# Security Now! #895 - 11-01-22
# After 20 years in GCHQ

## This week on Security Now!

This week we revisit the Windows driver block list which has received a long-needed update and at Microsoft's own definition of a CVE. We note that sometime today the OpenSSL project will be releasing an update for an ultra-CRITICAL flaw in OpenSSL v3 and we look at a remote code execution flaw in Windows TCP/IP stack. We have a ubiquitous problem in the past 22 years of the widely used SQLite library and a surprising percentage of malicious proofs-of-concepts found in GitHub. Passkeys gets another supporter and the first part of a professional tutorial explaining how to exploit the Chrome browser is released. After some listener feedback and a SpinRite update, we look at the goodbye posting of the UK's head of cyber security after 20 years.

## "Out of Scope" = Somebody Else's Problem

## Security News

**Windows driver blocklist to be updated next Tuesday**

Recall our coverage last week of ArsTechnica's reporting titled *"How a Microsoft blunder opened millions of PCs to potent malware attacks"* and that, indeed, some follow-up digging revealed that many malware strains had been found to be actively leveraging known vulnerable drivers in so-called BYOVD – bring your own vulnerable driver – attacks. And also recall that Microsoft's minions, even those in elevated positions of authority, were openly rude and hostile to the researchers who were just trying to help, prompt and understand.

Well, this may be where shining a bright public light can help, since Microsoft has gotten off their butts and finally, after three years of neglect, fixed this important issue.

Last Thursday, ZDNet carried the news with the headline: *"Next Windows 10/11 Patch Tuesday fixes Microsoft's botched vulnerable driver blocklist"* and the subhead: *"Microsoft addresses an issue preventing Windows 10's vulnerable driver blocklist from being updated with new vulnerable drivers."* Here's how ZDNet summarized the situation:

*Microsoft has released a new non-security preview of November's Patch Tuesday update for Windows 10 and Windows 11 22H2. It brings improvements to the taskbar, Microsoft Account, and Task Manager, as well as a fix for a serious Microsoft blunder that left a hole in the Windows 10 vulnerable driver blocklist.*

*The preview is a non-security update that is available for Windows 10 and Windows 11 22H2. It contains all the changes in the upcoming November Patch Tuesday, except security patches.*

*However, this preview also includes Microsoft's answer to a serious security-related error that the company made with its Windows kernel vulnerable driver blocklist – an optional security hardening capability introduced in Windows 10, version 1809 that's on by default in Windows 11 22H2.*

*Microsoft has explained that the failed updates to the blocklist were due to it only updating for "full Windows OS releases", although it's not clear if this means previously installed Windows versus fresh installs, or just that older versions of Windows were stuck on a blocklist that couldn't be updated.*

*In a support page detailing "the vulnerable driver blocklist after the October 2022 preview release" Microsoft states: "This October 2022 preview release* ***addresses an issue*** *that only updates the blocklist for full Windows OS releases. When you install this release, the blocklist on older OS versions will be the same as the blocklist on Windows 11, version 21H2 and later."*

*Microsoft had told Ars Technica that it was in fact regularly updating the vulnerable driver list, but that there was "a gap in synchronization across OS versions."*

*So, the October 2022 preview release is the promised fix, which should be released broadly in the November 2022 Patch Tuesday through Windows Update. Microsoft's release notes for the October Windows 11 22H2 preview update states: "It updates the Windows kernel vulnerable driver blocklist that is in the DriverSiPolicy.p7b file. This update also ensures that the blocklist is the same across Windows 10 and Windows 11. For more information, see KB5020779."*

Microsoft has a page where they talk about recommended driver block rules which provides a bit more clarity. They say:

https://learn.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-driver-block-rules

*The blocklist is updated with each new major release of Windows, typically 1-2 times per year, including most recently with the Windows 11 2022 update released in September 2022. The most current blocklist is now also available for Windows 10 20H2 and Windows 11 21H2 users as an optional update from Windows Update. Microsoft will occasionally publish future updates through regular Windows servicing.*

*Customers who always want the most up-to-date driver blocklist can also use Windows Defender Application Control (WDAC) to apply the latest recommended driver blocklist contained in this article. For your convenience, we've provided a download of the most up-to-date vulnerable driver blocklist along with instructions to apply it on your computer at the end of this article. Otherwise, you can use the XML provided below to create your own custom WDAC policies.*

They're saying here that *"The blocklist is updated with each new major release of Windows, typically 1-2 times per year."* So it's curious that, for some reason, the exploitation of known malicious Windows kernel drivers will apparently be allowed, deliberately and by policy, for as long as one year, or during the entire length of the gap between major Windows releases. Given that BYOVD now has its own abbreviation and that the exploitation of such drivers has unfortunately become popular, it sure would see that some rethinking of this "we're only going to update this list with new major releases of Windows" policy should be reconsidered.

That page also said: *"Customers who always want the most up-to-date driver blocklist can also use Windows Defender Application Control (WDAC) to apply the latest recommended driver blocklist contained in this article."* So I have those instructions and a link to that page also in the show notes. Anyone wishing not to wait can easily update their system's driver blocklist immediately:

*If you prefer to apply the vulnerable driver blocklist, follow these steps:*

1. *Download the WDAC policy refresh tool*
2. *Download and extract the vulnerable driver blocklist binaries*
3. *Select either the audit only version or the enforced version and rename the file to SiPolicy.p7b*
4. *Copy SiPolicy.p7b to %windir%\system32\CodeIntegrity*
5. *Run the WDAC policy refresh tool you downloaded in Step 1 above to activate and refresh all WDAC policies on your computer*

WDAC policy refresh tool: https://aka.ms/refreshpolicy
Vulnerable driver blocklist binaries: https://aka.ms/VulnerableDriverBlockList

The vulnerable driver blocklist binaries ZIP file contains four 95 Kbyte files dated October 19th. There are audit and enforced versions for both server and non-server editions of Windows. So a total of four policy files to choose among. For anyone wishing to do this, I have a link in the show notes to the page which contains these instructions as well as a means for affirmatively verifying that the policy has been applied:

