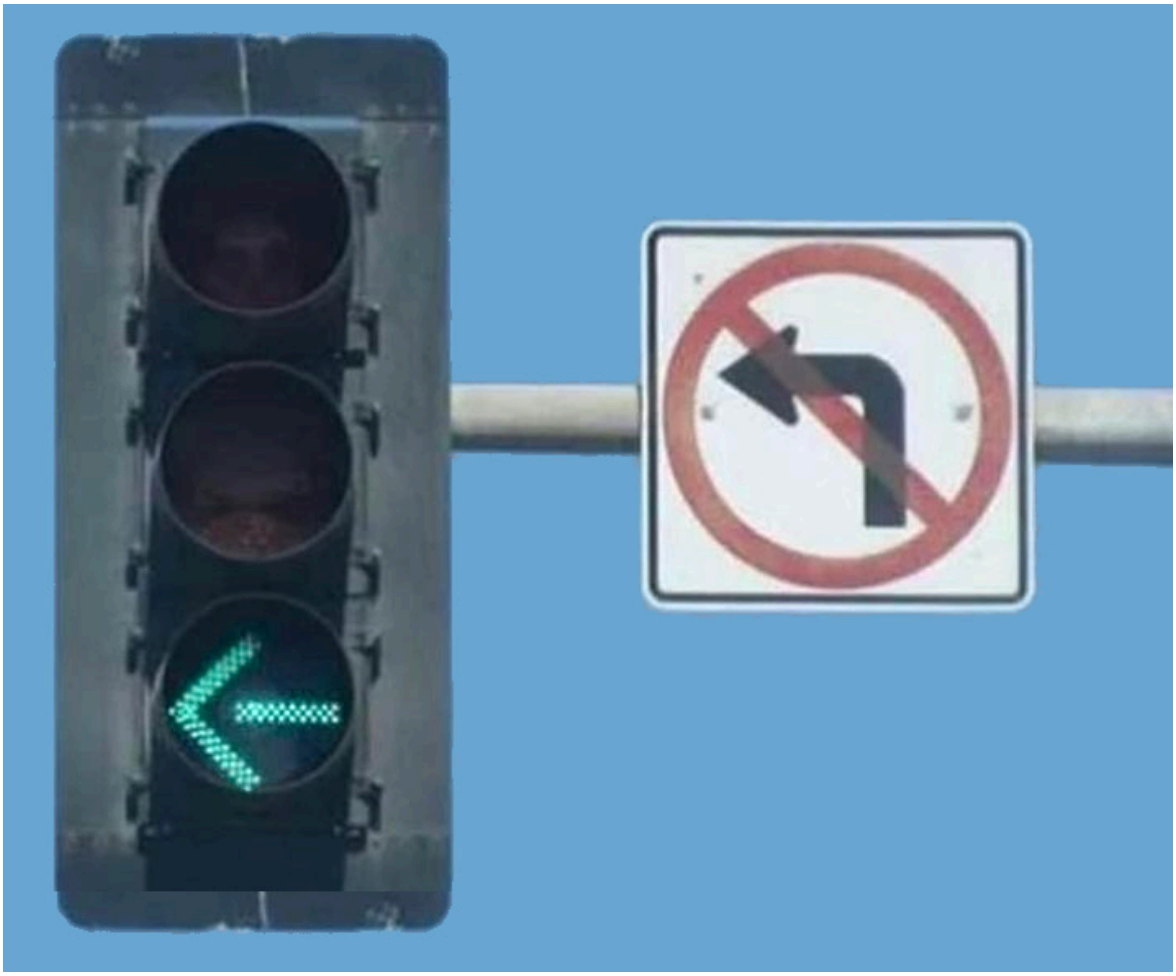
Security Now! #981 - 07-02-24

The End of Entrust Trust

This week on Security Now!

Why does everyone running OpenSSH need to patch immediately? Who just moved 50 bitcoins minted in 2010? (Sadly, it wasn't me.) How are things going with our intrepid Voyager 1? What features have I removed from GRC's email system? And what embarrassingly affordable commercial emailing system do I now recommend without reservation? Who's a "she" and not a "he"? What's recently been happening with SyncThing? Why do I use DNS for freeware release management? And what in the world happened to cause one of the industry's original SSL/TLS certificate authorities to fall from grace and lose all future access to Chrome's root store? Another really great episode of Security Now! is yours for the taking.

Perhaps we deserve to be taken over by the machines.



Security News

The regreSSHion Bug

The big news of the week which has everyone buzzing at the moment is a regression flaw that was discovered by Qualys in the widely used and inherently publicly exposed OpenSSH service. The developers of OpenSSH have historically been extremely careful, so this is the first OpenSSH vulnerability to be discovered in nearly 20 years. But nevertheless, when you hear that OpenSSH has an unauthenticated remote code execution vulnerability that grants its exploiter full root access to the system with the ability to create a root-level remote shell, affects the default configuration and does not require user interaction, that'll get anyone's attention.

So we have CVE-2024-6387. Here's what it's discoverer, Qualys, says about it:

> *The Qualys Threat Research Unit (TRU) discovered this unauthenticated Remote Code Execution (RCE) vulnerability in OpenSSH's server (sshd) in glibc-based Linux systems. This bug marks the first OpenSSH vulnerability in nearly two decades—an unauthenticated RCE that grants full root access. It affects the default configuration and does not require user interaction, posing a significant exploit risk.*
>
> *In Qualys TRU's analysis, we identified that this vulnerability is a regression of the previously patched vulnerability CVE-2006-5051, reported in 2006. A regression in this context means that a flaw, once fixed, has reappeared in a subsequent software release, typically due to changes or updates that inadvertently reintroduce the issue. This incident highlights the crucial role of thorough regression testing to prevent the reintroduction of known vulnerabilities into the environment. This regression was introduced in October 2020 (OpenSSH 8.5p1).*
>
> *OpenSSH is a suite of secure networking utilities based on the SSH protocol that are essential for secure communication over unsecured networks. It provides robust encryption, secure file transfers, and remote server management. OpenSSH is widely used on Unix-like systems, including macOS and Linux, and it supports various encryption technologies and enforces robust access controls. Despite a recent vulnerability, OpenSSH maintains a strong security record, exemplifying a defense-in-depth approach and a critical tool for maintaining network communication confidentiality and integrity worldwide.*

What we're dealing with in this instance is a very subtle and very tight race condition between multiple threads of execution.

