misspelled, yeah.

Steve: Because, you know, it looked close enough. So unless you looked very closely, you didn't realize.

Leo: I know you're going to put this in 7. Couldn't you just write a little utility? Just, you know, like Steve's little utilities that just fill the drive up, just said yeah, I see, I'm able to write a terabyte worth of stuff here.

Steve: Well, in order to do it right, it needs to be fast because these drives are big.

Leo: Yeah, it's a lot of data.

Steve: And then it needs to follow that fill with a complete trimming of the drive.

Leo: Right.

Steve: To formally release all the storage and show that it's no longer in use.

Leo: This is a real problem.

Steve: It is a real problem.

Leo: Golly.

Steve: Yeah.

Leo: Oh, well. What are you going to do with that 2TB thumb drive? I'll take it off your hands if you don't want it. I can use the storage. Oh, lord.

Steve: Wow, I know.

Leo: All right. Man-in-the-middle time.

Steve: And finally. Today's topic, The Man in the Middle, was inspired from another question from one of our listeners, Jon David Schober. He said: "Hey, Steve. Quick question regarding insecure HTTP traffic. Even if the site has no user interaction, like logins, wouldn't it being HTTP still put the user at risk for malicious injection of stuff like JavaScript miners, or overriding downloads to include malware? Seems like something that HTTPS would secure, even if the site were just read-only."

Okay, now, that's certainly a great point, and many of our listeners wrote to say the same thing. So I chose Jon's at random to represent all those who asked essentially the same question. And everyone is correct in their observation that the lack of authentication and encryption would mean that if someone could arrange to place themselves in a man-in-the-middle position and intercept traffic flow in an intelligent fashion, then an unsecured conversation could be altered.

So I wanted to take a moment to expand upon this, to first establish a bit of perspective and to also examine the practicality of man-in-the-middle attacks which are, I think, far more easily said than done. It's their practical difficulty that explains how we survived in a mostly unsecured Internet until Let's Encrypt came along relatively recently.

We should remember that the lack of persistent encryption was the way most of the Internet worked until only very recently, and things mostly worked just fine without everything encrypted all the time. As those of us who've been around for a while will recall, it was once the case that encryption was only employed on most websites when a login form was being delivered, and its user's name and password contents were being submitted. The sense was that it was only during that brief interchange where eavesdropping really needed to be prevented. I'll be the first to say that was probably never really true. But at all other times, most websites were using more economical HTTP connections whose traffic was not safe from even eavesdropping.

Our long-term listeners will recall "Firesheep." That was a Firefox add-on which provided a convenient user-interface for hijacking other people's logged-in session cookies, and thus their logged-in sessions. As I reported at the time, I tried it at my local Starbucks. And that is a brand we all recognize. I never impersonated anyone, of course. But the Firesheep web user interface quickly populated with all of the logged-on accounts of the other patrons with whom I was sharing free WiFi. And all I would have had to do was click on one of those icons to be logged into the websites they were using as them. It was a sobering experience.

Now, today, thanks to the pervasive use of TLS and HTTPS, we have communications privacy so that scenario cannot happen. So it's definitely a good thing that we have more security today than we did in the past. No question about it. But the use of HTTPS and TLS is also, let's remember, not an absolute guarantee. I'll remind our listeners of something that we covered at the time. A few years ago, the headline of a story in BleepingComputer read: "14,766 Let's Encrypt SSL Certificates Issued to PayPal Phishing

Sites." 14,766. Every one of those connections to those 14,766 - sounds like the number of votes I need - fake PayPal sites...

Leo: I love it.

Steve: ...was authenticated, encrypted, super secure, immune to any man-in-the-middle interception, and malicious. So, you know, just the fact that we have the padlock glowing and happy doesn't mean we're okay. We know that the reason our web browsers have been gradually downplaying the increasingly pervasive use of the HTTPS that they've been driving, which boasts its padlock, is to not give their users a false sense of security just because there's a padlock, and all is well. Well, thanks to Let's Encrypt, that may not be the case.

