When Hashes Collide

Description: This week, after quickly filling Leo in on last week's two most important pieces of news, guided by some great questions and comments from our listeners, we're going to look into the operating of hardware security modules (HSMs), fast file hash calculations, browser identity segregation, the non-hysterical requirements for truly and securely erasing data from mass storage, a cool way of monitoring the approaching end of Unix time, my plans to leave Twitter, and what I think will be a very interesting deep dive into cryptographic hashes and the value of deliberately creating hash collisions.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-940.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-940-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is in the house. So am I. And we have lots to talk about. We'll talk about fast file hash calculations, why you don't have to overwrite your hard drive over and over and over again. Steve explains the issues in wiping your hard drive, whether an SSD or a spinning drive, and then why he's leaving Twitter. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 940, recorded Tuesday, September 19th, 2023: When Hashes Collide.

It's time for Security Now!, the show where we cover the latest news from the security world with the king of all security news, Steve Gibson. Hi, Steve.

Steve Gibson: My hand looks pretty big. I'm going to move it back further so it gets smaller.

Leo: They told me - so I did some of the shows from Mom's house last week, and they told me my head was too big. So I'm going to do the same thing next time.

Steve: Well, let me tell you, I know.

Leo: It's easy to get too close, exactly. Thank you, Jason Howell, for filling in for me last week. I really appreciate that.

Steve: Oh, and by the way, I did want to mention, because you were concerned about audio volume, the last two podcasts have been absolutely uniform.

Leo: Good.

Steve: So whatever it was that was causing those weird volume diminishes, diminishments, diminishings...

Leo: Diminuendos.

Steve: Didn't happen.

Leo: We fixed them.

Steve: Yes.

Leo: Good.

Steve: So. We have a really fun podcast. This is going to be a goodie. As I said before we began recording, we're going to start by quickly filling you in on two important pieces of info. One I already know you know about because you mentioned it on the previous Sunday show, before your absence last week. But just need to keep synchronized, and you may have some things to add, as well, of course. Then, guided by some great questions and comments from our listeners, we're going to look into the operating, the more detailed operation of hardware security modules, since that question came up; the need for fast file hash calculations; browser identity segregation; the non-hysterical requirements for truly and securely erasing data from mass storage; a cool way of monitoring the approaching end of Unix time and you know, maybe the end of the simulation that we're all in.

Leo: I forgot that was your theory.

Steve: You know.

Leo: I hope to god I'm here in 2038, that's all I can say.

Steve: Also my plans to leave Twitter, Leo, and what I think will be a very interesting deep dive into cryptographic hashes and the value of deliberately creating hash collisions, thus Episode 940 of Security Now! is titled "When Hashes Collide."

Leo: Well, you're going to leave Twitter? I can't wait to hear that.

Steve: Yeah. Probably.

Leo: Maybe.

Steve: Maybe. Probably.

Leo: He's talking about now charging everybody, which I think would cause a grand exodus, to be honest.

Steve: And Leo, that is the catalyst, as a matter of fact.

Leo: Yeah, yeah, yeah. That's a little - that's going a little...

Steve: If that happens, that ends it, I mean, it just - that's the last straw.

Leo: I think he'll back down on that one. I can't imagine going through with that. But anyway, we'll find out.

Steve: Because it will kill Twitter. It will objectively kill it.

Leo: Yeah, yeah. Picture of the Week time, Steve.

Steve: So, yeah. We have - it's a two-frame picture. We have, like, two stuffed-shirt-looking guys, dudes, wearing coats and ties. And this looks like a real event, maybe from TV, because even though the second shot is the person over whose shoulder we're looking in the first frame, you can see like the brick in the building is the same. So it's actually, you know, this kid was actually, you know, facing these two stuffed-shirt guys. And in the background you can also read what looks like Campus Place and Imperial College and then something about Engineering New something. So anyway, the caption, the question being posed in the first frame is these two stuffed shirts are looking at him saying, "So." This is a job interview. "What makes you suitable for this job?" And this very serious-looking kid who does look sufficiently uncomfortable in his shirt and tie, which is good, he says: "I hacked your computer and invited myself for this interview."

Leo: That's one way to get a job.

Steve: So there.

Leo: I love it.

Steve: Okay. So Leo, I needed to make sure you knew about this. Last week's podcast was titled "LastMess," and that's because...

Leo: Wait, there's more?

Steve: Oh, baby. Yes.

Leo: Oh, geez.

Steve: Brian Krebs reported the very strong and still mounting evidence that selective decryption of the stolen LastPass vaults has been occurring and that the cryptocurrency keys stored in the decryption targets have had their funds emptied.

Leo: I did see that story. We actually reported on it on TWiT before I left.

Steve: Oh, okay.

Leo: Once LastPass was hacked, then the next question was are they going to be able to decrypt the vaults.

Steve: Right.

Leo: Brute-force decrypt them. Or is it a nation-state or maybe a competitor who's just trying to smear LastPass? And so we were just waiting. And it does seem like there's some evidence that - and what they're doing is, makes sense, cherry-picking the most valuable accounts.

Steve: They have no interest in logging into Facebook as Aunt Mabel.

Leo: No.

Steve: You know, these guys, they want one thing: money. And so they're going to target LastPass early adopters with low iteration counts because LastPass screwed up by never increasing those proactively. And the analysts who looked at this noted that these are not neophytes. These are people in the crypto industry. In fact, that's how they're being targeted; right? Because we know that the email address is in the clear. It's in plaintext.

So by scanning through email addresses in the 25 million-plus vaults that were exfiltrated and stolen, they're able to identify targets who they know by their email address the companies that they're with, and go, hey, there's a good chance that this guy has cryptocurrency. He may have made the mistake of giving LastPass his keys, didn't know it was a mistake at the time.

Leo: Well, a mistake I wish I had made, but didn't. So I can't get into my wallet. I kept looking in LastPass. So I don't have to worry about this. But that's interesting.

Steve: And the iteration count is also in the clear, as it must be, because you need to know the count in order to know how to iterate, in order to decrypt the vault.

Leo: Unbelievable. Unbelievable.

Steve: So they know where the low-hanging fruit is by low iteration counts. They know who the people are, meaning that they're likely targets. And then if the iteration count is low, they can put a whole bunch of GPUs on the task of doing a brute-force crack. Apparently they're able to crack a couple, I don't remember if it was a couple per week or per month. But it's been, you know, it must be per week because there have been enough of them since the theft. And apparently about $35 million worth of LastPass users' cryptocurrency has been stolen.

Leo: And that was the giveaway, or maybe not giveaway, but the clue that it might be because of the LastPass breach.

Steve: The one common factor.

Leo: Because that was what was in common among all these accounts.

Steve: Yup. Yup.

Leo: And it makes sense.

Steve: [Crosstalk] they're security-aware, they were early adopters.

Leo: They probably had strong passwords, but they were early adopters. And so the PBKDF2...

Steve: Well, actually one guy admitted to only having an eight-character password and a low iteration count. It was like, ooh.

Leo: He was the first.

Steve: And this guy thought, you know, maybe - this was two months ago, before the news broke. He said, maybe I should move my cryptocurrency. He lost $3.5 million. So he had already set up another wallet. And he was just procrastinating. He said, yeah, you know, he's like, he was ready to go. And he didn't do the transfer. So the 3.5 million was still sitting in his wallet whose keys were stored in LastPass, and now it's gone.

Leo: Wow.

Steve: So no new home for you. Okay. Oh, the other thing I just wanted to make sure, and I know you did talk about this, was the U.K. apparently blinking on this issue of, you know, no-exception CSAM monitoring. The OFAC or OFAM, their regulatory body, added a clause to this, you know, the online child protection act added the clause "where technically feasible and where technology has been accredited as meeting minimum standards of accuracy in detecting only child sexual abuse and exploitation content."

Well, that wasn't there originally. That got added. And so as far as all of the secure crypto companies are concerned, that's their out card. I mean, they're done. This is no longer a problem. The politicians get to say, oh, look, we enacted really like the strongest legislation we could. And all, you know, Apple and Signal and WhatsApp and everybody is like, okay. Oh, Telegram. Okay, yeah, great, glad you did that. You know, we will be able to keep offering our services because it is not technically feasible to do what you have asked us to do. And this added clause here late, just before the legislation closed, you know, gives us our out.