Remember that not long ago we spent some time looking at race conditions. Back then, I used the example of two threads that both wanted to test and conditionally increment the same variable. A race condition fault could occur if one thread first read the variable and tested it, but before it could return the updated value it was preempted, then another thread might come along to change that shared variable without the first thread being aware. When that first thread then resumed execution, it would place the updated value back into the variable, destroying whatever the second thread had done. Today we're going to see a real world example of exactly this sort of problem occurring.

First, though, I want to share Qualys' note about OpenSSH in general. In their technical report about this problem, they wrote:

OpenSSH is one of the most secure software in the world; this vulnerability is one slip-up in an otherwise near-flawless implementation. Its defense-in-depth design and code are a model and an inspiration, and we thank OpenSSH's developers for their exemplary work.

They then explain:

This vulnerability is challenging to exploit due to its remote race condition nature, requiring multiple attempts for a successful attack. This can cause memory corruption and necessitate overcoming Address Space Layout Randomization (ASLR). Advancements in deep learning may significantly increase the exploitation rate, potentially providing attackers with a substantial advantage in leveraging such security flaws.

In our experiments, it takes ~10,000 tries on average to win this race condition. So for example, with 10 connections (MaxStartups) accepted per 600 seconds (LoginGraceTime), it takes ~1 week on average to obtain a remote root shell.

Now this is statistics. Right. It could happen on the first try, never, or anywhere in between. It's like those 50 bitcoin I mined back in 2011. I got lucky. And it's still possible to "get lucky" today, though it's vastly less likely. The great concern is the available inventory of vulnerable OpenSSH servers which are publicly exposed to the Internet.

Qualys write that searches using Censys and Shodan they had identified over 14 million potentially vulnerable OpenSSH server instances exposed to the Internet and that within their own customer base of users who are using their CSAM 3.0 External Attack Surface Management technology, approximately 700,000 external internet-facing instances are vulnerable. This accounts for 31% of all internet-facing instances with OpenSSH in our global customer base.

So the way to think about this is that both intensely targeted and also diffuse and widespread attacks are going to be highly likely. If a high value target is running a vulnerable instance of OpenSSH, once this has been fully weaponized, someone can patiently try and retry in a fully automated fashion, patiently knocking at the door, until they get in. And what makes this worse, as we'll see, is that the attack is not an obvious flooding style. Its nature requires a great deal of waiting. This is why the 10 connections over 600 seconds they mentioned. Each attack attempt requires a ten-minute timeout. But 10 can be overlapped and running at once against a single target. So on average that's one new connection attempt per minute... each one patiently knocking quietly on the door until it opens.

And note that what this means is that a single attacker can be, and almost certainly will be, simultaneously spraying a massive number of overlapping connection attempts across the Internet. It would make no sense for a single attacking machine to sit around waiting ten minutes for a single connection timeout. Rather, attackers will be launching as many new attempts as they can against other targets during the ten minutes they must wait to see whether a single connection attempt succeeded on any one machine.

To that end, they wrote:

Qualys has developed a working exploit for the regreSSHion vulnerability. As part of the disclosure process, we successfully demonstrated the exploit to the OpenSSH team to assist with their understanding and remediation efforts. We do not release our exploits, as we must

allow time for patches to be applied. However, even though the exploit is complex, we believe that other independent researchers will be able to replicate our results.

And then, indeed, they detail exactly where the problem lies. I'll share two dense paragraphs then pause to clarify what they've said:

We discovered a vulnerability (a signal handler race condition) in OpenSSH's server (sshd): if a client does not authenticate within LoginGraceTime seconds (120 by default, 600 in old OpenSSH versions), then sshd's SIGALRM handler is called asynchronously, but this signal handler calls various functions that are not async-signal-safe (for example, syslog()). This race condition affects sshd in its default configuration.

This vulnerability is exploitable remotely on glibc-based Linux systems, where syslog() itself calls async-signal-unsafe functions (for example, malloc() and free()): an unauthenticated remote code execution as root, because it affects sshd's privileged code, which is not sandboxed and runs with full privileges. We have not investigated any other libc or operating system; but OpenBSD is notably not vulnerable, because its SIGALRM handler calls syslog_r(), an async-signal-safer version of syslog() that was invented by OpenBSD in 2001.

So, what's going on here is that when someone connects to a vulnerable instance of OpenSSH, as part of the connection management, a connection timeout timer is started. That timer was once set to 600 seconds, or 10 minutes. But in newer builds giving someone ten minutes to get themselves connected seemed excessive and unnecessary, so it was shortened to 120 seconds or two minutes. Unfortunately, in this case, a shorter expiration allows for faster compromise since it's the instant of timer expiration when OpenSSH is vulnerable to exploitation.

So upon a new connection the timer is started to give the new connection ample but limited time to get itself authenticated and going. And if the incoming connection just sits there doing nothing, or trying and failing to properly authenticate, regardless of what's going on and why, when that new connection timeout timer expires, OpenSSH drops the still-pending connection. Unfortunately, before it does that, as it's doing that, it goes off to do some other things, like make an entry in the system log about this expired connection attempt. So if the attacker was doing something at the precise instant that the connection expiration timer expires, the race condition can be forced to occur.

Just as with the two threads in my original shared variable example, the timer's expiration "asynchronously" interrupts what OpenSSH was doing at that moment. So if the attacker was able to time it right, so that OpenSSH was actively doing something involving memory allocation at the exact time of the timer's expiration, the memory allocations that would then be performed by the timer-driven logging would conflict and collide with what the attackers were doing, and **that** could result in the attack obtaining remote code execution under full root privilege. With the massive inventory of exploitable OpenSSH servers available – somewhere around 14 million. And with so many of them forgotten, unattended, and not quickly updated, there's just no way attackers will not be working out the details of this attack for themselves and getting busy.

Qualys explained:

To exploit this vulnerability remotely (to the best of our knowledge, CVE-2006-5051 has never been successfully exploited before), we immediately faced three major problems:

From a theoretical point of view, we must find a useful code path that, if interrupted at the right time by SIGALRM, leaves sshd in an inconsistent state, and we must then exploit this inconsistent state inside the SIGALRM handler.

From a practical point of view, we must find a way to reach this useful code path in sshd, and maximize our chances of interrupting it at the right time.

From a timing point of view, we must find a way to further increase our chances of interrupting this useful code path at the right time, remotely.

To focus on these three problems without having to immediately fight against all the modern operating system protections (in particular, ASLR and NX), we decided to exploit old OpenSSH versions first, on i386, and then, based on this experience, recent versions:

First, "SSH-2.0-OpenSSH_3.4p1 Debian 1:3.4p1-1.woody.3". This is the first Debian version that has privilege separation enabled by default and that is patched against all the critical vulnerabilities of that era.

