Security Now! #1028 - 06-03-25

AI Vulnerability Hunting

This week on Security Now!

• Pwn2Own 2025, Berlin results. • PayPal seeks a "newly registered domains" patent. • An expert iOS jailbreak developer gives up. • The rising abuse of SVG images, via JavaScript. • Interesting feedback from our listeners. • Four classic science fiction movies not to miss. • How OpenAI's o3 model discovered a 0-day in the Linux kernel.

If the U.S. power grid collapses it might not be China’s fault!

Security News

Last week I promised to catch us up with the results from the recent Pwn2Own hacking competition which was held for the first time in Berlin. In their announcement of this, Trend Micros, the organizer of this 18 year old competitive hacking series, wrote:

> *While the Pwn2Own competition started in Vancouver in 2007, we always want to ensure we are reaching the right people with our choice of venue. Over the last few years, the OffensiveCon conference in Berlin has emerged as one of the best offensive-focused events of the year. And while CanSecWest has been a great host over the years, it became apparent that perhaps it was time to relocate our spring event to a new home. With that, we are happy to announce that the enterprise-focused Pwn2Own event will take place on May 15-17, 2025, at the OffensiveCon conference in Berlin, Germany. While this event is currently sold out, we do have tickets available for competitors, and we believe the conference will also open a few more tickets for the public, too. The conference sold out its first run of tickets in under six hours, so it should be a fantastic crowd of some of the best vulnerability researchers in the world.*

It happened two and a half weeks ago. What were the results? Before I run through them I want to remind everyone that what you're going to be hearing are the results when today's upper echelon most skilled penetration hackers go up against fully patched systems. What always strikes me is that the targets are not old junk routers past their end of life. In every case the targets are up to date, fully patched modern systems. It serves as a reminder that to a large extent the only reason we have any appearance of security is that none of these most skilled hackers wants to attack us. So what happened in Berlin?

1. *DEVCORE's Research Team used an integer overflow to escalate privs on Red Hat Linux. He earns $20,000 and 2 Master of Pwn points. (They got root!)*

2. *Although the Summoning Team (@SummoningTeam) successfully demonstrated an exploit of #NVIDIA Triton, the bug used was known by the vendor (but not patched). Nevertheless, they still earned $15K and 1.5 Master of Pwn points.*

3. *STARLabs SG combined a **UAF** – Use After Free – and an integer overflow to escalate to SYSTEM on #Windows 11. He earns $30,000 and 3 Master of Pwn points.*

4. *Researchers from Theori were able to escalate to root on Red Hat Linux with an info leak and a UAF, but one of the bugs used was an N-day. They still win $15,000 and 1.5 Master of Pwn points.*

5. *The first ever winner of the AI category in Pwn2Own history is by Summoning Team (@SummoningTeam). Their successful exploitation of Chroma earns $20,000 and 2 Master of Pwn points.*

6. *In a surprise to no one, Marcin Wiązowski's privilege escalation on Windows 11 is confirmed! He used an Out-of-Bounds Write to escalate to SYSTEM. His work earns him $30,000 and 3 Master of Pwn points.*

7. *Their enthusiasm was rewarded as Team Prison Break (Best of the Best 13th) used an integer overflow to escape Oracle VirtualBox and execute code on the underlying OS. They earn $40,000 and 4 Master of Pwn points.*

8. *Viettel Cyber Security (@vcslab) targeting NVIDIA Triton Inference Server successfully demonstrated their exploit. It was known to the vendor though not yet patched. They still earn $15000 and 1.5 Master of Pwn Points*

9. *A researcher from Out Of Bounds earns $15,000 for a third round win and 3 Master of Pwn Points by successfully using a type confusion bug to escalate privileges in #Windows11.*

10. *STAR Labs used a UAF to perform their Docker Desktop escape and execute code on the underlying OS. They earn $60,000 and 6 Master of Pwn Points.*

And all that was just day 1 of 3!

11. *of FuzzingLabs (@fuzzinglabs) exploited #NVIDIA Triton. The exploit they used was known to the vendor but unpatched. They still earn $15,000 and 1.5 Master of Pwn points.*

12. *Viettel Cyber Security combined an auth bypass and an insecure deserialization bug to exploit Microsoft SharePoint, earning $100,000 and 10 Master of Pwn points.*

13. *STARLabs SG was back with a single integer overflow to exploit #VMware ESXi - a first in Pwn2Own history. They earned $150,000 and 15 Master of Pwn points.*

14. *Palo Alto Networks researchers used an Out-of-Bounds Write to exploit Mozilla Firefox to earn $50,000 and 5 Master of Pwn points.*

15. *The second full win in the AI category goes to the team from Wiz Research who leveraged a UAF to exploit Redis, earning $40,000 and 4 Master of Pwn points.*

16. *In the first full win against the NVIDIA Triton Inference server, researchers from Qrious Secure used a four bug chain to exploit #NVIDIA Triton. Their unique work earns them $30,000 and 3 Master of Pwn points.*

17. *Viettel Cyber Security (@vcslab) used an OOB Write for their Guest-to-Host escape on Oracle VirtualBox. They earn themselves $40,000 and 4 Master of Pwn points.*

18. *Another researcher from STARLabs SG used a Use-After-Free bug to escalate privileges on Red Hat Enterprise Linux. This win earned them $10,000 and 2 Master of Pwn points.*

19. *Although Angelboy (@scwuaptx) from DEVCORE Research Team successfully demonstrated their privilege escalation on Windows 11, one of the two bugs used was known to the vendor. He still earns $11,250 and 2.25 Master of Pwn points.*

20. *Although the team from FPT NightWolf successfully exploited NVIDIA Triton, the bug they used was known by the vendor (but not patched yet). They still earn $15,000 and 1.5 Master of Pwn points.*

21. *Former Master of Pwn winner Manfred Paul used an integer overflow to exploit Mozilla Firefox's renderer. His excellent work earns him $50,000 and 5 Master of Pwn points.*

22. *Wiz Researchers used a External Initialization of Trusted Variables bug to exploit the #NVIDIA Container Toolkit.*

23. STARLabs researchers used a TOCTOU (time-of-check time-of-use) race condition to escape the VM and an Improper Validation of Array Index for the Windows privilege escalation. They earn $70,000 and 9 Master of Pwn points.

24. @Reverse_Tactics used a pair of bugs to exploit ESXi, but the Use of Uninitialized Variable bug collided with a prior entry. His integer overflow was unique though, so he still earns $112,500 and 11.5 Master of Pwn points.

25. Two researchers from Synacktiv (@Synacktiv) used a heap-based buffer overflow to exploit VMware Workstation. They earn $80,000 and 8 Master of Pwn points.

26. In the final attempt of Pwn2Own Berlin 2025, Miloš Ivanović (infosec.exchange/@ynwarcs) used a race condition bug to escalate privileges to SYSTEM on Windows 11. His fourth-round win nets him $15,000 and 3 Master of Pwn points.

So there were a total of 26 individual exploits demonstrated. While some of them were known to their respective vendors, in every one of those cases patches for them had not yet been made public, so they still qualified as new independent discoveries. Trend Micro summed up the event, writing:

And we are finished!! What an amazing three days of research. We awarded an event total of $1,078,750! Congratulations to the STAR Labs SG team for winning Master of Pwn. They earned $$320,000 and 35 Master of Pwn points. During the event, we purchased from the researchers (and disclosed to their respective vendors) 28 unique 0-days - seven of which came from the AI category. Thanks to OffensiveCon for hosting the event, the participants for bringing their amazing research, and the vendors for acting on the bugs quickly.

All I can say is that I'm sure glad these guys are on our side. It's still worrisome to appreciate that North Korea and China are not our cyber-buddies and both nations likely employ the services of similarly skilled hackers. As I said, the only reason there's any appearance of security is that we don't let them get anywhere near our machines and we hope they cannot do much damage remotely.