So of course now the big question is, what will the EU do? But the gal that's running Signal was extremely happy, saying that this broke the inertia that had been mounting among the various nation-states about this, and it looks like now it's probably not going to happen. Again, they'll all claim that they enacted great legislation.

Leo: Right.

Steve: So they can always get themselves reelected.

Leo: It's perfect.

Steve: And it won't matter.

Leo: It's a perfect solution.

Steve: Yes. Everybody wins.

Leo: Everybody wins.

Steve: Everybody wins. Okay. So Chris Smith tweeted. He said: "I'll politely ask that you ponder the scalability of your personal HSM and whether it could even keep up with the I/O of your own domain. Yes, it would be awesome if prevailing Enterprise-class HSMs could commonly do more than just securely hold the keys. Rainbow crypto accelerator cards paved the way for ubiquitous TLS, and perhaps someday Enterprise HSM will similarly step up." Okay, now, so Leo, what Chris is referring to is that we learned how it was - we learned last week how it was that Microsoft lost that key.

Leo: Wasn't that a story?

Steve: Yes.

Leo: And I'm sure you said this. But credit to Microsoft for being very honest about this. Very forthright.

Steve: Except for one thing.

Leo: Oh.

Steve: So, yes, honest.

Leo: It was a very detailed technical description of what happened.

Steve: It was a technical description. The fumble was the key was in RAM.

Leo: Yeah.

Steve: When the server crashed. Which caused a snapshot of the crash dump to then, as a consequence of five bugs...

Leo: I've seen security experts tweet. You would be amazed at how many crash dumps contain secrets. That's the first place you look; right?

Steve: Yes. And of course, and so Microsoft knew this; right? So they actually had filters, secret filters in the pipeline that this crash dump moved through, and they all missed it. So they said, so there were like five bugs that all had to be there for this crash dump to make it onto a developer's computer out of the production environment into the operating environment, which then allowed China to come in and grab it. And but, okay. So yes. There were bugs. But the original sin was that this key was in RAM. And this brings us to Chris's point because I was saying that even GRC, I have a cute little HSM, which is where my cosigning...

Leo: What is an HSM? What is that?

Steve: So, yes. And that's, okay, so that's the issue. That's a Hardware Security Module. Okay. So we actually have two questions here. We have Chris's. So he's referring to my comment from last week about Microsoft having lost control of their secret signing key, which then allowed bad guys to sign on their behalf and thus gain access to their customers' enterprise email. And so the point is it should have never been in RAM.

The question raised is why, in this day and age, where we have hardware security modules - so an HSM is very much like a TPM, a Trusted Platform Module, which we all now have built into our motherboards. So maybe Microsoft will address this oversight. But I made the point that even GRC's code signing keys are locked up in HSMs and are thus completely inaccessible. Crash or no crash, my keys never exist in RAM and are thus never subject to theft.

Now, Chris reminds us that cryptographic signing imposes significant overhead, like compared to none, right, because that's what you're comparing it to. Either you don't have any signing, or you do. And if you're going to sign something, there's a computational cost to doing that. So he said: "And that the use of custom hardware for accelerating this process has historically been necessary." Okay. True. But the hardware for doing this, and at today's Internet scale, exists today.

Now, it's true that my little USB-attached HSM dongle would melt down if it were asked to sign at the rate that Microsoft probably needs. But apparently Microsoft already had software doing that on a general-purpose Intel chip running Windows, which is what crashed. Whereas, to Chris's point, special-purpose hardware can always be faster. So my point, for whatever reason, was Microsoft was not using such hardware. And it is difficult to discount that as being anything less than sloppiness. And it did bite them.

Now, in the show notes, Leo for the next question I have a picture of the HSM that I'm using.

Leo: Why, it's just a dinky little USB key.

Steve: Yes. That's it. Okay. So Rich said: "Hey, Steve. Thank you for all your contribution to the security community, and delighted to hear you're extending your tenure to four-digit episode numbers. In Episode 939 you mentioned that the GRC code-signing key is sequestered in a Hardware Security Module. I'm familiar with HSMs in finance and banking, but would you consider a technical teardown of what HSMs are, and how they operate without exposing the contained key material, for the Security Now! audience? Many thanks, Rich, in Sunny Devon, UK."

Leo: Oh, that's a nice question. And Devon is not sunny, so okay.

Steve: Oh. So maybe it was a - that's why he made a point - oh. It'd be like Sunny Seattle.

Leo: Yeah. Well, when it is sunny, though, it's such a red-letter day I can see why he might sign it that way. It's like a big deal.

Steve: Yeah, so, yeah. Okay. So Rich's question, of course, followed nicely from Chris's. So the idea is simplicity itself. The particular key I use came from DigiCert already containing the code signing key I obtained from them.

Leo: Oh. That's cool.

Steve: Now, that was several years ago, and I've since updated it several times. What's significant is that the signing keys which this little dongle stores are write-only. This little appliance - and this is true of HSMs on any scale - lack the ability to divulge their secrets. They just don't have it. Once a secret has been written and stored in the device's non-volatile memory, very much like a little memory stick, it can be used, but it cannot be divulged.

And this brings us to the obvious question: How can the key be used without it ever being divulged? The answer is that this HSM is not just storage for keys. It contains a fully working cryptographic microcomputer. So when I want to have some of my code signed, the signing software outside of the HSM creates a 256-bit SHA-256 cryptographic hash of my code. Then the resulting 32 bytes of the code's hash are sent into the HSM, where they're encrypted using its internal private key, which never leaves. It can't leave. There's no read key command. All you can do is give it something to do, and it does it.

So the 32 bytes of the hash of my code are sent into the key, and the encrypted result is returned. So the whole point of this is that all the work is done inside the HSM, by the HSM, and only the encrypted result is returned. There is no way for the key to ever be extracted from the hardware dongle. It doesn't have any "read key" command, only "write key."

And just to complete a statement of this entire process, the so-called "signature" that's then attached to my code is the matching public key, that is, the public key that matches the private key inside the dongle. So it's the matching public key which has been signed by a trusted root certificate authority, in my case by DigiCert. So when someone downloads and wishes to verify the signature of my code, the signature of its attached public code signing key is first checked to verify that the public key was signed by DigiCert. So that verifies DigiCert's assertion that this public key belongs to GRC. Then that public key is used to decrypt that encrypted hash which is the signature. And the private key can only correctly decrypt the hash if the hash was encrypted by its matching private key which GRC has and which is embedded in that HSM.

So then the rest of the code that was downloaded is hashed with SHA-256, and that hash is compared to the decrypted hash which the private key decrypted. Only if they match do we know a whole bunch of things. We know that GRC's matching private key encrypted the code, and that because the hashes match, not one single bit of the code has changed. Otherwise the hashes would never match. So although this whole process does have a bunch of moving parts, when everything lines up, it works.

Leo: It's quite clever. That's quite clever, yeah.

Steve: And it is really clever.

Leo: I'm thinking about, I don't know, there must be some analogy we can use to describe this. Essentially the key knows who you are. But it has a secret of yours that only the key has and will never reveal. But it can use that secret to create now public keys which it can then send out into the real world. And then somebody getting that public key - this is how public key crypto works - getting that public key knows that it's you because the public key can only be created by somebody who knows your secret, in effect; right? That's how I log into my SSH server, which is it has my public key. It doesn't have my private key. Only I have my private key. So when I log in with SSH, SSH says, yeah, this is him because the public key matches the private key, which only I have. And then they've added this really nice additional feature because you still need this hash of your code to verify the code's not been modified.

Steve: Right.

Leo: So they take that hash. They I guess hash it, right, with your public key? Or...

Steve: Well, so yeah. So they take the hash, I take the hash when I'm publishing this.

Leo: Right.

Steve: And I encrypted the hash.

Leo: Ah.

Steve: With my private key.