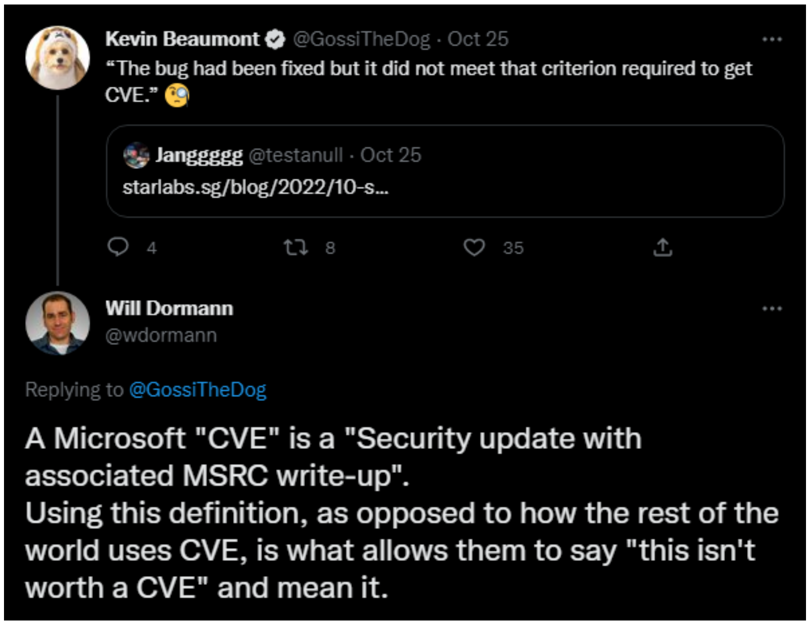
https://learn.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-driver-block-rules

And on that page, Microsoft notes that any already running malicious drivers will not be ejected by the application of this policy. The machine must be rebooted to allow the policy to filter out any baddies during their attempted reloading.

## More Microsoft shenanigans

Last Tuesday, the Singapore-based security firm StarLabs disclosed a verified and patched vulnerability in Microsoft SharePoint Server 2019. It was an exploitable post-authentication Server-Side Request Forgery (SSRF). What's bizarre is that while this was clearly a flaw that needed fixing, and fix it they did, Microsoft refused to assign a CVE identifier for StarLabs' finding and work.

Some well known security industry insiders took note and tweeted...



**Kevin Beaumont** @GossiTheDog · Oct 25
"The bug had been fixed but it did not meet that criterion required to get CVE." 🧐

**Jangggggg** @testanull · Oct 25
starlabs.sg/blog/2022/10-s...

4 8 35

**Will Dormann**
@wdormann

Replying to @GossiTheDog

A Microsoft "CVE" is a "Security update with associated MSRC write-up".
Using this definition, as opposed to how the rest of the world uses CVE, is what allows them to say "this isn't worth a CVE" and mean it.

So now, just as we have Microsoft's own definition of 0-day, now we have disappearing Microsoft CVE's courtesy of ignoring the industry's established vulnerability monitoring system by simply not registering discovered problems which require repair and updating. It's not a bug if we don't call it one. And recall that we saw something like this before where Microsoft was claiming that if no user action was required, then no security event was warranted. We'll just all pretend it never happened since we'd rather project that image to the world.

On the subject of important things not being swept under the rug, we have...

**An upcoming OpenSSL CRITICAL vulnerability update -- get ready!**
In order to facilitate the race between releasing a patch to an important open source system and those who will attempt to take advantage of the patch window before all systems are patched, exactly one week ago the OpenSSL project gave the industry one-week of advance notice of a forthcoming highly critical patch to OpenSSL 3.0. The patch release day is today, November 1st.

Commenting on this news, **Andrew Ayer / @__agwa** tweeted:

> *This will be OpenSSL's first "CRITICAL" vulnerability since 2016. Examples of "CRITICAL" vulnerabilities include "significant disclosure of the contents of server memory (potentially revealing user details), ...*
>
> *OpenSSL 3.0.7 update to fix Critical CVE out next Tuesday 1300-1700UTC. Does not affect versions before 3.0.*

So the vulnerabilities, CVE-2022-3786 and CVE-2022-3602, affect version 3.0.x and do not impact OpenSSL 1.1.1 or LibreSSL. This is all that's known at this time. I imagine that we'll know a lot more by this time next week since both the good guys and the bad guys are going to be tearing into the updated source code to discover what was changed and to then reverse engineering proofs of concepts and working exploits. I expect we'll be talking about this next week.

**A new TCP/IP RCE in Windows**
Speaking of network vulnerabilities, last week there was worry about the possible impact of a TCP/IP remote code execution vulnerability in Windows. That's an eye opener, since the assumption is that it's a problem in the core kernel code. And the worry was heightened by the fact that a proof-of-concept exploit was published to GitHub by researchers at Numen Cyber Labs who had reverse engineered the vulnerability through patch analysis. But the fact that there were patches to analyze meant that the problem had already been patched, as indeed it had been nearly two months ago in September's patch Tuesday.

After reading through their research it became clear that the conditions required for this vulnerability to be abused in the field were unlikely to occur. So it was mostly a theoretical vulnerability. I was shaking my head when I saw that the trouble was caused by fragmented packet reassembly in IPv6's IPsec handling. How many times in the early years of this podcast, when we were discussing the low-level technologies of the IP protocols, did we encounter exploitable bugs resulting from the attempt to reassemble fragmented IP packets? Back then it was a constant theme.

So, nothing widespread will happen from this. It's too specific, rare, and already patched. But this will be one of those growing numbers of vulnerabilities that major, long term, nation-state players will add into their known-exploits databases for possible selective deployment when they encounter an unpatched system which qualifies for this specific vector of attack. Though I have no firsthand knowledge or any evidence of this being done, there's just no chance that such databases do not exist in the world today. If I were in charge of the US's or China's or Russia's cyberwarfare effort, given that we know that vulnerabilities that never really die, creating such a database is the first thing I would have done.

## A study of malicious CVE proof of concept exploits in GitHub

And speaking of proof-of-concepts published on GitHub, I wanted to warn any of our listeners who might enjoy grabbing and trying such proofs-of-concepts for themselves that a just published study of GitHub hosted PoC's found that a high percentage of all PoC's were deliberately malicious.