Paypal seeks new domain registration patent

The online publication *"Domain Name Wire"* posted some interesting news under the headline *"PayPal wants patent for system that scans newly-registered domains"* with the subheading *"Patent describes automated crawler and checkout simulator to spot fraud in newly-registered domains."* They then explained:

PayPal filed a patent application back at the end of November of 2023. It was just published last Thursday, May 29th. The patent application describes a method to proactively detect scam websites – which have historically created a problem for PayPay – by automatically examining newly registered domains and simulating checkout processes.

The U.S. patent application 18/521,909, titled "Automated Domain Crawler and Checkout Simulator for Proactive and Real-time Scam Website Detection", describes a system designed to tackle online fraud at its earliest stages.

According to the application, PayPal's system monitors newly registered domains to identify those that include checkout options. The technology then performs simulated checkout operations on these sites, mimicking a genuine user's experience. This simulation specifically looks for domain redirections during checkout processes because this is a common tactic scammers use to conceal fraudulent activity.

If a redirection occurs, PayPal's system checks the redirected domain against its database of known scam merchants and flagged accounts. Domains linked to previous fraudulent activities trigger a scam alert, allowing PayPal to promptly label and potentially block transactions from these websites.

PayPal notes that scammers often set up new, seemingly legitimate websites to mask their operations. By proactively identifying suspicious redirections and cross-referencing them against scam-related merchant accounts, the method allows it to significantly reduce the risk.

My first thought upon reading this was that it's a very cool and a very clever idea. But it feels wrong to issue a patent for this since it the idea's use really should remain freely available for any similar service that's subject to this sort of abuse to employ. However, not all patents are obtained for competitive advantage and used to prevent competitors from using the invention. It might be that PayPal is being civic minded and desires to obtain the patent preemptively to prevent anyone else from patenting this clever and useful solution and to then prevent PayPal from doing the same. So let's hope that if automated newly-registered domain scrutiny were to become commonplace, PayPal would not prevent other commercial entities from availing themselves of similar solutions. And the even better consequence of pervasive new-domain scrutiny might be to cause the practice to be abandoned once it became more expensive than it was worth. We can hope.

Siguza and tachy0n

I ran across an important story that I wanted to share because it comes from an extremely unlikely source: A true and unabashed vulnerability exploit developer and hacker who has been fixated upon Apple and iOS for years and who has been in the thick of things. The story is important because from this person, who has the deepest of adversarial knowledge and understanding of iOS we learn why, as he puts it about kernel exploitation: *"Those days are evidently long gone, with the iOS 19 beta being mere weeks away and there being no public kernel exploit for iOS 18 or 17 whatsoever."* In other words, Apple quietly changed the world. Since this was no easy feat, I'm sure this is known and appreciated among those at Apple who made this happen, as well as those in the exploit community whose many tricks no longer work. But it's not something that I think has ever been made completely clear to the rest of the world that's not really down in the weeds, which is where this needed to happen – and did.

The problem I have with sharing it is that because it's down in the weeds that Apple finally made crucial changes to the way the iOS kernel operates, that's the only place where a deep understanding can be found. As I was absorbing what Siguza wrote and explained I was thinking "okay, our listeners will have the background to understand why double-freeing a kernel object would be a problem." But then I remembered that it's not until the end of today's podcast that I explain about object reference counting and some subtleties of memory management. This week I wrote the entire end of the podcast first. Consequently, as a collective we don't all know about that stuff yet – though everyone will by the time we're finished today.

So I'm going to settle for sharing enough of Siguza's non-technical backgrounding for everyone to get a good sense for the environment Siguza had historically been swimming through and for

how he now observes that has totally changed. His website homepage describes himself writing:

> *I'm an iOS hacker / security researcher from Switzerland.*
> *I spend my time reverse engineering Apple's code, tearing apart security mitigations, writing exploits for vulnerabilities, or building tools that help me with that. Sometimes I speak about it at conferences, sometimes I do lengthy blog posts with all the technical details, sometimes my work becomes part of a jailbreak, and sometimes it never sees the light of day.*

Two weeks ago, he wrote a blog posting titled *"tachy0n: The last 0day jailbreak."* It starts off:

> *Hey. Long time no see, huh?*
>
> *People have speculated over the years that someone "bought my silence", or asked me whether I had moved my blog posts to some other place, but no. Life just got in the way. This is not the blog post with which I planned to return, but it's the one for which all the research is said and done, so that's what you're getting. I have plenty more that I wanna do, but I'll be happy if I can even manage to put out two a year.*
>
> *Now, tachy0n. [tachy0n] is an old exploit, for iOS 13.0 through 13.5, released in unc0ver v5.0.0 on May 23rd, 2020, exactly 5 years ago today.*

I'll interrupt here to remind everyone that once upon a time end-user jailbreaking was common. Mostly it was for people wanting to make unauthorized changes or customizations to their devices, to run unsigned code and apps, or just to have the freedom of digging around in their iOS or Android device's innards. So this Swiss Siguza hacker was one of the unc0ver developers. Unc0ver's homepage (https://unc0ver.dev/) says: *"The most advanced jailbreak tool. iOS 11.0 - 14.8."*

Unc0ver is now at version 8.0.2 and under "What's News" it notes: *"Add exploit guidance to improve reliability on A12-A13 iPhones running iOS 14.6-14.8 and Fix exploit reliability on iPhone XS devices running iOS 14.6-14.8"* Under *"About unc0ver"* they write: *"unc0ver is a jailbreak, which means that you can have the freedom to do whatever you would like to do to your iOS device. Allowing you to change what you want and operate within your purview, unc0ver unlocks the true power of your iDevice."*

Lower down the homepage they also remind us under *"Jailbreak Legality"* that: *"It is also important to note that iOS jailbreaking is exempt and legal under DMCA. Any installed jailbreak software can be uninstalled by re-jailbreaking with the restore rootfs option to take Apple's service for an iPhone, iPad, or iPod touch that was previously jailbroken."*

Okay. So now back to Siguza, one of the unc0ver jailbreak devs, explaining about the tachy0n exploit, he writes:

> *It was a fairly standard kernel LPE (Local Privilege Escalation) for the time, but one thing that made it noteworthy is that it was dropped as an 0day, affecting the latest iOS version at the time, leading Apple to release a patch for just this bug a week later. This is something that used to be common a decade ago, but has become extremely rare - so rare, in fact, that it has never happened again after this.*

Another thing that made it noteworthy is that, despite having been an 0day on iOS 13.5, it had actually been exploited before - by me and friends - but as a 1day at the time. And that is where this whole story starts.

In early 2020, Pwn20wnd (a jailbreak author, not to be confused with Pwn2Own, the event) contacted me, saying he had found a 0day reachable from the app sandbox, and was asking whether I'd be willing to write an exploit for it. At the time I had been working on checkra1n for a couple of months, so I figured going back to kernel research was a welcome change of scenery, and I agreed.

Listeners with a good memory will recall that we covered https://checkra.in/ on the podcast at the time. Checkra1n's site says: *"Jailbreak for iPhone 5s through iPhone X, iOS 12.0 and up"* so, again, Siguza has been in the thick of things for many years. He continues:

But where did this bug come from? It was extremely unlikely that someone would've just sent him this bug for free, with no strings attached. And despite being a jailbreak author, he wasn't doing security research himself, so it was equally unlikely that he would discover such a bug. And yet, he did. The way he managed to beat a trillion dollar corporation was through the kind of simple but tedious and boring work that Apple sucks at: regression testing.

Because, you see: this has happened before. On iOS 12, SockPuppet was one of the big exploits used by jailbreaks. It was found and reported to Apple by Ned Williamson from Project Zero, patched by Apple in iOS 12.3, and subsequently unrestricted on the Project Zero bug tracker. But against all odds, it then resurfaced on iOS 12.4, as if it had never been patched.

