Security Now! #940 - 09-19-23

When Hashes Collide

This week on Security Now!

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.

Last week's "LastMess"

Last week's topic was important enough that I wanted to make sure Leo was not left out of the loop on this: Brian Kerbs reported the very strong and still mounting evidence that selective decryption of the LastPass vaults that were stolen **has been occurring** to vaults that are expected to contain protected cryptocurrency keys. And the owners of those LastPass vaults, many affiliated with the cryptocurrency industry, have subsequently had their funds stolen. I won't repeat more of last week's main topic other than to reiterate that the evidence is quite compelling and that several independent analysts who have reviewed the evidence have all concurred. Decrypting vaults – while possible if a vault's iteration count was never increased and if the user's password was not long and strong – is still expensive in time and resources. So they have no interest in logging into Facebook as aunt Mable. The bad guys are, as always, after one thing: Money. So they're going to target LastPass early adopters with low iteration counts who are likely to be trading in crypto, and who may have had their cryptocurrency keys stored in their LastPass vault at the time of the theft.

Our action item from this is: If you **did** have a cryptocurrency key stored in your LastPass vault it's probably worth creating a new wallet and transferring those funds in order to empty the wallet that might someday be cracked.

Who Blinked?

I also wanted to note that we discussed the significant backing down of the UK from their absolutist stance on encryption and CSAM monitoring. The battle is effectively over. Their updated final legislation 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."* This allows politicians to claim that they will have tough new legislation in place while also giving every one of the existing encrypted messaging providers an out by explaining that it is not technically feasible. Now we need to see how the EU may follow. But this dramatically slows the roll and breaks the anti-encryption momentum that appeared to be mounting.

Closing the Loop

Chris Smith / @4CSmith

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 HSM would 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.

Chris raises a good point. 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 customer's enterprise eMail. We now know that a crash dump of a system which contained the key **in RAM** created a snapshot of that RAM which later migrated into the hands of attackers thanks to a series of no fewer than five separate mistakes, each of which Microsoft now assures us have been remedied. The question raised was why, in this day and age where we have hardware security modules (HSM's) specifically to prevent this, Microsoft was not using those. I don't know. Perhaps Microsoft will address this. But I made the point that even GRC's code signing keys are locked up in HSM's and are thus completely inaccessible. Crash or no crash, they never exist in RAM and are thus never subject to theft.

Chris reminds us that cryptographic signing imposes significant overhead, and that the use of custom hardware for accelerating this process has historically been necessary. But the hardware for doing this – and at today's Internet scale – exists today. 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, whereas, to Chris' point, special-purpose hardware can always be faster. So to my point, for whatever reason, Microsoft was not using such hardware. It's difficult to discount that as anything less than sloppiness. And they did get well bitten.

Rich Carpenter / @_carpodiem_

Hey Steve. Thank you for all your contribution to the Security community, and delighted to hear you're extending your tenure to 4-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 tear down of what HSMs are, and how they operate without exposing the contained key material for the SecurityNow audience? Many thanks, Rich, in Sunny Devon Uk.

Rich's question followed nicely from Chris' question. The idea is simplicity itself. The particular key I use came from DigiCert already containing the code signing key I obtained from them. That was several years ago and I've since updated it several times. What's significant is that the

signing keys it stores are write-only. This little appliance – and this is true of HSM's on any scale – lack the ability to divulge their secrets. Once a secret has been written and stored in the device's non-volatile memory 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 SHA256 cryptographic hash of my code. Then, the resulting 32 bytes are sent into the HSM where they are encrypted using its internal private key and the encrypted result is returned. So the whole point of this is that all of 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 the entire process: The so-called "signature" that's then attached to my code is the matching public key which has been signed by a trust root certificate authority – in my case DigiCert. So, when someone downloads and wishes to verify the code's signature, the signature of its attached public code signing key is first checked to verify that the public key was signed by DigiCert. Then that public key is used to decrypt the signature that the HSM encrypted. Only if the public key and the HSM's private key are complimentary will the original hash of my code be restored. Then the just downloaded code is again hashed and the new hash and the decrypted hash are compared. They will only be equal if GRC's matching private key was used to encrypt the original code's hash, and if not one single bit of the code has been changed since it was signed – because that, too, would alter the new hash. It's a super slick system. It has a bunch of moving parts, but when everything lines up, it works!

Clint: (last name withheld to not embarrass this person)

Hey Steve, long time Security now follower my name is Clint, I wanted to bring to your attention that I just fell for a Verizon wireless fraud call. Since I wasn't aware of Verizon's Fraud policy, 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 password. I'm working on letting social security and other services I have know of potential activity that isn't 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 that's new going around.