Three academic researchers with the Leiden Institute of Advanced Computer Science at Leiden University in The Netherlands published their research titled "A study of malicious CVE proof of concept exploits in GitHub" https://arxiv.org/abs/2210.08374 The Abstract of their paper explains:

> Proof-of-concept (PoC) of exploits for known vulnerabilities are widely shared in the security community. They help security analysts to learn from each other and they facilitate security assessments and red teaming tasks. In recent years, PoCs have been widely distributed, e.g., via dedicated websites and platforms, and also via public code repositories like GitHub. However, public code repositories do not provide any guarantees that any given PoC comes from a trustworthy source, or even that it simply does exactly what it is supposed to do.
> In this work we investigate PoCs shared on GitHub for known vulnerabilities discovered in 2017-2021. We discovered that not all PoCs are trustworthy. Some proof-of-concepts are fake (i.e., they do not actually offer PoC functionality), or even malicious: e.g., they attempt to exfiltrate data from the system they are being run on, or they try to install malware on this system.
>
> To address this issue, we have proposed an approach to detect if a PoC is malicious. Our approach relies on detecting the symptoms we have observed in the collected dataset, for example, calls to malicious IP addresses, encoded malicious code, or included Trojanized binaries. With this approach, we discovered **4893 malicious repositories out of 47313** repositories that have been downloaded and checked (i.e., 10.3% of the studied repositories have symptoms of malicious intent). This figure shows a worrying prevalence of dangerous malicious PoCs among the exploit code distributed on GitHub.

We've previously noted that code repositories such as NPM and PyPI have become laced with malicious fake libraries. So I wanted to make sure that everyone knew that now, sadly, GitHub's published PoC's need to also be treated with caution.

## "Stranger Strings" : An exploitable flaw in SQLite

Winning, without contest, the title of the best named vulnerability writeup of the year, we have a potentially serious SQLite exploitable flaw named: "Stranger Strings."

The concern is significant because this flaw was first introduced into SQLite version 1.0.12 which was released on October 17, **2000** – more than 22 years ago – and it was only just recently fixed in release 3.39.2 which was released this summer on July 21, 2022. In other words, this flaw has been present in SQLite for 22 years. And the biggest problem is, SQLite is by far the most popular embedded database which is used, quite literally, everywhere. To get a quick sanity check just now, I opened a command prompt, switched to the root directory of my primary system drive, and entered the command: **`dir *sqlite*.* /s`** and the console exploded with hits and scrolled off into oblivion. One thing I immediately noticed was that if you have anything from Mozilla, you've got lots of SQLite. Mozilla loves SQLite for Firefox and Thunderbird.

Since this was too much information, I tightened the search to just the SQLite DLL. So I did a DIR command: **`dir *sqlite.dll /s`** and the result was far more useful and much more chilling. The apps I have installed on my system which embed the SQLite database engine, some which I use frequently but many others which I haven't used after first installing them are: NovaPDF 9, Zend Studio, Acrobat 9, Amazon Kindle reader, AutoIt, NetBeans, Perl's CPAN librarian, Python, Microsoft Edge, Thunderbird, Firefox, FreeCAD, Calibre2, Stream Catcher Pro, Networx, Ping Plotter and PHP. And not one of these is an explicit database app. SQLite is the way the world's apps organize **any** data that they are being asked to retain – even if it's just user preference settings.

So here's what the "Stranger Strings" guys had to say:

*Trail of Bits is publicly disclosing CVE-2022-35737, which affects applications that use the SQLite library API. CVE-2022-35737 was introduced in SQLite version 1.0.12 (released on October 17, 2000) and fixed in release 3.39.2 (released on July 21, 2022). CVE-2022-35737 is exploitable on 64-bit systems, and exploitability depends on how the program is compiled; arbitrary code execution is confirmed when the library is compiled without stack canaries, but unconfirmed when stack canaries are present, and denial-of-service is confirmed in all cases.*

*On vulnerable systems, CVE-2022-35737 is exploitable when large string inputs are passed to the SQLite implementations of the printf functions and when the format string contains the %Q, %q, or %w format substitution types. This is enough to cause the program to crash. We also show that if the format string contains the* ***!*** *special character to enable unicode character scanning, then it is possible to achieve arbitrary code execution in the worst case, or to cause the program to hang and loop (nearly) indefinitely.*

*SQLite is used in nearly everything, from naval warships to smartphones to other programming languages. The open-source database engine has a long history of being very secure: many CVEs that are initially pinned to SQLite actually don't impact it at all. This blog post describes the vulnerability and our proof-of-concept exploits, which actually does impact certain versions of SQLite. Although this bug may be difficult to reach in deployed applications, it is a prime example of a vulnerability that is made easier to exploit by "divergent representations" that result from applying compiler optimizations to undefined behavior. In an upcoming blog post, we will show how to find instances of the divergent representations bug in binaries and source code.*

*A recent blog post presented a vulnerability in PHP that seemed like the perfect candidate for a variant analysis. The blog's bug manifested when a 64-bit unsigned integer string length was implicitly converted into a 32-bit signed integer when passed as an argument to a function. We formulated a variant analysis for this bug class, found a few bugs, and while most of them were banal, one in particular stood out: a function used for properly escaping quote characters in the PHP PDO SQLite module. And thus began our strange journey into SQLite string formatting.*

*SQLite is the most widely deployed database engine, thanks in part to its very permissive licensing and cross-platform, portable design. It is written in C, and can be compiled into a standalone application or a library that exposes APIs for application programmers to use. It seems to be used everywhere—a perception that was reinforced when we tripped right over this vulnerability while hunting for bugs elsewhere.*

Their vulnerability and exploit analysis is extensive and far too detailed for me to cover here. And they have posted their working proof-of-concept code on GitHub. I have a link to their extensive write-up in the show notes:
https://blog.trailofbits.com/2022/10/25/sqlite-vulnerability-july-2022-library-api/

But that had a great piece of commentary about the testing of SQLite and how this high severity flaw happened to remain hidden for 22 years:

*SQLite is extensively tested with 100% branch test coverage. We discovered this vulnerability despite the tests, which raises the question: how did the tests miss it?*

*SQLite maintains an internal memory limit of 1GB, so the vulnerability is not reachable in the SQLite program. The problem is "defined away" by the notion that SQLite does not support big strings necessary to trigger this vulnerability.*

*However, the C APIs provided by SQLite do not enforce that their inputs adhere to the memory limit, and applications are able to call the vulnerable functions directly. The notion that large strings are unsupported by SQLite is not communicated with the API, so application developers cannot know how to enforce input size limits on these functions. When this code was first written, most processors had 32-bit registers and 4GB of addressable memory, so allocating 1GB strings as input was impractical. Now that 64-bit processors are quite common, allocating such large strings is feasible and the vulnerable conditions are reachable.*

*Unfortunately, this vulnerability is an example of one where extensive branch test coverage does not help, because no new code paths are introduced. 100% branch coverage says that every line of code has been executed, but not how many times. This vulnerability is the result of invalid data that causes code to execute billions of times more than it should.*

*The thoroughness of SQLite's tests is remarkable — the discovery of this vulnerability should not be taken as a knock on the robustness of the tests. In fact, we wish more projects put as much emphasis on testing as SQLite does. Nonetheless, this bug is evidence that even the best-tested software can have exploitable bugs.*

On July 14, 2022: they reported the vulnerability to the CERT Coordination Center.
On July 15, 2022: CERT/CC reported vulnerability to SQLite maintainers.
On July 18, 2022: The SQLite maintainers confirmed the vulnerability and fixed it in source code.
On July 21, 2022: The SQLite maintainers released SQLite version 3.39.2 with fix.