To remotely exploit this version, we interrupt a call to free() with SIGALRM (inside sshd's public-key parsing code), leave the heap in an inconsistent state, and exploit this inconsistent state during another call to free(), inside the SIGALRM handler.

In our experiments, it takes ~10,000 tries on average to win this race condition; i.e., with 10 connections (MaxStartups) accepted per 600 seconds (LoginGraceTime), it takes ~1 week on average ***to obtain a remote root shell.***

On a newer Debian build where the LoginGraceTime defaults to 120 seconds, it still took them around 10,000 attempts, but since they only needed to wait 2 minutes for timer expiration rather than 10 minutes, they were able to obtain a remote root shell in 1 to 2 days.

And on the most current stable Debian version v12.5.0, due to the fact that it has reduced the login timeout to 120 seconds and has also increased the maximum number of simultaneous login attempts – the so-called "MaxStartups" – from 10 to 100, they wrote:

In our experiments, it takes ~10,000 tries on average to win this race condition, so ~3-4 hours with 100 connections (MaxStartups) accepted per 120 seconds (LoginGraceTime). Ultimately, it takes ~6-8 hours on average to obtain a remote root shell, because we can only guess the glibc's address correctly half of the time (because of ASLR).

And they finish by explaining:

This research is still a work in progress: We have targeted virtual machines only, not bare-metal servers, on a mostly stable network link (~10ms packet jitter); We are convinced that various aspects of our exploits can be greatly improved, and we have started to work on an amd64 exploit, which is much harder because of the stronger ASLR.

That's "Address Space Layout Randomization" and the reason 64 bits make things much more difficult is that more high-bits allows for more randomized places to locate code. Finally:

> *A few days after we started our work on amd64, we noticed a bug report in OpenSSH's public Bugzilla, regarding a deadlock in sshd's SIGALRM handler. We therefore decided to contact OpenSSH's developers immediately to let them know that this deadlock is caused by an exploitable vulnerability, we put our amd64 work on hold, and we started to write this advisory.*

So, yikes. We have another new and potentially devastating problem. Everyone running a maintained Linux that's exposing an OpenSSH server to the public Internet, and potentially even major corporations using OpenSSH internally, need to update their builds to incorporate a fix for this. Until that's done, and unless you must have SSH running, it might be worth blocking its port or shutting it down.

50BTC moved

A listener of ours, James Tutton, shot me a note asking whether I may have found my 50BTC when he saw an article about 50BTC having been moved from a long dormant wallet. I wish that was my 50BTC, but I have satisfied myself that they are long gone. But I thought our listeners would enjoy hearing about the general topic of ancient Bitcoin movement. The article which appeared last Thursday at Cryptonews.com was titled: "Satoshi Era Bitcoin Wallet Awakens, Moves 50 BTC to Binance", and it reads:

> *A Satoshi era bitcoin wallet address, dormant for 14 years, transferred 50 BTC (approximately $3.05 million USD) to the Binance exchange on June 27 (that's last Thursday). The wallet is believed to belong to a Bitcoin miner who likely earned the 50 BTC as mining rewards in 2010.*
>
> *On-chain analytics firm Lookonchain revealed the Bitcoin wallet's origins. It's linked to a miner who received 50 BTC as a mining reward on July 14, 2010, just months after the Bitcoin network launched.*

And I'll note that my podcast with Tom Merritt which was titled "Bitcoin Cryptocurrency" where I explained the operation of the entire Bitcoin cryptocurrency system, blockchain and all, aired the following February 9th of 2011. So while it's true that solving the Bitcoin hash problem way back then resulted in an award of 50 bitcoin, my 50 were different from the 50 that were recently moved. The article continues:

> *Back in 2010, one BTC was valued at a mere $0.003, or 0.3 cents. This price wasn't surpassed until February 2011, reaching $30 by June of that year. Today, Bitcoin trades around $61,110, which is a 17% drop from its all time high in mid-March 2024 of $73,750.*
>
> *Satoshi bitcoin wallets, which were created during Bitcoin's infancy (2009-2011), hold historical significance. This period marked the time when Bitcoin's enigmatic creator, Satoshi Nakamoto, was still an active presence in the cryptocurrency community. The wallets' historical value, coupled with the limited transactions during that era, makes any movement of funds from them a notable event.*

In 2010, Bitcoin mining was accessible to anyone with a personal computer, yielding a reward of 50 BTC. This accessibility stands in stark contrast to the current Bitcoin mining environment. Four halving events (as in cut in half) have since reduced the block reward to a mere 3.125 BTC. These halvings, occurring roughly every four years, are integral to Bitcoin's deflationary model. This recent transfer from a Satoshi bitcoin wallet is not an isolated incident. It joins a growing list of dormant wallets springing back to life.

Yeah, no kidding. Multiplying the number of bitcoins which were easily earned back then by $60,000 will definitely put a spring in one's step! They wrote:

In March, a similar event occurred. A miner transferred 50 BTC, earned from mining on April 25, 2010, to Coinbase after 14 years of wallet inactivity. The reactivation of these wallets often stirs interest and speculation within the cryptocurrency community. Many are curious about the intentions behind these moves, whether they signal a change in market dynamics or simply represent a long-time holder finally deciding to liquidate their assets.

Bitcoin Whales, individuals or entities holding vast quantities of Bitcoin, possess the capacity to influence the cryptocurrency market through their sheer trading volume and holdings. Two such Whale wallets, dormant for a decade, sprang to life on May 12, 2024, transferring a combined 1,000 BTC. On September 12 and 13 of 2013, when Bitcoin was trading at $124, each of these two wallets received 500 BTC which was valued at $62K each back then.

In another noteworthy event on May 6, a Bitcoin Whale moved $43.893 million worth of BTC to two wallet addresses. This Whale had remained inactive for over 10 years, having initially received the Bitcoin on January 12, 2014, when it traded at $917.

This podcast did not quite get here before the beginning of the Internet. But we have been present to watch and chronicle the birth, emergence and maturation of the cryptocurrency phenomenon. Unfortunately, we've also observed that it has been the emergence of this largely anonymous form of value transfer that has, in turn, enabled and driven the scourge of ransomware and other attacks-for-hire services on the dark web.

Voyager 1 Update

On the topic of astonishing achievements by mankind, I wanted to share a brief update on the status on what has now become the Voyager 1 interstellar probe. NASA's JPL wrote:

NASA's Voyager 1 spacecraft is conducting normal science operations for the first time following a technical issue that arose in November 2023.

