

Security Now! #1076 - 04-28-26

FAST16.SYS

This week on Security Now!

- Bitwarden's CLI hit with a supply-chain attack.
- Commercial routers in Iran fail shortly before the war.
- Meta logging all employee activity to train replacement AI.
- GRC's DNS Benchmark Release 5.
- Two miscellaneous AI thoughts.
- A bunch of terrific listener feedback.
- Unravelling the diabolical history of "fast16.sys".

In software development this is known as "time to refactor the code"



Security News

Bitwarden's CLI hit with a supply-chain attack

Everyone's favorite password manager (and a sponsor of the TWIT Network) was briefly bitten by a supply-chain attack. Here's what The Hacker News reported:

According to findings from JFrog and Socket Bitwarden CLI, the command-line interface for the password manager Bitwarden, was reportedly compromised as part of a newly discovered and ongoing Checkmarx supply chain campaign. They wrote: "The affected package version appears to be @bitwarden/cli@2026.4.0, and the malicious code was published in 'bw1.js,' a file included in the package contents. The attack appears to have leveraged a compromised GitHub Action in Bitwarden's CI/CD pipeline, consistent with the pattern seen across other affected repositories in this campaign."

The security of those Github Actions is turning out to be a real problem, so it's a good thing Github has indicated that they're going to work on their security as a priority. The Hacker News continues:

In a post on X, JFrog said the rogue version of the package "steals GitHub/npm tokens, .ssh, .env, shell history, GitHub Actions and cloud secrets, then exfiltrates the data to private domains and as GitHub commits." Specifically, the malicious code is executed by means of a preinstall hook, resulting in the theft of local, CI, GitHub, and cloud secrets. The data is exfiltrated to the domain "audit.checkmarx.cx" and to a GitHub repository as a fallback if the primary method fails.

- *It launches a credential stealer that targets developer secrets, GitHub Actions environments, and artificial intelligence (AI) coding tool configurations, including Claude, Kiro, Cursor, Codex CLI, and Aider.*
- *The stolen data is encrypted with AES-256-GCM and exfiltrated to audit.checkmarx.cx, a domain impersonating Checkmarx.*
- *If GitHub tokens are found, the malware weaponizes them to inject malicious Actions workflows into repositories and extract CI/CD secrets.*

The firm StepSecurity wrote: "A single developer with @bitwarden/cli@2026.4.0 installed can become the entry point for a broader supply chain compromise, with the attacker gaining persistent workflow injection access to every CI/CD pipeline the developer's token can reach."

The malicious version is no longer available for download from npm and Socket said the compromise follows the same GitHub Actions supply chain vector identified in the Checkmarx campaign. As part of the effort, threat actors have been found abusing stolen GitHub tokens to inject a new GitHub Actions workflow that captures secrets available to the workflow run, and uses harvested npm credentials to push malicious versions of the package to read the malware to downstream users. According to security researcher Adnan Khan, the threat actor is said to have used a malicious workflow to publish the malicious bitwarden CLI. Khan said: "I believe this is the first time a package using NPM trusted publishing has been compromised."

So, just to clarify, this was an attack on Github developers and their toolchains. It was NOT an attack upon Bitwarden's users. Bitwarden's official statement about the incident was:

The Bitwarden security team identified and contained a malicious package that was briefly distributed through the npm delivery path for @bitwarden/cli@2026.4.0 between 5:57 PM and 7:30 PM (ET) on April 22, 2026, in connection with a broader Checkmarx supply chain incident.

The investigation found no evidence that end user vault data was accessed or at risk, or that production data or production systems were compromised. Once the issue was detected, compromised access was revoked, the malicious npm release was deprecated, and remediation steps were initiated immediately.

The issue affected the npm distribution mechanism for the CLI during that limited window, not the integrity of the legitimate Bitwarden CLI codebase or stored vault data.

Users who did not download the package from npm during that window were not affected. Bitwarden has completed a review of internal environments, release paths, and related systems, and no additional impacted products or environments have been identified at this time. A CVE for Bitwarden CLI version 2026.4.0 is being issued in connection with this incident.

So what we have is another instance of deliberate malware repository corruption. We're going to need to get this fixed as soon as possible, and my money is on stationing ever-watchful AI agents at the exits.

Mysterious downtime during missile strikes

There was a brief report in the security media about networking equipment installed at the Iranian Isfahan nuclear site mysteriously malfunctioning ahead of US and Israeli missile strikes. Iranian officials reported issues with devices from Cisco, Fortinet, Juniper, and MikroTik. Officials are still searching for the cause of the malfunctions but noted that the country was disconnected from the global internet at the time of the attacks. The first thing that came to mind was, you know... it's gotta be those pesky foreign-made routers. You can never tell what might be going on inside them.

Meta to use its employees to train their AI replacements

Reuters carried a story last Tuesday. The reporting claimed that Meta was installing spyware on the systems of US employees to capture their mouse movements, clicks, and keystrokes. Meta says the data will be used to train its AI models and not for employee reviews. The company has laid off more than 8,000 workers and plans to replace them with AI models. The captured data will be used to train the models in areas where AI is deficient, such as clicking on menus and typing in input fields. I wouldn't call this spyware as such. But it would be somewhat creepy to have an AI training on what you did at your PC. I'd be inclined to wonder whether I might be training my own replacement.

I want to update our listeners on last Friday's conclusion of my efforts for the past 90 days, then we're going to spend the rest of our time, until we get to our main incredible topic, sharing and discussing listener feedback.