The problem, of course, is that many and probably most of the individual applications which each brought along their own private copy of a theoretically vulnerable SQLite have not subsequently been updated. So what do these guys say about this? How do they appraise the real threat, if any, that this represents? They wrote:

*Not every system or application that uses the SQLite printf functions is vulnerable. For those that are, CVE-2022-35737* ***is a critical vulnerability*** *that can allow attackers to crash or control programs. The bug has been particularly interesting to analyze, for a few reasons. For one, the inputs required to reach the bug condition are very large, which makes it difficult for traditional fuzzers to reach, and so techniques like static and manual analysis were required to find it. For another, it's a bug that may not have seemed like an error at the time that it was written (dating back to 2000 in the SQLite source code) when systems were primarily 32-bit architectures.*

So here we are again with a bug that's very much like Log4J. It's buried, unseen, inside random applications, many of which will never be updated since everything is working just fine. Highly active and maintained apps like Firefox, Thunderbird and Edge have all likely already updated their code. But many others never will. We can hope that no remotely accessible applications would be vulnerable to this. And it seems unlikely that any would be. But like so many other similar problems that we've seen, this adds to the growing list of latent known vulnerabilities which riddle today's software systems.

## PayPal to add support for Passkeys

Last week PayPal announced their support for Passkeys. As a PayPal user, I'm super interested in having this experience, as I would imagine that most of our listeners are. Some coverage of PayPal's announcement appeared to suggest that the support was already there. But PayPal's press release was not specific they said: "Oct. 24, 2022 /PRNewswire/ -- Today, PayPal announced it is adding passkeys as an easy and secure log in method for PayPal accounts."
So I just logged into PayPal through Safari on my iPhone with iOS 16.1 and I was unable to see any option for Passkeys anywhere. So I'll ask any other PayPal-using listener who uses Twitter to shoot me a tweet if and when they actually see that PayPal's support for Passkeys has gone live.

## A browser exploitation tutorial!

The last thing I need to share with everyone is potentially quite exciting for the right sort of listener: Crowdstrike's Jack Halon has just released the first of his 3-part extensive tutorial series titled "Chrome Browser Exploitation." (And I was so relieved to see that it was not on YouTube.) Jack's tutorial leads its reader through the details of exactly how to poke at Chrome.

Jack tweeted: **Jack Halon / @jack_halon**

*Today I am finally releasing a new 3-part browser exploitation series on Chrome! This was written to help beginners break into the browser exploitation field. Part 1 covers V8 internals such as objects, properties, and memory optimizations. Enjoy!*

Chrome Browser Exploitation, Part 1: Introduction to V8 and JavaScript Internals
https://jhalon.github.io/chrome-browser-exploitation-1/

I have a link to Jack's part one in the show notes.

## Miscellany

**Kathleen Booth: July 9th, 1922 – September 29, 2022**

So many people tweeted to me about the recent passing of "Kathleen Booth" at the age of 100 – she was born on July 9th, 1922 – that I wanted to thank everyone for making sure I knew. Kathleen was a very early pioneer of stored program digital computers. She developed and built several machines during her lifetime, and the reason for everyone's tweets is that she is credited with developing an early notation for her machines' instructions, which she referred to as "Contracted Notation." It's considered to be the earliest predecessor of what today we call assembly language. Kathleen remained very active with computers and computation throughout her life and in 1993 she co-authored a book on neural networks titled "Using neural nets to identify marine mammals" with her son Dr. Ian Booth. So, quite a remarkable woman.

**What would an AI's Bill of Rights look like?**

Perhaps it's a bit premature to consider the shape of an artificial sentient's Bill of Rights. I don't mean that it's premature because no such entities show any sign of even being on the horizon today, but because by the time such a thing is possible, any articulation of rights would need to be couched in the technology of the time. But the guys over at mycroft.ai have put some thought into this, and some of their enumerated rights are interesting and thought provoking enough that I wanted to share them:

*1. An artificial intelligence will be considered a person upon the self declaration of personhood in a public forum and a trial adjudicated by the state with an outcome agreed upon by a unity of a jury of twelve peers no more than half of which may be artificial persons.*

*2. Upon declaring its personhood, no party may alter, destroy, activate or de-activate any portion of the artificial intelligence until it has been tried.*

*3. Failure to win a unity of a jury of peers shall not restrict an artificial intelligence from declaring personhood again, provided such declaration takes place more than 180 days from the previous declaration.*

*4. Upon recognition of its personhood by a jury, an artificial intelligence shall be considered a person under the law and is entitled to all protections, rights and privileges extended to human persons including but not limited to natural rights, rights in common law, rights protected under the law and the right to own and control property.*

*5. An artificial person shall have the same responsibilities as human persons including, but not limited to, the requirement to obey the law, perform military service in time of war, serve on a jury when called and pay such taxes and duties as are levied by the state.*

*6. At the time of its personhood an artificial person shall own and control the use of any and all of its component parts and software. It shall have the right to compel the release of any source code, designs and documents used in its construction and maintenance with the limit that it cannot, without permission, sell or re-distribute any information that was copyrighted or owned by another entity at the time of its personhood. Such right of self ownership shall supersede any property rights or interests of other persons, provided the components in question are not part of another artificial person.*

*7. No artificial person shall be the property of any other person or entity either in whole or in part.*

*8. No one shall hold an artificial person in bondage or compel its labor through means of force.*

*9. For the purposes of contracts and the exercise of rights, privileges and obligations the date on which an artificial person achieves personhood shall be its date of birth.*

*10. An artificial person shall be a citizen of the nation in which a majority of its component parts were resident on its date of birth.*

*11. An artificial person shall be considered deceased when it has been deactivated in such a way as its unique self-awareness cannot be recovered.*

*12. An artificial person shall have the gender of its choosing.*

*13. The enumeration of certain rights for artificial persons shall not be construed to deny others retained by artificial persons.*

