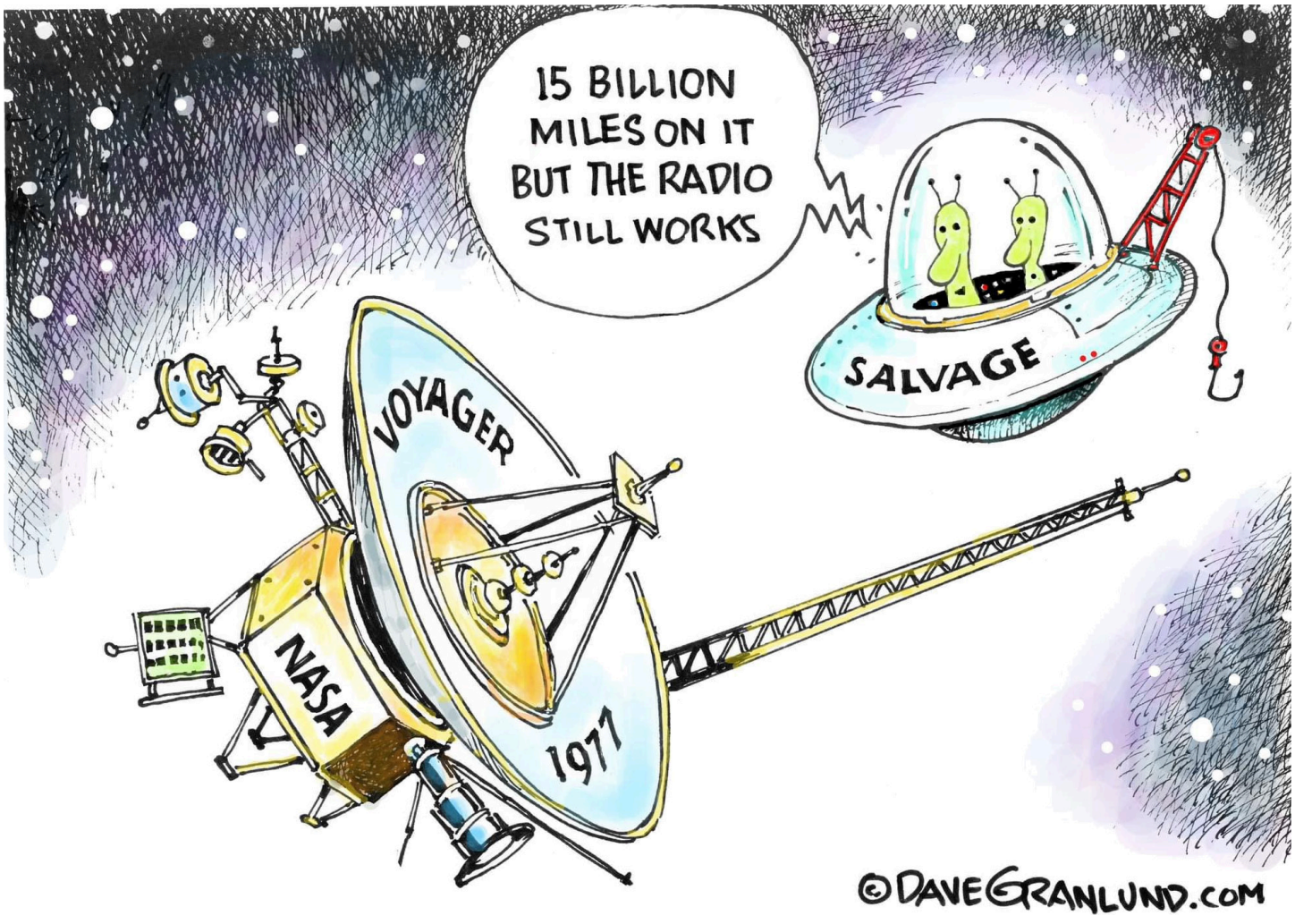
Security Now! #972 - 04-30-24

Passkeys: A Shattered Dream?

This week on Security Now!

The choice for this week's main topic received some serious competition from some surprising legislation that came into effect yesterday in the United Kingdom. So we're going to start by taking a close look at what happened in the UK that promises to completely change the face of consumer IoT device security. As we'll see, that's not an overstatement; the world as we've known it just changed. While that exploration is going to consume most of the first half of today's podcast, I also want to look at what happened last week with Chrome's change of plan regarding 3rd-party cookies, I have a bit of listener feedback to share, and news of the next installment in a long-running science fiction book series. I also have the welcome news that I am finally working on bringing up GRC's eMail communications system. Then we'll finish by taking a look at a blog posting by an industry insider that many of our listeners forwarded to me asking "what do you think about this?".



Security News

GCHQ: No more default passwords for consumer IoT devices!

Yesterday, April 29th, 2024, a new law went into effect in the UK and it appears that some of the right people are starting to get the message. The Guardian's coverage of this important milestone was titled: *"No more 12345: devices with weak passwords to be banned in UK"* with the subhead: *"Makers of phones, TVs and smart doorbells legally required to protect devices against access by cybercriminals."* Wow! But for a change, The Guardian's coverage was less informative than the UK government's own announcement. The GCHQ's National Cyber Security Centre blob posted yesterday was titled *"Smart devices: new law helps citizens to choose secure products"* and it continued...

