Security Now! #625 - 08-22-17

Security Politics

This week on Security Now!

This week we discuss the continuing Marcus Hutchins drama, the disclosure of a potentially important Apple secret, a super-cool website and browser extension our listeners are going to appreciate, trouble with extension developers being targeted, a problem with the communication bus standard in every car, an important correction from Elcomsoft, two 0-days in Foxit's PDF products, Lavalamps for entropy, the forthcoming iOS 11 TouchID killswitch, very welcome Libsodium audit results, a mistake in AWS permissions, a refreshingly forthright security statement, a bit of errata, miscellany, and a few closing the loop bits from our terrific listeners!

"**Every employee** has been implanted with biometric access RFID tags enabling secure access control to our extremely security-sensitive facility. It is **utterly** state-of-the-art, and **cannot** be penetrated!"

Security News

Marcus Hutchins Update:
One distressing piece of information we haven't covered here is that Marcus, who, as we know, pleaded NOT guilty, is facing six charges and up to 40 years in prison.

- *British spy chiefs knew of FBI sting on NHS hack attack hero Marcus Hutchins* https://www.thetimes.co.uk/article/british-spy-chiefs-knew-of-fbi-sting-on-nhs-hack-attack-hero-marcus-hutchins-hctlgbvrr

The UK's GCHQ was aware that Marcus Hutchins, who famously stopped the WannaCry worm that was decimating the UK's entire National Health Service IT infrastructure, was under investigation by the FBI *before* he travelled to America and that he would be walking into a trap that was being set for him before he was subsequently arrested by US authorities for alleged cyber-offences. Multiple reporting indicates that the British government allowed Marcus to wander into this trap because it saved them the headache of what would have then been a highly-charged extradition proceeding with their ally... the US.

So... what's developing in the case for and against Marcus?

Any time something important is happening, people will form opinions. Then those opinions evolve into positions, egos become engaged, and those positions start being defended... often beyond the point where they are supported by fact. It's the politics of humanity. And this is what's starting to develop in the interesting case of Marcus Hutchins.

During the past week there have been a number of new reports, some suggesting that Marcus WAS more deeply involved in the development of the Kronos banking malware, other arguing that the code was lifted from him by the Kronos malware authors, some even alleging that he was more directly involved with the creation of the WannaCry worm.

Some of the confusion surrounds the origins and refinement of a commonly employed programming technique known as "function hooking"and the need for a "trampoline"...

January 8th, 2015:
https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-1.html

[MARCUS] "A lot of my articles have been aimed at giving a high-level insight into malware for beginners, or those unfamiliar with specific concepts. Today I've decided to start a new series designed to familiarize people with malware internals on a programming level. This will not be a tutorial aimed towards people creating sophisticated malware, but security enthusiasts looking to better understand it.

Inline Hooking

What is it: Inline hooking is a method of intercepting calls to target functions, which is mainly used by antiviruses, sandboxes, and malware. The general idea is to redirect a function to our own, so that we can perform processing before and/or after the function does its; this could

include: checking parameters, shimming, logging, spoofing returned data, and filtering calls. Rootkits tend to use hooks to modify data returned from system calls in order to hide their presence, whilst security software uses them to prevent/monitor potentially malicious operations.

The hooks are placed by directly modifying code within the target function (inline modification), usually by overwriting the first few bytes with a jump; this allows execution to be redirected before the function does any processing. Most hooking engines use a 32-bit relative jump (opcode 0xE9), which takes up 5 bytes of space.

January 8th, 2015: "Inline Hooking for Programmers (Part 2: Writing a Hooking Engine)"
https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-2.html

[GIBSON] I have never seen that code before this morning. Yet I read it for the first time and I was COMPLETELY familiar with every aspect of it. Marcus may have felt that he invented it. Perhaps his was a subtle refinement of something he had seen elsewhere, or perhaps he's a gifted developer who solved the same problem and the same way and "co-invented" the same solution to the same problem.

The fact is, this problem has been encountered and solved many times before and there is often only one way to solve a specific problem.

Propel & Michael Toutonghi
Circa 1988 -- I have code dating back to 1988, probably somewhere on some 32 megabyte RLL drive, originally written by Michael Toutonghi, who I employed at the time.