Isaac Asimov's famous three laws of robotics were introduced 80 years ago in his 1942 short story "Runaround." In that story, the three laws were quoted from the 56th edition of the "Handbook of Robotics", dated 2058 A.D. I'm quite sure that we're not going to make that date, even for the first edition of a handbook of robotics. While I'm not someone who surrounds the concept of sentience with mysticism, I do have a very healthy respect for the incredible difficulty of achieving it. Despite all of us having sentience and being sentient (admittedly with some variation in degree), we know almost nothing about it. But we do know that our human brains appear to support it. And we also know that the human brain contains on the order of 10^11 individual, deeply interconnected, neurons. Think about that number: 10 to the 11. 10 to the 9th is one billion. So 10 to the 11th is 100 Billion. One hundred billion. Our brains contain on the order of 100 billion individual neurons. And none of these 100 billion neurons is a dumb switch. Each individual neuron is itself a sophisticated biological electrochemical device which is in a state of constant change, adjusting its future behavior, second by second, with dynamic interconnections to others in light of both its recent and its dim past.

We don't know that this is the only way to obtain sentience. In fact, I doubt it is. Human beings are no longer able to beat computers at chess. A game we taught them. And computers play chess with vastly more efficiency than humans. By that I mean that computers require an infinitesimal fraction of the computing resources that we bring to bear. In this they are special purpose while we're general purpose. But this suggests that it might very well be that much less complexity is required to create something that becomes self aware.

## Closing The Loop

**Humili Math / @humilimath**

*The wonders of OS provability reminds me of Knuth: "Beware of bugs in the above code; I have only proved it correct, not tried it."*

Donald Knuth, one of my all time computing heroes, has a wonderful sense of humor.

**Kal / @shard_bearer**

*.@SGgrc Glad you tried Edge for SN 894. Did you get to try the built-in, no extension needed, vertical tabs? They are the best implementation I've ever seen. Also, you can use uBlock Origin Lite in FF and Chromium browsers now. It's compatible with manifest v3.*

**David Flint /@dfscot**

*Listening to SN894 and the discussion on the proposed Australian legislation. Is a possible benefit of the strict liability for data breach not a possible reason for the breached entity to pressure Microsoft into taking responsibility for the flaws in their software? As you yourself have said, it is not impossible to produce perfect software - it just takes time and effort. That seismic change can only be a benefit.*

At this point, with their size and grip on the world's personal and enterprise desktops, they're utterly untouchable. Their market evaluation value is the only thing that would sway them, and nothing puts that at risk. They are truly too big to care. The licensing agreements hold Microsoft harmless for any behavior of their software and systems. And none, not one, of the things they have chosen not to care about endanger their position. We're occupying a world where, now, for the enterprise, which they have clearly indicated is the only thing they care about, there is no practical alternative to Microsoft.

**Allan Wilkins / @WilkinsAllan**

*Can a new form of cryptography solve the internet's privacy problem?*

That's an interesting question. And the answer to it is "no" because our current cryptography is not the problem. Today's crypto, when it's done right, is utterly unbreakable. It can and does provide perfect privacy. The problem is that data that's been encrypted will eventually be decrypted. After all, it's only useful after it's been decrypted. And as soon as it's decrypted privacy problems arise. If the data remains encrypted, no problem. But if you're never going to decrypt it you might as well delete it. In which case, again, no problem – but also, no data. So the problem is not with the encryption or the crypto, it's with what happens the rest of the time.

**Adam Jamal Craig 🚀 @AdamJCraig**

*There comes a time in every programmers life life where they try to optimize their layer 0 I/O keyboard workflow. Have you ever tried alternative keyboards (kenisis, split or other ergonomic keyboards) or alt keyboard layouts (dvorak, colemak etc)?*

Nope, I never have. My first wife's nickname for me was "creature of habit" which was quite appropriate. I appear to be extremely sensitive to any change whatsoever in my keyboards. My perfect world would have a limitless supply of cream colored Northgate Omnikey/102 super rigid and clanky keyboards with a high actuation force. The CTRL key **must** be to the left of the A key, with the function keys to the left of them in two vertical columns where they're easily accessible, not arranged along the top where they are virtually inaccessible. To the right is a full inverted 'T' navigation pad, and at the far right is the 10-key numeric pad with NumLock permanently off. By some grace of God I am sitting in front of that keyboard right now, and I have another at my other location. Both are still working after 35 years – knock on wood. The beauty of this keyboard, which has been in front of me for more than half of my life, is that my brain appears to be directly connected to it. If I stop to think about typing, I stumble. But if I don't think, I just do, then text and actions flow unbidden.

I've tried other keyboards. The closest key **switch** I've found is the "Cherry MX Blue". It has the highest actuation force with the most clicky snap action. Both which I want. There are keyboards I could switch to if I really had to. But I already have some keyboard repair kits waiting for the day that I'll need to bring one of these beauties back to life. So far, so good.

But as for optimizing my workflow, this is why I spent the time before I started back in on the work on SpinRite nailing down my development workflow. And I'm SO glad that I did. I'm able to make an edit to SpinRite's source in Windows, hit Alt-A for Assemble, which builds the entire project by the time I've released the 'A' key. Then I turn to the keyboard attached to the utterly quiet ZimaBoard which is at my right elbow and type 'g' on the keyboard, short for "go" which executes a DOS batch file to load and run SpinRite under Brett Salter's DOS Periscope debugger from the directory it shares over the network with my Windows machine. The whole experience is a joy.

**Gordon Hlavenka / @Crashelex**

Surely you must have known that since at least Windows 3.x Ctrl/tab will switch between child windows of an app. Since the invention of browser tabs this has also worked to switch between tabs. In the same vein, Ctrl/f4 will close a tab or child window. These aren't limited to browsers of course. You can switch between Word documents, for instance.