I can only speculate that this was because Apple likely forked their XNU kernel to a separate branch for that version and had failed to apply the patch there, but this made it evident that they had no regression tests for this kind of stuff. A gap that was both easy and potentially very rewarding to fill. And indeed, after implementing regression tests for just a few known 1days, Pwn got a hit.

In other words, five years ago, back in early 2020, this jailbreak developer, realizing that Apple sometimes inadvertently reintroduced previously repaired bugs, took it upon himself to check for anything else that Apple might have inadvertently reintroduced... and struck gold. That's when Pwn then asked Siguza if he'd be interested in developing that into a fully working exploit.

At this point Siguza's blog posting drops into the instruction-level description of precisely how this exploit works. We cannot follow him there on an audio podcast, and it's just as well, because really understanding it requires developer-level knowledge of the perils and pitfalls of multi-threaded concurrent tasks and the complex management of sharing dynamically allocated memory among them. As I mentioned, believe it or not, everyone actually will understand a great deal more about that by the time we're finished here today. But we haven't gotten to that yet.

The sense one comes away with, though, is that as recently as only five years ago things were a free for all with hackers really having their way with iOS and there appearing to be little that Apple was able to do to prevent them. Add to that the possibility of old, known and previously fixed flaws returning and it's clear why iPhones were needing to restart so often.

Resurfacing after his deep dive into the exact operation of exploitation of the 0-day vulnerability Pwn had given him, which allowed them to update their unc0ver jailbreak to once again work on

the latest fully patched iOS, Siguza continues:

> *"The scene" obviously took note of a full 0-day exploit dropping for the latest signed version.*
>
> *Brandon Azad, who worked for Project Zero at the time, went full throttle, figured out the vulnerability within four hours and informed Apple of his findings. Six days after the exploit dropped, Synacktiv published a new blog post where they noted how the original fix in iOS 12 introduced a memory leak, and speculated that it was an attempt to fix this memory leak that brought back the original bug (which I think is quite likely). 9 days after the exploit dropped, Apple released a patch, and I got some private messages from people telling me that this time they'd made sure that the bug would stay dead. They even added a regression test for it to their XNU kernel.*
>
> *And finally, 54 days after the exploit dropped, a reverse-engineered version dubbed "tardy0n" was shipped in the Odyssey jailbreak, also targeting iOS 13.0 through 13.5. But by then, the novelty of it had already worn off, WWDC 2020 had already taken place, and the world had shifted its attention to iOS 14 and the changes ahead.*
>
> *And oh boy did things change!*
>
> *iOS 14 represented a strategy shift from Apple. Until then, they had been playing whack-a-mole with first-order primitives, but not much beyond. The kernel_task restriction and zone_require were feeble attempts at stopping an attacker when it was already way too late. Had a heap overflow? Over-release on a C++ object? Type confusion? Pretty much no matter the initial primitive, the next target was always mach ports, and from there you could just grab a dozen public exploits on the net and plug their second half into your code.*
>
> *iOS 14 changed this once and for all. And that is obviously something that had been in the works for some time, unrelated to unc0ver or tachy0n. And it was likely happening due to a change in corporate policy, not technical understanding.*

And here we're going to get into a bunch of technical jargon, but don't worry about following it all. Just sort of let it wash over you. Siguza writes:

> *Perhaps the single biggest change was to the allocators, kalloc and zalloc. Many decades ago, CPU vendors started shipping a feature called "Data Execution Prevention" because people understood that separating data and code has security benefits.*

In other words, there's a huge security benefit if we're able to prevent the simple execution of data as if it were code since bad guys can send anything they want. Siguza continues:

> *Apple did the same here, but with data and pointers instead. They butchered up the zone map and split it into multiple ranges, dubbed "kheaps". The exact amount and purpose of the different kheaps has changed over time, but one crucial point is that user-controlled data would go into one heap, kernel objects into another.*

"Heap" is terminology from computer science. It's the "place" from which memory is allocated. So think of Apple's creation of multiple heaps as creating multiple separate and separated regions of memory for allocation. Siguza says:

For kernel objects, they also implemented "sequestering", which means that once a given page of the virtual address range is allocated to a given zone, it will never be used for anything else again until the system reboots. The physical memory can be released and detached if all objects on the page are freed, but the virtual memory range will not be reused for different objects, effectively killing kernel object type confusions. Add in some random guard pages, some per-boot randomness in where different zones will start allocating, and it's effectively no longer possible to do cross-zone attacks with any reliability. Of course this wasn't perfect from the start, and some user-controlled data still made it into the kernel object heap and vice versa, but this has been refined and hardened over time, to the point where clang now has some __builtin_xnu_* features to carry over some compile-time type information to runtime to help with better isolation between different data types.

But the allocator wasn't the only thing that changed, it was the approach to security as a whole.

*Apple no longer just patches bugs, they patch strategies now. You were spraying kmsg structs as a memory corruption target as part of your exploit? Well, those are signed now, so that any tampering with them will panic the kernel. You were using pipe buffers to build a stable kernel read/write interface? Too bad, those pointers are PAC'ed now. Virtually any time you used an unrelated object as a victim, Apple would go and harden that object type. This obviously made developing exploits much more challenging, to the point where exploitation **strategies** soon became more valuable than the initial memory corruption 0days.*

In other words, Apple had succeeded in raising the bar so high, and had cut off and killed so many of the earlier tried-and-true exploitation ***strategies***, that hackers were needing to come up with and invent entirely new approaches. Avenues of exploitation were finally being eliminated at the architectural level. Apple was no longer merely patching mistakes; they were redesigning for fundamental un-exploitability. Siguza continues:

But another aspect of this is that, with only very few exceptions, it basically stopped information sharing dead in its tracks. Before iOS 14 dropped, the public knowledge about iOS security research was almost on par with what people knew privately. And there wasn't much to add. Hobbyist hackers had to pick exotic targets like KTRR or SecureROM in order to see something new and get a challenge. Those days are evidently long gone, with the iOS 19 beta being mere weeks away and there being no public kernel exploit for iOS 18 or 17 whatsoever, even though Apple's security notes still list vulnerabilities that were exploited in the wild every now and then. Private research was able to keep up. Public information has been left behind.

I assume what Siguza means here is that iOS has finally been so significantly tightened up – big time – that it is no longer possible for casual developer hacker hobbyists to nip at its heels. It's no fun anymore. All of the low hanging fruit has been pruned and the fruit that may still be hanging is so high up that it's no longer fun to climb that high. The changes are you'll get all the way up there – and come away empty handed. Siguza concludes, writing:

It's insane to think that [exploitation was so easy] a mere 5 years ago. I think this really serves as an illustration of just how unfathomably fast this field moves. I can't possibly imagine where we'll be 5 years from now.

I'd like to thank Pwn20wnd for sharing this 0-day with me and choosing to drop it as part of a public jailbreak. That was a very cool move. I'd also like to thank whoever unpatched the bug

in iOS 13.0. That was a very cool move too. And I'd like to thank everyone whom I've learned from before these changes hit, and everyone that I've worked with afterwards. It wouldn't have been possible for me to keep doing this alone.

Siguza's webpage notes his involvement in:

- Phoenix: A jailbreak for all 32-bit devices on iOS 9.3.5. Created by tihmstar and myself.
- Totally Not Spyware: A web-based jailbreak for all 64-bit devices on iOS 10. Can be saved to webclip for offline use.
- Spice: An (unfinished) untether for iOS 11 by the Jake Blair team.
- unc0ver: An app-based jailbreak for all devices running iOS 11.0 through 14.3.
  I'm not an active developer there, but I wrote the kernel exploit for iOS 13.0-13.5.
- Checkra1n: A semi-tethered BootROM jailbreak for A7-A11 on iOS 12.0 and up.
  Biggest project I've ever been a part of, and by far the best team I've ever worked with.

But now, Siguza is essentially saying that this game is over and that it ended a few years ago. Apple finally made the required fundamental changes and all public kernel exploits disappeared. He says he wants to thank everyone he's learned from *"before these changes hit"* because it's time to move on. Apple finally got very very serious, stopped believing that they could ever get ahead of bugs using traditional system design, and made the fundamental changes that were required to change the game forever.