*From 29 April 2024, manufacturers of consumer 'smart' devices **must** comply with new UK law. The law, known as the Product Security and Telecommunications Infrastructure act (or PSTI act), will help consumers to choose smart devices that have been designed to provide ongoing protection against cyber attacks.*

The law means manufacturers must ensure that all their smart devices meet basic cyber security requirements. Specifically:

- *The manufacturer must not supply devices that use default passwords, which can be easily discovered online, and shared. If the default password is used, a criminal could log into a smart device and use it to access a local network, or conduct cyber attacks.*
- *The manufacturer must provide a point of contact for the reporting of security issues which – if ignored – could make devices exploitable by cyber criminals.*
- *The manufacturer must state the minimum length of time for which the device will receive important security updates. When updates are no longer provided, devices are easier to hack, or may stop working as designed.*

*Although most smart devices are manufactured outside the UK, the PSTI act also applies to **all organizations importing or retailing products for the UK market.** Failure to comply with the act is a criminal offense, with fines up to £10 million or 4% of qualifying worldwide revenue (whichever is higher).*

*The law applies to **any** 'consumer smart device' that connects either to the internet, or to a home network (for example by WiFi). This may include:*

- *smart speakers, smart TVs and streaming devices*
- *smart doorbells, baby monitors and security cameras*
- *cellular tablets, smartphones and games consoles*
- *wearable fitness trackers (including smart watches)*
- *smart domestic appliances (such as light bulbs, plugs, kettles, thermostats, ovens, fridges, cleaners and washing machines)*

The NCSC has produced a 'point of sale' (POS) leaflet for retailers to distribute in-store to their customers. It explains how the PSTI regulation affects consumers, and why it's important to choose smart products that protect against the most common cyber attacks.

The first thing I need to say to this is "Holy Crap!" — where did that come from?! Fines in the amount of the greater of £10 million or 4% of a manufacturer's qualifying worldwide revenue. THIS is the sort of legislation that can ***really*** make a profound overnight difference in consumer security. And since a great many manufacturers have shown through their actions that they need to be ***made*** to change, this is the change that's required to make them.

Since this is huge and potentially affects all products worldwide which might find their way to the UK because it impacts not only manufacturers but anyone who imports or retails such products, I wanted to get a bit more backstory... so I did a bit of digging. On the gov.uk website I found the actual legislation. It turns out that it's been quietly in the works for several years and it really is a law, not just the sort of watered-down milk toasty "recommendations" that we see too often in the U.S..

The Verge picked up on this news and in their coverage they noted that here in the United States, our FCC is trying something similar with its forthcoming "Cyber Trust Mark" program. They liken it to the federal Energy Star program, explaining that the Cyber Trust Mark logo indicates which products comply with the program's recommendations, which include strong default passwords. But like Energy Star, nobody is forcing companies to go along with it. And consumer product packaging has become so encrusted with certification and compliance logos that it's unclear whether anyone even notices. Consumers are focused upon three things: Does it do what I need?, what does it cost?, and what additional nice-to-have features does it offer? Whether or not a connected light switch has a default password is the last thing on anyone's mind. In other words, while the United **States** continues to be completely lame on this, the United **Kingdom** has taken the only action that has any chance of actually producing results. And, thanks to the fact that we still live in a blessedly globalized economy, everyone everywhere will obtain the security benefits that the **Kingdom** is now requiring as a matter of law.

Since this matters to everyone everywhere, and since it's going to change the face of Internet connected consumer technology, let's take a close look at what the legislation actually says. First of all, this comes from a non-profit organization, the European Telecommunications Standards Institute, or ETSI. This work has been underway for the past five years, having started back in February of 2019 with the publication of v1.1.1 and it's been quietly making its way forward year by year. The Baseline Requirements document is the one that's of most interest. It's a 34-page PDF that I've given the GRC shortcut of ETSI. So if you're curious to see more you can put grc.sc/etsi into your browser and you'll be bounced over to a document titled: *"Cyber Security for Consumer Internet of Things: Baseline Requirements"*. The document's introduction explains:

> *As more devices in the home connect to the Internet, the cyber security of the Internet of Things (IoT) becomes a growing concern. People entrust their personal data to an increasing number of online devices and services. Products and appliances that have traditionally been offline are now connected and need to be designed to withstand cyber threats.*
>
> *The present document brings together widely considered good practice in security for Internet-connected consumer devices in a set of high-level outcome-focused provisions. The objective of the present document is to support all parties involved in the development and manufacturing of consumer IoT with guidance on securing their products.*
> *The provisions are primarily outcome-focused, rather than prescriptive, giving organizations the flexibility to innovate and implement security solutions appropriate for their products.*

*The present document is not intended to solve all security challenges associated with consumer IoT. It also does not focus on protecting against attacks that are prolonged or sophisticated or that require sustained physical access to the device. Rather, the focus is on the technical controls and organizational policies that matter **most** in addressing the **most** significant and widespread security shortcomings. Overall, a baseline level of security is considered; this is intended to protect against elementary attacks on fundamental design weaknesses (such as the use of easily guessable passwords).*

The present document provides a set of baseline provisions applicable to all consumer IoT devices. It is intended to be complemented by other standards defining more specific provisions and fully testable and/or verifiable requirements for specific devices which, together with the present document, will facilitate the development of assurance schemes.

Many consumer IoT devices and their associated services process and store personal data, the present document can help in ensuring that these are compliant with the General Data Protection Regulation (GDPR). Security by design is an important principle that is endorsed by the present document.

My goal for today's discussion of this is to accurately convey how comprehensive this document and the legislation that backs it up, actually is. Here's the list of its main topics:

- No universal default passwords
- Implement a means to manage reports of vulnerabilities
- Keep software updated
- Securely store sensitive security parameters
- Communicate securely
- Minimize exposed attack surfaces
- Ensure software integrity
- Ensure that personal data is secure
- Make systems resilient to outages
- Examine system telemetry data
- Make it easy for users to delete user data
- Make installation and maintenance of devices easy
- Validate input data
- Data protection provisions for consumer IoT

Each of these major topics is broken down into multiple pieces and each of those pieces is tagged with one of four possible requirement levels: Mandatory, Recommended, Conditionally Mandatory or Conditionally Recommended.

The "Scope" of what the document covers is also very clearly laid out and no one gets a free pass here. It says:

The present document specifies high-level security and data protection provisions for consumer IoT devices that are connected to network infrastructure (such as the Internet or home network) and their interactions with associated services. The associated services are out of scope. A non-exhaustive list of examples of consumer IoT devices includes:

• connected children's toys and baby monitors;
• connected smoke detectors, door locks and window sensors;
• IoT gateways, base stations and hubs to which multiple devices connect;
• smart cameras, TVs and speakers;
• wearable health trackers;
• connected home automation and alarm systems, especially their gateways and hubs;
• connected appliances, such as washing machines and fridges; and
• smart home assistants.