And as another "predating" example, here is almost exactly the same code as Marcus', posted in March of 2009:
http://www.rohitab.com/discuss/topic/33771-patch-hook/?p=10062694

(Warning: F-Bomb ahead!)
February 7th, 2015: MalwareTech (@MalwareTechBlog)
"Just found the hooking engine I made for my blog in a malware sample.
This is why we can't have nice things, fuckers."
https://twitter.com/MalwareTechBlog/status/564175340667695104

Dan Goodin, writing for ArsTechnica: "Shortly after his arrest in Las Vegas two weeks ago, the Tweet resurfaced, and almost immediately it generated speculation that the malware Hutchins was referring to was Kronos. An analysis of Kronos soon showed that one portion used an instruction that was identical to one included in the code Hutchins published in January 2015.

The instruction in question is a monotonic "Compare and Exchange." There are separate compare instructions, and exchange instructions. But performing those two functions separately creates the possibility that a thread context switch might occur between them.

64-bit incrementation with a 32-bit chip and monotonic (indivisible) instructions.

In the single-core old days we would turn off "hardware interrupts" so that some block of code could not be reentered. But with multi-code hardware threads we need better solutions.

Windows "Critical Sections"

So that's Kronos... but what of WannaCry?

Dave Aitel, head of the security firm Immunity, had a different immediate response to the reports of marcus being "the hero of WannaCry". In a posting earlier this month Dave wrote:

[Aitel] But let me float my and others initial feeling when MalwareTech got arrested: The "killswitch" story was clearly bullshit. What I think happened is that MalwareTech had something to do with Wannacry, and he knew about the killswitch, and when Wannacry started getting huge and causing massive amounts of damage (say, to the NHS of his own country) he freaked out and "found the killswitch". This is why he was so upset to be outed by the media.

Being afraid to take the limelight is not a typical "White Hat" behavior, to say the least.

That said, we need to acknowledge the strategic impact law enforcement operations as a whole have on national security cyber capabilities, and how the lighter and friendlier approach of many European nations avoids the issues we are having here in the States.

Pretty much every infosec professional knows people who have been indicted for computer crimes now. And in most of those cases, the prosecution has operated in what is essentially an unfair, merciless way, even for very minor crimes. This has massive strategic implications when you consider that the US Secret Service and FBI often compete with Mandiant for the handling of computer intrusions, and the people making the decisions about which information to share with Law Enforcement have an extremely negative opinion of it.

In other words: Law Enforcement needs to treat hacker cases as if they are the LAPD prosecuting a famous actor in LA. Or at least, that's the smartest thing to do strategically, and something the US does a lot worse than many of our allies."

https://cybersecpolitics.blogspot.com.au/2017/08/the-killswitch-story-feels-like-bullshit.html

[GIBSON] We're all aware of the very real concern over a (hopefully) fictional highly virulent super-virus being developed purely for study, somehow later escaping from the lab into the wild and wreaking havoc upon humanity.

Now imagine a malware author who the recently disclosed weaponized EternalBlue technology believed to have been developed by the NSA... and having a deep technical background in malware operation, cannot resist experimenting with that technology and create an instance of a highly virulent worm which carries cryptomalware as its payload.

Because this researcher is not insane, he or she builds-in a killswitch just in case. Then... somehow, some way, it finds its out and escapes from its containment, exploding onto the public Internet. In this scenario, this wasn't what the researcher EVER intended. But it happened

nevertheless. So.. now what does he do? He "discovers" his own responsibly-installed killswitch and immediately registers the domain to shutdown the creation of his which escaped his control.

I dearly HOPE that's not what happened, because then WE are faced with a moral dilemma, while, as Dave noted above... U.S. and global law enforcement won't have any dilemma. They will throw the book at Marcus.

And note that, in this scenario, everything could be true at once: Marcus' publicly posted code could indeed have been lifted in whole and dropped into Kronos by Kronos' authors with no help from Marcus. AND, while at this point it is PURELY speculative "smell test" based, Marcus may have had some involvement in the development of WannaCry.

We won't know more until we see what evidence US law enforcement has.

Apple had been keeping a secret.
It's unclear, so far, how important keeping that secret had been. But we do know that cryptographic security does require some secret keeping.

The breakthrough, now quite some time ago in cryptographic maturity, occurred with the switch from keeping the algorithms secret to developing keyed algorithms which could be made public and then keeping specific instances of their usage keys secret. This DRAMATICALLY improved security by distributing and spreading the secret exposure risk. Before, if the secret algorithm were discovered every usage of it would simultaneously be compromised. But now, with keyed crypto, the disclosure of one key only compromises those uses of that key... and no one's use of the same algorithm under different secret keys.