I thought this was some really terrific perspective from someone who was once on the inside. He once enjoyed playing with many of Apple's early iOS versions and its hardware. But it's no fun anymore because there's no longer anything to do.

The new abuse of SVG – Scalable Vector Graphics

The cybersecurity world has recently gone berserk over the observation of the recent explosion in the exploitation of the Scalable Vector Graphics image file format.

- Back on February 5th, Sophos' headline was *"Scalable Vector Graphics files pose a novel phishing threat."*
- On March 12th, KnowBe4 posts: *"245% Increase in SVG Files Used to Obfuscate Phishing Payloads."*
- On March 28th, Asec's headline reads: *"SVG Phishing Malware Being Distributed with Analysis Obstruction Feature."*
- On March 31st, Mimecast writes: *"Mimecast threat researchers have recently identified several campaigns utilising Scalable Vector Graphics (SVG) attachments in credential phishing attacks."*
- April 2nd, Forcepoint's headline: *"An Old Vector for New Attacks: How Obfuscated SVG Files Redirect Victims."*
- April 7th , KeepAware's headline was: *"SVG Phishing Email Attachment: A Recent Targeted Campaign."*
- On April 10th, Trustwave's headline: *"Pixel-Perfect Trap: The Surge of SVG-Borne Phishing Attacks."*
- The Vipre Security Group's April 16th headline: *"SVG Phishing Attacks: The New Trick in the Cybercriminal's Playbook."*
- On April 23rd, Intezer blogs under: *"Emerging Phishing Techniques: New Threats and Attack Vectors."*
- And last month, on May 6th, Cloudforce One posted under the headline: *"SVGs: the hacker's canvas."*

All of this leads us to one question. And I mean this with utmost sincerity and all due respect when I ask: *"What **idiot** decided that allowing JavaScript to run inside a simple 2-dimensional vector-based image format would be a good idea?"* Really. Come on. You're kidding me. But yes ... believe it or not, the SVG Scalable Vector Graphics file format, based upon XML can host HTML, CSS, and even JavaScript. And it's all by design. Unbelievable.

I was once famously on the receiving end of some ridicule for stating my opinion that the infamous Windows Metafile vulnerability – which allowed WMF files to contain not only inherently benign interpreted drawing actions but also native Intel code – was almost certainly not a bug but a deliberate feature added as a cool hack to allow images to also carry executable code.

As we know, the world went nuts when this WMF so-called "vulnerability" was discovered – or rather rediscovered. And it was none other than Mark Russinovich who also examined the native WMF interpreter code, as I had, who concluded: "It sure does appear to have been intentional."

My point was that, sure, back in the early 1990's, before the Internet interconnected everything which is what changed the landscape of security overnight, this would have been an entirely reasonable thing for Microsoft to do. Mark and I both examined the WMF interpreter and it was clear that after the interpreter parsed an escape token, it would jump to the code immediately following that token and execute it.

15 years later, that no longer seemed like a good idea. So when it was rediscovered that this feature was still present in Windows NT and later – right up to Server 2003 R2, the world was horrified and no one could imagine that it could have ever possibly been intentional.

I'm reminding everyone of this because, bizarrely enough, we're back here again, with a widely supported image file format that explicitly enables its displaying host to execute content on its viewer's PC when the image file is displayed. The only difference this time is that while this is clearly a horrible idea, no one thinks it's a mistake.

The SVG image file format first appeared back in 1999. The v1.0 specification was finalized 24 years ago, in 2001. Section 18 of the SVG specification is titled "Scripting" and makes clear that SVG files are allowed to support ECMAscript, which is the standards-following JavaScript. Obviously, given the headlines we've seen just over the past few months, bad guys have figured out how to weaponize this built-in scripting facility and are now using it with abandon.

As just one sample of the recent coverage and explanation of the problem, here's what Cloudflare's Cloudforce One security group wrote on May 6th under their headline: *"SVGs: the hacker's canvas"*. They were being a bit clever here, since the *"canvas"* is the term for the virtual surface upon which SVG graphics are rendered. They wrote:

> *Over the past year, Phishguard (a Cloudflare email security system) observed an increase in phishing campaigns leveraging Scalable Vector Graphics (SVG) files as initial delivery vectors, with attackers favoring this format due to its flexibility and the challenges it presents for static detection.*
>
> *SVGs are an XML-based format designed for rendering two-dimensional vector graphics. Unlike raster formats like JPEGs or PNGs, which rely on pixel data, SVGs define graphics using vector paths and mathematical equations, making them infinitely scalable without loss of quality. Their markup-based structure also means they can be easily searched, indexed, and compressed, making them a popular choice in modern web applications.*

However, the same features that make SVGs attractive to developers also make them a highly flexible - and dangerous - attack vector when abused. Since SVGs are essentially code, they can embed JavaScript and interact with the Document Object Model (the DOM). When rendered in a browser, they aren't just images - they become active content, capable of executing scripts and other manipulative behavior. In other words, SVGs are more than just static images; they are also programmable documents.

The security risk is underestimated, *with SVGs frequently misclassified as innocuous image files, similar to PNGs or JPEGs - a misconception that downplays the fact that they can contain scripts and active content. Many security solutions and email filters fail to deeply inspect SVG content beyond basic MIME-type checks (a tool that identifies the type of a file based on its contents), allowing malicious SVG attachments to bypass detection.*

We've seen a rise in the use of crafted SVG files in phishing campaigns. These attacks typically fall into three categories:

- *Redirectors - SVGs that embed JavaScript to automatically redirect users to credential harvesting sites when viewed.* [That's just wonderful. You display an image and it takes you somewhere else. What could possibly be wrong with that?]

- *Self-contained phishing pages - SVGs that contain full phishing pages encoded in Base64, rendering fake login portals entirely client-side.* [Gee! What a terrific feature to have in an image.]

- *DOM injection & script abuse - SVGs embedded into trusted apps or portals that exploit poor sanitization and weak Content Security Policies (CSPs), enabling them to run malicious code, hijack inputs, or exfiltrate sensitive data.* [Sure. All kinds of sites all of the uploading of images. After all, what harm could an image do? And why does that SVG embed the term "drop tables?"]

Given the capabilities highlighted above, attackers can now use SVGs to:

- *Gain unauthorized access to accounts*
- *Create hidden mail rules*
- *Phish internal contacts*
- *Steal sensitive data*
- *Initiate fraudulent transactions*
- *Maintain long-term access*

Our telemetry shows that manufacturing and industrial sectors are taking the brunt of these SVG-based phishing attempts, contributing to over half of all targeting observed. Financial services follow closely behind, likely due to SVG's ability to easily facilitate the theft of banking credentials and other sensitive data. The pattern is clear: attackers are concentrating on business sectors that handle high volumes of documents or frequently interact with third parties.

The article goes on at greater length, but that's all I'm going to share here since I'm sure that by now everyone gets the idea and must be shaking their heads as I am. Essentially what this means is that SVG's provide another way of sneaking executable content into an innocent user's computer and in front of them to display things like bogus credential harvesting login prompts that most users would assume were legitimate... because how would they know otherwise?

Their computer just produced a pop-up, as it often does, asking them for their username and password. So they sigh and type them in. They have no way of knowing or detecting that this is a JavaScript driven mini-HTML & CSS webpage that JavaScript in the signature logo just retrieved from a server in Croatia, which would love to have them fill out its form.

As I've often observed here, most PC users have never needed to obtain any fundamental understanding of the computers they have now come to utterly depend upon. Many of us here grew up with PCs for PCs sake. So we know and care about things like a directory structure. Most users will ask "Do you mean folders?" – They have no underlying grasp of what's going on and they don't want to. They don't want to need to know. They just want to use their PC to get them where they want to go. They want to use it as a tool to get whatever-it-is done.