The present document provides basic guidance through examples and explanatory text for organizations involved in the development and manufacturing of consumer IoT on how to implement those provisions. Table B.1 provides a schema for the reader to give information about the implementation of the provisions. Devices that are not consumer IoT devices, for example those that are primarily intended to be used in manufacturing, healthcare or other industrial applications, are not in scope of the present document.

The present document has been developed primarily to help protect consumers, however, other users of consumer IoT equally benefit from the implementation of the provisions set out here. Annex A (informative) of the present document has been included to provide context to clauses 4, 5 and 6 (normative). Annex A contains examples of device and reference architectures and an example model of device states including data storage for each state.

The document does differentiate something it refers to as “constrained devices”, writing:

The present document addresses security considerations specific to constrained devices. EXAMPLE: Window contact sensors, flood sensors and energy switches are typically constrained devices.

To give everyone a sense for how well thought-out and specific this is, here’s the document’s definition of what’s means by a “constrained device”:

Constrained device: device which has physical limitations in either the ability to process data, the ability to communicate data, the ability to store data or the ability to interact with the user, due to restrictions that arise from its intended use

NOTE: Physical limitations can be due to power supply, battery life, processing power, physical access, limited functionality, limited memory or limited network bandwidth. These limitations can require a constrained device to be supported by another device, such as a base station or companion device.

EXAMPLE 1: A window sensor's battery cannot be charged or changed by the user; this is a constrained device.

EXAMPLE 2: The device cannot have its software updated due to storage limitations, resulting in hardware replacement or network isolation being the only options to manage a security vulnerability.

EXAMPLE 3: A low-powered device uses a battery to enable it to be deployed in a range of locations. Performing high power cryptographic operations would quickly reduce the battery life, so it relies on a base station or hub to perform validations on updates.

EXAMPLE 4: The device has no display screen to validate binding codes for Bluetooth pairing.

EXAMPLE 5: The device has no ability to input, such as via a keyboard, authentication information.

*NOTE: A device that has a wired power supply and can support IP-based protocols and the cryptographic primitives used by those protocols is **not** constrained.*

The document is too long and detailed for me to go through in detail here, but I want to again give everyone a sense for how thorough and serious this is. So, for the crucially important case which the document labels “No universal default passwords”, they say:

Where passwords are used all consumer IoT device passwords shall be unique per device or defined by the user. There are many mechanisms used for performing authentication, and passwords are not the only mechanism for authenticating a user to a device. However if they are used, following best practice on passwords is encouraged.

Many consumer IoT devices are sold with universal default usernames and passwords (such as "admin, admin") for user interfaces through to network protocols. Continued usage of universal default values has been the source of many security issues in IoT and the practice needs to be discontinued. The above provision can be achieved by the use of pre-installed passwords that are unique per device and/or by requiring the user to choose a password that follows best practice as part of initialization, or by some other method that does not use passwords. For example, during initialization a device generates certificates that are used to authenticate a user to the device via an associated service like a mobile application.

To increase security, multi-factor authentication, such as use of a password plus OTP procedure, can be used to better protect the device or an associated service. Device security can further be strengthened by having unique and immutable identities.

Where pre-installed unique per device passwords are used, these shall be generated with a mechanism that reduces the risk of automated attacks against a class or type of device.

Pre-installed passwords must be sufficiently randomized. Passwords with incremental counters (such as "password1", "password2" and so on) are easily guessable. Further, using a password that is related in an obvious way to public information (sent over the air or within a network), such as MAC address or Wi-Fi SSID, can allow for password retrieval using automated means.

Authentication mechanisms used to authenticate users against a device shall use best practice cryptography, appropriate to the properties of the technology, risk and usage. Where a user can authenticate against a device, the device shall provide to the user or an administrator a simple mechanism to change the authentication value used.

When the device is not a constrained device, it shall have a mechanism available which makes brute-force attacks on authentication mechanisms via network interfaces impracticable. For example, a device has a limitation on the number of authentication attempts within a certain

time interval. It also uses increasing time intervals between attempts. Or the client application is able to lock an account or to delay additional authentication attempts after a limited number of failed authentication attempts. This provision addresses attacks that perform "credential stuffing" or exhaust an entire key-space. It is important that these types of attacks are detected by the consumer IoT device and defended against, whilst guarding against a related threat of "resource exhaustion" and denial of service attacks.

What I just summarized is broken into five individual provisions in the document, but each and every one of them is tagged as "Mandatory". So, for example, if a device offers password-based authentication it can no longer be shipped from the factory with a generic default password and the device must also incorporate proactive defenses against brute force and credential stuffing attacks. It must incorporate some form of lock-out mechanism.

This legislation changes everything. It takes the well-understood but still not often implemented best practice from the high-end enterprise level and mandates its use for a residential doorbell. This is huge.

Section 5.3 is titled "Keep software updated" and picking some bits from it, it says:

Developing and deploying security updates in a timely manner is one of the most important actions a manufacturer can take to protect its customers and the wider technical ecosystem. It is good practice that all software is kept updated and well maintained.

All software components in consumer IoT devices should be securely updateable. When the device is not a constrained device, it shall have an update mechanism for the secure installation of updates. "Securely updateable" and "secure installation" means that there are adequate measures to prevent an attacker misusing the update mechanism.

Measures can include the use of authentic software update servers, integrity protected communications channels, verifying the authenticity and integrity of software updates. It is recognized that there are great variances in software update mechanisms and what constitutes "installation". An anti-rollback policy based on version checking can be used to prevent downgrade attacks. Update mechanisms can range from the device downloading the update directly from a remote server, transmitted from a mobile application or transferred over a USB or other physical interface. If an attacker compromises this mechanism, it allows for a malicious version of the software to be installed on the device.