This seems significant since Clint explains that he's also a long time Security Now! Podcast listener. The lesson here is "skepticism first" and never – for one moment – let down our guard. It's unfortunate because it's not the way humans are built. 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 take action. Clint now knows that his first reaction should have been to place his own call to Verizon. In fact he knew it then, but these scammers count on their targets getting caught up in the moment and reacting rather than taking the time to think things through. So I cannot say it often enough. It doesn't matter how bits of encryption you might have. The human factor remains the weakest link and the bad guys are incessantly clever. I dread the day I make such a mistake, and if I could type with my fingers crossed I would.

Stand_Bold / @stand_bold

Greetings from a dedicated listener from Bharat, India! 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: install the 7-zip file archiver. This adds a CRC SHA verification option to the Windows right-click menu. 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!

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 and I recall deliberately installing it many years ago. That's the approach I prefer. Following the rule that we tend to more often do the things that are quick and easily done, I think it makes sense for file checksumming to be as quick and simple as possible. So I've placed a link to the Windows shell extension solution I've been using for years. It hasn't been updated since 2009 bit it has support for both 32 and 64 bit Windows systems. Unfortunately, it does not support SHA256, so it cannot be used to compare checksums from VirusTotal: http://code.kliu.org/hashcheck/. Its author offers its source code so it could easily be enhanced. I also found two other similar open source solutions, on Github. All of the links are in today's notes: https://github.com/gurnec/HashCheck https://github.com/namazso/OpenHashTab

Chris Shearer / @cbshearer

Hello Steve! This week you mentioned running a separate browser for segregating extensions. Chrome and Edge have a 'profiles' feature where you can run the browser separately as a different identity with different extensions/sessions/cookies etc. I have used this to maintain sessions to multiple Azure, Gmail or internal app sessions simultaneously. This is great for separating dev, prod, test, personal, etc. they even get separate toolbar icons (in Windows) so you can launch right into the one you want. In edge/chrome you click your avatar or icon and there is 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. I believe FF has similar features but I don't use it as often. 999 and beyond+

I had completely overlooked profiles, so I'm glad Chris reminded me and that 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 added superfluous and unnecessary extras. And, sure enough, entering "about:profiles" into Firefox's URL bar took me to my "profiles" page where I saw that I currently only have one profile, but there was an option to create more. Nice! Thanks, Chris!

Atli Davidsson / @atlisd

Hi Steve, Long time listener and Spinrite owner. I am 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 is a lot of mixed information on how many times to overwrite the drive, so I picked 7 as some article suggested. What would be the sufficient times to overwrite the drive?

Some say 3, some say 1. I used 3 times for my SSD a few days back and now using 7 for a large spinning drive which will probably take days. I probably should have at lest swapped those numbers and used a larger value for the SSD? Any thoughts? Looking forward for the next few hundred Security Now episodes :-)

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 v7 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. Since that future utility is not yet available, here's what needs to be done in the meantime:

The first thing to appreciate is that there are similarities and differences between the wiping needs for spinning versus solid state mass 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. 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. 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 SSD 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 10,000. Still, 10,000 writes to every bit is a lot of writing.

The problem is that if only 5 gigabytes of 128GB was written over and over, the remaining 123 GB would remain brand new and unused while the first 5GB that was being read and written over and over would be fatiguing due to the stress of being rewritten over and over. So the process known as "wear leveling" was quickly introduced. Wear leveling introduces a "mapping layer" between the user and the physical media. This is known as the FTL, for Flash Translation Layer. So now, say 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 other lesser-used physical region of memory and the FTL (the translation layer) is updated to point the LOGICAL address of that memory region to the PHYSICAL location where the updated memory actually resides. This beautifully spreads the fatiguing 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 one location, then rewritten to another. And the FTL was updated so that the next read of that 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 0's, or whatever, to assure that its old data has been permanently eliminated.

Now, 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 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.

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 data. 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 that device will be free to discard or give away.

The story with hard drives is somewhat murkier due to their long history and some bad advice that has outgrown its usefulness, coupled with some panic 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. 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, known as 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. And that was probably the last time that it might have been even theoretically possible to recover any data that had previously been overwritten. Everything since then has been hysteria.

When discussing solid state memory previously, I was careful to mention that once the electrons had been removed from a storage cell, there is no remnant of the data that had previously been recorded there. No footprint remains. That's what'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. And 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. 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 present previously.

That is the source of the concern over the need to overwrite and overwrite and overwrite a hard drive in order to adequately "push back into history" the drive's original data; essentially burying it so that it is beyond any recovery. None of that has been necessary for the past twenty 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 twenty years. Today, it's truly a miracle that it's possible to read back even what was most recently written; 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 bare 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 as the drive's data channel decides exactly what to write so that the user's data will have the greatest probability of later being read back; and perhaps then only after errors in that process have been corrected.