So how was it that we survived as long and as well as we did before most websites had their communications encrypted all the time? The answer is, there is a world of practical difference between passive eavesdropping, which is trivial, you know, you can do it in your local Starbucks - or could - and active interception. Eavesdropping, as in the Firesheep example, is trivial, whereas active traffic interception is not something that's practically available to common malicious hackers. And that explains how the pre-encrypted Internet managed to survive until we got it encrypted.

I've often talked about practical traffic interception. Once upon a time, antiviral content scanners were installed at the network borders of corporations to maximize and filter all of the traffic passing through on the fly. Encrypted connections, however, posed a problem. So the solution was to create "middleboxes," which would fully proxy such connections by intercepting and terminating the TLS handshakes at the box, then establish a second connection to their user's browser within the organization. Since the traffic-intercepting middlebox needed to create certificates to stand in for those remote websites it was proxying, they needed to have some means for browsers to trust the certificates they were creating and presenting.

The controversial and underhanded way this was done for a while was by obtaining a certificate from an already trusted certificate authority where that certificate itself had bits set in it which permitted it to sign other certificates. In that way it was possible for the middlebox to transparently create certificates on the fly as if it were a trusted authority. The industry quickly clamped down on that practice since allowing random hardware to create globally trusted certificates was extremely dangerous and prone to abuse. So today, any TLS proxying middlebox that wishes to create trusted certificates on behalf of other websites can only do so by installing, you guessed it, its own self-signed root certificate into every browser's trusted root store.

Leo: Nice callback, Steve.

Steve: This essentially makes it a trusted certificate authority, but only within the organization that's behind it, being protected by it, and whose browsers have installed its certificate.

So both in the earlier unencrypted days and today, when nearly everything is encrypted, it's possible and practical for hardware to be installed into a network which allows for traffic interception; filtering; and, potentially, alteration. But, again, such interception hardware resides in a privileged position that is not available to common hackers.

I was trying to think of an instance where traffic was actually being intercepted against the wishes and best interests of its users. And I remembered that there was a time when some slimy ISPs were caught injecting their own crap into the unencrypted HTTP web pages of their customers. Now, there is a perfect example of why we would want and would benefit from having end-to-end encryption of our web traffic. No one wants to have their ISP monitoring their activity, literally everything that they do, let alone injecting their own crap into content that our web browsers receive.

So, yes, there is an example, which was not theoretical, of an entity in a privileged position - our ISP whom we're paying to carry our data and connect us to the Internet - betraying our trust and not only spying on our activity, but injecting its own traffic into our connections. Again, to actually pull off a man-in-the-middle attack, however, requires that the attacker arrange to have a privileged position where they're able to intercept, capture, buffer, analyze the traffic that's flowing past. It is easier said than done. But as we've always seen, where there's a will, there's a way. And as those examples show, adding encryption to connections makes that job decidedly more difficult.

In doing a little more brainstorming, I came up with one way that Google's new "always try HTTPS first" approach would help. One of the other attacks we talked about in the past was the protocol downgrade. In such an attack, an attacker would arrange to convert all of a webpage's HTTPS URLs to HTTP. The browser wouldn't know any difference and would make its queries over HTTP. But with Chrome's automatically trying HTTPS upgrade, those rewritten HTTPs would be treated as HTTPS's, thus thwarting the attempt to downgrade connections for the sake of keeping a browser's communication unencrypted.

I think that what Google is doing is a good thing. Trying to upgrade an HTTP URL to HTTPS prevents any protocol downgrade horseplay and provides more security for its users. And like many of our listeners who hope that we don't lose all use of good old HTTP - especially for local connections to our own appliances where encryption and authentication offer little, if any, value - I doubt that losing HTTP is in the cards anytime soon. After all, it took quite a while for FTP to finally be deprecated and removed from our browsers, and the File Transfer Protocol was never as popular as HTTP remains today. Actually I'm somewhat surprised by Google's statement that 5 to 10% of their traffic that they're monitoring of Chrome's own users remain over HTTP. That seems surprisingly high today.

Leo: Yeah.

Steve: You know, they can't just be all from me clicking on my non-encrypted...