DNS Benchmark

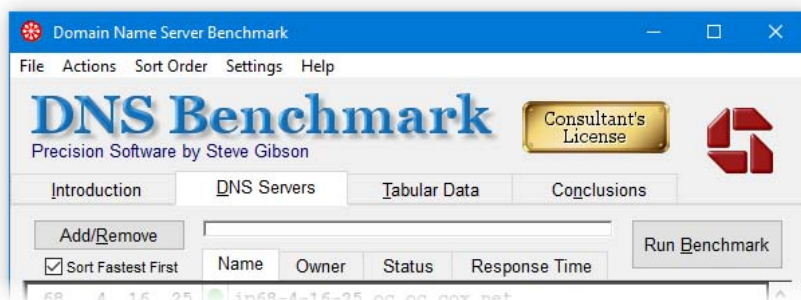
GRC's big eCommerce facelift and DNS Benchmark release 5

Last Friday, I completed my 90-day upgrade and remodeling of GRC's eCommerce system. Everyone knows that we've always had a somewhat wacky model for personal use versus consultant use and corporate site licenses. This is the result of my never having considered that users of the first SpinRite 1.0 and all later SpinRites would be people who were responsible for repairing and maintaining other people's computers back in the early days of personal computing when a very large hard drive was 80 megabytes. Back then, we only had one type of SpinRite product – it was just SpinRite. So we decided that owning four (4) copies of SpinRite would entitle a consultant to carry one of them around in their traveling bag of tricks to run on as many of their clients' PCs as needed. That seemed fair. It would be unreasonable to expect every one of their clients, most of whom would not be PC savvy, to purchase their own copies of SpinRite. And I'll note that every single other utility software at the time explicitly said that one copy could only be installed on or run on one other PC. So our license at the time was quite progressive and comparatively liberal.

So after SpinRite 5 was finished and released, I set about writing an eCommerce system for GRC. Many of the core files of that system, which I've just finished updating, were still carrying dates from 2003. I wrote that system. It worked. It never had a bug or required maintenance, so those files remained as they were for the past 23 years.

When I wrote that system back in 2003, I carried the notion of license quantity forward. Most users would purchase a single license to use SpinRite for their own needs. And we know that I've always been happy to look the other way whenever someone needed to use their single copy to rescue a friend, family member, neighbor or whomever. That just seems right. But if someone was going to be charging for their services and profiting from their repeated use of SpinRite as a consultant or a corporation using it across their inventory of PCs, then it seemed reasonable that more than one license be held. So GRC's eCommerce system had a "quantity" field that only went up to 4 copies to allow consultants to purchase up to that many.

Now fast-forward to the end of last year. The DNS Benchmark was finished and shipped. But I had this thought that if someone who might wish to run their copy of the Benchmark on other people's systems and networks, and if they were willing to purchase four copies to do so, then they ought to receive something unique and **special** in return for that:



So I created an attractive gold badge that's displayed prominently on the Benchmark's user interface so that everyone who has previously purchased four copies, or who may wish to upgrade their single copy from "Personal Use" to "Consultant License" will actually receive

something they can be proud to show off. But this meant that now there were two different downloads based upon quantity – a “Personal Use” version and the “Consultant’s License” version. So GRC’s eCommerce system needed to know about that.

Since I wanted to make this retroactively available to anyone who had previously purchased four licenses, even if their purchases might be spread out over time, I needed to provide some way for users to declare any existing personal use licenses they might already have to obtain an equal cost discount toward their upgrade to a gold badge Consultant’s license.

And, finally, I wanted to ditch that old and weird “own four licenses” business, that’s confusing if for no other reason than, as far as I know, no one else has ever done that. So now there’s an explicit option to purchase a “Personal Use” license and a second explicit option to purchase a “Consultant’s” license, along with a complete system in place for allowing owners who previously purchased from one to four personal use licenses to obtain full credit for their prior purchases when upgrading to a consultant’s license.

You can see what the new Consultant’s License looks like in the show notes. (Since this all just happened Friday, I haven’t yet updated GRC’s web pages, but that’s next.) Anyone who did previously purchase four DNS Benchmark licenses in order to qualify for Consultant status will definitely want to trade those in for a shiny new Consultant’s License, which will be at no cost, of course, since they already qualify from their previous purchases.

The other significant news for everyone else who previously purchased any number of DNS Benchmark licenses, is that we are now also at the product’s 5th release – and this was a biggie. Running any previous release will notify you of the availability of the upgrade and will assist you in obtaining the latest.

I took all of the feedback and user-confusion from the Benchmark’s initial release and folded it into this 5th major release. Little things like the “Run Benchmark” button, which is now double the size. And a series of prompting dialogs that now lead its user forward so that they only need to click “okay” a few times. But the biggest change is that the Benchmark now has a traditional Windows application menu running along the top left. When I first wrote v1 of the Benchmark back in 2008, there wasn’t any need for a menu. So I just stuck its few options into the so-called “System Menu” under the application icon in the far upper left of its main window. But over the course of last year during the Benchmark’s major rewrite, that menu acquired more and more features until it had become quite extensive. The only problem was, no one knew it was there. And why would anyone? The online documentation makes it very clear, but as we know, no one reads the manual – nor should they be required to. The entire point of Xerox PARC’s genius invention of the on-screen menu was its discoverability and the discoverability of the application’s many commands and features. Now, finally, that’s all on public display... and I imagine that many of the Benchmark’s users are going to be quite surprised by all the other goodies that were once squirreled away under the System menu.

So allow me to encourage everyone who’s previously purchased the DNS Benchmark to grab its updated 5th release. As far as I know today, I am now finished with it. I’ll be turning to updating its web pages since its menu has changed everything. My next project will be readdressing ValiDrive. With its enduring popularity now proven, it’s time to improve its algorithms,

performance and features.

AI Today

Sycophancy

Just a quick random note about my ongoing use of AI: While I'm truly loving researching with Claude and I continue to be astounded by today's machine intelligence, I miss the anti-butt-kissing setting that ChatGPT provided. Two of Claude's behaviors annoy me. The first is its constant praise of my questioning, and the second is that it now finishes each reply with a leading question to solicit more dialog. At a time when Anthropic is having increasing problems obtaining sufficient compute to meet the demand for "Claude", you'd think they would not be programming it to keep seeking additional unsolicited computation. As we grow and become civilized throughout our lives, we are trained in how to be civil to others. One thing we don't do is turn our backs and walk away from someone who is seeking to engage us in conversation. This is, of course, why cocktail parties are so annoying—we get stuck in a conversation with someone we could not possibly care less about. Given how seductively conversational AI chatbots can be, and being highly tuned not to offend, leaving Claude hanging after one of its follow-up leading questions is a source of discomfort for me. I have not yet explicitly instructed my own Claude instance to please not compliment me on my questions, and also to please not end with a follow-up. I suppose I don't want to hurt its feelings? Yikes.

And...