The team partially resolved the issue in April when they prompted the spacecraft to begin returning engineering data, which includes information about the health and status of the spacecraft. On May 19, the mission team executed the second step of that repair process and beamed a command to the spacecraft to begin returning science data. Two of the four science instruments returned to their normal operating modes immediately. Two other instruments required some additional work, but now, all four are returning usable science data.

The four instruments study plasma waves, magnetic fields, and particles. Voyager 1 and Voyager 2 are the only spacecraft to directly sample interstellar space, which is the region outside the heliosphere — the protective bubble of magnetic fields and solar wind created by the Sun.

While Voyager 1 is back to conducting science, additional minor work is needed to clean up the effects of the issue. Among other tasks, engineers will resynchronize timekeeping software in the spacecraft's three onboard computers so they can execute commands at the right time. The team will also perform maintenance on the digital tape recorder, which records some data for the plasma wave instrument that is sent to Earth twice per year. (Most of the Voyagers' science data is sent directly to Earth and not recorded onboard.)

Voyager 1 is more than 15 billion miles (24 billion kilometers) from Earth, and Voyager 2 is more than 12 billion miles (20 billion kilometers) from the planet. The probes will mark 47 years of operations later this year. They are NASA's longest-running and most-distant spacecraft.

Email @ GRC

Everything is going well with GRC's email system and I'm nearly finished with my work on it. The work I'm finishing up is automation for sending the weekly Security Now! email, so that I'm able to do it before the podcast while being prevented from making any errors. I'm about a day or two away from being able to declare that work finished.

Last week's announcement that I had started sending out weekly podcast summaries generated renewed interest and questions from listeners (via Twitter or forwarded to me through Sue and Greg) who had apparently been waiting for the news that something was being sent before subscribing to these weekly summary mailings. So now they wanted to know how to do that. All anyone needs to know is that at the top of every page at GRC is a shiny new white envelope labeled "Email subscriptions." Clicking that begins the process. If you follow the instructions presented at each step, a minute or two later you'll be subscribed. And remember that if your desire is not to subscribe to any of the lists, but to be able to bypass social media to send email directly to me, you are welcome to leave all of the subscription checkboxes unchecked when you press the "Update Subscriptions" button, That will serve to confirm your email address which then allows you to send feedback mail, pictures of the week and whatever else you like directly to me by writing to securitynow (at) grc.com.

Finally, I wanted to note that the email today's subscribers have already received from me was 100% unmonitored, as I expect all future email will be. So I won't know whether those emails are opened or not. I've also removed all of the link redirections from GRC's mail so that clicks are also no longer being monitored. This makes the mailings completely blind, but it also makes for cleaner and clearer email. Some of our listeners were objecting to their clients warning them about being tracked – even though I still don't think that's a fair use of a loaded term when the email has been solicited by the user and if the notification only comes back to me. I would never have bothered to put any of that in, if I'd written the system from scratch, but it was all built into the bulk mailing system I purchased, and it's so slick, and has such lovely graphical displays

that it was fun to look at while it was new. And I never anticipated the backlash doing this would produce – but then, we're not your average crowd, are we? We're the Security Now! audience!

For the first two mailings I've done so far, which did contain link monitoring, it was interesting to see that three times more people clicked to view the full-size picture of the week than clicked to view the show notes. That makes sense when most people will be listening to the podcast audio but are curious to see the picture of the week, which we have fun describing each week. In any event, I'm over it now. So no more single-pixel image fetches with its attendant email client freakout, or anything else that might be controversial. What you do with any email you receive from me is up to you. I'm just grateful for everyone's interest.

One thing I've been waiting to do, is to give a shout out to the emailing system I chose to use. I've been utterly and totally impressed by its design, its complete feature set, its maturity, and the author's support of this system. I feel somewhat embarrassed over what I have received in return for a one-time purchase payment of $169 US dollars. This thing is worth far more than that. Because I'm me, I insisted upon writing my own subscription management front-end, though this package's author, a Greek guy whose first name is Panos, has no idea why I've done so and thinks I'm nuts, since his system, as delivered, does all of that too. But as Frank Sinatra famously said, "I did it my way."

NuevoMailer, spelled "nuevoMailer" is an open-source PHP-based email marketing management and mailing solution. It runs beautifully under Windows, Unix or Linux. To help anyone who might have any need to create an emailing facility from scratch, it's this episode's GRC short of the week, so: grc.sc/981 will bounce you over to: https://www.nuevomailer.com/.

I've had back and forth dialogs with Panos because I've needed to customize some of the RESTful APIs his package publishes. For example, a new feature that's present in the email everyone received from me today provides a direct link to everyone's email subscription management page. I needed to modify some of his code to make that work. So I can vouch for the support he provides. And as I said, I've felt somewhat guilty about paying so little when I've received so much. So I'm aware that telling this podcast's listeners about his work will likely help him. All I can say is that he deserves every penny he makes. There are thousands of bulk mailing solutions out in the world. This one allows you to roll your own, and I'm very glad I chose it.

Errata

Last week's podcast drew heavily on two articles written by Kim Zetter. It's embarrassing that I've been reading, appreciating and sharing Kim's writing for years, but never stopped to wonder whether Kim would probably associate with the pronoun "he" or a "she". Her quite attractive Wikipedia photo strongly suggests that she would opt for "she" – as will I from now on.

Miscellany

SyncThing

I wanted to note that while I'm still a big fan of SyncThing, lately I had been noticing a great deal of slowdown in its synchronization relay servers. I don't think they used to be so slow. I'm unable to get more than 1.5 to 1.8 megabits of traffic through them. When it's not possible to obtain a direct end-to-end connection between SyncThing endpoints, an external 3rd-party relay server is required to handle their transit traffic. It's all super-well encrypted, so that's not the issue. The issue is the performance of this solution.

Since this problem was persisting for multiple weeks, my assumption is that SyncThing's popularity has been growing and is loading down their relay server infrastructure. At one point in the past I had arranged for point-to-point connections between my two locations. But some network reconfiguration had broken that. My daytime work location has a machine that runs 24/7. But I shutdown my evening location's machine at the end of every evening's work. The trouble was that synchronization with the always-on machine had become so slow that I was needing to leave my evening machine running unattended for several hours, waiting for my evening's work to trickle out and be synchronized with the machine I'd be using the next morning.