Leo: Ah, you've encrypted it. So it only can be decrypted by you, essentially, by your key.

Steve: No, it's encrypted by me. But it could only be decrypted...

Leo: By the public key.

Steve: ...by the public key.

Leo: Got it.

Steve: And the public key is signed by DigiCert attesting that it's GRC's public key.

Leo: Very clever. That's a nice system. I like that.

Steve: It's beautiful, yeah.

Leo: That's taking kind of the simple public key crypto I use to do my SSH and adding this additional feature that it verifies some code.

Steve: Yes.

Leo: You could do that, when you download code from open source, you download the code, but you'll also download a hash that needs to match the key that is the public key that is posted on the key servers to make sure that that developer did write this code, and it's unmodified. So we see that all the time without this certificate thing.

Steve: Right. And the nice thing for Windows users is if you right-click on the file and bring up the little property dialog, one of the tabs up there will say "digital signature." And so you're able to click on it, and it says, you know, this signature's been verified, and it shows who signed it.

Leo: So it shows DigiCert in this case, yeah.

Steve: Yes, yeah. And so not only does Windows itself, you know, all of Windows' wacky security stuff, see that a valid signature signed the code, which tends to calm down Windows antiviral kneejerk reactions, but then the user is able to easily see who signed it themselves at a given time.

Leo: Very nice. Very nice.

Steve: It's just a slick system.

Leo: Public key crypto is amazing. It's brilliant.

Steve: Yeah. It is so cool. And Leo, you are going to love this podcast's topic when we get there.

Leo: Okay. Okay, good.

Steve: Because something more very cool is going to happen. Okay. So Clint, and I withheld his last name so as not to embarrass him, he said: "Hey, Steve. Longtime Security Now! follower. My name is Clint. I wanted to bring to your attention that I just fell for a Verizon wireless fraud call."

Leo: Oh.

Steve: Uh-huh. He said: "Since I wasn't aware of Verizon's fraud policy" - meaning that they will never call you - "I responded to a call that said someone tried to log into my account. In doing so, I actually gave them information they obviously didn't already have. After calling to talk to Verizon directly, I've changed my account pin and my password. I'm working on letting Social Security and other services I have know of potential activity that would not be me. Hopefully nothing comes of this as I changed my info quickly. But I'm still being proactive on this. Just thought I would share this in case it's something everyone needs to be aware of new that's going around."

Okay. So it's not that new. But this seems significant since Clint explains that he's also a longtime Security Now! podcast listener. The lesson here is, not knowing whether or not this is Verizon's policy, which really shouldn't factor in, it's "skepticism first." And never, for one moment, let down our guard. It's unfortunate because it's not the way humans are built; right? We tend to believe our senses, which is what stage magicians depend upon. So when something reasonable and believable appears to happen, most people's first reaction is to believe it and to take some action. Clint now knows that his first reaction should have been to place his own call to Verizon, rather than accepting a call ostensibly from Verizon. And in fact he knew it then; right?

But these scammers count on their targets getting caught up in the moment and reacting to an emergency rather than taking the time to think things through. So I cannot say this often enough, which is why I've decided to say it again. It doesn't matter how many bits of encryption you have. The human factor remains the weakest link, and the bad guys are incessantly clever. I dread the day when I will make such a mistake; you know? And it could happen, you know, to any of us. If I could type with my fingers crossed, I would. So anyway, just another reminder.

Stand_Bold said: "Greetings from a dedicated listener from Bharat, India." He says: "I found your site around 2004, and it's always given me peace of mind. It was comforting to learn everything was secure then, and thanks to your weekly episodes, I remain confident. Referring to last week's episode about checksum verification with VirusTotal, here's a handy tip for Windows users unfamiliar with the command line."

Because I had mentioned that you could use certutil-hashfile and the filename space SHA-256. And somebody subsequently noted that SHA-256 is the default, so you don't actually have to be specific about that. Anyway, so he said: "Install the 7-Zip file archiver." He says: "This adds a CRC SHA verification option to the Windows right-click menu." He said: "I frequently use it to confirm file checksums. By the way, it's delightful to learn you'll be continuing the show for the foreseeable future. Many thanks."

Okay. So this listener's comment caused me to check my own Windows right-click file context menu, and I have a checksumming option there, too. I wanted to mention that there are a number of simple checksumming add-ons for Windows and Linux desktops. My file properties dialog has a "checksums" tab. I was just talking about how it also has a "digital signatures" tab when the file you've right-clicked on has a digital signature. I always get a "checksums" tab. And I recalled deliberately installing it many years ago. And right-clicking on a file and using this tab is the approach I prefer. I think it makes sense for file checksumming to be, because it's so useful, I mean, it's like a signature; right? So it now allows you to very quickly make sure that two things are the same. The shell extension is a solution I've been using for years. And having something at your fingertips means, as we know, you'll tend to do it more often.

So I have a couple links in the show note. The one I was using was - it's called "hashcheck." It's open source. It has not been updated since 2009, unfortunately. It's only unfortunate because it works great because it does not support SHA-256, which I think any state-of-the-art checksummer probably should. It's not that SHA-1, which it does support, is not good enough. It's that VirusTotal is using SHA-256, and other things you may be wanting to cross-reference with use SHA-256. So it should, too. For what it's worth, the source code is published. He offers the source code. It's got a Visual Studio project in it. So it would not be difficult for one of our listeners to update that.

And in fact, if one of our listeners wants to, I will be happy to tell everybody about it because it's really very clean and very cool. And I'm not going to take the time away from my work to do that because I know that we have some listeners who could. So the link is in the show notes. There are also two other solutions on GitHub, both open source, one called HashCheck and one called OpenHashTab. So the options abound. And I just wanted to say, hey, you know, it's easy to add that. Oh, and there is also one for Linux desktops for the rest of our population.

Leo: We do it by command line, Steve. We don't need no extensions.

Steve: Leo, I know that you get out your remarkable tablet, and you just do the math by hand.

Leo: I calculate it by hand, of course.

Steve: That's what real - real men calculate 256 SHA by hand.

Leo: Yeah, no, there's plenty of command line tools that'll do that on Linux, and a lot of times in Linux you've got the terminal open, and you just say, well, check the hash on this.

Steve: Yup. Okay. So finally, before our second break, Chris Shearer said: "Hello, Steve. This week you mentioned running a separate browser for segregating extensions.'" Actually for segregating, like, you know, logged-on cookies and so forth. I was saying, if you normally use Firefox, you could use Chrome and not have extensions loaded there and so forth.

Anyway, he said: "Chrome and Edge have a 'profiles' feature where you can run the browser separately as a different identity with different extensions, sessions, cookies, et cetera. I've used this to maintain sessions to multiple Azure, Gmail, or internal app sessions simultaneously. This is great for separating dev, prod, test, personal, et cetera. They even get separate toolbar icons in Windows so you can launch right into the one you want." Meaning launch the browser with the profile that you want.

He said: "In Edge and Chrome you click your avatar or icon, and there's a profiles area where there is an option to make a new one, or use a guest profile, either of which will be completely vanilla." Then he finishes: "I believe Firefox has similar features, but I don't use it as often." He says: "999 and beyond plus."

Okay. So I had completely overlooked profiles, so I'm glad Chris reminded me and anyone else who might have a use for them. As always, I strongly prefer using features that are built into our existing systems rather than adding superfluous and unnecessary extras. And sure enough, in Firefox, entering "about:profiles" took me to my "profiles" page where I saw that I currently have only one profile in Firefox, but there is an option to create more. So that's very cool. And thanks, Chris, for pointing that out. And, you know, and that would allow you to have like your banking or super secure profile where you don't load all the other extensions, all of those requiring trust because they all have access to whatever you do while you're working with your browser. So a very nice tip. And Leo, let's tell our listeners why we're here.

Leo: Yes.

Steve: And then I'm going to do a deep dive into what's actually required to securely erase mass storage.

Leo: That's a good subject. And we've talked about this before, especially with SSDs. It's not...

Steve: The question is this whole business of overwriting hard drives.

Leo: Right. Oh, oh, yeah, yeah. Okay, good. Can't wait, yeah.