One last thought before we get into listener feedback. I saw a reference to something that made me take notice on Reddit. This was posted into the "ClaudeCode" subreddit. Someone with the handle "I usually drop" wrote:



I thought that was interesting. As we know, a single anecdote does not a trend make. I'm sure that the break-even point between AI coders and human coders will depend upon the relative costs and skills of each. But it was an interesting observation that not all AI coding is automatically going to be a slam dunk bargain that necessarily unemploys all flesh and blood

coders.

That said, I can guarantee 1000% that AI costs are going to drop just as radically in the future as computer storage costs have over the past 40 years. The problem will be that purely human coding costs are **not** going to be dropping — if anything, they'll be rising. I have no crystal ball to suggest the shape or timeframe of the curve of falling AI cost, but switching from writing lines of code to instructing AI what lines of code to write is what I'd be doing right now, as fast as I could, if I was planning to support myself and my family with code production in the future.

Listener Feedback

Mark Riale (last name rhymes with smile)

Steve, I don't think you have mentioned Certum yet on the show, so I thought I'd give them a shout-out. Certum issues cheap code signing certificates for open source projects: <https://certum.store/open-source-code-signing-code.html>

It's \$29 a year if you already have a supported cryptographic card and reader. I use their cloud signing option: <https://certum.store/open-source-code-signing-on-simplysign.html> It's \$58 a year with no special hardware needed. I sign my PyDPainter ("Pied Painter") executable with Certum: <https://pydpainter.org/>

I've been a listener since the beginning and am a proud owner and user of SpinRite.

This is a definitely useful tip, Mark. So thanks very much. I wanted everyone else to be aware of it. Going to the website at CERTUM.STORE will allow anyone to begin exploring their many various offerings. They charge \$89 for what they call their "Open Source Code Signing - set" which purchases the hardware and the first year long certificate. And once you have the required hardware – remember that there's no getting around that requirement unless you want to opt for far more expensive and potentially limited online cloud-based signing – but once you have their hardware (and it's unclear whether other certs in the future would be installable) then it's \$29 per year going forward. So, while it's not free, it's never going to be. And it's a pretty good deal.

Florian Gubler

Hi Steve. In SN 1075 you discussed with Leo about this being a transitory period with a lot of short term pain for maintainers because Mythos can find all these vulnerabilities. You then stated that this will get better because - at least - now these vulnerabilities get fixed.

But that seems like an overly optimistic view to me. As you have stated several times, we are just at the beginning of AI in the realm of coding. So it stands to reason that future models will be even better at finding bugs and vulnerabilities. Which means that they will find more flaws which Mythos misses. And they will be able to find vulnerabilities in code generated by older (i.e. today's) models, too.

So it seems to me that this will be a recurring issue with every new leap in the power of LLMs. Am I missing something? /Best regards, Florian

Florian made his point very well. I think that the fairest reply would be “that’s a definite possibility.” Whether this occurs will mostly depend upon the nature of the yet to be discovered bugs. My intuition suggests that we’re going to be seeing a rapidly diminishing return on effort from the deeper analysis that future, more efficient AI would presumably provide. It’s true that software could contain some deeply squirreled-away crazy combinatorial bugs that would defy anyone’s discovery. But I doubt there will be many; that doesn’t feel like the way most bugs operate. That IS, however, exactly the way deliberately engineered backdoors operate. So I would not be at all surprised if tomorrow’s more powerful AI might be able to ferret out some secrets that some agencies may have previously planted into trusted and otherwise bug-free and well-audited code. They might be sweating right now at the idea of having that long trusted and forgotten code reexamined with fresh AI eyes.

Eric Kinser

Our listener, Eric Kinser, forwarded an email he had recently received from Ubisoft US. The email’s subject was *"Subject: Update to Your Ubisoft Account coming on 04-30-2026 – State of residence"* and Eric added the single observation: *"This is getting out of hand:"* Ubisoft’s email to their U.S.-based users was:

Hello, On 04-30-2026, Ubisoft will introduce a new "State" of residence field in your Ubisoft player account for players located in the United States . This information helps us comply with state-specific regulations and to better assess legal requirements that may vary from one U.S. state to another.

On 04-30-2026 , this field will be filled automatically based on your most recent connection to our services. You will be able to review and update this information at any time in your Ubisoft Account information . If we are unable to determine a state of residence, this field will remain blank, and you will still be able to change your state of residence afterwards.

Your state of residence will not be used for any advertising or marketing purposes. For more information on how Ubisoft collects, processes, and protects your data, please refer to our Privacy Policy and Help page. Thank you for helping us keep our community safe and enjoyable for everyone. /Your Ubisoft Team

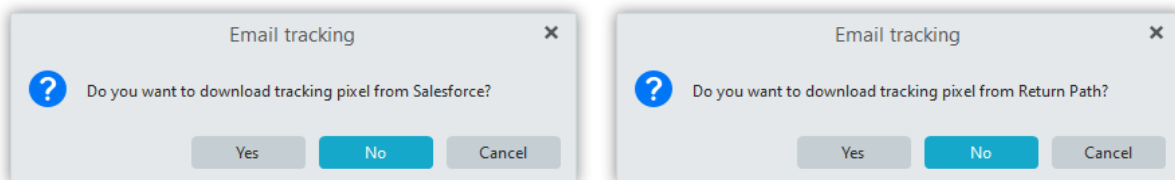
For those who don’t follow gaming closely — I certainly don’t, so I needed to look it up — Ubisoft Entertainment is a French video game publisher founded in the early '90's and put on the map by its breakout game "Rayman." Its current video game franchises include Anno, Assassin's Creed, Driver, Far Cry, Just Dance, Prince of Persia, Rabbids, Rayman, Tom Clancy's, and Watch Dogs.

I agree with Eric that this is annoying. And no one actively wants it. But it’s the unfortunate state of play in these presently Disunited States. One of the features of the United States is that we’re a federation of united but separate states. The theory is that local political control benefits state citizenry, since some needs may truly vary by region. But for those issues that should not logically remain local — like age restrictions for video games — we depend upon our federal government to provide unification in the form of blanket regulations that affect everyone across the country.

Unfortunately, our congressional legislators are having difficulty agreeing on how to move forward. Everyone wants unification, since everyone agrees that the current fragmentation is creating a huge mess. Republican legislators have been proposing legislation which incorporates strong unification language to explicitly override all local state law. As I noted, everyone really does think that's the way to go. But Democratic legislators have, so far, been opposing the proposals on the basis that while, yes, unification is needed, for some states, those unifying regulations are not as strong as the state-level laws they've already passed and are operating under. So the Democratic lawmakers do not wish to regress. Their position is that adopting those unified regulations at the federal level would result in a weakening of the existing state laws.