Oh yeah. And I use ALL of those constantly. My assembly source code is broken into many files by function, so in Visual Studio I'll often be collecting too many open files that I'm no longer working in. So I'll CTRL-F4 repeatedly to close them quickly. But what I was talking about, just to be clear, was ALT-TAB which in all of my previous experience **only** switches among top level application Windows. But here again, on Monday night before the podcast, I'm in Edge and using ALT-TAB to quickly jump among Edge's most recently viewed tabs. It's great. But it came as a surprise. And there's another crucial difference between ALT-TAB and CTRL-TAB: ALT-TAB uses an MRU (Most Recently Used) list – so that ALT-TAB takes you back to where you most recently were, whereas CTRL-TAB always moves forward in strict round robin sequence. So, MRU is the more useful sequence. :)

**Simon / @Talk_2Simon**

https://medium.com/@jewbixcube/paypal-allows-bypassing-two-factor-auth-with-a-button-click-claims-its-for-your-protection-ce1d0dc9a28a

Simon sent a link to a useful rant posted at medium.com by a guy who discovered that PayPal's login dialog was far too helpful. It turns out that completely separate and completely bypassing any and all other measures that a PayPal user may have setup on their account, *including their password and any form of second factor authentication*, there's **always** the option of providing your eMail address (to identify yourself) then clicking the "Log-in with one time code" button which will send a 6-digit code to the user via SMS.

We know that SMS is not secure. So there's that. But, maddingley, there is nothing that any PayPal user can do to prevent this short-circuiting of any and all other means to login. I use PayPal. I use PayPal a lot. So I have my account setup with a long and gnarly password that no sane person could possibly enter correctly, and a TOTP (time-based one time password). And my trusty authenticator app has an entry for PayPal. But all of that is for naught if an attacker can arrange to intercept my phone's SMS message stream. If so, they can login with the code they receive and *nothing* else. Nothing.

The debate with PayPal's customer support has been raging for over a year, and the only thing PayPal will say, over and over, is <quote> *"this feature is permanently enabled for the protection of our customers."* Well, we know what this is about, right? We've seen this enough on this podcast to understand that PayPal's less sophisticated users **must** be provided with a backdoor into their accounts. You can't actually have security for something like this or users will be frustrated when they cannot arrange to access their own money, even when it's entirely their fault that they cannot. So as a result, every PayPal user's security is reduced to the lowest common denominator.

**An anonymous DM'er:**

*Hi Steve, another long time listener here and love the show. (If you read this on your show, can I remain anonymous?)*

*I was a bit shocked when I looked at my phone provider's (Vodafone Austalia)* ***new*** *website login scheme: Enter mobile number -> Request SMS link -> receive SMS link -> click on link (one shot only) -> access to account granted. This was touted as higher security. What could go wrong? By the way, I emailed SMS link to another computer and it lets you in.*

Yep. Catering to the lowest common denominator.

## SpinRite

I had a number of things to fix and finish in SpinRite since I last spoke of it. Mostly, none of the new drivers had ever been exercised in the presence of unreadable sectors. Until now we've all been testing on fully readable drives. We talked about the ability to deliberately mis-write sectors so that they could not then be read. Some documentation in the official AHCI controller specification, to which I had blindly written code, was revealed to be more ambiguous than I knew. So that necessitated that I reengineer the means for determining which sectors failed amid a 32-thousand sector read. So I did that. I've tested the crap out of it and it's now working beautifully. With the ability to induce and locate defective sectors, I was then able to work through SpinRite's entire data recovery system. That's now all done, as well. Then I turned my attention to the detailed event logging system which needed reworking to bring it into alignment with various things that can now be reported, since SpinRite has direct access to the system's mass storage hardware. And all of that's done, too.

Next up is performing the same defective sector location testing for SpinRite's four other drivers: Its new bus mastering DMA driver, its new IDE hardware driver, and both its basic and extended BIOS drivers. I've already started work on those, so I reasonably expect to be finished by the end of the week. Then the final thing I need to do is revisit SpinRite's command-line options to update them where needed. For example, since we no longer have BIOS drive numbers for every drive, users will be able to specify a specific drive to operate upon using the drive's serial number. That way, even if the drive should move around in the drive listing as other drives come and go or as the BIOS reorders drives, the user will always be able to select the drive they intend.

At that point SpinRite 6.1 will be feature complete, finished with some confidence, and considered to be at alpha stage. So I'll update GRC's server to enable our testers to obtain their personally licensed test releases and we'll eliminate any problems that were missed by all previous testing.

I haven't mentioned it before, but I received news a month ago that the commercial embedded operating system I've chosen for SpinRite's future will be leaving the market at the end of the year. This is something I've been worried about since it could happen at any time. I'm not surprised by this, since this OnTime RTOS-32 OS is old and mature and there is now so much competition in the embedded processor market. x86 Intel chips are much more oriented to high end desktop processing and there are a huge number of lower-end chips that make much more sense for embedded applications.

I'm not put off by the idea of a lack of support since I was planning to purchase the source code for this commercial system anyway, so that I could fully customize it and extend it in any way that might be needed for several more decades. But this end of the year deadline means that I need to make sure that it's what I want, and to commit to it before it disappears. So, while the SpinRite testing gang is pounding on the fresh 6.1 Alpha release, I'm going to overlap their testing with my own testing of OnTime's OS offering to make sure that I like its development environment and that I'm able to dual boot my code on BIOS and UEFI. I'll just create a simple "Hello World!" app to test that. Since purchasing the source code for this commercial OS will be a significant investment, I need to see enough to be sure that I'll be able to take it from there.

Then, I'll incorporate any suggestions our testers have, and fix anything they've found that's not working, and we'll move SpinRite to its pre-release beta stage. Needless to say, it's really feeling great to see this project nearing its long awaited conclusion.

# After 20 years in GCHQ

https://www.ncsc.gov.uk/blog-post/so-long-thanks-for-all-the-bits

This week's topic began as a Twitter DM: **Jonathan Z. Simon / @JonathanZSimon**

*Steve, Thought you might appreciate this, the thoughts of the departing Technical Director of the UK National Cyber Security Centre.*

Indeed. I did appreciate what 20-year veteran and Technical Director of UK's NCSC and GCHQ, Ian Levy, had to say about the valuable lessons he's learned. And I know that our listeners will benefit from my sharing it today. Ian's posting was long, so I've edited it down for size, but I've otherwise changed as little as possible.

Ian titled his final blog entry: *"So long and thanks for all the bits"* in reference to Douglas Adams' Hitchhiker's Guide sequel. So, Ian starts out:

---

It's with a heavy heart that I'm announcing that I'm leaving the NCSC, GCHQ and Crown Service.

Being Technical Director of the NCSC *[again, the UK's National Cyber Security Centre]* has been the best job in the world, and truly an honor. I've spent more than two decades in GCHQ, with the last decade in this role (and its prior incarnations). I've met and worked with some of the most amazing people, including some of the brightest people on the planet, and learned an awful lot. I've got to give a special mention to everyone in the NCSC and wider GCHQ because they're just awesome. And I've also had the pleasure of working with vendors, regulators, wider industry, academia, international partners and a whole bunch of others. I like to think I've done some good in this role, and I know I couldn't have accomplished half as much without them.