Steve: Uh-huh. Okay. So Atli, A-T-L-I, Atli Davidsson, he said: "Hi, Steve. Longtime listener and SpinRite owner. I'm giving away an external USB hard drive and want to securely erase the drive. A few years back I stumbled upon the format 'P' option to specify how many times to overwrite the drive. There's a lot of mixed information on how many times to overwrite the drive, so I picked seven, as some article suggested. What

would be the sufficient times to overwrite the drive? Some say three. Some say one. I used three times for my SSD a few days back, and now using seven for a large spinning drive which will probably take days. I probably should have at least swapped those numbers and used a larger value for the SSD? Any thoughts? Looking forward to the next few hundred Security Now! episodes."

Okay. So as I've mentioned previously, since Atli's problem is going to become increasingly common for many of us, GRC's planned second commercial product to follow SpinRite 7 will be an easy-to-use, fast, and extremely capable secure data erasure utility. Its sole purpose will be to safely and securely remove all data from any form of mass storage. It will understand every nuance, strategy, and variation available and will take care of every detail. But since that future utility is not yet available, here's what needs to be done in the meantime. And of course even after it's available you could still do this and be doing just as good a job. But it won't be, you know, that advice won't be updated constantly and automatic, and it won't, you know, you won't necessarily know all the features that each device offers. So that's where it makes sense, I think, to bring everything together into a single place.

Okay. But the first thing to appreciate is that there are similarities and differences between the wiping needs for spinning versus solid-state storage. The differences are largely due to their differing physics for bit storage, and their similarities are largely due to the need for defect management and wear leveling. So let's look at solid-state storage first.

The first thing to note is that "electrostatic data storage," which is the physics underlying all current solid-state storage, has no notion of remnants or residual data. That is, in the individual bits. The individual bits of data are stored by injecting and removing electrons to and from individual storage cells. This process leaves no trace. Once those electrons have been drained off, nothing remains to indicate what that cell may have ever contained, if anything. Consequently, nothing whatsoever is gained from multiple writes to an SSD. The bigger problem with SSDs is wear leveling.

As I've mentioned before, this process of electrostatically injecting and extracting electrons across an insulating barrier inherently fatigues that barrier over time. Earlier single-level cell (SLC) storage was rated to have an endurance of at least 100,000 write cycles. The move to multi-level cell (MLC) storage has reduced that by a factor of 10 to around 10,000. Okay. Still, 10,000 writes to every bit is a lot of writing. So that's why in general solid-state storage is still regarded as being more reliable than mechanical storage.

But the problem is that, if only 5GB of a 128GB drive, solid-state drive, was written over and over, those remaining 123GB would remain brand new and unused, while the first 5GB that was being read over and over and over would be fatiguing due to the stress of being written over and over. Which is a real thing for this form of solid-state storage. So the process known as "wear leveling" was quickly introduced. Wear leveling introduces a "mapping layer" which is interposed between the user, you know, in the outside world, and the physical media. This is known as the FTL, for Flash Translation Layer.

So now say, just say for example that the first 4K bytes of memory is read from a Flash memory. The user then makes some changes to that data and rewrites it back to the same place. What actually happens is that the rewrite occurs to some lesser-used physical region of memory; and the FTL, that flash translation layer, is updated to point the LOGICAL address of that memory region to the PHYSICAL location where the updated memory now actually resides. This beautifully spreads the fatiguing that's being caused by writes out across the entire physical surface of the memory to prevent the overuse of any highly written memory addresses.

But there's a downside to this when it comes to data erasure. Remember that the 4K bytes were originally read from the beginning of the drive, at that location. But then they were rewritten to another location. And the FTL was updated so that the next read of that same 4K byte region will be from the most recently written location where the updated data was stored.

But what happened to the original 4K byte region that was read first? When the FTL is updated to point its address to the new location, it becomes stranded. It is literally unaddressed physical memory. If the FTL does not have a mapping for it, it cannot be reached from the outside. It's still there. It still contains an obsoleted copy of the 4K bytes region's previous data. But there's no way to access it any longer. And if there's no way to access it any longer, there's no way to deliberately overwrite it with all zeroes or whatever, to assure that its old data has been permanently eliminated. You know, you can't get to it.

Now, okay. You might say, if there's no way to access it, then no one else can either. But that's probably not true. Solid state memory controllers, which is where this FTL lives, contain undocumented manufacturing commands to allow external manipulation of the FTL. So while it's not easy to do, it's likely that there are powers higher up that could perform recovery of all obsoleted data, even after all of the user accessible data had been wiped clean. And if the information was important enough, it's also possible to desolder the memory chips, which are separate from their controller, and access them directly, thus bypassing the FTL entirely. And there are videos on YouTube showing how this can be done.

It is for all of these reasons that solid-state mass storage added special "secure erase" features to erase data that cannot be accessed through the normal I/O operations of reading and writing from the outside. When a secure erase command is sent to solid-state memory, all of the device's memory is wiped clean, whether or not it is currently accessible to its user. This command is carefully obeyed by today's manufacturers since there's no upside to being found not to be doing this correctly.

So in short, the only thing that needs to be done for any solid-state drive which supports the secure erase feature is to arrange to trigger the command, then sit back and wait for the command to complete. Once that's done, the device will be free to be discarded or given away to somebody else. It will be empty.

Leo: So, I mean, this is a big deal because for a long time we said you can't really effectively erase solid-state because of this slack space problem.

Steve: Right.

Leo: And that's good to know.

Steve: You actually can.

Leo: Yeah. That's really good to know.

Steve: Although you need to know how to issue the command. There are various levels of command. Some drives will allow you to securely erase only the unused area so you get to keep the used area.

Leo: This is why we need Steve to write an app, by the way.

Steve: That's why I believe there will be a place for this.

Leo: Complex, yeah.

Steve: And for what it's worth, I would still do one pass of overwriting just because, you know, belt and suspenders. Do one pass of overwriting, which does not fatigue the device very much when you consider it's got 10,000 available. And then tell it now erase yourself, and that way, you know, you know for sure. But what's interesting, Leo, is the story with hard drives is somewhat murkier due to their long history and some bad advice that has outgrown its usefulness. And that's coupled with some panic, you know, that aliens may have more advanced technology than the NSA, which would enable them to somehow unearth layers of previously written data from long ago. Okay.

Leo: Here comes my favorite line of the whole thing. Go ahead. Keep going.

Steve: So here's what I believe to be true for any hard drives manufactured since around the year 2000. The earliest hard drives used an extremely simple encoding of user data into magnetic flux reversals. It was a modified form of frequency modulation, so we of course call it MFM. Then, by dividing time into smaller quanta, and using the precise timing of flux reversals to convey additional information, 50% more data could be stored using the same total number of flux reversals. We called that run-length limited coding, or RLL.

Leo: Oh, I remember those days, yeah.

Steve: And that was probably the last time that it might have even been theoretically possible to recover any data that had previously been overwritten. Everything since then has been hysteria.

Leo: I love your line in here. Go ahead, say it. It's a miracle that it's possible to read back even what was most recently written.

Steve: Yes.

Leo: That they work at all is a miracle.

Steve: That's right. When discussing solid-state memory previously, I was careful to mention that once the electrons had been removed from a storage cell, there was no remnant of the data which had previously been recorded. No footprint remains. That's not entirely true for magnetic storage, and that was the original source of the hysteria.

To some degree, magnetism is additive because residual magnetism arises from the alignment of individual magnetic domains, and it's their vector sum that determines the

resulting overall magnetic field vector. Thus the greater the number of domains having the same alignment, the stronger the resulting field. So when a new magnetic field is imposed upon a ferrous substance which was previously magnetized, the resulting magnetic field will be a composite which is influenced by the field that was there previously.

If the previous field was aligned with the newly imposed field, the result will be a somewhat stronger field. But if the previous field had the opposite alignment, although the new field will dominate, the result may be ever so slightly weaker as a result of the memory of the previous opposite field still being present. So in theory, if we assume that a new magnetic field pattern was uniformly written, tiny variations occurring when that magnetic field is later read back might be due to the data that was previously present. In other words, something remains, however tiny.