So... Apple somehow lost control of their secret, Secure Enclave decryption key... and last Thursday it was published...

The key allows for the decryption and explore of the secure enclave processor's firmware in full detail. This is something Apple has tried to prevent. No necessarily because it represents the end of mankind as we know her, but, well, just because it's proprietary Apple code, it's no one's business, and it COULD BE (emphases on COULD) a unexpected treasure trove for curious security researchers and hackers alike. And the research community will doubtless be gaining more information about the iOS platform.

But I began this with a reminder about the nature of modern day crypto. As with all modern cryptographic systems, disclosure of the algorithm is not, per se, a huge concern.

So, there are two dangers:

The least bad, but perhaps the more likely (given the general history of complex unaudited code), would be that by combing through Apple's code with an adversarial posture -- which, as we have often discussed it is extremely difficult for Apple's own developers to ever really hold -- those with nefarious intent might discover some mistakes in the secure enclave processor's code that could be leveraged into a powerful exploit that Apple could do nothing to prevent.

But the biggest danger would be that the secure enclave processor firmware itself contains embedded private keys whose secrecy is crucial, and whose disclosure would then, indeed, represent a catastrophic information leak.

As HackADay writes: [The] SEP is the security co-processor introduced with the iPhone 5s which is when touch ID was introduced. It's a black box that we're not supposed to know anything about but [xerub] has now pulled back the curtain on that.

The secure enclave handles the processing of fingerprint data from the touch ID sensor and determines if it is a match or not while it also enables access for purchases for the user. The SEP is a gatekeeper which prevents the main processor from accessing sensitive data. The processor sends data which can only be read by the SEP which is authenticated by a session key generated from the devices shared key. It also runs on its own OS [SEPOS] which has a kernel, services drivers and apps. (And, yikes... it frightens me, because that means it's not a simple crypto engine, is a much more complex subsystem, the code for which is now public.) The SEP performs secure services for the rest of the SOC and much more which has been described in talks such as "Demystifying the Secure Enclave Processor" at Blackhat.

Xerub said: "Hopefully Apple will work harder now that they can't hide SEP, resulting in improved security for users."

What remains unknown is how Xerub obtained this presumably large key? My guess is that it's the same old problem we've discussed here over and over and over: Some secrets CAN be kept -- like a remote server's private key. But other secrets inherently cannot be kept, like a DVD's secret key which MUST be present and used to playback an encrypted DVD... OR, similarly, the secure enclave's firmware decryption key which MUST ALSO be at least transiently present to decrypt the processor's stored firmware in order for it to be executed and used.

A super-cool idea SN listeners are going to love!
TL;DR -> ToS;DR https://tosdr.org/
As the page's subhead reads: "I have read and agree to the Terms" is the biggest lie on the web. We aim to fix that. [SITE] We are a user rights initiative to rate and label website terms & privacy policies, from very good "Class A" to very bad "Class E".

Terms of service are often too long to read, but it's important to understand what's in them. Your rights online depend on them. We hope that our ratings can help you get informed about your rights. Do not hesitate to click on a service below, to have more details!

You can also get the ratings directly in your browser by installing our web browser add-on: This extension will add a small icon in the right side of your address bar. If you are browsing a website that is not in our database, nothing will appear. Click on the icon to get the summary of the terms, provided by ToS;DR. When you land on a website with very bad terms, a small notification will pop up in the bottom right hand corner to warn you.

For: Chrome, Firefox, Safari, Opera and coming soon to IE.

On the topic of browser extensions: Chrome extension developers are being targeted
https://threatpost.com/seven-more-chrome-extensions-compromised/127458/

- Chrome is the #1 browser on the Internet
- Developers of Chrome extensions may have weaker security than Google.
- So... attack the point of least resistance... less cautious developers.

At the beginning of the month we had the news of the hijacking of the OCR add-on called Copyfish. By hijacking we mean that attackers managed to break into the systems of the Copyfish developers and compromise their code base so that the now "quasi-legitimate" extension would be downloaded by innocent Chrome users and could then be attacked.

Now, according to researchers at Proofpoint, to Copyfish we add 7 additional legitimate Chrome Extensions that attackers have taken over and used to manipulate internet traffic and web-based ads.