An update shall be simple for the user to apply. The degree of simplicity depends on the design and intended usage of the device. An update that is simple to apply will be automatically applied, initiated using an associated service (such as a mobile application), or via a web interface on the device. If an update is difficult to apply, then that increases the chance that a user will repeatedly defer updating the device, leaving it in a vulnerable state.

Automatic mechanisms should be used for software updates. If an automatic update fails, then a user can, in some circumstances, no longer be able to use a device. Detection mechanisms such as watchdogs and the use of dual-bank flash or recovery partitions can ensure that the device returns to either a known good version or the factory state.

Security updates can be provided for devices in a preventative manner, as part of automatic

updates, which can remove security vulnerabilities before they are exploited. Managing this can be complex, especially if there are parallel associated service updates, device updates and other service updates to deal with. Therefore, a clear management and deployment plan is beneficial to the manufacturer, as is transparency to consumers about the current state of update support.

In many cases, publishing software updates involves multiple dependencies on other organizations such as manufacturers that produce sub-components; however, this is not a reason to withhold updates. It can be useful for the manufacturer to consider the entire software supply chain in the development and deployment of security updates.

It is often advisable not to bundle security updates with more complex software updates, such as feature updates. A feature update that introduces new functionality can trigger additional requirements and delay delivery of the update to devices.

The device should check after initialization, and then periodically, whether security updates are available. If the device supports automatic updates and/or update notifications, these should be enabled in the initialized state and configurable so that the user can enable, disable, or postpone installation of security updates and/or update notifications.

It is important from a consumer rights and ownership perspective that the user is in control of whether or not they receive updates. There are good reasons why a user may choose not to update, including security. In addition, if an update is deployed and subsequently found to cause issues, manufacturers can ask users to not upgrade their software in order that those devices are not affected.

The device shall use best practice cryptography to facilitate secure update mechanisms. Security updates shall be timely. "Timely" in the context of security updates can vary, depending on the particular issue and fix, as well as other factors such as the ability to reach a device or constrained device considerations. It is important that a security update that fixes a critical vulnerability (i.e. one with potentially adverse effects of a large scale) is handled with appropriate priority by the manufacturer. Due to the complex structure of modern software and the ubiquity of communication platforms, multiple stakeholders can be involved in a security update.

So what we're talking about here amounts to nothing less than the forced and immediate maturation of more than a decade of lazy consumer product security design. End user security is finally being prioritized over device development and support costs and convenience. It had to happen sometime and all indication is that sooner or later it was going to need to be forced. That happened yesterday.

What happened with Chrome and 3rd-party cookies?

While we were recording last week's podcast, the news dropped that Google's plans for Chrome's full phase-out of 3rd party cookies would not be occurring this year, as planned. For this week's podcast I had hoped to follow-up on that news to learn and then report on what was going on. The beginning of my research into Google's interactions with the UK's CMA, their Competition and Markets Authority, suggested that everything is actually going quite well overall, only somewhat slower than expected. My preliminary examination suggested that the UK's concerns are more along the lines of whether Google's new system goes far enough. Where concerns have

previously been raised that smaller advertisers may be disadvantaged, the UK appears to now be discounting those concerns and complaints.

But I ran out of time for research and we've run out of space for today's podcast. So I'll be following up next week with a much better understanding of what's going on.

Closing The Loop

Guillermo García / @gmogarciag

Hi Steve, Listening to the feedback on the counter race condition, I wonder what would happen to a process that wants to increase the counter if the previous owner of the counter was switched out of context and did not return the ownership before being switched off. Would this active process get stuck and have to wait for the previous one to regain context and return it? Again, many thanks!

Many of our listeners reported that they found the discussion of object ownership within a multi-threaded environment very interesting. But at no point did I talk about what happens when something goes wrong. For example, notice that nothing actually prevents the shared counter from being incremented by a thread that doesn't first acquire ownership of the object. In this instance there is no enforcement. It's all and only by agreement among the process's threads. And since they are all part of the same process, it's in their best interest to abide by the rules.

But it doesn't take a deliberate act to mess something up. A typical bug in a complex multi-threaded environment is that a thread will acquire ownership then follow some code path that causes it to fail to release its ownership. At that point that counter can never be incremented again and all any thread that needs to may stall waiting for an object's ownership to be released.

Who among us, especially back in the early days of Windows, has not had an application lock up and freeze. Or its menuing UI stops responding. Or the app apparently dies and refuses to do anything more? Some things might still be functioning whereas other things suddenly become non-responsive. One of the most common causes of such things happening is that, somehow, the ownership of a shared object was not freed by its owner. Threads can sometimes get into trouble. If a thread attempts to divide by zero, that thread will be terminated by the operating system. Or if a thread mistakenly attempts to execute some data, an illegal instruction can be encountered and, again, the thread will be immediately killed. In any event, if that thread happened to own some shared objects at the time of its termination, they would likely not be freed and other threads, or even a re-spawn of the terminated thread, might then be unable to succeed after that. Shutting down and restarting the application might be the only way to clear out stuck ownership. Not surprisingly, many solutions for these sorts of problems have been created over time. For example, there's a system known as "Structured Exception Handling" which allows a thread to protect the system and itself from their own possible misdeeds. I've implemented Structured Exception Handling in assembler and I've used it when my code had no choice, for some reason, other than to try to do something that might fail catastrophically. And there are entirely different ways to manage shared object ownership than the exchange instruction approach which I chose specifically to demonstrate the simplest possible solution.

Before I close out this conversation, I would be remiss if I didn't mention one of the classic problems with multi-threaded environments which is known as the "deadlock". A deadlock can be created when two threads each separately own an object but also need ownership of another object that the other owns. In other words, say that there are two objects, both of which need to be simultaneously owned by a thread in order to complete some work. One thread currently owns the first object and the second thread currently owns the second object, and each of them needs to obtain ownership of the object that the other one already has. Both threads will patiently wait for something that will never occur, since neither will relinquish its ownership of the object it owns until it, however briefly, obtains ownership of the object it needs. Neither will ever succeed and we have a classic deadlock.