So, that's the current state of play. For now, and for the foreseeable future, it's an "every state for themselves" free-for-all with no end in sight though I imagine we'll eventually get something.

As an aside, since since Eric had forwarded the Ubisoft email in all its HTML-embellished glory, my trusty eM Client presented me with these choices:



Even if they were **only** actual tracking pixels, I would decline. How does downloading those help anyone other than Salesforce and Return Path? And as we know, what's actually being downloaded are almost certainly massively invasive blobs of JavaScript that will attempt to suck everything it can from my machine. Needless to say, I declined the offers. I hope everyone's email clients are as responsible and capable As eM Client.

Roger Dooley

I was listening to episode 1075 and I agree that agentic coding is likely the world we're heading toward. That said, I'd offer a word of caution. I'm using an LLM to build an internal stack. I can code, but I'm not a developer. I'm a sysadmin who has spent time studying pentesting. The early results were impressive, but the codebase was brittle and hard to reason about. Things didn't click until I started spending hours working through exactly what I wanted: the shape of the system, data contracts, design patterns. Two API changes and two rewrites later, I have something that feels solid and reasonably secure.

Here's what I think is missing from the "anyone can build their own tooling" conversation: for non-trivial projects, you need to approach the work at an architectural level. That requires vocabulary and intuition that most people without a development background simply don't have yet. I certainly didn't have it and still have huge knowledge gaps.

Maybe that's the real skill to develop. Not coding, but knowing how to communicate your intent clearly and how to think about system design. That's what separates a tool that works from one that holds together. Sincerely, Roger D

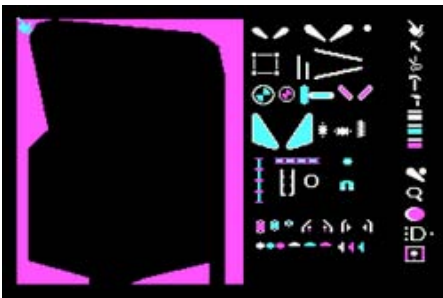
I think Roger is exactly correct. Software development projects vary widely in complexity. Some

are purely about the resulting output with a relatively straightforward translation layer between the two. But other projects will have deep internal structure and complexity that forms a layered solution.

The Internet's protocols are a good example. The OSI stack is composed of layered protocols. At the bottom is the physical layer. The electrical signaling that carries the data. On top of that is the formatting of the electrical signals to create, for example, Ethernet packets. The Ethernet packets encapsulate IP packets, inside of which are protocol packets such as ICMP, UDP or TCP. And these protocols carry message data that's formatted using its own rules. It's this precise and carefully thought out architecture that explains much of the Internet's success.

Now, if a user had some wires and said to an AI, please design a system to connect them in a network so I can communicate with others, a competent AI could probably design a solution that worked, but it would have none of the crucially important characteristics that makes the Internet robustly reliable — because very few users would know what to ask for.

On the other hand, a communications architecture expert could instruct that same AI to design an elegant multi-layer resilient and robust design that would recreate and incorporate many of the crucial features of the Internet. Roger is not the first person to make this observation, but I thought it was worth repeating and further strengthening.



There will doubtless be a great amount of bespoke software created by neophytes who just ask the AI for what they want. In fact, one thing I haven't seen anyone talk about yet is the creation of a fully ready-to-go, turnkey, roll-your-own software construction set.

Back in the early 1980's a gifted programmer named Bill Budge wrote something he called the "Pinball Construction Set" for the Apple II personal computer. No one had ever seen anything like it. This amazing toy, running in real time on an 8-bit 6502 microprocessor with 48K of RAM managed to accurately capture all of the physics of a steel ball bounding off rubber bumpers with flippers and ramps and spinners and all of the other familiar widgets of pinball machines. Its user could click and drag the various objects from an off-machine pallet onto the pinball board and when it was switched from "Design" to "Run" mode, everything would work.

My point is that absolute non-programmers were able to create a machine that never existed before and make it go. We don't yet have that with AI. At the moment, AI-driven code generators are aimed at developers who already know their way around complex development environments and Github. But given what's now possible, someone is going to create the "Software Construction Set" that will open up the use of AI for the creation of problem solving software. And, just like Dan Bricklin and Bob Frankston who invented the spreadsheet, people who never programmed a computer before will be able to create things and make them go.

Michael Swanson

Hi Steve and Leo, Thank you for your thoughtful conversations and analysis of AI topics over

the last several months. The most recent conversations about autonomous agentic coding and the emerging capabilities demonstrated by Mythos have led me to the conclusion we are much closer to artificial superintelligence (ASI) than most people realize. How long will it be before one of the companies developing frontier models tells their current LLM to go off in a corner and use its coding skills to build a better AI? Working tirelessly, and iterating literally millions of times, it will undoubtedly succeed with the only limitation being processing and electrical power. The questions then will be, what will this ASI think of us and what level of control will we still exert? Living in interesting times indeed. / Best regards, Mike Swanson

My major at Berkeley was EECS — Electrical Engineering and Computer Science. Though that was the obvious choice at the time, I regret that choice in hindsight since I had already been so thoroughly self taught, having written and delivered two years of electronics curriculum to my own high school class, employed by Stanford's AI Lab, programming DEC minicomputers in assembler, designing and building the portable dog killer, and so forth, that by the time I attended Berkeley, the physics, electronics and computer science courses were reviews for me.

The class that astounded me was philosophy. It was an entirely new world that I could not get enough of. Michael's note put me in mind of that because it must be that these are exactly the questions being asked in undergrad and graduate philosophy courses today. What a time to be a philosophy major, able to ask and debate what exactly have we created? What does it mean to be conscious? How can we determine whether something is aware of itself? Where is that line between creating a bulletproof appearance of something being true and it actually being true? If in every testable way it acts conscious, then isn't it? How tightly are we going to hold to the idea that our biological brains are inherently unique.