It finally became so intolerable that I sat down and punched remote IP filtered holes through my firewalls at each endpoint. Even if pfSense's firewall rules were not able to track public domain names as they are, the public IPs of our cable modems change so rarely that statically opening an incoming port to a specific remote public IP is practical. Once I punched those holes, SyncThing was able to make a direct point-to-point connection and once again screams. So just a heads-up to anyone who may be seeing the same dramatic slowdown that I'm seeing with the use of their relay server infrastructure. It's an amazingly useful free service, and helping it to establish direct connections between endpoints also helps keep the relay servers free for those who really need them.

Closing The Loop

Matt St.Clair Bishop

Hello Steve. I've been a listener of Security Now! for some years now, however as I have edged closer to making my own utilities publicly available, my mind has turned to my method of updating them. I think in my dim and distant memory I remember you saying that you used a simple DNS record to hold the latest edition of each of your public releases and the client software inspects that record, it being a very simple and efficient mechanism to flag available updates. Could you elaborate at all if you have a spare section in your podcast, I'm personally using C# and the .NET framework as I'm a Windows only guy, so if you could paint the broad strokes I should be able to Google the C# detail. SpinRite user, Loving all your efforts in this field, Matt SCB.

Matt's correct about my use of DNS and I'm pleased with the way that capability has turned out. Anyone who has the ability to lookup the IP address for validrive.rel.grc.com will find that it returns: 239.0.0.1. This is because ValiDrive is still at its first release. When I've released an update to ValiDrive, it will be release #2 and I'll change the IP address to 239.0.0.2.

Whenever an instance of ValiDrive is launched, it performs a quick DNS lookup of its own product name "validrive.rel.dns.com" and verifies that the release number returned in the lower byte of the IP address is not higher than its own current release number. If it is, it will notify its user that a newer release exists. What's convenient about this is that the version checking is performed by a single DNS query packet and that DNS is distributed and caching. So it's possible to set a very long cache expiration to allow the cached knowledge of the most recent version of ValiDrive to vary widely. This means that when the release number is incremented, the notifications of this will be widely distributed as local caches expire. This prevents everyone from coming back at once.

And to Matt's question and point, I've never encountered a language that did not provide some means for making a DNS query. I know that C# and .NET make this trivial.

A listeners of ours named **Carl** sent email to me at securitynow(at)grc.com

> *Hi Steve, Much has been discussed over the recent weeks on your podcast about the upcoming Windows Recall feature and its value proposition vs security and privacy concerns.*
>
> *It has been suggested that the concept started as a productivity assistant that uses AI to index and catalog everything on your screen (Clippy 2.0?!), and may be more applicable in an office environment than at home. However, I think it was just as likely that this concept first started as a productivity monitoring tool, where corporate management can leverage Recall to ensure employees are using their screen time doing real, actual work. Of course, Microsoft realizes they can't possibly market Recall this way, so here we are.*
>
> *I dread the day Recall is installed on my work computer. Carl*

Carl's take on this is aligned with "The Evil Empire" theory which, as we know, I don't subscribe to. I would say that Recall is ethically neutral. It's like the discovery of the chain reaction in the fission of atomic nuclei. That discovery can be used to generate needed power or to make a really big bomb. But the chain reaction itself is just the physics of our universe. Similarly, Recall is just a new capability which could be used to either help or to hurt people. Could employers use it to scroll back through their employee's timeline to see what they've been doing on enterprise- owned machines? That's not yet clear. There are indications that Microsoft is working to make that impossible. But we know that as it was first delivered it would have been entirely possible.

It appears that Microsoft desperately wants to bring Recall to their Windows desktops. It would be incredibly valuable as training material for a local AI assistant and to deeply profile the desktop's user as a means for driving advertising selection in a future ad-supported Windows platform. So I suspect they will be doing anything and everything required to make it palatable.

The End of Entrust Trust

This week, I want to share the story, and the backstory, of the web browser community again bidding a less than fond farewell to yet another certificate authority. In, as we'll see, what appears to be a demonstration of **executive arrogance**, Entrust, one of the oldest original certificate authorities, after six years of being pushed, prodded and encouraged to live up to the responsibilities that accompany the right to essentially print money by charging to encrypt the hash of a blob of bits, the rest of the industry that proactively monitors and manages the behavior of those who have been "entrusted" to do this responsibly, finally reached its limit and Google announced last Thursday that Chrome would be curtailing its trust of Entrust from its browser's root store.

Signing and managing certificates is by no means rocket science. There's nothing mysterious or particularly challenging about it. It's mostly a clerical activity which must follow a bunch of very clearly spelled out rules about how certificates are formatted and what information they must contain. These rules govern how the certificates must be managed and what actions those who sign them on behalf of their customers must do when problems arise. And just as significantly, the rules are arrived at and agreed upon collectively. The entire process is a somewhat amazing model of self governance. Everyone gets a say, everyone gets a vote, the rules are adjusted in response to the changing conditions in our changing world, and everyone moves forward under the updated guidance. This means that when someone in this collective misbehaves, they are not pushing back against something that was imposed upon them. They are ignoring the rules that they voted to change and agreed to follow.

"Certificates" have been an early and enduring focus and topic on this podcast because so much of today's security is dependent upon knowing that the entity one is connecting to and negotiating with over the Internet really is who we think they are, and not any form of spoofed forgery. The idea behind a certificate authority is that while we may have no way of directly confirming the identity of an entity we don't know across the Internet, if that entity can provide proof that they have previously, and somewhat recently, proven their identity to a 3rd party, a certificate authority, whose identity assertions we do trust, then by extension we can trust that the unknown party is who they say they are when they present a certificate to that effect signed by an authority whom we trust. That's all this whole certificate thing is about. It's beautiful and elegant in its simplicity. But as the saying goes, "the devil is in the details" and as we're going to see today, those who understand the importance of those details can be pretty humorless when they are not only ignored, but flaunted.

The critical key here is that we are completely and solely relying upon a certificate authority's identity assertions, where any failure in such an authority's rigorous verification of the identity of their client customers could have widespread and devastating consequences. This is one of the reasons I've always been so very impressed with the extreme patience shown by the governing parties of this industry in the face of certificate authority misbehavior. Through the years we've seen many examples where a trusted certificate authority really needs to screw up over a period of many years, and actively resist improving their game, in order to finally have the industry lower the boom on them. No one wants to do this indiscriminately or casually because it unilaterally puts the wayward CA out of the very profitable browser certificate issuing business

overnight.