But Guillermo's question highlights something else that I didn't talk about. He talked about multiple processes sharing objects, in other words, inter-process object sharing, whereas all of my discussion has been about multiple threads within a single process, intra-process object sharing. It is possible to share objects between processes. For example, Windows allows this by using names to identify the objects. Then separate processes that know the common name for an object can open the object to obtain a handle, very much like opening a file, after which operating system calls can be used to check the shared object's status, obtain and release ownership. And it's also possible to set timeouts while waiting for an object's ownership to be granted. If that amount of time passes, the object wait will be ended and the waiting process will be notified that the object never became available during the allotted time.

And even returning to our original example with the exchange instruction, remember that a thread that wants to obtain ownership attempts to obtain it and the result of the exchange instruction informs it whether or not it was successful. If it was not successful it is fully able to decide what to do next. It can go do other things and try again later. Or it might ask the operating system to put it to sleep for some length of time. That's an extremely friendly thing to do since the thread is voluntarily giving up the rest of its running time slice which allows the OS to schedule other threads. Then when the thread is re-awaked it can again attempt to obtain ownership.

This all fascinates me and I've never encountered anything as pure and clean and gratifyingly complex as coding. I know it's not for everyone. But if it is, it can be terrifically rewarding.

Vx-underground / @vxunderground

*Yesterday The New York Times unveiled that General Motor's had **accidentally** enrolled millions of people into its "OnStar Smart Driver+" program. If consumers chose to not enroll through the phone app – it would do it anyway. Unenrolling requires consumers to contact OnStar customer support line. However, some people do not trust them and have turned to stripping the electronic devices from their car. The OnStar Smart Driver+ data was being sold to LexisNexis, and insurance companies, to modify insurance rates.*

Sci-Fi

On the science fiction reading front, I wanted to mention to our many listeners who've been enjoying Ryk Brown's ongoing Frontiers Saga, that book 11 of 15 in his 3rd of 5, 15-book story arc became available yesterday in the US from Amazon. I received Ryk's announcement that his latest novel titled *The First Ranger* is now available for download in the US, though international availability may lag a bit, as is apparently common.

I've received so much feedback through the years from our listeners who've enjoyed following this adventure, that I wanted to make sure everyone knew that book 11 was here now. With the release of this latest book, there are now 41 full-length novels in the series.

Ryk's writing is primarily character driven. He offers us fully formed individuals with very distinct, and at times annoying, personalities. In that way it's a bit like Star Trek, where it's less about whiz-bang science fictional technologies than about how the various characters whom we've come to know well deal with what comes their way. I find it to be very satisfying. And in addition to many in this podcast audience, many of my friends and family members feel the same way.

For those who haven't ever looked at the series, it's available through Amazon's Kindle Unlimited plan and the novels are not expensive if just purchased outright. Here's what I do know: Anyone starting the first book will know within one hour whether they have just started into a journey that already has 40 additional books, each which is just as compelling, waiting for them.

SpinRite

On my own work front, last week I finished updating various GRC pages with the news that 6.1 was available. This is not 6.1's documentation which is still sorely needed. This is just enough to hold us over until I have eMail communication up and running, after which I'll plow into SpinRite's extreme need for documentation.

Since there's been such strong interest shown by those without access to a Windows machine for preparing SpinRite's boot media, I expanded upon GRC's BootAble page with instructions for Mac and Linux users. So, with that done, with 6.1 really working very well without anything known that can be fixed, I've finally turned my attention to implementing a modern email system for GRC... and I know, because I keep hearing it, how many of our listeners will be glad to finally have an alternative to Twitter.

Passkeys: A Shattered Dream?

Last Friday, a thoughtful posting by a guy named William Brown, the author of a popular WebAuthn package for RUST, generated significant attention within the security community. His Webauthn package is "webauthn-rs" which describes itself as "Webauthn Framework for Rust Web Servers". To remind everyone how and where Webauthn fits within the overall Passkeys solution, Webauthn is the protocol and specification that a Passkeys client on the user's side, uses to communicate with a web server that supports Webauthn. So, for example, just as a web server will offer some form of username and password login, possibly with additional factors such as TOTP (time-based) one time passwords or something else, such a server might also offer support for the Webauthn protocol as a means for allowing remote clients to identify and authenticate their identify over a network. In the case of this author's RUST implementation of Webauthn, he described Webauthn by writing:

> *Webauthn is a modern approach to hardware based authentication, consisting of a user with an authenticator device, a browser or client that interacts with the device, and a server that is able to generate challenges and verify the authenticator's validity.*
>
> *Users are able to enroll their own tokens through a registration process to be associated to their accounts, and then are able to login using the token which performs as a cryptographic authentication.*
>
> *This library aims to provide useful functions and frameworks allowing you to integrate webauthn into Rust web servers. This means the library implements the Relying Party component of the Webauthn/FIDO2 workflow. We provide template and example javascript and web asm bindings to demonstrate the browser interactions required.*

The only thing I'll note about what this author wrote, and it might be significant, is that this appears to have first been written back in the earlier FIDO1 era when hardware dongles were the only way the FIDO group was willing to roll. As we know, the requirement for purchasing a piece of hardware, while potentially ensuring greater security, was finally accepted to be a bar too high so the FIDO group capitulated to allow software-only systems the privilege of authenticating with what evolved into FIDO2. So my point is that the author of this Webauthn crypto library for RUST appears to have started this back at the hardware-only dongle stage of FIDO1 and he simply changed "FIDO1" to "FIDO2" in his introduction. This may be significant for what he subsequently wrote and published last Friday, since the introduction of FIDO2, with its accompanying Passkeys, promised to make this work far more relevant.