That is the source of the concern over the need to overwrite and overwrite and overwrite a hard drive in order to adequately push it back, push what was there back into history, you know, the original data; essentially burying it so that it is beyond any recovery. None of that has been necessary for the past 20 years.

This was only theoretically possible when the encoding between the user's data and the drive's flux reversal patterns were trivial, as they were back in the early MFM and RLL days. They have not been trivial for the past 20 years. Today, and here's your line, Leo, it's truly, and I mean this, a miracle that it is possible to read back even what was most recently written. It's just astonishing how dense these hard drives have become. And many drives have become utterly dependent upon their built-in error correction to accomplish even that. Today's magnetic flux reversal patterns which are recorded onto the disc's surface bear only a distant relationship to the data that the user asked to be recorded. Where it was once true that bits of user data created flux reversal events on the drive, today large blocks of user data are massaged and manipulated. There's something called "whitening" is done to them, you know, they're bleached.

Leo: It's kind of amazing these things work at all.

Steve: It is insane, Leo, how far this technology has gone.

Leo: It's because of the density of the bits; right? I mean, we're just storing...

Steve: Yes. It's because of the density and because it is so difficult to get any more of them in the same amount of space. Yet they just, you know, management keeps demanding it, so the engineers go, well, I guess we could run it through a pasta maker in order to...

Leo: No, let's whiten it.

Steve: I don't know, shred it and then recombine it or something.

Leo: I've got to say, though, a lot of credit to technology. I mean, you and I both, I'm sure back in 2000, thought there's no way we'll still be using spinning drives in 2023.

Steve: People had been predicting the demise of the spinning hard drive for a decade.

Leo: Since the '90s, yeah.

Steve: Yes. And we're still here, baby.

Leo: They keep upping the density. I mean, you can get 20TB drives now.

Steve: It's insane.

Leo: It's unbelievable.

Steve: Okay. So these large blocks of user data are massaged and manipulated, and then the result is what's written back down. And then somehow this process is reversed. And after it's reversed, it's unmassaged and dewhitened, and the contrast is turned back up. And then error correction is used in order to give you back what you asked to have written in the first place. That's what's actually going on. So given all of these facts, here is my own plan for GRC's Secure Spinning Disc Data Erasure Program, which will closely follow after SpinRite 7.

Actually, I'm going to use pretty much all of SpinRite 7 in order to create the secure wiping utility. When GRC's program is asked to securely erase a spinning hard drive, it will use an extremely high quality random data generator to fill the entire drive with noise. Pure noise. Then it will go back to the beginning and do it again. And that's it. If its user wishes to follow that with a wipe to all ones or all zeroes for cosmetic purposes so that the drive appears to be empty rather than full of noise, they may elect to do so. But that won't be needed for security.

The theory is that it's always possible to read what was most recently written. Right? I mean, that's what a drive does. So in this case that'll be the noise that was written by the final second pass over the drive. Then also, by the sheerest, most distant theory of data recovery that really no longer has any support, it might theoretically be possible, and you may need to consult the aliens for this, to get something back from that first noise writing pass. But all that anyone would then ever get back is noise, the noise that was first written, absolutely unpredictable noise with no rhyme nor reason. And THAT noise will have absolutely completely obscured any data that was underneath it.

Leo: Yeah. I've been saying this for years. You don't need these 18 writes and overwrites and all that stuff.

Steve: No. That's - yes.

Leo: That's long gone.

Steve: Given today's complexity of the mapping between the data the user writes and the pattern of flex reversals that result, there is no possibility of recovery of any original user data following two passes of writing pure noise on top of that data. And thanks to

SpinRite's speed improvements, this two-pass noise wipe will be able to be done at probably around .25TB per hour. So, you know, because this utility will be using all of SpinRite 7's technology. So it will be feasible to wipe multi-terabyte drives in a reasonable amount of time and absolutely know that the data is gone.

Now, spinning hard drives do still also have the problem of inaccessible regions. When defects are detected after the drive is in the user's hands, those defective regions which contain some user data at the time are spared out, taken out of service, and are no longer accessible. Although the amount of user data is generally very small, if such spinning drives offer any form of "secure erase" feature, and many do now, GRC's utility will follow its two passes of noise writing with the drive's own secure erase command, which is designed to eradicate data even from those spared out sectors in order to get to all the final little bits.

And one final note before I close this answer: There are also drives that have built-in AES encryption hardware that's always active. When the drive comes from the factory it will have an initial default key already established. So it looks like just any regular drive. But this means that any user data that is ever written to such a drive will have first passed through the AES cipher using the drive's built-in key, and that everything that's ever written to the physical drive media will be encrypted on the drive, whether it's spinning or it's solid-state. Both technologies now have this available.

And that means that securely eradicating all recoverable traces of the user's data will be as simple as replacing and overwriting that one key with a new randomly generated key. At that point nothing that was ever previously written to the drive will ever be recoverable. And again, GRC's utility will know if the drive offers that feature, and offer it. And, you know, if you wanted to, belt and suspenders and, I don't know, a parachute, you could still do something that took longer, but there's really no point in doing that.

So to finish answering Atli's question: Due to the physics of electrostatic storage, that is, SSDs, there's no need to ever perform more than a single erasure pass over any current solid state storage if you did not trust the drive's secure erase command. Then, if possible, follow that with a secure erase command if one's available. The drive's manufacturer will typically have a utility that knows how to do that with their own drives. And due to the physics of electromagnetic storage, it might theoretically be useful to perform a second overwriting pass, and if possible I would do that with random noise.

That's what GRC will do. And again, follow that with a secure erasure command if one is available. And, finally, if you're lucky enough to have a drive with built-in encryption, just instruct it to forever forget its old key and obtain another one, and the entire drive will then be scrambled with nonsense that nobody will ever be able to decode. So now everyone knows everything I know about securely erasing data from today's mass storage devices.

Leo: That's so huge because you've debunked, I mean, I've tried to tell people this for years, but you've debunked years of misinformation. Please still run defrag disk optimization on their SSDs and things like that. I mean, we carry old habits with us, I guess, is the problem.

Steve: Yeah.

Leo: And thankfully people listen to you, so good job. Good job.

Steve: Okay. So I think I have two more pieces, and then we'll get to our main topic.

Ethan Stone, he said: "Hi, Steve. Apropos of yesterday's show, here's a link to the California Attorney General's 7/31 press release regarding a new investigation of 'connected vehicles.'" Okay. So Ethan is referring back to last week's coverage of the Mozilla Foundation's revelations which they titled "It's Official: Cars Are the Worst Product Category We Have Ever Reviewed for Privacy." And Leo, that's another one of our main topics that you missed last week.

Leo: I read that article. It was incredible. Unbelievable.

Steve: Okay. So you know that they're even saying that they can report on the sex life of the occupants of the car.

Leo: Yeah, unbelievable, yeah.

Steve: So, yeah. So anyway, for what it's worth, the California Privacy Protection Agency, that's the CPPA, enforcement division, they wrote: "...today announced a review of data privacy practices by connected vehicle manufacturers and related connected vehicle technologies. These vehicles are embedded with several features including location sharing, web-based entertainment, smartphone integration, and cameras. Data privacy considerations are critical because these vehicles often automatically gather consumers' locations, personal preferences, and details about their daily lives."

CPPA's Executive Director, Ashkan Soltani, said: "Modern vehicles are effectively connected computers on wheels. They're able to collect a wealth of information via built-in apps, sensors, and cameras, which can monitor people both inside and near the vehicle. Our Enforcement Division is making inquiries into the connected vehicle space to understand how these companies are complying with California law when they collect and use consumers' data."

I think we can presume that, since then, Soltani or his people been informed of Mozilla's connected vehicle privacy research. That ought to make the CPPA's hair curl, or more likely catch on fire. So, yeah, Leo, I'm glad you saw that. And, you know, yikes.

Leo: Mozilla's been doing great work, I have to say, along these lines, these privacy lines. That's really good, yeah.