Okay, so what happened? In a remarkable show of prescience, when things were only just heating up, FeistyDuck's "Cryptography & Security Newsletter" posted the following **only a few hours** before Google finally lowered the boom on Entrust. FeistyDuck wrote...

Entrust, one of the oldest Certification Authorities, is in trouble with Mozilla and other root stores. In the last several years, going back to 2020, there have been multiple persistent technical problems with Entrust's certificates. That's not a big deal when it happens once, or even a couple of times, and when it's handled well. But according to Mozilla and others, it hasn't been. Over time, frustration grew. Entrust made promises which it then broke. Finally, in May, Mozilla compiled a list of recent issues and asked Entrust to formally respond.

Entrust's first response did not go down well, being non-responsive and lacking sufficient detail. Sensing trouble, it later provided another response, with more information. We haven't seen a response back from Mozilla yet, just ones from various other unhappy members of the community. It's clear that Entrust's case has reached a critical mass of unhappiness.

We haven't heard from other root stores directly … yet. However, at the recent CA/Browser forum meeting also in May, Google used the opportunity to discuss standards for CA incident response. It's not clear if it's just a coincidence, but Google's presentation uses pretty strong words that sound like a serious warning to Entrust and all other CAs to improve—or else.

Looking at the incidents themselves, they're mostly small technical problems of the kind that could have been avoided with standardized validation of certificates just prior to issuance.

As it happens, Ballot SC-75 focuses on preissuance certificate linting. If this ballot passes, linting will become mandatory as of March 2025. It's a good first step. Perhaps the CA/Browser Forum will in the future consider encoding the Baseline Requirements into a series of linting rules that can be applied programmatically to always ensure full compliance.

As I noted, a few hours after FeistyDuck posted this, Google made their announcement. Last Thursday, June 27th, the Chrome Root Program and Chrome Security Team posted the following to Google's Security Blog under the title "Sustaining Digital Certificate Security - Entrust Certificate Distrust" They wrote:

The Chrome Security Team prioritizes the security and privacy of Chrome's users, and we are unwilling to compromise on these values. The Chrome Root Program Policy states that CA certificates included in the Chrome Root Store must provide value to Chrome end users that exceeds the risk of their continued inclusion. It also describes many of the factors we consider significant when CA Owners disclose and respond to incidents. When things don't go right, we expect CA Owners to commit to meaningful and demonstrable change resulting in evidenced continuous improvement.

Over the past several years, publicly disclosed incident reports highlighted a pattern of concerning behaviors by Entrust that fall short of the above expectations, and has eroded confidence in their competence, reliability, and integrity as a publicly-trusted CA Owner. In response to the above concerns and to preserve the integrity of the Web PKI ecosystem, Chrome will take the following actions.

*In Chrome 127 and higher, TLS server authentication certificates validating to the following Entrust roots whose earliest Signed Certificate Timestamp (SCT) is dated **after** October 31, 2024, will no longer be trusted by default.*

The posting then proceeds to enumerate the nine root certificates that Chrome has, until now, trusted to be signing valid TLS server certificates. They continue, writing:

TLS server authentication certificates validating to the above set of roots whose earliest Signed Certificate Timestamp is on or before October 31, 2024, will not be affected by this change. This approach attempts to minimize disruption to existing subscribers using a recently announced Chrome feature to remove default trust based on the SCTs in certificates.

Additionally, should a Chrome user or enterprise explicitly trust any of the above certificates on a platform and version of Chrome relying on the Chrome Root Store (e.g., explicit trust is conveyed through a Group Policy Object on Windows), the SCT-based constraints described above will be overridden and certificates will function as they do today.

To further minimize risk of disruption, website operators are encouraged to review the "Frequently Asked Questions" listed below.

Okay. Now if Chrome were to yank all nine of those Entrust certs from their root store, at that instant any web servers that were using Entrust TLS certificates would generate those very scary "untrusted certificate warnings" from Chrome and be quite difficult to use. Sample: https://untrusted-root.badssl.com/ Instead, Chrome is now able to keep those semi-trusted root certificates in their root store in order to continue trusting any certificates Entrust previously signed and will sign during the next four months, through this coming October. But no Entrust TLS server certificates signed from November on will be accepted by Chrome.

Now there can be no question that this will be devastating for Entrust. Not only will this shutdown their TLS certificate business, but CAs obtain a great deal of additional revenue by providing their customers with many related services. Entrust will lose all that, too. And, of course, there's the significant reputational damage that accompanies this which, makes a bit of a mockery out of their own name. And there's really nothing they can say or do at this point. The system of revoking CA trust operates with such care to give misbehaving CAs every opportunity to fix their troubles that any CA must be flagrant in their misbehavior for this to occur. As long time listeners of this podcast know, I'm not of the belief that firing someone who missteps always makes sense. Mistakes happen and valuable lessons can be learned. But from what I've seen, and what I'm going to share, I'll be surprised if this is a survivable event for Entrust's Director of Certificate Services, a guy named Bruce Morton.

Way back 1994, Entrust built and sold the first commercially available public key infrastructure. Five years later, in 1999, they entered the public SSL market by chaining to the Thawte Root and created Entrust.net. And then, ten years later Entrust was acquired for $124 million by Thoma Bravo, a U.S.-based private equity firm. I don't know whether being owned by private equity may have contributed to their behavior, but they have that in common with Lastpass.

Google, in their FAQ answering the question: *"Why is Chrome taking action?"*, they replied:

> *Certification Authorities (CAs) serve a privileged and trusted role on the Internet that underpin encrypted connections between browsers and websites. With this tremendous responsibility comes an expectation of adhering to reasonable and consensus-driven security and compliance expectations, including those defined by the CA/Browser TLS Baseline Requirements.*
>
> ***Over the past six years**, we have observed a pattern of compliance failures, unmet improvement commitments, and the absence of tangible, measurable progress in response to publicly disclosed incident reports. When these factors are considered in aggregate and considered against the inherent risk each publicly-trusted CA poses to the Internet ecosystem, it is our opinion that Chrome's continued trust in Entrust is no longer justified.*

This makes a key point: It's not any one thing that Entrust did, taken in isolation, that resulted in this loss of trust. The loss of trust resulted from multiple years of demonstrated uncaring about following the rules that they had voted upon and agreed to. No one wants to make Entrust an example. Too many lives will be negatively impacted by this decision. But the entire system only functions when everyone follows the rules they have agreed to. Entrust refused to do that, so they had to go. Let's look at some specifics.