So given all of these facts, here's my own plan for GRC's secure spinning disc data erasure program which will closely follow after SpinRite 7: 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 0's 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. That's what drives do. 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 to get something back from that first noise writing pass. But all that anyone would get back is noise. Absolutely unpredictable noise with no rhyme or reason. And THAT noise will have completely obscured any data that was underneath it. Given today's complexity of the mapping between the data the user writes and the pattern of flux reversals that result, there's 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 can be done at around ¼ terabyte per hour. So it will be feasible to wipe multi-terabyte drives.

Spinning hard drives do also have the problem of inaccessible regions. When defects are detected, those defective regions – which contained 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, GRC's utility will follow its two-passes of noise writing with the drive's own secure erase which is designed to eradicate data even from spared out sectors.

And one final note before I close this topic: 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. 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 drive will be encrypted on the drive – whether spinning or solid state. And that means that securely eradicating all recoverable traces of the user's data is as simple as replacing and overwriting that key with a new randomly generated key. At that point nothing that was ever previously written to the drive will ever be recoverable.

So to finish answering Atli's question: Due to the physics of electrostatic storage, there's no need to ever perform more than a single erasure pass over any current solid state storage. Then, if possible, follow that with a secure erasure command if one is available. The drive's manufacturer will 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 two overwriting passes and, if possible, do so with random noise data. And again, follow that with a secure erasure command if one is available. And if you're lucky enough to have a drive with built-in encryption, just instruct it to forget its old key and obtain another.

Now everyone knows everything I know about securely erasing data from today's mass storage devices. And, once I get SpinRite 7 launched, I plan to produce another utility which uses most of SpinRite 7's core to tell its users about all of secure erasure features that any of their drives contain and work with them to securely erase their data from any of their drives.

Ethan Stone / @ethstone

Hi Steve. Apropos of yesterday's show, here's a link to the CA AG's 7/31 press release re a new investigation of "connected vehicles": https://cppa.ca.gov/announcements/

Ethan is referring 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."*
He pointed me to the CPPA, which is the California Privacy Protection Agency. This past July the CPPA announced that they were launching an inquiry into exactly this. The Agency wrote:

The California Privacy Protection Agency's (CPPA) Enforcement Division today announced a review of data privacy practices by connected vehicle (CV) manufacturers and related CV 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 will have been informed of Mozilla's connected vehicle privacy research – that ought to make the CPPA's hair curl – or more likely catch on fire.

Carl Marcus / @CarlMarcus2

Steve - I am happy you reported on Mozilla's coverage of car insecurity but sad that you did not do your usual techie dive. Specifically, how do the cars communicate with their servers? I have skimmed the report. I own a Subaru, and my understanding is that, since I did not buy the service that keeps the car always connected to Subaru (for example, for emergency help) there is no way for my personal data to reach their servers. Your comments?

I didn't do my usual techie dive because that wasn't the focus of that reporting. I wasn't attempting to deeply understand or offer any cure for those problems. Mozilla understood that there wasn't much that consumers could do since they needed to drive cars and there was no brand that was any better than the others. They are all truly terrible.

In answer to Carl's question about his Subaru's phoning home to Subaru Central, if his automobile has any sort of native connection to the Internet then it is doubtless using that connection to contact and report to Subaru. If all it has is built-in mapping that cannot be updated online. If there's no obvious Internet connectivity and podcast offerings and if he only, for example, has SiriusXM satellite radio, then his car would be considered to be disconnected. Subaru has something they call "Subaru Starlink" and it's definitely quite a lot of connectivity. The bottom line is that individual owners, having now been forewarned, will need to decide what's going on with their particular car on a car by car basis.

Miscellany

The first of two pieces of miscellany comes 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. It is beautifully done: https://retr0.id/stuff/2038/. For ease of future access during the next 15 years, I gave it a GRC shortcut of "2038", that being the year of doom: https://grc.sc/2038.

The clock is a beautiful conception which I admire. 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 8-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. Unfortunately, UNIX time is a 32-bit SIGNED value (whoops!). So we only get half of the normal unsigned range. 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, before the time wrapped around back to zero. But being a signed value means that half of the total 4.3 billion values that can be represented by a 32-bit quantity are reserved to represent negative numbers. So the instant the high-bit of the counter turns on, which would be when the RED hand of the clock points straight down to 80 hex, UNIX time goes negative and the simulation we've all been enjoying so much crunches in on itself, otherwise known as game over.

Everyone pays for Twitter?

And 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 the platform. Elon is all about the free market. Well, this is the free market in action. Objectively, he is managing to destroy Twitter, now renamed to 'X', which, as an aside, is about the worst branding imaginable. 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.