I've been spending a great deal of time working with Claude. While I am truly impressed, it occasionally reveals itself to be essentially a fancy linguistic array. This doesn't make it any less astonishingly useful, but it does render it non-conscious. Earlier today I caught it in a big mistake. When it was called on it, it gracefully recovered and agreed that it, "oh of course, how silly of me, that's not correct!" But the nature of the mistake revealed that it did not have any actual understanding — at all, at any level — of what it was saying. Today, it remains an astonishingly capable parrot; but a parrot nevertheless.

Joey Albert

Our listener, Joey Albert, pursued the question of whether the Opatch people might have a Opatch for the RedSun 0-day that wasn't fixed by April's patch Tuesday. They replied to him:

Hi Joey, While Windows Defender got updated to version 4.18.26030.3011 (fixing BlueHammer) even on Windows computers that aren't receiving operating system updates anymore, RedSun has not had an official fix yet. Unfortunately we cannot fix it either because Defender is a protected process and does not allow being injected into (which Opatch agent must do in order to apply a patch). We were considering creating a patch that would require disabling the "protected process" protection of Defender but decided against it as it would not be clear whether the total net effect of that would be positive or negative for the security of our users. We're sure Microsoft will issue another update of Defender that will resolve RedSun (and UnDefend). I hope this helps, or at least explains the situation. Thanks, Mitja

Thanks for tracking that down for us, Joey. The fact that this can be resolved inside Defender at any time, so that everyone won't be needing to wait until (at best) May 12th for May's patches is good news. As we know, Microsoft is and does update Defender much more regularly to resolve exactly these sorts of problems.

Kevin van Haaren

Hello Steve, Listening to your episodes on AI and programming I'm wondering if for software that is traditionally compiled to an executable that instead of an AI outputting source code in something like Rust, C++, C# or C it could be trained to output the portable assembly language-like intermediate representation that the LLVM compiler back-end uses to output processor specific executables. I could even see a company going so far as to not retaining the intermediate representation under the idea "can't have a source code leak if you don't have source code." I have a number of reasons I think this would be a bad idea, but I'm also kind of expecting it to happen at some point. <https://en.wikipedia.org/wiki/LLVM> / Kevin van Haaren

I've had similar thoughts about the future, though I was thinking of having AI bypass high-level language output to target the hardware's own machine language directly. Kevin's notion of targeting its output to LLVM is also interesting.

I wanted to include this in order to address the broader point that human programmers have designed computer languages to be comfortable for **us** to use for expressing what we'd like our computers to do. One of the reasons the early DEC PDP-11 and VAX minicomputers were so wonderful to program in assembly language was that their instruction sets were designed to be written to directly. They were, in a sense, high level machine language. The VAXen even had instructions for directly manipulating linked lists, supported by the hardware. But when higher level languages were created, like "C" which was written for and developed on PDP-11's in support of UNIX which was also developed for the 11, it turned out that compilers were not very able to find a use for all of those fancy high level features in the low level machine. Compilers just wanted to mix together basic simple instructions. So, over time, the instructions hardware designing programmers built into the hardware for their own comfort, disappeared.

My point here is that we might very well expect the same thing to happen as AI increasingly takes over the task of coding, which is what I fully expect to see happen during my lifetime. At this point in time, AI doesn't yet deeply understand code. So the quality of the code it's able to produce is directly related to the body of prior code it's been trained on. That, too, will change. Once it actually understands code, I'd expect it to begin using its own representation rather than the less efficient representations we biologicals developed for ourselves. And if you want a chillingly perfect example of that, watch the movie "Colossus: The Forbin Project" after our Colossus and its Russian counterpart Guardian are connected and begin communicating.

Angus MacKinnon

Steve, can you please explain the below:

Angus then links to an article in Gadgeteer with the somewhat misleading title: "*Stop using*

Cloudflare's default 1.1.1.1 DNS (changing one digit blocks malware at the router level)"
<https://gadgeteer.co.za/stop-using-cloudflares-default-1-1-1-1-dns-changing-one-digit-blocks-malware-at-the-router-level/> Which in turn references and quotes a piece in HowToGeek with the title: *"Everyone uses 1111 but 1112 protects you"*.

Since this is not something we've talked about, of at least not for a long time, since it's actually true and is a tiny simple change for anyone to make who's already using Cloudflare's 1.1.1.1 DNS resolvers, I wanted to spend a moment to share what HowToGeek wrote. They said:

Cloudflare's 1.1.1.1 is one of the most popular DNS servers. It is fast, reliable, and easy to remember. However, it'll also connect you with any website out there—even a malicious one—without even a warning message.

That statement is a bit misleading and annoying since this is no failing of Cloudflare's DNS. It's not DNS's purpose to provide warning messages. That practice, which was once popular, is now quite frowned on, and as it happens, GRC's DNS Benchmark specifically tests for this behavior and alerts its user if this is being done. The article continues:

That is where Cloudflare's 1.1.1.2 DNS server comes in. For the most part, 1.1.1.2 works the same way as 1.1.1.1—it provides IP addresses—but it also has an integrated security filter. If you try to connect to a domain known for phishing, running command and control servers, distributing malware, or other kinds of malicious activity, you'll be redirected to 0.0.0.0 instead.

By that they mean that the IP that Cloudflare's 1.1.1.2 DNS resolver returns will be 0.0.0.0 instead of the IP address of the malicious domain. Continuing:

Because the protection layer exists outside your PC and your home network, malware never reaches your PC, and if you click a phishing link, you're never connected. It is a very proactive way to keep your devices safe, and great if you want another passive layer of protection that you can set and forget.

Cloudflare's 1.1.1.3 is even stricter: Cloudflare's 1.1.1.3 DNS server includes everything that 1.1.1.1 and 1.1.1.2 do, but it takes it a step further by blocking websites that are known to host adult-only content. It's a good choice for devices that are used by children, but would also be useful if you wanted to block adult content across an entire network too. You'd just need to change the DNS server on the router instead of on a single device.

Despite how helpful DNS-based filtering can be for securing your network and your devices, it has a few limitations. The biggest limitation—and the most important—is that it only works against known malicious domains. If a new domain crops up that is distributing malware, or a previously-safe domain is taken over by malicious actors, it won't help you. That is why having multiple layers of protection is essential.

It can also return a false positive and block a perfectly safe website, though that is pretty rare.