Steve: Yeah. So I have two pieces of miscellany. The first of two pieces - oh, Leo, this is so cool. Go to grc.sc/2038, obviously 2038, grc.sc/2038. Okay. This is courtesy of our listener Christopher Loessl who sent the link to a super-cool "end of the world as we've known it" Unix 32-bit countdown clock.

Leo: Oh, that's cool.

Steve: It is beautifully done. Okay. I just really admire this. It's a simple, very clean JavaScript app that runs in the local browser. Since 32-bit Unix time - similar to a 32-bit IP address - can be broken down into four eight-bit bytes, this clock has four hands, with the position of each hand representing the current value of its associated byte of 32-bit

Unix time. It's brilliant. Since a byte can have 256 hex values ranging from 00 to FF, the clock's face has 256 index marks numbered from 00 to FF. So, collectively, the four hands are able to represent the current 32-bit Unix time, since each hand is a byte of that time. Unfortunately, Unix time is a 32-bit SIGNED value. Whoops. So we only get half of the normal unsigned 32-bit value range.

Leo: Did they do that so they could have negative time? Why did they...

Steve: I just don't think anyone paid attention.

Leo: They never thought that people would still be using it in 2038, that's obvious.

Steve: I don't think they even thought about it. I think they just said, okay, 32 bits, that's big.

Leo: That's huge.

Steve: And we'll make it an integer. Unfortunately, unless you said "unsigned integer," it's a signed integer.

Leo: Right. Remember, though, this is 1970 or '69, when an eight-bit computer was like, wow. So a 32-bit value must have seemed infinite.

Steve: Well, and that's why, of course, the Internet Protocol, IP, only got 32 bits. Because they were like, come on, 4.3 billion computers? We only have two. We have two right now.

Leo: Exactly.

Steve: One on each end of this packet link.

Leo: Yeah.

Steve: So if Unix time had originally been defined as an unsigned integer, we would have 68 additional years added to the 15 that we still have left.

Leo: We're almost 90% of the way to the end of time.

Steve: Correct.

Leo: Wow.

Steve: Before the time would wrap around. So anyway, so what's going to happen is when the red hand on this clock gets down to pointing straight down at 80, Unix time goes negative because the high...

Leo: [Laughing]

Steve: It's not good, Leo. It's not good. Nope. What's going to happen is the simulation that we've all been enjoying so much will crunch in on itself.

Leo: Oh, no.

Steve: And that's otherwise known as "game over."

Leo: Okay. Yikes.

Steve: Yeah. But this is a wonderful clock. Grc.sc/2038. I commend it highly. It's just a beautiful...

Leo: So the green hand is seconds.

Steve: Well, yes. And you could see above it the lowest byte of Unix time in hex right above the clock dials...

Leo: Is the last few digits, yeah, yeah, yeah.

Steve: Yes. And so that's the one you're able to see moving.

Leo: Right.

Steve: And then every time the green one goes around once, the black one, which is the second most significant byte...

Leo: Yeah, goes up one.

Steve: It clicks up by one.

Leo: Yeah. And then there's the orange, which is whatever the next four significant bytes. But there ain't no red. Actually, the orange is bytes two and three, or three and four.

Steve: Is the third most significant.

Leo: Yeah, yeah. I'm sorry. And then the red is the first two. Yeah, yeah, yeah.

Steve: Right. And so as soon as red gets halfway around, unfortunately halfway because it's a signed value...

Leo: Close.

Steve: ...it hits eight zero. That means the high bit is on in the 32-bit signed value, and that's a negative event.

Leo: Wow. Wow.

Steve: And so, again, that's - at that point it's over.

Leo: We will solve this. We solved the 1999 problem. We'll solve this one.

Steve: I've got sun coming in off of - look at that.

Leo: That's apocalyptic.

Steve: Not only in the U.K. do we have sun.

Leo: Yes.

Steve: Maybe. Okay. So speaking of "game over," yesterday Elon Musk complained that, to exactly no one's surprise, since taking the helm at Twitter, advertising revenue has dropped 60%. Apparently, everyone except Elon knows exactly why this has occurred. Major brands have alternatives, and they don't wish to have their ads appearing next to controversial and often obnoxious tweets posted by the raving lunatics who Elon has deliberately allowed back onto his platform. You know, Elon is all about the free market. Well, this is the free market in action, Elon. Objectively, he is managing to destroy Twitter, now renamed "X," which as an aside is about the worst branding imaginable.

Leo: Oh, yeah.

Steve: You know a brand is bad when everyone is continually forced to refer to it as "X, formerly known as Twitter." In other words, the letter "X" is not sufficiently distinctive to establish a brand.

Leo: Well, go ahead, do a Google search for "X" and tell me what you find. I mean, it's nuts. Whoever thought this was a good idea? By the way, he's named all his children with X, as well. So he just loves this letter for some reason.

Steve: Wow. Wow. He's been casting around for more revenue. First, as we know, he decided that he'd make people pay for the use of the blue verified seal. I had no problem whatsoever with losing my years-long verified designation since my entire use of Twitter is to connect with our listeners here. And no one here needs me to be any more verified than I already am.

But then a month ago, in mid-August, Elon decided that hadn't worked. So he decided to also take away the use of the TweetDeck UI for all non-paying users. That was my sole interface to Twitter, so losing it was painful. I refused to pay this extortionist any money. For several weeks I attempted to use the regular Twitter web browser user interface. I hated it in so many ways. I tried to get accustomed to it, but I finally gave up.

So I decided two weeks ago that because I was using Twitter for the podcast, to broadcast each week's show note link, and now also the Picture of the Week, and to communicate with our Twitter-using listeners, I would pay this man $8 per month - $2 per Security Now! Episode - in return for having access to a user interface that made assembling these show notes, communicating with our listeners, and announcing each week's show note link far less painful. So for the past two weeks I've been back to using TweetDeck, and joy has returned to the valley.

But still not satisfied with his revenue stream, and apparently unable to combat the storm of automated bot postings - which, by the way, the previous management with a much larger staff was managing to hold at bay - yesterday Elon announced that everyone, yes, everyone, all users of Twitter, would soon need to pay for access to his service. Elon has now stated that he will be moving Twitter - X - behind a paywall. And I suspect that if he really goes through with it, this will finally spell the end of Twitter, at least the Twitter that we've known. Now, many of our listeners have communicated that they don't use Twitter and never will. Period. I mean, like three quarters of the listeners of this podcast I'm estimating.

Leo: Oh, yeah. It's just tech journalists and crazy people.

Steve: Well, I have 64,000 followers who are our listeners.

Leo: I have a half a million followers. I don't know if they're listeners, though.

Steve: Yeah, well, I know that mine, I'm saying that all of this communication that I share is via Twitter.

Leo: Right.

Steve: So, for example, just yesterday I received a DM from a listener who had just joined Twitter to send me that DM. He said, "I created an account so I could send this message to you."

Leo: See? You see the damage you're doing, Steve? Get off of that place.

Steve: So, yes. This is not acceptable, and it's apparently about to become even less acceptable. So I wanted to let all of our listeners know that because I value it so highly, I am arranging for an alternative conduit for our communication.

Leo: Oh, good. Thank you.

Steve: Now, this sentiment of "I don't use Twitter and I never will" is strong among the majority of our listeners. I understand that.

Leo: Well, especially as time goes by.

Steve: Yes.

Leo: It's gotten worse and worse and worse.

Steve: Yes. I also understand, though, that it's actually less specific than Twitter. The real underlying sentiment is "I don't use social networking, and I never will."

Leo: Well, that's true, yeah, yeah.

Steve: So Twitter was just the specific instance of social networking that was most prominent at the time.

Leo: Right. By the way, you may not remember it, but some of our listeners do. I worked for years to get you on Twitter. You were very resistant.

Steve: Yeah.

Leo: But this is a decade ago. But you did not want to get on there. You were one of them.

Steve: I was one of our listeners.

Leo: Yeah.

Steve: And many of them have outlasted me in this. So the point is, I don't believe that moving to Mastodon, Discord, Bluesky, Threads, LinkedIn, Tumblr, Instagram, or any other...