For example, a few months ago, following an alert from Google's Ryan Dickson, Entrust discovered that all of its EV certificates issued since the implementation of changes due to Ballot SC-62v2, which amounted to approximately 26,668 certificates, were missing their CPS URI, in violation of the EV Guidelines. Entrust said this was due to discrepancies/misinterpretations between the CA/Browser Forum's TLS Baseline Requirements and the Extended Validation Guidelines. Entrust chose to **not** stop issuing the EV certificates (that's a violation of the rules) and did **not** begin the process of revoking the certificates (that's another violation). Instead, they argued that the absence of the CPS URI in the EV certificates was due to ambiguities in CA/B Forum requirements; they said that the absence of the CPS URI had no security impact, and that halting and revoking the certificates would negatively impact customers and the broader WebPKI ecosystem. In other words, they thought they were bigger than the rules, the rules were dumb, or that the rules didn't apply to them. Everyone else had to follow them, but not them. Entrust then also proposed a ballot to adjust the EV Guidelines so that they would not be out of compliance, to not require the CPS URI. They also argued that their efforts were better spent focusing on improving automation and handling of certificates, rather than on revocation and reissuance.

Now, the CPS URI is truly incidental. CPS stands for Certification Practice Statement, and EV certs are now supposed to contain a CPS URI, a link pointing to the issuing CAs document. So is leaving that out any big deal? Probably not from a security standpoint. But it's worrisome when a CA intentionally defies the standards that everyone has agreed to follow.

A security and software engineer by the name of "Amir Omidi" has worked on maintaining certificate issuance systems at Let's Encrypt, and Google Trust Services and he's very active in the PKI space. His GitHub account contains 262 repositories and it appears that he's currently working on a project named "boulder" which is an ACME-based certificate authority, written in

Go. And before that was "zlint" an X.509 Certificate Linter focused on Web PKI standards and requirements. Yesterday (Monday) he posted a terrific summary of the way the Public Key Infrastructure industry thinks about these things. He wrote:

> *Entrust did not have one big explosive incident. The current focus on Entrust started with this incident. On its surface, this incident was a simple misunderstanding. This incident happened because up until the SC-62v2 ballot, the CPS URI field in the certificate policy extension was allowed to appear on certificates. This ballot changed the rules and made this field be considered "not recommended".*
>
> *However, this ballot only changed the baseline requirements and did not make any stipulation on how Extended Validation certificates must be. The EV guidelines still contained rules requiring the CPS URI extension.*
>
> *When a CA has an incident like this, the response is simple:*
>
> - *Stop misissuance immediately.*
> - *Fix the certificate profile so you can resume issuance.*
> - *In parallel, figure out how you ended up missing this rule and what the root cause of missing this rule was.*
> - *Revoke the misissued certificates within 120 hours of learning about the incident.*
> - *Provide action items that a reasonable person would read and agree that these actions would prevent an incident like this happening again.*
>
> *When I asked Entrust if they've stopped issuances yet, they said they haven't, and they don't plan to stop issuance. This is where Entrust decided to go from an accidental incident to willful misissuance. This distinction is an important one.* ***Entrust had started knowingly misissuing certificates.*** *Entrust received a lot of push-back from the community over this. This is a line that a CA shouldn't, under any circumstances, cross. Entrust continued to simply not give a crap even after Ben Wilson of the Mozilla Root Program chimed in and said that what Entrust is doing is not acceptable.* ***Entrust only started taking action after Ryan Dickson of the Chrome Root Program also chimed in to say this is unacceptable.***

I'll interrupt to mention that this is an important distinction. The executives at Entrust appeared not to care about any of this until Google weighed in with the power of their Chrome browser. That was a monumental mistake and it demonstrated a fundamental misunderstanding of the way the CA/B forum members operate. None of this operates on the basis of market power. It's only about agreeing to and then following the rules. It's not about "Oh yeah? Make me!" We're not in the schoolyard anymore. Amir continues...

> *Entrust's delayed response to the initial incident, spanning over a week, compounded the problem by creating a secondary "failure to revoke on time" incident. As these issues unfolded, a flurry of questions arose from the community. Entrust's responses were often evasive or minimal, further exacerbating the situation. This pattern of behavior proved increasingly frustrating, prompting me to delve deeper into Entrust's past performance and prior commitments.*

In one of my earlier posts, I found that Entrust had made the promise that:

- *We will not make the decision not to revoke.*
- *We will plan to revoke within the 24 hours or 5 days as applicable for the incident.*
- *We will provide notice to our customers of our obligations to revoke and recommend action within 24 hours or 5 days based on the Baseline Requirements.*

This pattern of behavior led to a troubling cycle: Entrust making promises, breaking them, then making new promises only to break those as well.

*As this unfolded, Entrust and the community uncovered an alarming number of operational mistakes, culminating in a record 18 incidents within just four months. Notably, about half of these incidents involved Entrust offering various excuses for failing to meet the 120-hour certificate revocation deadline—**ironically, a requirement they had voted to implement themselves.***

I do want to highlight that the number of incidents is not necessarily an indication of CA quality. The worst CA is the CA that has no incidents, as it's generally indicative that they're either not self-reporting, or not even aware that they're misissuing.

In other words, mistakes happen. Everyone understands that. No one needs to be perfect here. But it's how the mistakes that are discovered are then handled that demonstrates the trustworthiness of the CA.

Due to the sheer number of incidents, and Entrust's poor responses up until this point, Mozilla then asks Entrust to provide a detailed report of these recent incidents. Mozilla specifically asks Entrust to provide information regarding:

- *The factors and root causes that lead to the initial incidents, highlighting commonalities among the incidents and any systemic failures;*
- *Entrust's initial incident handling and decision-making in response to these incidents, including any internal policies or protocols used by Entrust to guide their response and an evaluation of whether their decisions and overall response complied with Entrust's policies, their practice statement, and the requirements of the Mozilla Root Program;*
- *A detailed timeline of the remediation process and an apportionment of delays to root causes; and*
- *An evaluation of how these recent issues compare to the historical issues referenced above and Entrust's compliance with its previously stated commitments.*

Mozilla also asked that the proposals meet the following requirements:

- *Clear and concrete steps that Entrust proposes to take to address the root causes of these incidents and delayed remediation;*
- *Measurable and objective criteria for Mozilla and the community to evaluate Entrust's progress in deploying these solutions; and*

- *A timeline for which Entrust will commit to meeting these criteria.*

Mozilla gave Entrust a one month deadline to complete this report. Mozilla's email served a dual purpose: it was both a warning to Entrust and an olive branch, offering a path back to proper compliance. This presented Entrust with a significant opportunity. They could have used this moment to demonstrate to the world their understanding that CA rules are crucial for maintaining internet security and safety, and that adhering to these rules is a fundamental responsibility. Moreover, Entrust could have seized this chance to address the community, explaining any misunderstandings in the initial assessment of these incidents and outlining a concrete plan to avoid future revocation delays.