So, Elon has been casting about in search of more revenue. First, he decided that he'd make people pay for the use of the blue verified seal. I had no problem 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 notes 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 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 it 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. Many of our listeners have communicated that they don't use Twitter and never will. Just yesterday, I received a DM from a listener who joined Twitter just to send me his DM.

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.

This sentiment of "I don't use Twitter and I never will" is strong among the majority of our listeners. I understand that. I also understand that it's less specific than Twitter. The real underlying sentiment is "I don't use social networking and I never will." Twitter was just the specific instance of social networking that was most prominent at the time. Therefore, I don't believe that moving to Mastodon, Discord, Bluesky, Threads, LinkedIn, Tumblr, Instagram or any other social media platform is the answer for this particular community. Even the fact that there are so many disjoint and competing platforms demonstrates the problem.

The answer, I believe, is good old fashioned eMail. We all have it. We all already have it. It works and it does exactly what we all want. I've mentioned previously 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 the link to the week's show notes, a brief summary of the podcast and the picture of the week. And anyone will be free to unsubscribe or subscribe from multiple accounts as they wish.

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 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.

When Hashes Collide

Security and privacy news is obviously 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 (or X'd) via DM

In addition to being an avid listener to 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, etc. 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 time scale 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 we should 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 insights on the technique of "Pseudonymization" (https://en.wikipedia.org/wiki/Pseudonymization) for personally identifiable information (PII) data.

Ray's note to me links 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."*

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, SHA256, converts anything that's input (like someone's password) into a fixed length of 256 binary bits. 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, by the way, why we call it a "hash" – it makes a hash out of whatever we put in.)

However, the output of today's typical 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 of a cryptographic hash.

Turning now 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 – which is a far smaller population. So I wanted to use this question and Ray's particular application to talk about something we've never discussed before, which is the use of only a portion of an entire cryptographic hash.

Those listeners who have 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 block of input, every single bit of the output will have a 50/50 chance of being a 0 or a 1. And, if any one bit of the input is changed, on average, some collection of half of the output bits will be changed. So that's just very cool. Think about that. All of the bits are exactly equally relevant. Every bit has exactly a 50/50 chance of being a 0 or a 1. And when a single bit of the hash's input is changed, on average half of the hash's output bits will be changed.

One way to think of this is that when something is put into the hash what comes out is a large pseudo-random 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 pseudo-random number output is independent of all of the other bits – it does not care about any of its neighbors – and given any input, it will have an equal chance of being a 0 or a 1.

What that tells us is that there is no need, whatsoever, to use it all.

The only reason modern cryptographic hashes are becoming longer is to increase the universe of possible outputs and to keep the hash's internal algorithm 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. So approximately 115 followed by 75 '0's. 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, counter-intuitive 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 8-bit byte from the hash of his patron's aggregated personal information – perhaps he just uses their eMail address. Since an 8-bit byte can have 256 possible bit combinations, this would have the effect of dividing the entire population of the library's patrons into 256 groups. No matter what personal data is put in, a number from 0 to 255 will come out.

Presuming that the library has thousands of local patrons, there will obviously be collisions since all of those thousands 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 8-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 about the behavior of any one individual due to the presence of these deliberately engineered hash collisions.

So, say that 8-bits is too few. Ray needs more resolution. He wants to be able to collect meaningful statistical data by keeping his post-anonymized patrons more separated – but not provably separate. He wants to leave a modicum of deliberate uncertainty, introduced by hashing collision, 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 the year (we'll ignore leap years for the moment) and equally probable birthday occurrences throughout the year (an even distribution), 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 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. The number is as low as it is because the question is "any two" having the same birthday. In other words, 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 be assumed.

So there are two different goals each with a different property. Ray wants sufficient resolution among his library's patrons. 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 it's https://grc.sc/840. This will allow anyone to play with these numbers: https://kevingal.com/apps/collision.html

If we test it 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 chance of collision drops to 47.57%. So this confirms that 23 people is the minimum number needed to reach 50%.

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. How many hash bits would we need to use for there to be a 50% chance of any collision among that population of 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.

Poking the numbers into that page we find that using just 24 bits of the hash's value, instead of all 256 bits – so only three bytes of storage – will result in a 52.5% chance of any two patrons having the same 24-bit identifier when their personally identifiable information is hashed. Three bytes is also sort of cool, since those 24 bits, instead of being divided into three bytes of 8 bits, can be divided into four groups of 6 bits. And 6 bits can be mapped into a 64-character alphabet of printable characters. This means that each of the library's 5,000 patrons can have their personally identifiable information hashed and turned into a 4-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 4-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.

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 fewer number of bits, and that which bits are used makes absolutely no difference.