Leo: Pebble, got to have Pebble.

Steve: I don't even know what that one is.

Leo: Pebble is T2. They rebranded to Pebble.

Steve: So I don't think that's the solution, you know, for this particular community, for our listeners. Even the fact that there are so many disjoint and competing platforms now demonstrates the problem.

Leo: Right.

Steve: The answer, I believe, is good old-fashioned email.

Leo: Oh. Okay.

Steve: We all have it. We all already have it. It works, and it does exactly what we all want. I've previously mentioned that it has been my intention to set up good old tried-and-true emailing lists. I'll create one for Security Now! and invite this podcast's listeners to join it. Each Tuesday before the podcast I'll send a link to the week's show notes, a brief summary of the podcast, and the Picture of the Week, much as I've been doing on Twitter. And anyone will be free to unsubscribe, resubscribe, subscribe multiple accounts, do whatever they want. Since I very much want the communication to be both ways, I also need a means for receiving email from our listeners while avoiding the inevitable spam. And I believe I have an innovation that will make that practical, too.

So I just wanted to make certain that everyone knew, in advance of the apparent pending end of Twitter as a practical social media communications platform for us, that I would be arranging for its replacement. And I already know that this will be good news for the majority of our listeners who are not, who never have been, and never will be participating on Twitter or any other social media platform. So.

Leo: Good. I think email is as universal as you can get.

Steve: Yup, it is. And I think I have an extremely cool way of avoiding incoming spam which all of our listeners are going to get a huge kick out of, even get a kick out of using, just to try it. So I will have news of that here at some point.

Leo: Good.

Steve: And in the meantime, Leo, our last sponsor, and then I'm going to share something I think everyone's going to find really interesting.

Leo: Are you going to completely abandon Twitter? Or what's the deal? Are you going to stick around until he starts charging?

Steve: The only thing that email replacement doesn't create is a community. And there is, although I don't participate in it, there are a lot of people, you know, who are tweeting @SGgrc and seeing each other's tweets, and so it kind of creates a community there. But if I, I mean, literally, the only thing I use it for is to talk to our listeners and to receive DMs and to look at my feed. So, and, you know, I've been complaining. It's like, I can't also have a Mastodon presence because I just can't hold - I'm already, you know, I'm so spread thin. I've got GRC's NNTP newsgroups. I've got GRC's forums. I would love just to move everybody to email. So that's my plan.

Leo: Yeah. Well, if you ever change your mind, TWiT.social is always open for you, and all your friends are there.

Steve: Well, I'm already @SGgrc over at Infosec.social. I created, I grabbed that because that's what...

Leo: Yeah, just to have it, yeah, yeah, yeah.

Steve: Just in case it ever happens. But, I mean, there are so many listeners, Leo, who say "I don't do Twitter."

Leo: Right. Yeah, well, those people probably wouldn't do Mastodon, either. So that makes sense.

Steve: They won't go anywhere; right.

Leo: Yeah, yeah. And now let's get back to Steve and our subject of the day.

Steve: When Hashes Collide.

Leo: Colliding hashes.

Steve: So obviously, security and privacy news is important and interesting. But our listeners always tell me that they really enjoy our occasional deep dives into theory and wish we would do more of those. That's kept me on the lookout for such opportunities; and this week, in addition to our previous discussion of secure data erasure, I found another one thanks to a question posed by one of our listeners.

Ray V. tweeted: "In addition to being an avid listener of Security Now!, I'm also a System Administrator for a large public library system in Ohio. Libraries often struggle with data, being especially sensitive around data related to patrons and patron behavior in terms of borrowing, library program attendance, reference questions, et cetera. The common practice is for libraries to aggregate and then promptly destroy this data within a short time frame, which is typically one month.

"However, administrators and local government officials, who are often instrumental in allocating library funding and guiding operational strategies, frequently ask questions on a larger timescale than one month to validate the library's significance and its operational

strategies. Disaggregation of this data to answer these types of questions is very difficult and arguably impossible. This puts people like me, and many others like me, in a tough spot in terms of storing and later using sensitive data to provide the answers to these questions of pretty serious consequence, like what should we spend money on, or why should we continue to exist.

"I'm sure you're aware, but there are many interesting historical reasons for this sensitivity, and organizations like the American Library Association (ALA) and other international library associations have even codified the protection of patron privacy into their codes of ethics. For example, the ALA's Code of Ethics states: 'We protect each library user's right to privacy and confidentiality with respect to information sought or received and resources consulted, borrowed, acquired or transmitted.' While I deeply respect and admire this stance, it doesn't provide a solution for those of us wrestling with the aforementioned existential questions. In this context, I'd be immensely grateful if you could share your thoughts on the technique of 'pseudonymization.'" And he gives a link to Wikipedia's definition of "pseudonymization," he says, "for personally identifiable information (PII) data."

So Ray's note to me links, as I said, to Wikipedia's article on pseudonymization which begins by explaining: "Pseudonymization is a data management and de-identification procedure by which personally identifiable information fields within a data record are replaced by one or more artificial identifiers, or pseudonyms. A single pseudonym for each replaced field or collection of replaced fields makes the data record less identifiable while remaining suitable for data analysis and data processing."

So, okay. Clearly what Ray is asking for is some way to process the data that is collected, and in such a way that it can be used for statistical analysis and essentially who's doing what, but while also deliberately obscuring the who. So the first thing that springs to mind is that this is a perfect application for cryptographic hashing. As we know, a hash is a one-way information-lossy transform that's able to take input of any length and turn it into an output of fixed length. Today's most popular cryptographic hash we've referred to a number of times on this show already, SHA-256. It converts anything that's input, like someone's password, into a fixed length 256-bit binary blob. And unlike a cryptographic cipher, which was designed to be reversible to allow for decryption, hashes are deliberately designed to not be reversible. They are a one-way scrambling of all of the bits which are input to the hash. Which is, of course, by the way, why we call it a "hash"; right? It makes a hash out of whatever we put in.

Okay. However, the output of today's super-strong hashes can be overly long for some applications. We don't always need to use all of the bits produced by today's hashes. And this is what I want to explore today: the applications for, and the implications of, the deliberate use of fewer bits, far fewer bits, of a cryptographic hash.

Turning to Ray's use, it's not necessary for him to be able to uniquely identify every individual known human and unknown alien throughout the universe. Ray only needs to aggregate and disambiguate the human patrons to his local library system, which is a far smaller population. So I wanted to use this question and Ray's particular application to talk about something we have never discussed before, which is the use of only a portion of an entire cryptographic hash.

Those listeners who've been with us from the start may recall that part of the definition of a cryptographically strong hash is that the result is some number of bits where, for any given input block, every single bit of the output will have a 50/50 chance of being a zero or a one. In other words, there is no bias to any of the individual bits. And if any one bit of the input is changed, on average, some set, some collection of half of the output bits will invert. So that's just very cool math. You know, think about that. All of the bits are exactly equally relevant. Every bit has exactly a 50/50 chance of being a zero

or a one. And when a single bit of the hash's input is changed, on average some half of the hash's output bits will be changed.

So one way to think of this is that when something is put into the hash, what comes out is a large pseudorandom number. During repeated uses, as long as the same thing is put in, the same thing will come out. And here's the important thing to understand: Every single bit of that pseudorandom number output is independent of all the other bits. That bit, any one of the bits doesn't care about any of its neighbors. And given any input, it will have an equal chance of being a zero or a one.

What that tells us is that there is no need whatsoever to use it all. The only reason modern cryptographic hashes have become longer is to increase the universe of possible outputs and to keep the hash's internal algorithm being a black box which remains highly resistant to cryptographic analysis. We never want powerful computers to be able to unscramble the eggs. But 256 bits is not what Ray needs to uniquely identify his library's patrons. 256 bits gives us 115*10^75 individual combinations. That's 115 followed by 75 zeroes. Ray's application is far better served by only using enough bits to reduce the probability of a hash collision to an acceptable level. A hash collision occurs when two different inputs result in the same output, given the number of bits.