Before I share what he wrote, Friday, I wanted to note that the section following that brief introduction was titled "Blockchain Support Policy". That caught my eye because I thought "What? What does blockchain have to do with Webauthn". This author is apparently of the same opinion since under that heading he wrote:

> *This project does not and will not support any blockchain related use cases. We will not accept issues from organizations (or employees thereof) whose primary business is blockchain, cryptocurrency, NFTs or so-called "Web 3.0 technology".*

Amen, bother! There's been so much nonsense surrounding the "Blockchain will solve all of society's ills" – and especially within the identify authentication space – that it's easy to imagine how much of that this guy may have been fending off through the years. Elsewhere he notes that his library has passed a security audit performed by SUSE Linux's product security and that other security reviews are welcome. And as a total aside, I thought it was also interesting that on the topic of compatibility he wrote: "Known Supported Keys/Hardware: We have extensively tested a variety of keys and devices, not limited to:

- Yubico 5c / 5ci / FIPS / Bio
- TouchID / FaceID (iPhone, iPad, MacBook Pro)
- Android
- Windows Hello (TPM)
- Softtokens

And under "Known BROKEN Keys/Hardware" he notes

- Pixel 3a / Pixel 4 + Chrome - Does not send correct attestation certificates, and ignores requested algorithms. Not resolved.
- Windows Hello with Older TPMs - Often use RSA-SHA1 signatures over attestation which may allow credential compromise/falsification.

Okay. So Friday, he gave his blog posting the title that I re-used for today's podcast "Passkeys: A Shattered Dream" – although I added the question mark. His posting was not a rhetorical question. His was meant as a statement.

Before I share what William has written, I wanted to take a moment to note that in order to do justice to his choice of words, I'm going to again need to use a term on this podcast that makes me uncomfortable. Although the fact that the term was the American Dialect Society's Word of the Year for 2023 suggests that it's a term we're all destined to be encountering more and more often. That term is "enshittification". I'm somewhat eased about using it due to its lineage and the fact that Wikipedia does not shy away from devoting a rather extensive page to its definition, description and discussion with extensive examples of this happening. Wikipedia describes "enshittification" as:

> *"Enshittification is the pattern of **intentional decreasing quality** observed in online services and products such as Amazon, Facebook, Google Search, Twitter, Bandcamp, Reddit, Uber, and Unity. The term was used by writer Cory Doctorow in November 2022, and the American Dialect Society selected it as its 2023 Word of the Year. Doctorow has also used the term **platform decay** to describe the same concept."*

And, for what it's worth, allow me to commend this Wikipedia page to our listeners. I found it to be somewhat gratifying and affirming because while I was reading and agreeing with everything it said, I felt a bit less like the crotchety old timer yelling at kids to get off his lawn. In other words, objectively and, sadly, **deliberately**, some things actually are getting worse.

Also, not far into this, he refers to a system whose name is spelled “Kanidm” which its home page explicitly explains is pronounced “kar - nee - dee - em.” It’s a sprawling open source identity management platform developed in Rust by the SUSE Linux project. So it appears that William’s Webauthn library was adopted into that multi-faceted identify project to provide its Webauthn functionality. So when we hear him refer to “ker-nee–dee-um”, he’s refer to his library’s participation in that project.

So here’s what William Brown, a quite well-informed author of a high quality Webauthn library for RUST web servers wrote about Passkeys:

At around 11pm last night my partner went to change our lounge room lights with our home light control system. When she tried to login, her account could not be accessed. Her Apple Keychain had deleted the Passkey she was using on that site.

This is just the icing on a long trail of enshittification that has undermined Webauthn. I'm over it at this point, and I think it's time to pour one out for Passkeys. The irony is not lost on me that I'm about to release a new major version of webauthn-rs today as I write this.

(The Dream)

In 2019 I flew to my mate’s place in Sydney and spent a week starting to write what is now the Webauthn library for Rust. In that time I found a number of issues in the standard and contributed improvements to the Webauthn workgroup, even though it took a few years for those issues to be resolved. I started to review spec changes and participate more in discussions.

At the time there was a lot of optimism that this technology could be the end of passwords. You had three major use cases:

- *Second Factor*
- *Passwordless*
- *Usernameless*

Second Factor was a stepping stone toward the latter two. Passwordless is where you would still type in an account name then authenticate with PIN+Touch to your security key.

And usernameless is where the identity for your account was resident [and thus] discoverable on the key. This was (from my view) seen as a niche concept by developers since really - how hard is it for a site to have a checkbox that says "remember me"?

This library ended up with Kanidm being (to my knowledge) the very first OpenSource identiyu management platform to implement “passwordless” (which is now passkeys). The user experience was wonderful. You went to Kanidm, typed in your username and then were prompted to type your PIN and touch your key. Simple, fast, easy. For devices like your iPhone or Android, you would do similar - just use your Touch ID and you're in.

It was so easy, so accessible, I remember how it almost felt impossible. That authentication could be cryptographic in nature, but so usable and trivial for consumers. There really was the idea and goal within FIDO and Webauthn that this could be "the end of passwords".

This is what motivated me to continue to improve webauthn-rs. It's reach has gone beyond what I expected with parts of it being used in Firefox's authenticator-rs, a whole microcosm of Rust Identity Providers (IDPs) being created from this library and my work, and even other language's Webauthn implementations and password managers using our library as the reference implementation to test against. I can not understate how humbled I am by the influence webauthn-rs has had.

(The Warnings)

However, warnings started to appear that the standard was not as open as people envisioned. The issue we have is well known - Chrome controls a huge portion of the browser market, and development is tightly controlled by Google.

An example of the effect of this was the "Authenticator Selection Extension" of the WebAuthn specification. This specification extension is important for sites that have strict security requirements because the extension supports the attestation of the make and model of the authenticator in use. If you know that the website's attestation will only accept certain devices, then the browser should filter out and only allow those acceptable devices to participate.

However, Chrome never implemented it, that alone led to the entire feature being removed. It was removed because Chrome never implemented it. This demonstrates that if Chrome doesn't like something in the specification they can just veto it without consequence.