Leo: But it would include that; right? Yeah, it would include that. So, and I think that - I think we've talked about this before. Somebody said that in enterprise it's very common that you would have an HTTP link because it's within the network of...

Steve: Okay. That makes sense. So, right, that Intranet traffic would not be encrypted, yes.

Leo: Exactly, right, right.

Steve: Anyway, so answering the question does everything need to be HTTPS, I'm still not convinced. I understand the arguments, and I agree with our listeners who pointed to the potential for man-in-the-middle attacks. We've just reviewed how difficult, but possible, such attacks actually are. They require an attacker to be in a privileged position with some serious hardware. HTTPS makes that more difficult, but not impossible in every case.

It seems to me that the way things are today is probably right. If some random website, like the example I gave last week of Ctyme.com, which publishes that wonderful list of PC and DOS interrupt APIs, if they don't care enough to encrypt their site, even now that certificates to do so are freely available with minimal effort, then that ought to be up to them. Am I going to avoid using such a site because it's not encrypted? No. I am put in much greater danger by downloading files over HTTPS from file archives. That's truly terrifying. And I avoid doing so at all costs, but sometimes there's no choice. And of course the first place that file goes is over to VirusTotal to have it scanned to make sure that, as far as anyone knows, it's okay.

As for HTTP-only sites, I think it ought to be the site's and their visitors' decision whether the site wants to remain unencrypted, and whether their visitors want to visit. Overly demonizing the lack of authentication and encryption to me seems wrong, especially when Let's Encrypt is cranking out fraudulent TLS certificates at a phenomenal rate which are then being used to spoof legitimate websites which have full authentication and encryption. So there.

Leo: Yeah. I mean, this always bugged me. I'm reading, I've mentioned this many times, but I'm reading a book about political power called "The Power Broker" about a guy who designed New York City over the wishes of the electorate because he wasn't elected. But the question, the big question is do the ends justify the means. And so this always bugged me that Google, the ends Google was going for made sense. HTTPS should be everywhere. But the means bug the hell out of me, which is we will give you a better ranking in the search results if you do HTTPS. Because those two shouldn't be related.

Steve: They're using their power.

Leo: And it's a way of using your power to achieve something that on the face of it is good, but is it ever okay to misuse your power even if the ends justify the means. And it really bugged me, and I still don't think so.

Steve: Yeah. I agree. And in fact you've just summed up exactly what infuriates everyone about national politics is the constant display of the people who have power abusing that power.

Leo: Well, and if you read this book, you realize it's much worse than you thought. And the guy who wrote it, Robert Caro, is now working on - he's got four volumes out, a fifth volume on his LBJ biography, and there's a perfect example of somebody you could look at and say, you know, in many cases what he did, like the Civil Rights Act, really was great. But then if you look at the means, he was very corrupt; you know? And is it okay ever to use your power in that way? And I don't know if it is. I think it ends up tarnishing the ends.

Steve: No, it's about ethics; right?

Leo: It's ethics, yeah.

Steve: It's sort of beginning to be a lost form.

Leo: Well, that's really the eye-opener on this Robert Moses book is politics has always been - it's the worst. Just the worst. And it's filled with people who have that notion that, well, okay, it's okay to cheat and lie and steal a little bit because I'm trying to do something good here. And that's the problem; you know? That's the problem. Steve, always really a great show. This is really fun. I'm so glad that we are going to continue on. Here we are at 937. It's right about now that I would start to be thinking there's 62 more episodes, I don't know.

Steve: Yup. I couldn't stand the countdown.

Leo: No, I don't want the countdown.

Steve: It's gone.

Leo: And you know what, as I said, you can quit at Episode 1000, if you want. You're only really obligated to go to a thousand. And then you've satisfied your promise.

Steve: No bait-and-switch here, baby.

Leo: No, that's not even bait-and-switch. And if at any time it becomes onerous, you just say, yeah, you know, I think I do want to retire. Because I feel the same way; you know? I'm going to keep doing this as long as it's fun. As soon as it's not fun, there's no reason to beat yourself up.

Steve: Thanks, buddy. Bye.