Remember that, counterintuitive though it might be, all of the bits of a good hash's output are equally valuable. So say that Ray just took the lowest eight-bit byte from the hash of his patrons' aggregated personal information. Perhaps he just uses their email address, for example. Since an eight-bit byte can have 256 possible combinations, this would have the effect of dividing the entire population of the library's patrons evenly into 256 groups. No matter what personal data is put in, a number from zero to 255 will come out. And they're evenly divided because the hash is good. If all the bits have an equal chance of being a one or a zero, then given a population, you pour them through the hash, even if you only use eight bits from the output, they're going to be evenly spread among those essentially 256 bins. They're going to be grouped by those 256 possibilities.

Now, that one byte is colliding because we've ignored the other 31 bytes on purpose in this example; right? So even though two different email addresses might wind up hashing to the same least significant byte, they're absolutely different everywhere else. But we're deliberately choosing to ignore all those for the purpose of this exploration.

So presuming that the library has thousands of local patrons, there will obviously be collisions since all of those thousands will get grouped into just those 256 bins. The downside of using so few bits is that it results in a loss of resolution. It's no longer possible to track specific individuals since multiple individuals will definitely be sharing the same eight-bit byte identifier. For some applications, this may be an advantage since it would tend to enhance the privacy of an individual. With so many patrons grouped into so few bins, thus guaranteed to be sharing bins, no claim could be made by looking at any one bin about the behavior of any one individual due to the presence of these deliberately engineered hash collisions.

So say that eight bits is too few. I think it probably is. Ray needs more resolution because he wants to be able to perform some useful data analysis. He wants to be able to collect meaningful statistical data by keeping his post-anonymized, or rather pseudonymized, patrons more separated, but not provably separate. He wants to leave a modicum of deliberate uncertainty, introduced by hashing collisions, to keep it from being possible to prove that any one patron had some specific behavior in the past.

So now the question is, how many patrons does Ray's library have? How many bits from the hash should he use? How separated should his patrons be? And what's an acceptably low level of collision to create behavior deniability? And this, of course, brings us to real-world statistical hash collisions and the famous "birthday paradox."

The birthday paradox refers to the surprising fact that assuming 365 days in a year, we'll ignore leap years for this purpose, and equally probable birthday occurrences throughout the year, which is to say an even distribution of birthdays, the number of people required in any group for there to be a 50% chance that any two of them will share the same birthday is, surprisingly, only 23. Given 365 days in a year, the number being as low as 23, for there to be a 50% chance of a birthday collision often surprises non-statisticians.

Leo: Yeah, it's not really a paradox, it's really just an example of innumeracy among people, I guess.

Steve: Right. Right, example.

Leo: It's not that it's any given date. It's any two will share the same date. And there's a big difference.

Steve: Yes, yes. And in fact it's exactly that. Every pair of people in the group is compared, and a group of 23 people contains 253 pairs of people. So there is a great deal more opportunity for birthday collision than might have intuitively been assumed.

Leo: Exactly. Right, right.

Steve: So we have two different goals here with a different property. Ray wants sufficient resolution among his library's patrons in order to do data processing. Meaning that he doesn't want there to be a very high likelihood of a birthday paradox collision among any of his patrons. So to make that determination we rely upon the birthday paradox math. And in this case, the more bits of the hash we use, the lower the chance of any collision, and the more distinct the categorization.

But as Ray explained in his note, libraries also have the goal of protecting the privacy of their patrons against government or law enforcement queries into the past behavior of any individual patron. Since privacy protection is enhanced by using fewer bits from the hash, these two goals are conflicting, and a compromise must be reached. This is made worse by the fact that the birthday paradox does not apply in this second instance of privacy and plausible deniability. Here, the question is not what is the chance that any two patrons will share the same hash bin. The question is what is the chance that someone else shares the same bin as the targeted individual. And there, unfortunately, the chances are far lower for any given number of hash bits.

So let's put some numbers to this. I found a nice webpage containing a calculator for these birthday paradox questions. It allows the input of the number of items, which would be the number of patrons, and the number of buckets or bins. I've also made it this week's GRC shortcut of the week, so you can get to it, grc.sc/840. I'm sorry, I did say 840. It's Episode 940. Oops. Typo. So the link is in the show notes, and I will fix this to be a shortcut at 940 as soon as we're done recording this. So GRC.sc/940. And in the meantime you can also get to it at 840 because I fumble-fingered on my typo, on the shortcut.

Okay. So going to this page, if we test it first with the classic birthday paradox, by putting 23 items into 365 buckets, that web page tells us that the chance of a collision between any two is 50.73%, just over 50%, so that looks right. If we take one person away from the group, the page shows that the chance of collision drops to 47.57%. So

this confirms that 23 people is the minimum number needed to reach a 50% chance of collision.

I have no idea how large Ray's patron population may be. So let's use 5,000 patrons as a working number. That sounds like it's probably in the ballpark. So how many hash bits would we need to use for there to be a 50% chance of a collision among that population of 5,000 patrons? When you think about it, that would provide very adequate separation among his patrons, for there to only be a 50% chance that any two of them would collide.

So poking the numbers into that page, and I did some experimenting because you have to know what powers of two, how many bins various powers of two are, it turns out that using just 24 bits, three bytes, of the hash's value, instead of all 256 bits, will result in a 52.5% chance of any two patrons from a population of 5,000 having the same 24-bit identifier when their personally identifiable information is hashed. Now, it turns out three bytes is also sort of cool because those 24 bits, instead of being divided into three bytes of eight bits, can be divided into four groups of six bits. That's also 24; right? And six bits can be mapped into a 64-character alphabet of printable characters. This means that each of the library's 5,000, more or less, patrons can have their personally identifiable information hashed and turned into a four-character printable token which probably, but not certainly, uniquely identifies them within that 5,000 patron population.

This protects their privacy, allows their pseudonymous history to be collected over time into bins identified only by those tokens, with only a 52.5% chance that any two of the 5,000 patrons will be assigned the same four-character token, thus having their individual histories mixed. This is low enough to provide sufficient inter-patron separation, and also high enough to introduce sufficient uncertainty into the data collection system to prevent the token-tagged history from being legally evidentiary. That is, you cannot prove that all of any person's tokens are only from their borrowing habits because the system is deliberately fuzzy just enough to preserve the value of statistics, but to keep it from being useful as legal evidence.

So the goal of this exercise was to highlight another less-appreciated and extremely cool aspect of cryptographically strong hashes, which is that all bits are created equal, that there can be some very useful real-world applications for deliberately using some many fewer number of bits, and that which bits are used makes absolutely no difference.

Leo: Well. Very nice. And it's cool that somebody put this hash collision calculator online.

Steve: Yeah. Isn't that neat? Yeah.

Leo: Yeah. It's very fun.

Steve: And the thing I did not mention is that I was using powers of two, right, because I'm just using some number of bits. The problem is, when you get up like to 24 bits to be the number of bins, and then if you want to add one more bit, you go from 24 to 25, well, that, you know, that's a big jump. That's a big jump in value. So the other way to do this, what I'm trying to say is you might want a number between what binary bits can provide you. And you can get that by taking the entire 256-bit value and doing a long division by the number that you want.

So say that you didn't - you'd want it somewhere between, just to make it easy, 128 and 256. That's seven bits or eight bits; right? Say that you wanted for some reason exactly 150, which falls between 128 and 256. Well, you can get 150 equally distributed if you take the entire large hash and divide it by 150. You basically take it modulus 150. And there you'll get a result between zero and 149 equally distributed values. So anyway, just a little addition to that. But anyway, a very cool application for a cryptographically strong hash where you don't want it to be, you know, to uniquely identify one from a population.

Leo: Right.

Steve: The way we do if we're signing code, for example.

Leo: Right. People are saying, "We were told there'd be no math in this show." Well, you were told wrong. I'm sorry. That's exactly what this show is all about. Very interesting stuff. You know, if some of that went over your head, the show notes are a great place to go. Steve's got them at GRC.com. And you can actually read what Steve just said. Transcripts will also help. And then work it out on your own because it's worth doing, and I think it'll become clear if you do. But sometimes it's hard just to hear it and understand it.