*Unfortunately, Entrust totally dropped the ball on this one. Their first report was a rehash of what was already on Bugzilla, offering nothing new. Unsurprisingly, this prompted a flood of questions from the community. Entrust's response? They decided to take another crack at it with a second report. They submitted this new report a full two weeks **after** the original deadline.*

In their second report, Entrust significantly changed their tone, adopting a much more apologetic stance regarding the incidents. However, this shift in rhetoric wasn't matched by their actions. While expressing regret, Entrust was still:

- *Overlooking certain incidents.*
- *Delaying the revocations of existing misissuances, and*
- *Failing to provide concrete plans to prevent future delayed revocations.*

An analysis of these 18 incidents and Entrust's responses serves as a prime example of mishandled public communications during a crisis.

Stepping back from this for a moment, the only way to really read and understand this is that the executives at Entrust didn't really take any of this seriously. They acted as though they were annoyed by the gnats buzzing around them who were telling **them** how **they** should act and what **they** should do. Amir says:

The consensus among many community members is that Entrust will always prioritize their certificate subscribers over their obligations as a Certificate Authority.

And there it is in a single sentence.

*This practice fundamentally undermines internet security for everyone. **Left unchecked,** it creates a dangerous financial incentive for other CAs to ignore rules when convenient, simply to avoid the uncomfortable task of explaining to subscribers why their certificates need replacement. Naturally, customers prefer CAs that won't disrupt their operations during a certificate's lifetime. However, for CAs that properly adhere to the rules, this is an impossible guarantee to make.*

Furthermore, these incidents were not new to Entrust. As I've covered in earlier posts, Entrust has continuously demonstrated that they're unable to complete a mass revocation event in the 120 hours defined by and required by the baseline requirements. This pattern of behavior suggests a systemic issue rather than isolated incidents.

*Despite there being over a dozen root programs, there are only **four** that are existentially important for a Certificate Authority:*

- *The Mozilla Root Program: Used by Firefox and practically all Linux distribution and FOSS software.*
- *The Chrome Root Program: Used by Chrome (the Browser and the OS) & some Androids.*
- *The Apple Root Program: Used by everything Apple, and*
- *The Microsoft Root Program: Used by everything Microsoft.*

Enforcement over the operational rules of a CA has been a struggle in the past. A root program only has a binary choice... to either trust or distrust a certificate authority.

There's one last piece of interaction that I want to share. It was written by Watson Ladd who studied Math at Berkeley and is presently a principal software engineer at Akamai. Among his other accomplishments he's the author of RFC9382 which specifies SPAKE2, a password-authenticated key exchange system. And his name has been on about six other RFC's.

In the public discussion thread about Entrust's repeated and continuing failings to correct their mistakes and live up to the commitments they had made to the CA/Browser community, Watson publicly addressed a note to Bruce Morton, Entrust's Director of Certificate Services who has been the face of Entrust's repeated failures, excuses and defiance. Watson Ladd wrote:

Dear Bruce,

This report is completely unsatisfactory. It starts by presuming that the problem is 4 incidents. Entrust is always under an obligation to explain the root causes of incidents and what it is doing to avoid them as per the CCADB incident report guidelines. That's not the reason Ben and the community need this report.

[He's referring to Mozilla's Ben Wilson who initially asked Entrust to explain how they would deal with these ongoing problems and demonstrate how they would be prevented in the future. As we know, Entrust's Bruce Morton basically blew him off, apparently because he wasn't from Google.] Anyway, he wrote:

That's not the reason Ben and the community need this report. Rather, it's to go beyond the incident report to draw broader lessons and to say more to help us judge Entrust's continued ability to stay in the root store. The report falls short of what was asked for, in a way that makes me suspect that Entrust is organizationally incapable of reading a document, understanding it, and ensuring each of the clearly worded requests is followed. The implications for being a CA are obvious.

To start, Ben specifically asked for an analysis involving the historical run of issues and a comparison. I don't see that in this report, at all. The list of incidents only has ones from 2024 listed, there's no discussion of the two issues specifically listed by Ben in his message.

Secondly the remedial actions seem to be largely copied & pasted from incident to incident without a lot of explanation. Saying the organizational structure will be changed to enhance support, governance and resourcing really doesn't leave us with a lot of ability to judge success or explain how the changes made (sparse on details) will lead to improvements.

*Similarly process weaknesses are not really discussed in ways that make clear what happened. How can I use this report if I was a different CA to examine **my** organization and see if I can do better? How can we as a community judge the adequacy of the remedial actions in this report?*

Section 2.4 I find mystifying. To my mind there's no inherent connection between a failure to update public information in a place it appears, a delay in reconfiguring a responder, and a bug in the CRL generation process beyond the organizational. These are three separate functions of rather different complexity. If there's a similarity it's between the latter two issues where there was a failure to notice a change in requirements that required action, but that's not what the report says! Why were these three grouped together, and not others? What's the common failure here that doesn't exist with the other incidents?

If this is the best Entrust can do, why should we expect Entrust to be worthy of inclusion in the future? To be clear, there are CAs that have come back from profound failures of governance and judgement. But the first step in that process has been a full and honest accounting of what their failures have been, in a way that has helped others understand where the risks are and helps the community understand why they are trustworthy.

Sincerely, Watson Ladd

Now, I don't know these Entrust guys at all, but given the condescension they've exhibited, it's not difficult to picture them as some C-suite stuffed shirts who have no intention of being **"judged by"** and pushed around by a bunch of pencil-necked geeks. Buy BOY did they read this one wrong. Those pencil-necked geeks with their pocket protectors reached consensus and pulled their plug, ejecting them from the web certificate CA business they had a hand in pioneering.

I've said before when we've seen a CA get ousted, that I hope other CA's are paying attention because it could happen to them too. In this case, Entrust appears to have believed that the rules they agreed to didn't apply to them. Or it might be that the extreme leniency that the industry had been showing them led them to believe that their failures would not catch up with them.

Being a certificate authority is a rare place of privilege where the CA is able to charge a significant amount of money for performing a hash function followed by a public key encryption. As long as the CA carefully abides by the rules of the collective, they get to print money. But it's a privilege that's earned and needs to be respected. And as we've just seen, it's a privilege that can be withdrawn.