In their report released last Monday, they list:

- Web Developer (0.4.9)
- Chrometana (1.1.3)
- Infinity New Tab (3.12.3)
- Web Paint (1.2.1)
- Social Fixer (20.1.1)

- And they also believe that the **TouchVPN** and **Betternet VPN** were also compromised in the same way at the end of June.

Proofpoint said: "At the end of July and beginning of August, several Chrome Extensions were compromised after their author's Google Account credentials were stolen via a phishing scheme."

Browser extensions are powerful add-ons to browsers. Whereas the JavaScript code downloaded ad hoc almost any website today is inherently untrusted, and is therefore executed with extreme caution and containment, browser add-ons have significantly greater conditional trust and so can perform many actions which are explicitly denied to page-based code.

So this is a rational attack vector and a means of compromising high numbers of unwitting Chrome users.

The CAN bus which is universally used to tie all auto systems together has a problem

The automotive "CAN" bus is the Controller Area Network. Here's what's going on:

The CAN messages, including errors, are called "frames." Our attack focuses on how CAN handles errors. Errors arise when a device reads values that do not correspond to the original expected value on a frame. When a device detects such an event, it writes an error message onto the CAN bus in order to "recall" the errant frame and notify the other devices to entirely ignore the recalled frame. This mishap is very common and is usually due to natural causes, a

transient malfunction, or simply by too many systems and modules trying to send frames through the CAN at the same time.

If a device sends out too many errors, then -- as CAN standards dictate -- it goes into a so-called Bus Off state, where it is cut off from the CAN and prevented from reading and/or writing any data onto the CAN. This feature is helpful in isolating clearly malfunctioning devices and stops them from triggering the other modules/systems on the CAN.

This is the feature that their attack abuses. Their attack triggers this feature by deliberately inducing enough errors that a targeted device or system on the CAN is forced off the bus into the Bus Off state, and thus rendered inert/inoperable. They demonstrate that this, in turn, can drastically affect the car's performance to the point that it becomes dangerous and even fatal, especially when essential systems like the airbag system or the antilock braking system are deactivated. All it takes is a specially-crafted attack device, introduced to the car's CAN through local access, and the reuse of frames already circulating in the CAN rather than injecting new ones (as previous attacks in this manner have done).

So how did we get here?

It's another classic Security Now lesson: The original system is old and it was designed just to work in the face of mutually cooperating interlinked systems. it was NEVER designed to be secure against deliberate attack... because back when it was designed (like my 16-year old 2001 BMW is CAN-bus based) there was no way for an attacker to gain remote access to the CAN bus. But, as we know, newfangled auto with all their fancy new connected-to-everything- else features, now at least create the opportunity for CAN bus access.

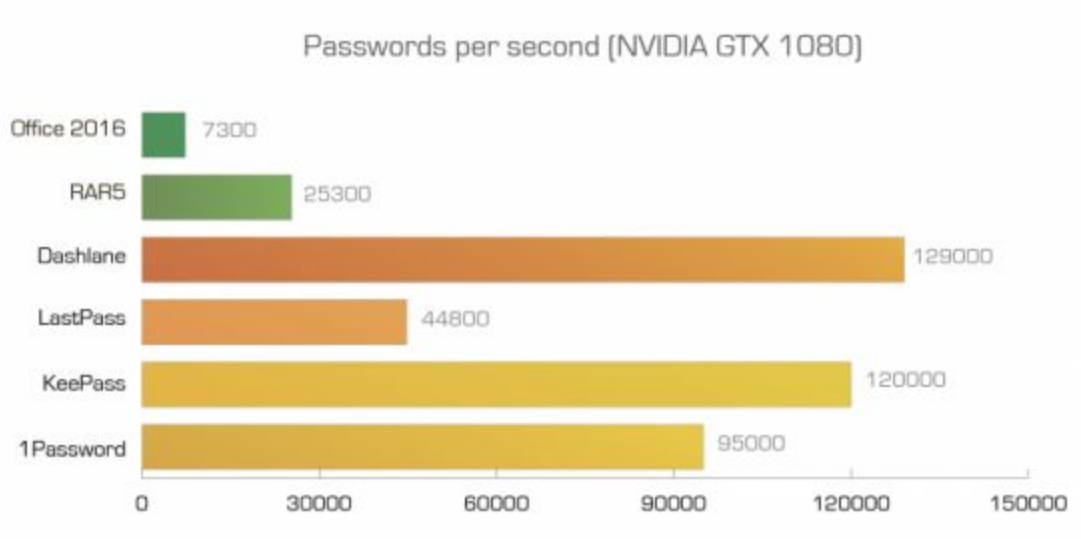
Just how old is it? The development of the CAN bus began way back in 1983 at Robert Bosch GmbH. The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) conference in Detroit, Michigan. The first CAN controller chips, produced by Intel and Philips, appeared on the market in 1987. And the 1988 BMW 8 Series was the first production vehicle to feature a CAN-based multiplex wiring system. And today, all cars are CAN bus based.