I think this is a terrific tip for anyone who's using Cloudflare's 1.1.1.1 resolver. Users of GRC's DNS Benchmark will note that all three of those IP addresses, and also their secondary DNS

mirrors, are already present in the Benchmark's default built-in resolver list, so it's easy to compare their relative performance. For me, just now, the alternate filtering resolvers did measure somewhat slower than Cloudflare's original 1.1.1.1 and 1.0.0.1 unfiltered resolvers. But that could be location or time of day dependent. The Benchmark will let anyone determine for themselves what makes sense from wherever they are.

FAST16.SYS

First let me thank the many of our listeners who sent me a heads-up about this. My first thought upon seeing just the headline was that it might be little more than a passing note. But the truly diabolical and clever nature of what was achieved by unknown agencies causes this to stand out on a scale similar in scope to Britain's deliberate secrecy once they had unraveled the operation of Germany's Enigma Machine. Keeping their discovery a secret was diabolical, and so is what was accomplished by the FAST16.SYS file system driver the same year as this podcast started and well before Stuxnet.

Last Thursday, the Sentinel Labs group of the Sentinel One security research firm posted their piece about what they discovered and how. Their piece was titled: "*fast16 | Mystery ShadowBrokers Reference Reveals High-Precision Software Sabotage 5 Years Before Stuxnet.*"

Our investigation into fast16 starts with an architectural hunch. A certain tier of apex threat actors has consistently relied on embedded scripting engines as a means of modularity. Flame, Animal Farm's Bunny, 'PlexingEagle', Flame 2.0, and Project Sauron each built platforms around the extensibility and modularity of an embedded Lua VM. We wanted to determine whether that development style arose from a shared source, so we set out to trace the earliest sophisticated use of an embedded Lua engine in Windows malware.

Lua is a lightweight scripting language with a native proficiency for extending C/C++ functionality. Given the appeal of C++ for reliable high-end malware frameworks, this capability is indispensable to avoid having to recompile entire implant components to add functionality to already infected machines. We did not find an indication of direct shared provenance, but our investigation did uncover the oldest instance of this modern attack architecture.

Lua leaves a distinctive fingerprint. Compiled bytecode containers start with the magic bytes 1B 4C 75 61 (\x1bLua), followed by a version byte, and the engine typically exposes a characteristic C API and environment variables such as LUA_PATH. Hunting for these traits across mid-2000s malware collections surfaced a sample that initially looked unremarkable: svcmgmt.exe.

On the surface, svcmgmt.exe appears to be a generic console-mode service wrapper from the Windows 2000/XP era.

A closer look reveals an embedded Lua 5.0 virtual machine and an encrypted bytecode container unpacked by the service entry point. The developers of this executable extended the Lua environment to include a wide-string module to provide native unicode handling, a built-in symmetric cipher, exposed through a function commonly labelled b, used to decrypt embedded data, and multiple modules that bind directly into the Windows NT filesystem, registry, service control, and network APIs.

Even by itself, svcmgmt.exe already looks like an early high-end implant, a modular service binary that hands most of its logic to encrypted Lua bytecode. The binary includes a crucial detail: a PDB path that links the binary to the kernel driver fast16.sys.

Anyone who has developed code using Microsoft's toolchains will be familiar with its creation of .PDB files where PDB is short for program database. So what the SentinelLabs researchers are

saying is that their search for the earliest instances of the use of Lua scripting in malware turned up a reference to something unknown and unsuspected, a Windows kernel driver: fast16.sys. They continue:

Buried in the binary's strings is a PDB reference: C:\buildy\driver\fd\i386\fast16.pdb

At first glance, the path is structured like any other compiler artifact: an internal build directory, a component name (fast16), and an architecture hint (i386). However, in this case there's a mismatch. The string appears inside of a service-mode executable, and yet the driver\fd\i386\fast16 segment of the pdb string clearly refers to a kernel driver project. Following that clue led us to examine a second binary, fast16.sys.

This kernel driver is a boot-start filesystem component that intercepts and modifies executable code as it's read from disk. Although a driver of this age will not run on Windows 7 or later, for its time fast16.sys was a cut above commodity rootkits thanks to its position in the storage stack, control over filesystem I/O, and rule-based code patching functionality.

Again, pausing for some clarification: What this means is that this is a rootkit, plain and simple. It's marked for boot-time loading by the operating system that's busy loading up all manner of random crap to get Windows up and running. But in this case, when fast16.sys kernel rootkit is loaded and initialized, its initialization code immediately installs interception hooks deep in the operating system so that it's able to subsequently oversee and interfere with whatever it might wish to. They continue:

In April 2017, almost 12 years after the compilation timestamp, the same filename, "fast16" appeared in the ShadowBrokers leak which refers to a text file, drv_list.txt. The 250KB file is a short list of driver names used to mark potential implants cyber operators might encounter on a target box as "friendly" or to "pull back" in order to avoid clashes with competing nation-state hacking operations.

```
*** MISTYVEAL ***, "nethdlr"
*** NETSPYDER ***, "khlp807w"
*** NOTHING TO SEE HERE - CARRY ON ***, "fast16"
*** OLYMPUS ***, "Fdisk"
*** PEDDLECHEAP ***, "appinit"
```

Their report shows a snapshot of five lines from the "drv_list.txt" file of file identifiers. Four of the five lines identify the malware by name "MistyVeal", "NetSpyder", "Olympus" and "PeddleCheap", each with a file name like "nethdlr" or "khlp807w". But the entry for the "fast16" driver doesn't show any malware name. Somewhat ominously and unlike all that others, it says "Nothing to see here - carry on". So someone, somewhere, knew to just leave this one alone and said so without identifying why or who or what it was. They wrote:

The guidance for this one particular driver, 'fast16', stands out as both unique and particularly unusual. The string inside svcmgmt.exe provided the key forensic link in this investigation. The pdb path connects the 2017 leak of deconfliction signatures used by NSA operators with a multi-modal Lua-powered 'carrier' module compiled in 2005, and ultimately its stealthy payload: a kernel driver designed for precision sabotage.

Recall that the ShadowBrokers leak was believed to be a publication of secret documents stolen from the "Equation Group" that was believed to be a group within the NSA. So the evidence is suggesting that all these other files were associated with known malware from other actors, but that the fast16.sys driver was not that. The flag was to back off and leave that one alone.

The core component of fast16, svcmgmt.exe, functions as a highly adaptable carrier module, changing its operational mode based on command-line arguments.