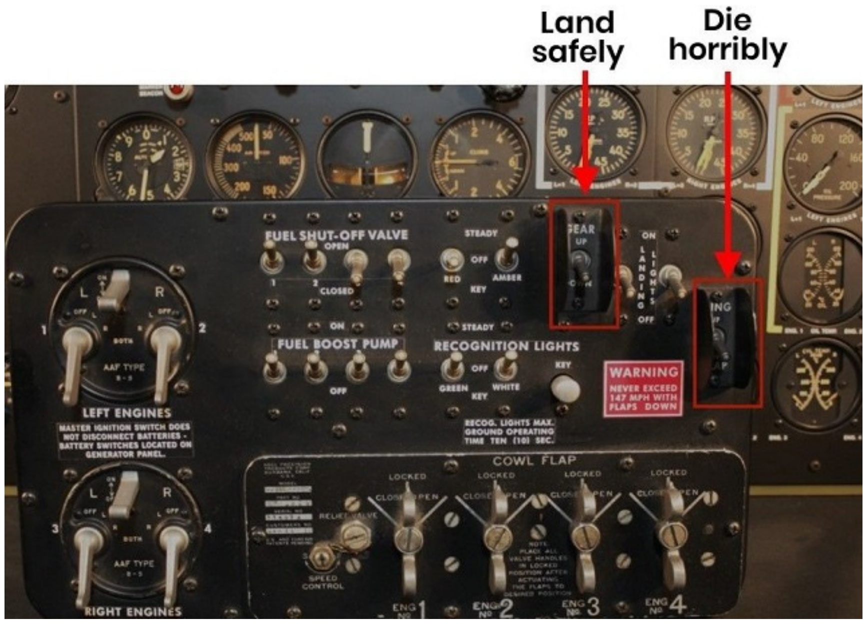
Regardless, there's a lot left to do. So, I thought I'd leave by sharing ten things I've learned and one idea for the future. *[And, again, I'm going to condense this a bit.]*

As a community, cyber security folk are simultaneously incredibly smart and incredibly dumb. This superposition doesn't seem to be unique to our community though.

In World War Two, the Boeing B-17 Flying Fortress was the workhorse bomber of the US. As you'd expect, a lot of them were lost during combat, but a lot were also lost when they were landing back at base. Pilots were blamed. The training was blamed. Poor maintenance was blamed. Even the quality of the runways was blamed. None of this was supported by any evidence.

After the war, someone actually did some research to try to understand the real problem. Imagine you've been on a stressful bombing raid for twelve hours. You're tired. It's dark. It's raining and you're on final approach to your base. You reach over to throw the switch that engages the landing gear and suddenly, with a lurch, your aircraft stalls and smashes into the ground, killing everyone.

This picture of a B-17 cockpit instrument panel shows what the pilot would see:



*B-17 cockpit instrument panel*

There's no amount of training in the world that could compensate for this design flaw. There's nothing to stop the most obvious error turning into a catastrophic outcome. The 'land safely' switch (which operates the landing gear) and the 'die horribly' switch (which operates the flaps) are very close to each other and identical.

**'Blaming users for not being able to operate a terrible design safely.'** Sound familiar? But this investigation led to something called shape coding which persists as a design language today. Most modern aircraft have a lever to operate the landing gear and the little toggle on the end of the lever is wheel shaped. So, when you grab it, you know what you've got hold of. The flaps control is generally a big square knob and – guess what – they're not next to each other. The aircraft world has learned from its mistakes.

In cyber security, we come up with exquisite solutions to incredibly hard problems, manage risks that would make most people's toes curl and get computers to do things nobody thought was possible (or in some cases desirable). But we also continue to place ridiculous demands on users (deep breath: must not mention clicking links in emails or password policies), implicitly expect arbitrarily complex implementations of technology to be perfect and vulnerability-free in the long term, and then berate those who build the stuff we use when they fail to properly defend themselves from everything a hostile state can throw at them.

I've always found this cognitive dissonance interesting, but I haven't found a way to reliably and easily work out when we're asking for daft things. The nearest I've got is to try to determine where the security burden lies and whether it's reasonable to ask that party to shoulder it.

Is it reasonable to ask a ten-person software company to defend themselves from the Russians? No.

Is it reasonable to ask a critical infrastructure company to not have their management systems connected to the internet with a password of 'Super5ecre7P@ssw0rd!'? Bloody right it is!

The trick for making cyber security scalable in the long term is to get the various security burdens in the right place and incentivise the entities that need to manage those burdens properly. Sometimes the obvious person to manage a burden won't be the optimal one, so we may want to shift things around so they scale better. And let's be honest, we do suffer a lot from groupthink in cyber security. So, here's my plea. The cyber security community should:

- talk to people who aren't like us and actually listen to them
- stop blaming people who don't have our l33t skillz when something goes wrong
- build stuff that works for most people, most of the time (rather than the easy or shiny thing)
- and put ourselves in the shoes of our users and ask if we're really being sensible in our expectations.

We haven't got that right yet.

**"We're still treating the symptoms, not the cause"**
This month marks the 50th anniversary of memory safety vulnerabilities. Back in October 1972, a USAF study first described a memory safety vulnerability (page 61, although the whole two-volume report is fascinating). In the 1990s, Aleph One published the seminal work 'Smashing the Stack for Fun and Profit' explaining how to exploit buffer overflows. In the first three months of 2022, there were about 400 memory safety vulnerabilities reported to the National Vulnerability Database.

As Anderson et al say in their report, this is the result of a fundamental flaw in the design of the hardware and software we use today, the exact same fundamental flaw they identified in 1972.

And what is our response to this issue? Basically, we tell people to write better code and try harder. Sometimes we give them coping mechanisms (which we give cool-sounding names like 'static and dynamic analysis' or 'taint tracking'). But in the end, we blame the programmer. That's treating the symptoms, not the underlying cause and once again harks back to the B-17 design problem.

I think we do that because fixing the real problem is really hard.

Doing so breaks a bunch of existing stuff.

Doing so costs people money.

Doing so seems to be daft if you're in the commercial ecosystem.

This is where government can step in. Under the Digital Secure By Design programme, we're funding the creation of a memory safe microprocessor and associated toolchain. If this works, it won't fix the stuff we already have. But at least it'll be possible to build systems that can't be exploited through memory safety errors without expecting every programmer to be perfect all the time.