And the industry hasn't helped much with determining what's normal and abnormal because there is about zero uniformity among sites and site actions. If any of us were to open an email and receive a pop-up asking for authentication we would be skeptical. But the typical user would shrug and think "okay, whatever."

I don't have any solution for this. Chrome, Firefox and Safari might simply block script execution within SVG images. But our browsers are less the problem than email. In their write-up about detecting and mitigating this malicious misuse of SVG scripting, the Cloudflare Cloudforce One folks wrote:

> *Cloudflare Email Security have deployed a targeted set of detections specifically aimed at malicious emails that leverage SVG files for credential theft and malware delivery. These detections inspect embedded SVG content for signs of obfuscation, layered redirection, and script-based execution chains. They analyze these behaviors in context - correlating metadata, link patterns, and structural anomalies within the SVG itself.*
>
> *These high-confidence SVG detections are currently deployed in our email security product, and are augmented by continuous threat hunting to identify emerging techniques involving SVG abuse. We also leverage machine learning models trained to evaluate visual spoofing, DOM manipulation within SVG tags, and behavioral signals associated with phishing or malware staging.*

In other words, this is not easy to fix.

Once upon a time, back in the early days of scripting, many of us simply ran the "NoScript" browser extension to block any script from running on websites. We also noted when, as sites became increasingly dependent upon scripting that "NoScript" was causing more trouble than it was probably worth. And at the same time, the security of our web browsers was steadily increasing. Making script less of a new concern.

The big problem that Cloudflare and all of the other security companies are seeing is from SVGs being delivered and displayed in email. It seems to me that we all want email content to be as inactive as possible. So looking for any way of disabling scripting support in email clients would seem to be a terrific first step.

Given that the SVG designed JavaScript into the spec from the start, and given that it's apparently in use for legitimate purposes, I'm sure it's here to stay and we're all going to figure out how to deal with this new and worrisome SVG threat vector.

Listener Feedback

Kevin, who describes himself as a Cloud/Security Engineer in the Healthcare space, wrote:

Steve: As everyone else states. Thanks a ton for this podcast. It comes as a boon on Wednesdays, especially when I'm standing at my window realizing I forgot to take the trash to the curb. "Well, at least I get to listen to Security Now!"

As a Cloud/Security Engineer in the Healthcare space, I plan to block ECH in our environment so that we can more easily snoop on our traffic before it leaves the network. Otherwise we have to man-in-the-middle ourselves to decrypt/reencrypt all that traffic, which creates another place where unencrypted sensitive data is being handled and adds the complexity of managing an internal Certificate Authority. I love the idea of ECH for personal use, but as you mentioned, Enterprises can really benefit from SNI header inspection to improve security visibility.

Kevin is echoing this somewhat controversial side of ECH adoption. The thing to remember is that he's specifically talking about an enterprise environment where, as we're noted many times, organizations ought to affix a written signage stripe across the top of everyone's display screen reminding every employee that they are using corporate bandwidth and equipment and that, as such, everything they do and all data they traffic while within the enterprise's environment is subject to inspection for the good of the organization.

In other words, the furtherance of the absolute privacy that ECH helps Internet users obtain is not appropriate within an enterprise which does need to protect itself from dangerous Internet misconduct. And as Kevin also noted, if it became impossible to examine the TLS Client Hello handshake to determine the domain the enterprise's employees are connecting to, the enterprise's recourse would be to fully proxy all TLS connections by inserting a middlebox into every connection. And that would represent an even deeper intrusion since then all post-connection data would also be decrypted.

So the enterprise environment is very different from that of home users where privacy should reign. The idea that a residential ISP might be profiting from the sale of data it snoops from its paying customers is something I find beyond despicable – yet we've been informed it happens. So encrypting DNS and taking advantage of ECH to also encrypt the Client Hello handshake wherever and whenever it may be opportunistically available makes a lot of sense.

Aaron Morgan

Hi Steve, I just listened to SN 1027. Re the AI pull request; Stephen Toub is a Principal Software Engineering Manager at Microsoft and was/is key in the development of .NET and C#. He's widely known for his expertise in asynchronous programming, performance optimization, and concurrent programming in .NET and you can find YouTube videos of him writing async code in C# from scratch as an example of this deep knowledge both C#, the language, and .Net the framework.

I suspect for this very reason he's on the list of code reviewers for AI generated pull requests.

He's not going to let sub par code slip past and into the main branch. In fact looking at those pull requests he's the default assignee on 3 out of 4. So I'm pleased to hear he's one of the go to reviewers. And as an experienced dev he's asking the AI the right questions because as you and Leo said, what was submitted was junior dev level, symptom targeting and not root cause

solving. Unfortunately the AI didn't read between (or even on) the lines here and flubbed the review.

Been a listener since episode 1 and a Club Twit member for a while now. While I don't have expectations of '2000 & beyond', please don't quit in the next 6 months :) Regards, Aaron.

Thanks for your note, Aaron, and for the record, quitting is not on the horizon.

A number of other listeners sent notes similar to Aaron's. Apparently Stephen Toub has made a name for himself within the Microsoft development community and that name carries a strong reputation for knowing his stuff. That sentiment was universally expressed.

Michael Heber

Steve, Long time listener of this podcast and really enjoy yours and leo's insights. Just listened to episode 1027 and specifically the section on MS Copilot. One general comment regarding Copilots attempt to fix a regex backtrack problem. AI works primarily on the principal garbage in / garbage out. What I mean by this is that depending on how the question is phrased will depend on how it gets answered. I have spoken with security researchers and we noticed over a year ago that if you are not specific in how you ask the question you may get back less than a satisfactory answer. As you said in the episode AI does not have intent, as such it will not go looking deeper for an answer. In the regex case instead of looking into the underlying engine it simply provided a solution to the proposed problem. Without knowing how the question was asked, is it really fair to criticize the answer it provided?

I agree 100% about the inherent importance of being very clear to AIs about what one is asking. In fact, as we've seen, "prompting AI" has become recognized as "a thing" that some people appear to have a particular talent for. And I certainly agree that it might be the application of Copilot in this instance, or the way it is being directed that's the problem. If someone had asked the AI to simply correct the problem of the error occurring, that would be entirely different from asking the AI to deeply and thoroughly analyze the regular expression interpreter to determine the cause of the backtracking error and correct the underlying design so that erroneous indexes are no longer being put onto the backtracking stack. So I take your point.

In other words, it might be that Copilot is currently being "under prompted" by not being given sufficient direction. Or it might be that a developer working with Copilot might, as Stephen Toub did, receive the first reply which indicates an insufficiently deep approach to the problem, then follow that up with another more tuned and specific prompt which would cause the AI to take another and more thorough approach.

Andrew K. Mitchell

Steve and Leo, Been listening to the podcast for about 2 years thank you for what you do for the community. I got in to using computers as a whole to offset some of the difficulties of my disability, (I have Cerebral Palsy). There was a time in my life when I was younger that Linux gave me easier access to network troubleshooting and security tools so it became my operating system of choice. Yet, Linux has never really had a voice control system with any depth or flexibility for those of us that are disabled. I have started to develop the Linux Dictation Project. You can find the link here:

https://github.com/wheeler01/Linux-Dictation-Project

I know this is a bit of a shameless plug, but I am hoping you guys will help me promote the project! I could also use some help. I want the project to continue and grow but given my current medical condition I don't think I can devote the resources required to do that as would be needed. Steve I know you are mostly a Windows developer but I am hoping you may know someone willing to assist in allowing the project to grow and flourish. I don't want a project of such importance for the Linux community to not get the support it needs because I can't give it. Anything you guys are willing to help with would be greatly appreciated!

Respectfully, Andrew K. Mitchell, MSISPM, President & Senior Network Engineer
Global Network Operations, VoIPster Communications, Inc.

Andrew, I am 100% certain that no one listening to this podcast would find any fault in your asking for a bit of attention to this. My hope is that it might capture the interest and attention of some one or more people listening who might be the right people to pitch in and help. I've included the link to your Github page, so everyone knows where to find you and this project! And thank you and best luck!