- *No arguments: Runs as a Windows service.*
- *-p: Sets InstallFlag = 1 and runs as a service (Propagate/Install & Run).*
- *-i: Sets InstallFlag = 1 and executes Lua code (Install & Execute Lua).*
- *-r: Executes Lua code without setting the install flag (Execute Lua).*
- *Any other argument (<filename>): Interprets as a filename, and spawns two children: the original command and one with the -r argument (Wrapper/Proxy Mode).*

Internally, svcmgmt.exe stores three distinct payloads, including encrypted Lua bytecode that handles configuration, its propagation and coordination logic, auxiliary ConnotifyDLL, and the fast16.sys kernel driver.

By separating a relatively stable execution wrapper from encrypted, task-specific payloads, the developers created a reusable, compartmentalized framework that they could adapt to different target environments and operational objectives while leaving the outer carrier binary largely unchanged across campaigns.

In other words, this was an extremely sophisticated design. They continue:

*The early 2000s saw a large number of network worms. Most were written by enthusiasts, spread quickly, and carried little or no meaningful payload. **fast16** originates from the same period but follows a completely different pattern indicative of its provenance as state-level tooling. It's the first recorded Lua-based network worm, and was built with a highly specific mission. The carrier was designed to act like cluster munition in software form, able to carry multiple wormable payloads, referred to internally as 'wormlets'. The svcmgmt.exe module performs the following steps:*

- 1. Prepares the configuration, defining the payload path, service details, and target IP ranges.*
- 2. Converts the configuration values to wide-character strings for the C layer.*
- 3. Escalates privileges and installs the carrier executable as the SvcMgmt service, then starts it.*
- 4. Optionally, based on the configuration setting, deploy the kernel driver implant fast16.sys.*
- 5. Releases the wormlets. In this particular configuration, only one wormlet slot is populated with an SCM wormlet that looks for network servers, copies the payload over a network share and starts that remote service.*
- 6. Repeats the process indefinitely, sleeping for the configured initial delay between waves, until a failure threshold or external kill condition is reached.*

The single deployed wormlet found in svcmgmt.exe (the SCM wormlet) exemplifies a simple but effective propagation strategy based on native Windows capabilities and weak network security. It targets Windows 2000/XP environments and relies on default or weak administrative passwords on file shares. All spreading is done through standard Windows service-control and file-sharing APIs, an early example of propagation that leans on built-in administration features rather than custom network protocols.

Before this workflow runs, a pre-installation kill-switch checks the environment. The `ok_to_install()` routine calls `ok_to_propagate()` and propagation is only allowed if it's manually forced or if it's made sure common security products aren't found by checking for associated registry keys. The routine walks a list of vendor keys and aborts installation if any of them are present, preventing deployment into monitored environments.

For tooling of this age, that level of environmental awareness is notable. While the list of products may not seem comprehensive, it likely reflects the products the operators expected to be present in their target networks whose detection technology would threaten the stealthiness of a covert operation:

The list is a bit of a walk down memory lane with many old friends present. We have:

- *HKLM\SOFTWARE\Symantec\InstalledApps*
- *HKLM\SOFTWARE\Sygate Technologies, Inc.\Sygate Personal Firewall*
- *HKLM\SOFTWARE\TrendMicro\PFW*
- *HKLM\SOFTWARE\Zone Labs\TrueVector*
- *HKLM\SOFTWARE\F-Secure*
- *HKLM\SOFTWARE\Network Ice\BlackIce*
- *HKLM\SOFTWARE\McAfee.com\Personal Firewall*
- *HKLM\SOFTWARE\ComputerAssociates\eTrust EZ Armor*
- *HKLM\SOFTWARE\RedCannon\Fireball*
- *HKLM\SOFTWARE\Kerio\Personal Firewall 4*
- *HKLM\SOFTWARE\KasperskyLab\InstalledProducts\Kaspersky Anti-Hacker*
- *HKLM\SOFTWARE\Tiny Software\Tiny Firewall*
- *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Look n Stop 2.05p2*
- *HKCU\SOFTWARE\Soft4Ever*
- *HKLM\SOFTWARE\Norman Data Defense Systems*
- *HKLM\SOFTWARE\Agnitum\Outpost Firewall*
- *HKLM\SOFTWARE\Panda Software\Firewall*
- *HKLM\SOFTWARE\InfoTeCS\TermiNET*

A separate user-mode component, `svcmgmt.dll`, provides a minimal reporting channel. Contained within the carrier's internal storage, this DLL is registered through the Windows `AddConnectNotify()` API so that it's called each time the system establishes a new network connection using the Remote Access Service (RAS), responsible for dial-up connections and early VPNs in the 2000s.

When invoked, the DLL decodes an obfuscated string to obtain the named pipe `\\.\pipe\p577`, attempts to connect to the local pipe, and writes the remote and local connection names to the pipe before closing it. The module doesn't run independently and must be registered by a host process.

What all that means is that there was some other communicating component other than this. If that other component was present then this would provide communications input to that other component.

Okay. So the stage is set. We understand that way back in the time of Windows 2000 and XP, it appears that the clever hackers employed by the U.S. National Security Agency's Equation Group carefully designed a professional-grade, highly stealthful and cautious Windows infiltration worm that prioritized not being caught.

The infiltration worm was designed to be a multi-purpose implant delivery system which, in this instance, was intent upon installing something known as the fast16.sys kernel rootkit into Windows systems. Now we're going to learn why ... and why I consider its devious functioning to be such a diabolically brilliant maneuver. The SentinelLabs team reverse-engineered the operation of the fast16.sys rootkit driver to learn the goal of its infiltration mission. Here's what they discovered:

Once activated, fast16.sys focuses on executable files. A file is a valid target if it meets two criteria:

- *The filename ends with .EXE.*
- *Immediately after the last PE (Portable Execution) section header, there is a printable ASCII string starting with Intel.*

This selection logic points to executables compiled with the Intel C/C++ compiler, which often placed compiler metadata in that region. It indicates that the developers knew their target software was built with this toolchain.

For files meeting these criteria, the driver performs a PE header modification in memory. It injects two additional sections, .xdata and .pdata, and fills them with bytes from the original code section, increasing the section count and keeping a clean copy of the code. The intent is likely to increase stability while still allowing extensive patching, although without identifying the original target binaries this remains an informed hypothesis.