Later the justification for this not being implemented was: "We never implemented it because we don't feel that authenticator discrimination is broadly a good thing. ... they [users] should have the expectation that a given security key will broadly work where they want to use it."

I want you to remember this quote and its implications: Users should be able to use any device they choose without penalty.

Now I certainly agree with this notion for general sites on the internet, but within a business where we have policy around what devices may be acceptable, the ability to filter devices does matter.

This makes it possible to go to a corporate site, and apparently successfully enroll a security key, only to then have it fail to register (even better if this burns one of your resident key slots that can not be deleted without a full reset of your device). This might happen since the identity provider rejected the device's attestation. That's right, even without this, IDP's can still "discriminate" against devices without this extension, but the user experience is much worse, and the consequences far more severe in some cases.

The kicker is that Chrome has internal feature flags that they can use for Google's needs. They can simply enable their own magic features that control authenticator models for their policy, while everyone else has to have a lesser experience.

The greater warning here is that many of these decisions are made at "F2F" or Face to Face meetings held in the US. This excludes the majority of international participants leading some voices to be stronger than others. It's hard to convince someone when you aren't in the room, even more so when the room is in a country that has a list of travel advisories for foreign travelers including "Violent crime is more common in the US than in Australia", "There is a persistent threat of mass casualty violence and terrorist attacks in the US" and "Medical costs in the US are extremely high. You may need to pay up-front for medical assistance".

The points he's making here is that Google has outrageously outsized power to decide what does and does not succeed in the world due to their unilateral control of their Chrome browser. That which Chrome does not support, dies. And he's also observing something that might not ever occur to those of us who are happily camped out here in the United States, which is that, unfortunately, the US can apparently be somewhat frightening and expensive for volunteer open source developers wishing to have their voices heard from other countries. His point is, those voices are too easy for Google to ignore.

This brings to mind something that Stina Ehrensvärd often mentioned to me through the years. After founding Yubico in Sweden, she understood the critical importance of geographic location. So she deliberately uprooted her young family and relocated to Silicon Valley. She knew that if she was going to succeed she needed to be where the action was... and specifically to be able to attend face to face meetings with Google executives and others. In the list of authenticators on William's webauthn-rs site, Yubico's products are all mentioned first because when it mattered she was there in person... and also, truth be told, because, as both Leo and I know, it's often quite difficult to say 'no' to Stina!

Anyway, William continues…

> *(The Descent)*
>
> *In 2022 Apple announced Passkeys.*
>
> *At the time this was just a really nice "marketing" term for passwordless, and Apple's Passkeys had the ability to opportunistically be usernameless as well. It was, all in all, very polished and well done.*
>
> *But of course, thought leaders exist, and Apple hadn't defined what a Passkey was. One of those thought leaders took to the FIDO conference stage and announced "Passkeys are resident keys" while at the same time they unleashed a passkeys dev website.*
>
> *The issue is described in detail in another of my blog posts, but to summarize, this push to* ***resident keys*** *means that [physical hardware] security keys are excluded because they often have extremely low limits on storage, the largest being 25 for Yubikeys. That simply won't cut it for most people who have more than 25 accounts.*

And yes, I know, that by now many of our listeners are thinking "But SQRL doesn't have any of those problems!" Right. That's true. But neither does SQRL have any adoption, so who cares?

William then coins a term that Cory Doctorow might appreciate. He terms the period following the announcement of Passkeys "The Enshittocene Period", writing...

> *Since then, Passkeys are now seen as a way to capture users and audiences into a platform. What better way to encourage long term entrapment of users then by locking all their credentials into your platform, and even better, credentials that cannot be extracted or exported in any capacity.*

Both Chrome and Safari will try to force you into using either hybrid where you scan a QR code with your phone to authenticate. To use a hardware security key requires clicking through multiple menus. And even their default is not a good experience, taking more than 60 seconds work in most cases. The UI is beyond obnoxious at this point. Sometimes I think the password game has a better UX.

The more egregious offender is Android, which won't even activate your security key if the website sends the set of options that are needed for Passkeys. This means the identity provider gets to choose what device you enroll without your input. And of course, all the developer examples only show you the options to activate "Google Passkeys stored in Google Password Manager". After all, why would you want to use anything else?

A sobering pair of reads are the Github Passkey Beta and Github Passkey threads. There are instances of users whose security keys are not able to be enrolled as the resident key slots are filled. Multiple users describe that Android can not create Passkeys due to platform bugs. Some devices need firmware resets to create Passkeys. Keys can be saved on the client but not the server leading to duplicate account presence and credentials that don't work, or worse lead users to delete the real credentials.

The helplessness of users on these threads is obvious - and these are technical early adopters. The very users we need to be advocates for changing from passwords to passkeys. If these users cannot make it work how will people from other disciplines fare?

Externally there are other issues. Apple Keychain has personally wiped out all my Passkeys on three separate occasions. There are external reports we have received of other users whose Keychain Passkeys have been wiped just like mine. Consequently, as users we have the expectation that keys won't be created or they will have disappeared when we need them most.

In order to try to resolve this, the working group seems to be doubling down on more complex JS apis to try to patch over the issues that they created in the first place. All this extra complexity comes with fragility and more bad experiences, but without resolving the underlying core problems. It's a mess.

(The Future)

At this point I think that Passkeys will fail in the hands of the general consumer population. We missed our golden chance to eliminate passwords through a desire to capture markets and promote hype.

Corporate interests have overruled good user experience once again. Just like ad-blockers, I predict that Passkeys will only be used by a small subset of the technical population, and consumers will generally reject them.

To reiterate - my partner, who is extremely intelligent, an avid computer gamer and veterinary surgeon has sworn off Passkeys because the user experience is so crappy. She wants to go back to passwords.

And I'm starting to agree - a password manager gives a better experience than passkeys.

That's right. I'm here saying passwords are a better experience than passkeys. Do you know how much it pains me to write this sentence? And yes, that means multi-factor authentication

with time-based one time passwords is still important for passwords that require memorisation outside of a password manager.