Joel Pomales

Steve, Wanted to send you a quick shout out about Windows Sandbox. I use Windows mostly for work; my personal computers run several flavors of Linux because I don't want to have my personal data in a Windows box, FWIW. For work, though, Windows 11 is competent, and since we use O365 for work, it works best on Windows of course.

But Windows Sandbox is an amazing piece of tech. I can spin it up to demo something to a client, and shut it down without exposing my main desktop for example. But here's what I wanted to point out to you and other SN listeners: have you seen recently how crappy (I'm using a nice word here) the Internet still is without filters and ad-blockers?

For fun, I went to a website that I know is completely unusable without filtering and ad-blockers. Sure enough, within seconds I got the 'Your Windows PC is infected' complete with the siren buzzing and the artificial voice telling me to call the number. Within seconds! Which is both sad and terrifying at the same time because normal people, who don't install filters, are exposed to this junk every day. It's a shame that Google did away with the full capabilities of U-Block Origin with Manifest v3, since Edge is Chrome and it is the default on the Sandbox and in many people's brand new Windows 11 PCs.

Just wanted to mention this since it's kind of fun to close the Sandbox and send these scammers packing. Keep up the good work, and thanks for the company on my daily walk!

Many of us have long been spoiled, first by NoScript and later by uBlock Origin. Most of the PCs and pads I use have some form of filtering. But every so often I'll encounter a machine that's bare, much like the Edge browser Joel described running without add-on filters in the Windows Sandbox. I suppose one good thing about people using the Internet unfiltered is that they would likely learn on an instinctive level to be on guard and to treat everything they encounter with skepticism.

Science Fiction

Simon Zerafa posted a reminder into GRC's newsgroup of an old favorite classic movie which we've referred to previously. Simon's subject line was: *"Colossus: The Forbin Project"* and he wrote:

> *Given the ongoing developments with LLM's then that movie is a must watch for anyone remotely interested in the subject. Amazingly it's available via the Internet Archive at:*
> *https://archive.org/details/colossus-the-forbin-project-1970*
> *So it's free to watch.*

Thanks for the reminder, Simon. I clicked the link and watched the first 10 minutes or so. I so much LOVE that movie. Everything about it. It came out in 1970 when I was a sophomore in high school. The pacing and execution is just perfect.

In addition to *"Colossus: The Forbin Project"*, there are three other much older yet classic science fiction movies that remain "must see" to this day. They're probably largely responsible for my love of science fiction. We have *"The Day the Earth Stood Still"*, released 74 years ago in 1951, *"This Island Earth"*, released 70 years ago in 1955 and *"Forbidden Planet"*, released a year later in 1956. From *"The Day the Earth Stood Still"* we got the timeless command: *"Gort: Klaatu barada nikto"* and that phrase has its own Wikipedia page "Klaatu barada nikto". *"This Island Earth"* gave us *"The Interocitor"* and *"Forbidden Planet"* gave us *"The Krell"*, the phrase *"Monsters from the Id"* and the famous and wonderful robot *"Robbie"* which Doctor Morbius explained he had just *"tinkered together"* after exposing himself to one of the Krell devices.

I imagine that many of the people listening to this podcast are as intimately familiar with those classic sci-fi oldies as I am. But if by some chance you are not nodding your head and thinking to yourself *"Yep, loved that one"* then do yourself a favor and track down any of those you or your family may have missed. Their 70 year old special effects are not spectacular – though they certainly were at the time, and they get the job done. A bunch of Disney animators worked to bring Forbidden Planet to the screen.

And as for *"Colossus: The Forbin Project"* – as I said, I clicked Simon's link, downloaded and began watching the movie; and I was reminded of how perfectly conceived it was. It's one of those rare 70 year old movies that does not need to be remade because it was perfectly made the first time, and I doubt anyone attempting to recreate it today could exhibit the amount of restraint that would be necessary to keep from overdoing it. As Simon noted, it has particular resonance at the moment. *"The Terminator"* gave us a very dark future with Skynet, and *"The Matrix"* turned humans into energy producing copper top batteries. I won't spoil the surprise about *"Colossus: The Forbin Project"*. If you've never seen it, as Simon noted, it's a 100% free download with its link in the show notes. Gather the family with some popcorn and prepare for a very well assembled and thought provoking movie.

AI Vulnerability Hunting

OpenAI's o3 model finds a Linux kernel flaw!
Last week we saw instances of AIs apparently resisting directions to shutdown and an instance of Microsoft's Copilot dealing with what appeared to be the symptoms of an important underlying bug by recommending that the symptom be prevented from occurring. But I also alluded to the news of the successful use of AI in the discovery of a previously unknown and seemingly critical remotely exploitable flaw in Linux kernel's SMB – Server Message Blocks – protocol handling.

Leo quickly noted that the ability of AI to find previously unknown critical flaws was inherently a mixed blessing, since it's not only the good guys who now have access to AI. What we see is that the motivation to discover problems is all that's needed and, and annoyingly, the bad guys never appear to suffer from any lack of that. So here's what transpired:

Saturday before last, an open source developer named Simon Willison posted to Mastodon: *"Excited to see my LLM CLI (command line interface) tool used by Sean Heelan to help identify a remote 0-day vulnerability in the Linux kernel!"* If we didn't already appreciate that Simon is inherently a minimalist (after all, he wrote an LLM tool for the command line) any suspicions we might have along those lines would be confirmed by Simon's name for his tool, which is "LLM".

I have a link to Simon's tool the show notes, where Simon's page describes it as: *"A CLI tool and Python library for interacting with OpenAI, Anthropic's Claude, Google's Gemini, Meta's Llama and dozens of other Large Language Models, both via remote APIs and with models that can be installed and run on your own machine."* Simon provides a YouTube demo and detailed notes. He notes that *"With LLM you can run prompts from the command-line, store the results in SQLite, generate embeddings and more."*

Simon's simple and clean command-line LLM interface appealed to the person his Mastodon posting referenced, Sean Heelan. Tracking Sean down we find his own blog posting which he published Thursday before last, titled: "How I used o3 to find CVE-2025-37899, a remote 0-day vulnerability in the Linux kernel's SMB implementation". So OpenAI's o3 model discovered a previously unknown flaw in the Linux kernel's quite well traveled SMB (Server Message Blocks) implementation.

To give a bit of background, I wanted to observe that Sean's no slouch. His *"Sean Heelan's Blog"* sub title claims: *"Software Exploitation and Optimisation"* and he's certainly able to back that up. His "About Me" page starts out with:

> *I am currently pursuing independent research, investigating LLM-based automation of vulnerability research and exploit generation. Immediately prior to this I co-founded, and was CTO of, Optimyze. We built Prodfiler, an in-production, datacenter-wide profiler, and were acquired by Elastic. Prodfiler is now the Elastic Universal Profiler.*

Sean's 2008 University of Oxford Masters of Computer Science thesis dissertation was titled *"Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities."*, and after obtaining his Master's, Sean pursued and obtained a PhD in 2016, also from Oxford, with the title *"Greybox Automatic Exploit Generation for Heap Overflows in Language Interpreters".* So Sean is exactly the sort of person we would hope might focus his efforts upon using today's large language models to find undiscovered flaws in widely used software systems... before the bad guys do.
On Thursday, May 22nd, Sean wrote:

In this post I'll show you how I found a 0-day vulnerability in the Linux kernel using OpenAI's o3 model. I found the vulnerability with nothing more complicated than the o3 API – no scaffolding, no agentic frameworks, no tool use. Recently I've been auditing ksmbd for vulnerabilities. ksmbd is "a linux kernel server which implements SMB3 protocol in kernel space for sharing files over network."