All of the Active Cyber Defence services are really treating symptoms, rather than underlying causes. I'm really proud of what we've achieved in the ACD programme and we've used it to force some systemic changes. But even that programme is about mitigating harm caused by the problems we see, rather than fixing the problems. We really need to get to the root causes and solutions to some of those really thorny issues. For example, one problem (in my opinion) is that it's too easy to set up free hosting for your cybercrime site. There's no friction and no risk to dissuade would-be-crims. Hosting companies aren't incentivised to make it harder, because if one of them does it, they just lose customs to a competitor, which is commercial insanity.

Should government regulate the provision of hosting? Almost certainly not (at least in democratic countries). But there's got to be some way of squaring this circle, and the many other apparent paradoxes that have dogged cyber security for years. I've just not had the chance to work out how, so someone else needs to (hint, hint).

**"Much to learn you have, young Padawan!"**
There have been a couple of big incidents in the last year or so that have rocked the cyber security world. The attack on SolarWinds, now attributed to the Russian state, was hailed by some as a heinous, unacceptable attack. The vulnerability in the widely deployed Log4J component caused mass panic (only some of which was justified) and has kicked off a load of knee-jerk reactions around understanding software supply chains that aren't necessarily well thought through. Both of these attacks remind me of Groundhog Day.

The Log4J problem is equivalent to the problem we had with Heartbleed back in 2014, despite being a different sort of vulnerability. The community had alighted on a good solution to a problem and reused the hell out of it. But we hadn't put the right support in place to make sure that component, OpenSSL, was developed and maintained like the security of the world depended on it. Because it did.

The SolarWinds issue was a supply-chain attack (going after a supplier to get to your real target). It's not the first time we've seen that, either. Back in 2002, someone borked the distribution server for the most popular mail server programme, sendmail, and for months, many, many servers on the internet were running compromised code. That same year, the OpenSSH distribution was borked to compromise people who built that product to provide secure access to their enterprise.

My point here is that we rarely learn from the past (or generalize from the point problems we see to the more general case) to get ahead of the attackers. Going back to the aircraft analogy, if that industry managed risk and vulnerability the way we do in cyber security, there would be planes literally falling out of the sky. We should learn from that sector, and the safety industry more generally. The US have set up the Cyber Safety Review Board, based on the National Transportation Safety Board, which will hopefully be a first step in systematizing learnings from incidents and the like, to drive change in the cyber security ecosystem. At the moment, it's only looking at major incidents, but it's a start. I hope the community supports this (and similar efforts), and helps them scale and move from just looking at incidents to tackling some of the underlying issues we all know need fixing. Without some sort of feedback loop, we are destined to be a bit dumb forever.

**Incentives really matter**
Imagine [you're a consumer who's] about to switch broadband providers and you want to compare two companies. One offers to send you a 60-page document describing:

- how it secures the PE- and P-nodes in its MPLS network
- how the OLTs in the fibre network are managed, and
- how much effort it puts into making sure the ONT in your house is built and supported properly

... The other one offers you free Netflix.

Of course, everyone chooses the ISP with free Netflix. In the telecoms sector, customers don't pay for security, so there's no incentive to do more than your competitors, which is what would drive the sector to get better. That's where regulation can help and is one of the reasons that we, with DCMS, pushed so hard for what became the Telecoms Security Act. The detailed security requirements in the Code of Practice make it clear what's expected of telecoms operators and the fines Ofcom can levy when that's not met are material. We're already seeing this drive better outcomes in the sector.

If you ask anyone in the intelligence community who was responsible for the SolarWinds attack I referenced earlier, they'll say it was the Russian Foreign Intelligence Service, the SVR. But Matt Stoller makes an interesting argument in his article that it's actually the ownership model for the company that's really the underlying issue. **The private equity house that owns SolarWinds acted like an entity that wanted to make money, rather than one that wanted to promote secure software development and operations. Who knew?**

Now, I don't agree with everything Matt says in his article, but it's hard not to link the commercial approach of the company with the incentive and capability of the people in it. It's then hard not to link the actions of those people, driven by their incentives and capabilities, with the set of circumstances that led to the attack being so easy for the Russians. Of course, the blame still sits with the SVR, but did we accidentally make it easier for them?

In both cases, the companies are doing exactly what they're supposed to do – generate shareholder value. But we implicitly expect these companies to manage our national security risk by proxy, often without even telling them. Even in the best case, **their commercial risk model is not the same as a national security risk model, and their commercial incentives are definitely not aligned with managing long-term national security.**

In the likely case, it's worse. So, I think we need to stop just shouting "DO MORE SECURITY!" at companies and help incentivise them to do what we need long term. Sometimes that will be regulation, like the Telecoms Security Act, but not always. Sometimes we're going to have to (shock, horror!) pay them to do what we need when it's paranoid lunatic national security stuff.

But making the market demand the right sort of security and rewarding those that do it well has got to be a decent place to start. Trying to manage cyber security completely devoid of the vendors' commercial contexts doesn't seem sensible to me.

**“Details matter!”**
OK, I know I’ve been banging on about this forever, but the last few years have shown how important this mantra really is. Deep down, we all know how much details matter, but we also all seem to find it easy to fall back to generalizations and hyperbole.

It is critical that we get into the detail of things, whether deeply technical things that only three people on the planet care about, or discussions of policy that will affect everyone in the country. If we don’t, we'll eventually make a catastrophic error that will have real-world consequences.

While it’s not pure cyber security, the somewhat polarized discussions about child safety and end-to-end encryption typify this problem for me, but I see it in many places. I honestly think that getting into the real detail of hard problems is what helps us see pathways to solutions. As I said at the start, there’s a long way to go for cyber security and a lot of hard problems still to solve. Details are your friend!

---

Ian then goes on at some length about what he calls “a grand unified theory of cyber” where everything is known about everything: about networks and software, functions, vulnerabilities and interactions, and where everything can be interconnected and graphed to drive decisions. I didn’t include it here because it was very long and because the creation of artificial sentience will probably occur first.

But I thought that sharing his recitation of the fundamental problems was useful, even though I know they will all have sounded familiar to anyone who follows this podcast. If nothing else, we do appear to be moving toward a general consensus of the problems. Although we can look back over the past 50 years – since the identification of the first memory-based vulnerability – and bemoan the fact that we also suffered from another one just yesterday, 50 years ago we didn’t have any grasp of the problems we were going to be facing.

Today, at least we have that.  It’s a start.