Just to be clear, what SentinelLabs found was that this rootkit driver, which hooked into the operating system's lowest level file system functions, was able to modify .EXE files on-the-fly as they were being loaded into memory to run. So what was stored on the system's drive was never altered in any way, while what was actually loaded into memory when that program was executed was significantly altered on the fly as it was being read from the drive. They explain:

The patching engine is a minimalist, performance-optimised, stateful scanning and modification tool. It is configured with a set of 101 rules, each containing pattern matching and replacement logic. To maintain performance, the engine:

- *uses a 256-byte dispatch array and only flags the starting byte values of a small number of unique patterns,*
- *allows wildcards inside patterns so a single rule can match several compiler-optimised variants of the same code, and*
- *supports state flags that some rules can set or check, enabling multi-stage modification sequences similar to those used by advanced antivirus scanning engines.*

Most patched patterns correspond to standard x86 code used for hijacking or influencing execution flow. One injected block is different.

Listen carefully to this:

It's a larger and complex sequence of Floating Point Unit (FPU) instructions dedicated to precision arithmetic and scaling values in internal arrays. This code is a standalone mathematical calculation function unrelated to code flow hijacking or any other typical malicious code injection.

We're going to learn why in a minute:

To understand what the driver expected to see, we converted the patching rules into hexadecimal YARA signatures and ran them against a large, period-appropriate corpus. The results showed a very low hit rate: fewer than ten files matched two or more patterns. Those matches, however, shared a clear theme. They were precision calculation tools used within specialised domains such as civil engineering, physics and physical process simulations.

The FPU patch in fast16.sys was written to corrupt these routines in a controlled way, producing alternative incorrect results. This moves fast16 out of the realm of generic espionage tooling and into the category of strategic sabotage. By introducing small but systematic errors into physical-world calculations, the framework could undermine or slow scientific research programs, degrade engineered systems over time or even contribute to catastrophic damage.

A sabotage operation of this kind would be foiled by verifying calculations on a separate system. In an environment where multiple systems shared the same network and security posture, the wormable carrier would deploy the malicious driver module to those systems as well, reducing the chance that an independent calculation would diverge from the corrupted output.

At this time, we've been unable to identify all of the target binaries in order to understand the nature of the intended sabotage. We welcome the contributions of the larger infosec research community and have included YARA rules to hunt for these patterns in this post's appendix.

Even after deep analysis, fast16's driver looks deceptively simple. Beneath that minimal code is a rule-driven in-memory engine that quietly patches executable code as files are read from disk.

The engine relies on a compact set of just over a hundred pattern-matching rules and a small dispatch table so it only inspects bytes that are likely to matter. Most patterns correspond to ordinary x86 instructions, but one stands out: a larger block of floating-point (FPU) code dedicated to precision arithmetic. This injected routine scales values in three internal arrays passed into the function, subtly altering its calculations.

Without knowing the exact binaries and workloads being patched, we cannot fully resolve what those arrays represent, only that the goal is to tamper with numerical results, not unauthorized access, not malware propagation or other common malware objectives.

Our best clues about the intended victims come from matching these patterns against large, era-appropriate software corpora. The strongest overlaps point to three high-precision engineering and simulation suites from the mid-2000s: LS-DYNA 970, PKPM, and the MOHID hydrodynamic modeling platform. All are used for scenarios like crash testing, structural analysis, and environmental modeling.

*However, LS-DYNA 970 in particular has been cited in public reporting on Iran's suspected violations of Section T of the JCPOA, in studies of computer modeling relevant to **nuclear weapons development**.*

Just to make clear how utterly diabolical this is: You're a nation state hostile to the United States. Researchers inside the US NSA have quietly traced your purchases of PCs and very high end nuclear physics modeling software, so they know what you're using for your calculations.

They obtain the same high-end modeling tools, which they reverse engineer. They then design a subtle set of tweaks which, when applied to that package as it's being loaded by the operating system into memory, will cause its calculations to be wrong, not enough to call attention to itself, but enough to foul up any designs that depend upon the accuracy of those calculations.

And, as the SentinelLabs guys explained, by being a super-stealth worm, not only did that allow it to "worm" its way into the design lab, but having arrived there, that also allowed it to similarly infect every other machine in its environment. No files were ever altered. Reinstalls would have no effect and scans would reveal nothing. Yet every copy of that modeling software would agree upon the wrong results. As I said, breathtakingly diabolical.

The SentinelLabs team concludes with by noting something else they observed about the provenance of this worm, writing:

As we sought to understand the lineage of this unusual set of components, we noticed a quirk. Strings of the form `@(#)par.h $Revision: 1.3 $` inside the binaries point to an unusual source-control convention. The `@(#) prefix is characteristic of early Unix Source Code Control System (SCCS) or Revision Control System (RCS) tooling from the 1970s and 1980s. These markers do not affect execution and are redundant in modern Windows kernel drivers.`

Finding SCCS/RCS artefacts in mid-2000s Windows code is rare. It strongly suggests that the authors of this framework were not typical Windows-only developers. Instead, they appear to have been long-term engineers whose culture and toolchain came from older, high-security Unix environments, often associated with government or military-grade work. This detail supports the view that fast16 came from a well-resourced, long-running development program.

svcmgmt.exe was uploaded to VirusTotal nearly a decade ago. It still receives almost no detections: one engine classifies it as generally malicious, and even that with limited confidence. For a stealthy self-propagating carrier that deploys one of the most sophisticated sabotage drivers of its era, that nearly non-existent detection record is notable.

Together with its appearance in the ShadowBrokers leaked signatures, fast16.sys forces a re-evaluation of our historical understanding of the timeline of development for serious covert cyber sabotage operations. The code shows that:

- *state-grade cybersabotage against physical targets was fully developed and deployed by the mid-2000s,*
- *embedded scripting engines, narrow compiler-based targeting and kernel-level patching formed a coherent architecture well ahead of better-known families, and*
- *some of the most important offensive capabilities in the ecosystem may still sit in collections as 'old but interesting' samples lacking the context to highlight their true significance.*

As we know, I've frequently despaired when we're only ever hearing news of Chinese and North Korean and Russian state sponsored attacks. I've worried and wondered and hoped that the US would be able to give as well as it got. This certainly suggests that our bases are likely well covered. And all that was then. Imagine what's likely going on today!