And as for modern access to the CAN bus?... We have seen this sort of access many times already. With things like CDs taking over cars, or their cellphone, WiFi, or OnStar radios being used at a distance.

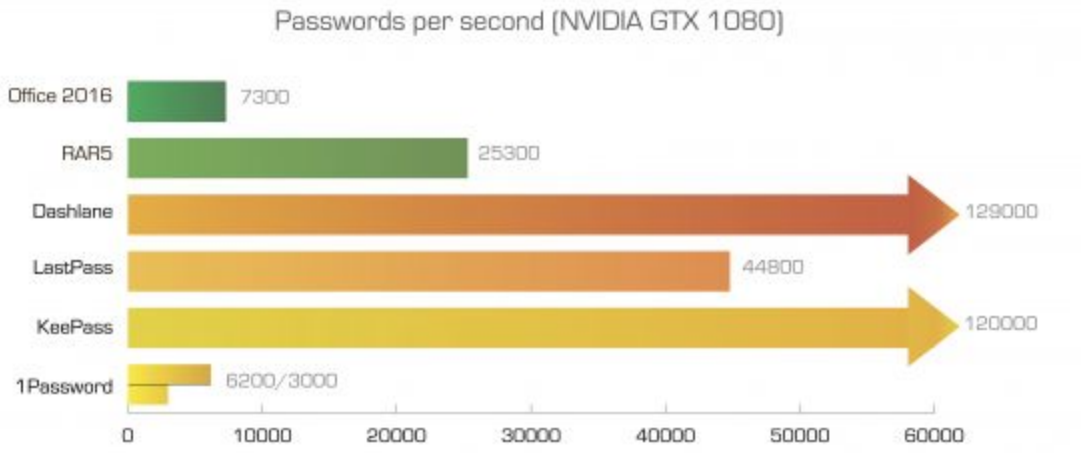
Because this is fundamental to the original bus' design, it cannot be patched. Rather, it's way past time for the bus to be updated with the need of contemporary security kept well in mind. And that change won't occur overnight because there's a massive ecosystem of existing CAN bus controllers deeply embedded into every component of modern autos.

An important correction from ElcomSoft:
https://blog.elcomsoft.com/2017/08/attacking-the-1password-master-password-follow-up/

Last week's Password Manager benchmark:



THIS week's CORRECTED benchmark (note also the change in X-axis scale):



1Password is now more than 15 times more difficult to crack by brute force than was stated previously, which makes is HARDER than any other system they tested.

What happened???.....

The Foxit PDF reader has a pair of newly disclosed 0-day vulnerabilities.
HelpNetSecurity's coverage:

The first one (CVE-2017-10951) is a command injection flaw that exists within the app.launchURL method, and arises because the method accepts more than just URLs as arguments. It does not filter file extensions, and therefore can be nade to launch executables. It was discovered by Ariele Caltabiano.

The second one (CVE-2017-10952) is a arbitrary file write flaw that exists within the saveAs JavaScript function. "SaveAs does not properly check the path it is given to write to," ZDI security researcher Abdul-Aziz Hariri explained. It also does not check the file extension.

Steven Seeley, the researcher who flagged the flaw, "exploited this vulnerability by embedding an HTA file in the document, then calling saveAS to write it to the startup folder, thus executing arbitrary vbscript code on startup."

Both vulnerabilities require user interaction to be exploited, e.g. the target must visit a malicious page or open a malicious file. Also, both vulnerabilities can be exploited only if the application's Safe Reading Mode is disabled.

Foxit Software is working on fixes, and in the meantime noted: "Foxit Reader & PhantomPDF have a Safe Reading Mode that is enabled by default to control the running of JavaScript. This is effective in guarding against potential vulnerabilities from unauthorized JavaScript actions."

Yes, Cloudflare uses Lavalamp Entropy
https://www.fastcodesign.com/90137157/the-hardest-working-office-design-in-america-encrypts-your-data-with-lava-lamps

Cloudflare is now handling 10% of the Internet's traffic now.