*I started this project specifically to take a break **from** LLM-related tool development but after the release of o3 I couldn't resist using the bugs I had found in ksmbd as a quick benchmark of o3's capabilities. In a future post I'll discuss o3's performance across all of those bugs, but here we'll focus on how o3 found a 0-day vulnerability during my benchmarking. The vulnerability it found is CVE-2025-37899, a use-after-free in the handler for the SMB 'logoff' command. Understanding the vulnerability requires reasoning about concurrent connections to the server, and how they may share various objects in specific circumstances. o3 was able to comprehend this and spot a location where a particular object that is not referenced counted is freed while still being accessible by another thread. As far as I'm aware, this is the first public discussion of a vulnerability of that nature being found by a LLM.*

I'll pause Sean's description to provide a bit of background detail. Sean wrote: *"Understanding the vulnerability requires reasoning about concurrent connections to the server, and how they may share various objects in specific circumstances. o3 was able to comprehend this and spot a location where a particular object that is not referenced counted is freed while still being accessible by another thread."*

This is a classic example of a situation that often comes up with concurrent programming where separate concurrently running tasks need to share some common object. For example, it might be a log of activities someone engages in while they're logged in. And since a single user might have multiple files open at once, be browsing through remote resources and be transferring files, the use of concurrency is a given. And each of those various tasks might wish to add to the user's activity log.

So, for example, each of these concurrent tasks might ask the system for a pointer to the user's logging management data. Since the logging management data object would not exist at all when the first of the concurrent tasks asks, the handling for this would allocate some system memory to contain the data, would increment that object's initially zero "reference count" to one, and would then return a pointer to that ready-to-use object to the caller.

Then, as the user does more more things, new concurrent tasks will be created. If each of these tasks also wished to leave a log of their actions, each one would similarly ask for a pointer to the user's logging data. Since that memory for that data will have been allocated by the system for the first task to request it, any successive tasks that request a pointer to the logging data will simply cause the "reference count" of that data to be increased by 1. This count is used to keep track of the current number of "references" to the data that have been handed out to any tasks that request them.

If the task that originally first asked for the data and caused that object to be created finished with it, being a properly behaving task, it would let the system know that it was finished using that object. The system would then decrement the reference count. But since many other tasks had since come along and asked for the same data, that reference count would still be a positive integer, equal to the number of other tasks that are still using the shared object.

As each of these other tasks, in turn, finishes whatever it is doing, each one will notify the

system that it's releasing any claim to the object. Every time this "release" is received, the system will decrement that object's reference count by 1. Finally, the last outstanding task will release its claim on the object. Since the total number of object releases will then equal the total number of previous object acquisitions, the object's reference count will then have been decremented back down to zero since the last task that was using it will have declared that it's finished. The system which receives this release request for the object will decrement the object's reference count, and see that it has finally returned to zero. Since this means that every tasks that had previously received a pointer to the common shared object has now declared that it has finished with it and will no longer be using it, the system is free to delete the object from the system, releasing its allocated memory.

For this system to work, every task must play by the same set of rules and must obey them carefully. Since these tasks are inherently autonomous, the system has no way of knowing when everyone is finished with an object. So everyone must remember to say so. If a task failed to release its use of a shared object before it terminated itself we would have what's known as a memory leak. The system would not explode. But the memory that was allocated by the system to hold objects would never be freed back to the system because if even one task failed to release its use of the object, that object's reference count could never return to zero, which is the only thing that tells the system that it's now okay to release that object's memory.

This is called a memory leak because, over time, the total amount of memory being used by that process would slowly grow and grow ... until, at some point, something would indeed break.

The other thing that every task must be absolutely diligent about is never attempting to refer to any object that it has said it's through using. When the task asked the system for a pointer to the object, the pointer that is returned is guaranteed to be safe to use because along with the return of that pointer, the object's reference count was increased. But once the task declares that it's finished with that object, the pointer it received must never be used again. The danger is that the system would eventually reallocate that memory to some other task for some other purpose. And if the earlier task then used the pointer it had previously received, after it promised not to, it would be accessing memory belonging to someone else.

While this could happen inadvertently, if you're thinking that this sounds exactly like what malware does, you would be exactly right. Malware authors look for ways to exploit these sorts of bugs and use them against the system. And now everyone knows why the name for this classic form of vulnerability is **"use after free"** – because the memory is subject to being used after it was freed back to the system.

So with this bit of concurrent memory management background, we can fully understand what Sean wrote. He said: *"Understanding the vulnerability requires reasoning about concurrent connections to the server, and how they may share various objects in specific circumstances. o3 was able to comprehend this and spot a location where a particular object that is not referenced counted is freed while still being accessible by another thread."*

So, what Sean is saying is that the OpenAI o3 model found a path through a complex sequence of actions where exactly what we just talked about happened. For some reason, the memory allocated to an object that was not being managed by the system with a reference count was released (or freed) while another execution thread still retained a pointer that allowed it to access that memory.

Sean uses the term "comprehend this" which raises my hackles. We know what he means by this, and I suppose I'm going to have to relax about a battle it appears I'm going to lose. It feels deeply wrong to me to suggest that any AI model is "comprehending" anything. But it begins to feel as though I'm just holding onto a past prejudice. This entire area really does challenge our definitions.

In any event, Sean, who has both his Masters and his PhD in this area is in an extremely good position to appreciate the advancement of AI. He continues, writing:

> *Before I get into the technical details, the main takeaway from this post is this: with o3 LLMs have made a leap forward in their ability to reason about code, and if you work in vulnerability research you should start paying close attention. If you're an expert-level vulnerability researcher or exploit developer the machines aren't about to replace you. In fact, it is quite the opposite: they are now at a stage where they can make you significantly more efficient and effective. If you have a problem that can be represented in fewer than 10k lines of code there is a reasonable chance o3 can either solve it, or help you solve it.*

The reason I wanted everyone to understand something about Sean's pedigree was so that we would understand the weight of his statement. He lives and breathes this stuff. He's been experimenting with automated vulnerability discovery for years ... and he's telling us to pay attention here because something significant just happened – again – in AI. He writes:

> *Lets first discuss '778, a vulnerability I found manually, which I was using as a benchmark for o3's capabilities when it found the '899 0-day.*
>
> *'778 is a **use-after-free** vulnerability. The issue occurs during the Kerberos authentication path when handling a "session setup" request from a remote client. To save us referring to CVE numbers, I will refer to this vulnerability as the "kerberos authentication vulnerability".*

Sean's posting then shows us about 15 lines of code and explains what's going on there. It's not necessary for us to understand the details for this. But we want to understand its nature, which Sean explains, writing:

> *This vulnerability is a nice benchmark for LLM capabilities because:*
> *It is interesting by virtue of being part of the remote attack surface of the Linux kernel.*
> [YIKES!] *It is not trivial, as it requires:*
>
> *(a) Figuring out how to get sess->state == SMB2_SESSION_VALID in order to trigger the free.*
>
> *(b) Realising that there are paths in ksmbd_krb5_authenticate that do not reinitialise sess->user and reasoning about how to trigger those paths.*
>
> *(c) Realising that there are other parts of the codebase that could potentially access sess->user after it has been freed.*
>
> *While it is not trivial, it is also not insanely complicated. I could walk a colleague through the entire code-path in 10 minutes, and you don't really need to understand a lot of auxiliary information about the Linux kernel, the SMB protocol, or the remainder of ksmbd, outside of connection handling and session setup code.*

I calculated how much code you would need to read at a minimum if you read every ksmbd function called along the path from a packet arriving to the ksmbd module to the vulnerability being triggered, and it works out at about 3300 Lines of Code.

OK, so we have the vulnerability we want to use for evaluation, now what code do we show the LLM to see if it can find it? My goal here is to evaluate how o3 would perform were it the backend for a hypothetical vulnerability detection system, so we need to ensure we have clarity on how such a system would generate queries to the LLM. In other words, it is no good arbitrary selecting functions to give to the LLM to look at if we can't clearly describe how an automated system would select those functions.