So do yourself a favor. Get something like Bitwarden or if you like self hosting get Vaultwarden. Let it generate your passwords and manage them. If you really want passkeys, put them in a password manager you control. But don't use a platform-controlled passkey store, and be very careful with physical hardware security keys. And if you do want to use a security key, only use it to unlock your password manager and your email.

Within enterprise, there still is a place for attested security keys where you can control the whole experience to avoid the vendor lock-in parts. It still has rough edges though. Just today I found a browser that has broken attestation, which is not good. You still have to dive through obnoxious UX elements that attempt to force you through the default QR code path even though your identity provider will only accept certain security models, so you're still likely to have some confused users.

Despite all this, I will continue to maintain webauthn-rs and it's related projects. They are still important to me even if I feel disappointed with the direction of the ecosystem.

But at this point, in Kanidm ("kar-nee-dee-em") we're looking into device certificates and smartcards instead. The UI is genuinely better. Which says a lot considering the [state of the] PKCS11 and PIV specifications. But at least PIV won't fall prone to attempts to enshittify it.

PIV stands for "Personal Identity Verification" – it's a standardized physical smartcard system that's heavily used by government and military. The technology to create digital identity cards has been around for a long time, but they are so fraught with their own problems that they aren't an alternative to solve the web's authentication needs.

I think that, for me, the thing that's so sad is that Cory Doctorow's term, and the examples of Enshittification that Wikipedia documented, make very clear that these are deliberate usury outcomes. The shortest of Wikipedia's examples is what Uber did. Wikipedia writes:

App-based ridesharing company Uber gained market share by ignoring local licensing systems such as taxi medallions while also keeping consumer costs artificially low by subsidizing rides via venture capital funding. Once they achieved a duopoly with competitor Lyft, the company implemented surge pricing to increase the cost of travel to riders and dynamically adjust the payments made to drivers.

Nearly all of the problems William observed in his posting are things we've independently noted from the start as inherent problems with the way Passkeys have been rolled out. As I've observed on several occasions, the fact that passkeys were implemented in a non-portable way, as a vehicle for creating implicit platform lock-in, is almost a crime.

But the new thought that William proposes is something that had never occurred to me. Perhaps it's because I've been blinded by Passkeys' superior public key technology which offers so many potential authentication benefits. Even though the benefits are theoretical, I've never questioned whether or not Passkeys would eventually become the new standard for the web. But William writes: *"At this point I think that Passkeys will fail in the hands of the general consumer*

population. We missed our golden chance to eliminate passwords through a desire to capture markets and promote hype." When I read that the first time I was surprised. But at the same time, I have still not adopted passkeys. I've never registered a single passkey. I don't have even one, anywhere. I don't encounter websites offering passkey authentication, so there's that. But mostly, because authentication matters crucially to me, I want to feel that I'm in control of my authentication. And that starts with thoroughly and deeply understanding it. I do thoroughly and deeply understand passkey's underlying cryptography... but as William explains, what's then been done with that underlying crypto has been made deliberately opaque as a means of "just trust us" individual platform lock-in. The problem is, my authentication is far too important for me to entrust to any company that might choose to, dare I say, enshittify it.

Having unique, per-site, insanely long high-entropy passwords – that I can touch and feel and copy and paste and see – stored and managed by a cross-platform password manager, which is everywhere I need it to be, allows me to really understand the status of my authentication.

One thing I also have is a long and growing list of TOTP one-time passcodes. And the reason I'm absolutely comfortable with that is that, again, they're tangible things that I can control, see and understand.

Has the entire techie insider industry just been playing with itself this whole time? Have we been imagining that authentication can and should be made entirely invisible, because Passkeys can theoretically make that happen? Will end users who don't know anything about the underlying technology say "Well, I don't know how it works, but that was certainly easy!" ?? But then, what about when it doesn't work? What about when someone needs to logon from a device that's outside of the provider's walled garden? These are all problems we've previously identified and questions we've asked before. I've always assumed that this was just the typical extreme adoption inertia we always see. It never occurred to me that Passkeys might ultimately fail to **ever** obtain critical mass and to eventually become more dominant than passwords.

While poking around to get a broader perspective I encountered a recent piece in Wired titled "I Stopped Using Passwords. It's Great—and a Total Mess" with the intro: "Passkeys are here to replace passwords. When they work, it's a seamless vision of the future. But don't ditch your old logins just yet." The author explained that, as William said, things didn't always work. He also noted that having multiple clients all popping up and asking whether you want to save passkeys with them had become annoying. But the biggest problem he had was remembering where he had stored which passkey. As someone who spends some time pondering which of the multiple streaming providers carries the show my wife and I have been watching, that definitely resonated.

Our advice at the start of this passkeys saga was to wait until a single provider offered Passkeys support across every platform that might conceivably be needed, since passkeys portability wasn't something that anyone was even talking about back then. In fact, back then it was clearly an overt password lock-in move.

So I wanted to share the news that Bitwarden, the solution that William's posting referred to and a sponsor of the TWiT network, earlier announced on the 10th of this month that passkeys for the iPhone and Android clients had just entered Beta testing. So I'm very glad to see that my

chosen open source password manager will soon be offering passkey support.

Now what's going to be needed, based upon the experience of the author of the Wired article, will be the ability to assign a single passkey handler to a platform, much as we currently assign a handler for URL links. Having all of a platform's passkey-aware clients popping up solicitations to store a passkey with them seems like it would quickly become annoying. On the other hand, it doesn't happen often.

On balance, I still don't feel much pressure to give up my use of passwords since they're working perfectly for me. The other factor is that website login has become so persistent that I rarely need to re-authenticate to most sites. Each of the browsers I use carries a static cookie for each of the sites I frequent, so I'm already known everywhere I go.

So for the foreseeable future, I expect to hang back and wait. The dust is still settling on passkeys, and passkeys doesn't solve any problem **I** have today.