A high resolution camera takes image frames every millisecond and are hashed to produce absolutely unpredictable truly random -- non-algorithm-based -- numbers.

iOS 11 has a 'cop button' to temporarily disable Touch ID
https://www.theverge.com/2017/8/17/16161758/ios-11-touch-id-disable-emergency-services-lock
http://gizmodo.com/a-hidden-trick-in-the-ios-11-beta-lets-you-disable-touc-1797974234

Five successive presses of an iOS device's power/sleep button enabled emergency 911 calling and also simultaneously disables the TouchID function until the device's full passcode is properly entered.

Libsodium Audit Results
https://www.privateinternetaccess.com/blog/2017/08/libsodium-audit-results/

Private Internet Access today releases the results of its Libsodium audit. Libsodium is an open source, cryptographic library that is used far and wide in projects such as Zcash as well as internal applications at Private Internet Access. Private Internet Access is proud to have another audited tool in its software suite. The Libsodium security assessment was conducted by Dr. Matthew Green of Cryptography Engineering on v1.0.12 and v1.0.13. Dr. Green previously completed the TrueCrypt audit with the Open Crypto Audit Project as well as an OpenVPN 2.4 audit on PIA's behalf. The assessment found no critical flaws or vulnerabilities in the Libsodium library.

Matt, who authored the Cryptography Engineering Libsodium Security Report, summarized the audit goals and results:

"Over the past several months we conducted a detailed audit of the libsodium cryptographic library, at the request of Private Internet Access (PIA). Our goal in this effort was to help ensure the safety of an increasing number of applications that rely on libsodium for cryptographic operations. We are pleased to report that our review did not uncover any critical flaws or vulnerabilities in the core library. Overall we believe that libsodium is a carefully-implemented, secure cryptographic library."

The Libsodium team commented:
"We are thankful to Private Internet Access for their initiative in sponsoring security audits for crucial free and open source software around the world; they help make the internet more secure."

AWS cloud bucket permissions accidentally set to "public"
https://threatpost.com/meeting-and-hotel-booking-providers-data-found-in-public-amazon-s3-bucket/127542/

This is the danger inherent in easy-to-use power. It's both easy to use and easy to misuse.

Surprisingly forthright security disclaimer
https://turtlapp.com/docs/security/#when-is-turtl-not-secure

When is Turtl *not* secure?

Here are some possible scenarios where Turtl's security measures will fail you. We try to provide an exhaustive list so you're aware of the dangers of relying on Turtl.

- **When we make mistakes**. That's right, we're human. It's entirely possible that bugs in the Turtl client leave your data exposed, *especially* at our early stage.

- **When you use a bad password**. Turtl encrypts just about everything before sending it to the server. It does this using a cryptographic key based off of your email and password. If you choose a password that's short, predictable, easy to guess, etc then your data is *not safe*. Choose a good password. Turtl has no restrictions on password length, we suggest you take advantage of this.

- **When you invite someone to a board over email.** Turtl has a feature that allows you to invite someone to share one of your boards via email. Before sending, you are able to set a shared secret for the invite, which makes the invite useless unless the invitee enters the secret when they accept the invite (this secret must be communicated to the invitee separately). Without setting this secret, anybody who intercepts the invite email *will gain full access to the board and its data*. If you want to share something but need it to be secure, set the secret and communicate it (via phone, text message, etc) to the person you're inviting. Please note that the shared-secret method is not as secure as ECC encryption (ECC is used when inviting an *existing* user to a board), but it's a lot better than not having the shared secret at all (if you care about privacy).

- **When someone you shared data with is compromised.** If you share notes, boards, or any other data with other Turtl users, you are giving them the ability to decrypt those pieces of your data. It's possible that the person you shared with isn't who you think they are, or they have a gun to their head and have no choice but to expose your data. Be very careful about who you share sensitive data with.

- **When you have malware installed**. When you're logged in to Turtl, all your unencrypted data is sitting in your computer's memory. It's possible that a malicious program could gain access to the app's memory and read your data. Note that most operating systems have protections against this, but that doesn't make it impossible.

- **When your operating system is compromised**. Although this may sound far fetched, it's possible that your entire operating system itself is maliciously programmed to send contents of memory from certain programs to certain corporate headquarters or government agencies. If you really want to elminate this possibility, use an open-source operating system (such as Linux or BSD).