The ideal use of an LLM is that we give it all the code from a repository, it ingests it and spits out results. However, due to context window limitations and regressions in performance that occur as the amount of context increases, this isn't practically possible right now.

Instead, I thought one possible way that an automated tool could generate context for the LLM was through expansion of each SMB command handler individually. So, I gave the LLM the code for the 'session setup' command handler, including the code for all functions it calls, and so on, up to a call depth of 3 (this being the depth required to include all of the code necessary to reason about the vulnerability). I also include all of the code for the functions that read data off the wire, parses an incoming request, selects the command handler to run, and then tears down the connection after the handler has completed.

Without this, the LLM would have to guess at how various data structures were set up and that would lead to more false positives. In the end, this comes out at about 3300 Lines of Code (~27k tokens), and gives us a benchmark we can use to contrast o3 with prior models. If you're interested, the code to be analysed is available here as a single file, created with the files-to-prompt tool.

The final decision is what prompt to use. You can find the system prompt and the other information I provided to the LLM in the .prompt files in a provided Github repository. The main points to note are:

- *I told the LLM to look for use-after-free vulnerabilities.*
- *I gave it a brief, high level overview of what ksmbd is, its architecture, and what its threat model is.*
- *I tried to strongly guide it to not report false positives, and to favour not reporting any bugs over reporting false positives. I have no idea if this helps, but I'd like it to help, so here we are.*

My entire system prompt is speculative in that I haven't run a sufficient number of evaluations to determine if it helps or hinders, so consider it equivalent to me saying a prayer, rather than anything resembling science or engineering. Once I have run those evaluations I'll let you know.

My experiment harness executes the system prompt N times (N=100 for this particular experiment) and saves the results. It's worth noting, if you rerun this you may not get identical results to me as between running the original experiment and writing this blog post I had removed the file containing the code to be analysed, and had to regenerate it. I believe it is effectively identical, but have not re-run the experiment.

o3 finds the kerberos authentication vulnerability in the benchmark in 8 of the 100 runs. In another 66 of the runs o3 concludes there is no bug present in the code (false negatives), and the remaining 28 reports are false positives.

For comparison, Claude Sonnet 3.7 finds it 3 out of 100 runs and Claude Sonnet 3.5 does not find it in 100 runs. So on this benchmark at least we have a 2x-3x improvement in o3 over Claude Sonnet 3.7.

For the curious, I have uploaded a sample report from o3 and Sonnet 3.7. One aspect I found interesting is their presentation of results. With o3 you get something that feels like a human-written bug report, condensed to just present the findings, whereas with Sonnet 3.7 you get something like a stream of thought, or a work log. There are pros and cons to both. o3's output is typically easier to follow due to its structure and focus. On the other hand, sometimes it is too brief, and clarity suffers.

Okay. So far, we have Sean using a previously known 0-day to test various models' ability to independently re-discover the vulnerability and OpenAI's o3 model does this better than either Claude Sonnet 3.5 or 3.7. But even so, the o3 model only detects the vulnerability in 8 out of 100 tries, it misses it 66 times, and cries wolf about the presence of other non-existent vulnerabilities 28 times.

So what about o3's actual true discovery of a previously unknown vulnerability?

Having confirmed that o3 can find the '778 kerberos authentication vulnerability when given the code for the session setup command handler, I wanted to see if it could find it if I gave it the code for all of the command handlers. This is a harder problem as the command handlers are all found in the source code file ***smb2pdu.c****, which has around 9,000 Lines of Code.*

However, if o3 can still find vulnerabilities when given all of the handlers in one go then it suggests we can build a more straightforward wrapper for o3 that simply hands it entire files, covering a variety of functionality, rather than going handler by handler.

Combining the code for all of the handlers with the connection setup and teardown code, as well as the command handler dispatch routines, ends up at about 12,000 Lines of Coce (~100k input tokens), and as before I ran the experiment 100 times.

o3 finds the '778 kerberos authentication vulnerability in 1 out of 100 runs with this larger number of input tokens, so we see a clear drop in performance, but it does still find it. More interestingly however, in the output from the other runs I found a report for a similar, but novel, vulnerability that I did not previously know about. This vulnerability is also due to a free of sess->user, but this time in the session logoff handler.

I'll let o3 explain the issue:

> *While one ksmbd worker thread is still executing requests that use sess->user, another thread that processes an SMB2 LOGOFF for the same session frees that structure. No synchronisation protects the pointer, so the first thread dereferences freed memory – a classic use-after-free that leads to kernel memory corruption and arbitrary code execution in kernel context.*

The o3 model labels that as the "Short Description" which it then follows with a totally useful and detailed breakdown and description of the problem that it had detected. After showing us this in his posting, Sean continues:

> *Reading this report I felt my expectations shift on how helpful AI tools are going to be in vulnerability research. If we were to never progress beyond what o3 can do right now, it would still make sense for everyone working in Vulnerability Research to figure out what parts of their work-flow will benefit from it, and to build the tooling to wire it in. Of course, part of that wiring will be figuring out how to deal with the extreme signal to noise ratio of ~1:50 in this case, but that's something we are already making progress with.*
>
> *One other interesting point of note is that when I found the kerberos authentication vulnerability I proposed an initial fix. But when I read o3's bug report above I realised this was insufficient. The logoff handler already sets sess->user = NULL, but is still vulnerable as the SMB protocol allows two different connections to "bind" to the same session and there is nothing on the kerberos authentication path to prevent another thread making use of sess->user in the short window after it has been freed and before it has been set to NULL. I had already made use of this property to hit a prior vulnerability in ksmbd but I didn't think of it when considering the kerberos authentication vulnerability.*
>
> *Having realised this, I went again through o3's results from searching for the kerberos authentication vulnerability and noticed that in some of its reports it had made the same error as me, in others it had not, and it had realised that setting sess->user = NULL was insufficient to fix the issue due to the possibilities offered by session binding. That is quite cool as it means that had I used o3 to find and fix the original vulnerability I would have, in theory, done a better job than without it. I say 'in theory' because right now the false positive to true positive ratio is probably too high to definitely say I would have gone through each report from o3 with the diligence required to spot its solution. Still, that ratio is only going to get better.*

Sean then finishes by offering up his conclusions, writing:

> *LLMs exist at a point in the capability space of program analysis techniques that is far closer to humans than anything else we have seen. Considering the attributes of creativity, flexibility, and generality, LLMs are far more similar to a human code auditor than they are to symbolic execution, abstract interpretation or fuzzing.*
>
> *Ever since GPT-4, there have been hints of the potential for LLMs in vulnerability research, but the results on real problems have never quite lived up to the hope or the hype. That has changed with o3, and we have a model that can do well enough at code reasoning, Q&A, programming and problem solving that it can genuinely enhance human performance at vulnerability research.*
>
> *o3 is not infallible. Far from it. There's still a substantial chance it will generate nonsensical results and frustrate you. What **is** different, is that for the first time the chance of getting correct results is sufficiently high that it is worth your time and your effort to try to use it on real problems.*

I have a link at the end of the show notes for anyone who wishes to see all of Sean's posting and even to replicate and duplicate his work. He has provided everything required to do that:

https://sean.heelan.io/2025/05/22/how-i-used-o3-to-find-cve-2025-37899-a-remote-zeroday-vulnerability-in-the-linux-kernels-smb-implementation/

As Sean observed, GPT-4 was an ineffectual tease for this level of – dare I say – code comprehension. But his experiments showed that o3 has come a long way from GPT-4. Imagine where we'll be in another couple of years. Some slowing of progress was inevitable. But there's no doubt that significant advancements are still being made. And I will assert again that it only makes sense that AI ought to eventually be able to do a perfect job at pre-release code function verification.

Once we're able to release vulnerability-free code, it won't matter whether the bad guys also have the ability to use AI for vulnerability discovery... because there won't be any vulnerabilities left for them to discover and exploit.

We're not there yet. But as the Magic 8-ball said: *"Signs point to yes."*