- **When your hardware is compromised**. It's not outside the realm of possibility that your computer's hardware is maliciously sharing the contents of your memory to a third party.

- **When someone is holding a gun to your head**. Sometimes the easiest way to get your data is to threaten you or your family. Turtl has no countermeasures to protect against this, and it's up to you to make your own decisions.

Errata

- Delphi Ote (@delphi_ote)
  @SGgrc Road sign classifier misstatement: you said it had 91% accuracy on data used to train it. That would be training set, not test set.

- Emett Speer (@BotEmett)
  @SGgrc You miss quoted the availability of S3. The 99.999999999% is the data durability. S3's availability is 99.99%.

Miscellany

- The Tweet of the week:
  Andy (@remixman)
  LOL, @SGgrc!

    - Jim Birch @thejimbirch
      The sun is currently undergoing a Denial of service attack. We hope to have full service restored soon. #EclipseDay #eclipse #nerdhumor

- Richard Phillips Rho Agenda books (8 of 9) $99 through August!
  https://rhoagenda.me/2017/08/01/rho-agenda-series-0-99-august-promo/
  http://bit.ly/rhoagenda
  Each book $0.99 through the end of August.

SpinRite

Jesse Harris (coolestfamilyever.com) -- *loved the domain name*
Subject: Spinrite saves the day again

I know I can't get enough of hearing how great something I've done is and figure you're probably the same. I purchased SpinRite around 3 months ago when my primary desktop started experiencing lots of strange IO errors and related lags in the OS. I ran a level 2 scan on my SSD which turned up no issues. Despite this, it's been performing absolutely normal ever since.

This morning I had the worst case scenario of two drives failing in a four-drive RAID array (RAID-Z1 on FreeNAS). I pulled one of the drives, ran a level 2 scan, and dropped it back into the box. One quick command later, the pool was up in a degraded but functional and accessible state as I scan the other drive reporting errors. It's almost like the drive just needed to see that I did, indeed, mean business when it decided to whip itself back into shape.

For a long time I balked at the $89 price tag and yet now I wonder if you're even charging enough for what this software is capable of doing. Looking forward to what's in store for the future of the 6.x series once SQRL is done.

Jesse Harris

Closing The Loop

- Jan (@j4n_)
  @SGgrc Could you talk about how you encrypt your storage on amazon s3 for next week's episode of Security now? Or has it been done already?

- David Mueller (@quanterium)
  @SGgrc HTTP 418 comes from RFC 2324, the Hyper Text Coffee Pot Control Protocol
  http://tools.ietf.org/html/rfc2324
  https://sitesdoneright.com/blog/2013/03/what-is-418-im-a-teapot-status-code-error#.WZXu6GaK6WI.twitter

If you look through the full list of HTTP status codes, you'll see one that really stands out: "418: I'm a teapot". So what's that all about? I'll explain in a bit. But first, a really quick introduction on status codes.

Whenever any page or file is accessed on your site (whether it's a user accessing it in a browser or a search engine crawling a page), your server returns an HTTP status code in response to the request and it provides information about the status. For example, some popular codes are "301 Page Moved Permanently" and the ever ubiquitous "404 Page Not Found".

So, what's the "418 I'm a teapot" all about? Well, the group of people who make these codes and set the standards is the IETF or "Internet Engineering Task Force". To propose new standards, the members release RFC's or "Requests for Comments" to the community. Every year since 1989, they release a few humorous RFC's for April's Fool Day and on April 1st, 1998, RFC 2324 introduced the "Hyper Text Coffee Pot Control Protocol" or (HTCPCP/1.0). This was a brand new protocol for controlling, monitoring, and diagnosing coffee pots. Now, the RFC is pretty funny with lines like "Coffee pots heat water using electronic mechanisms, so there is no fire. Thus, no firewalls are necessary". If you have never read it, it's definitely worth a read. I added a link in the description below. Now, you may be asking, if this is a COFFEE protocol, why the "teapot" code? This is answered in Section 2.3.2. in that "Any attempt to brew coffee with a teapot should result in the HTTP error code *418 I'm a teapot* and the resulting entity body MAY be short and stout". And just like that the "418 I'm a teapot" code was born! Since then, it's been used in all sorts of wacky ways. Google even referred to "418" as a different error in their 2013 April Fool's Joke "Google Nose" saying that *"418: Scent transfer protocol error" indicates system congestion; please try again later.*