Security Now! #724 - 07-23-19

Hide Your RDP Now!

This week on Security Now!

This week we start off with something bad that we unfortunately saw coming. We then look at the changing security indication feedback in browsers, the challenge of keeping browsers compatible with important but non-standards-compliant web sites, the failure and repair of Incognito browsing mode, the possibility of a forthcoming "super Incognito mode" for Firefox, a new super-fact TLS stack written in the Rust programming language, Microsoft's promised open source release of their voting machine election software, yet another widely deployed, exposed and exploitable Internet server. Then a quick bit of miscellany, some terrific SQRL news, and we look at a recent and quite sobering report from Sophos about attacks on exposed RDP servers.

Real Facebook Cert signed by DigiCert Fake Facebook Cert signed by KZ

6308 9198 9198 2019-07-17 12:43 No Time Name *

Majority Record		This Record	
CN	*.facebook.com	CN	*.facebook.com
O	Facebook, Inc.	O	Facebook, Inc.
C	US	C	US
Not Before	2019-06-06T00:00:00Z	Not Before	2019-07-16T12:39:52Z
Not After	2019-09-04T12:00:00Z	Not After	2020-07-15T12:39:52Z
SHA1	C5:22:F1:15:F8:B2:AD:AE:12:63:BC:8D:5F:A7:B	SHA1	5F:55:F8:28:2C:9B:AA:79:0A:5C:C2:76:CD:D7:81:7C:BC:
MD5	EC:B8:53:F1:12:34:C8:35:22:23:F5:78:3F:4E:A6	MD5	F6:9F:EF:F3:07:84:D1:D4:F2:48:6A:FA:58:C3:F2:FA
subectAltName	*.facebook.com messenger.com *.fbcdn.net *.fb.com *.m.facebook.com fb.com *.facebook.net *.xx.fbcdn.net *.xz.fbcdn.net *.messenger.com *.fbsbx.com *.xy.fbcdn.net facebook.com	subectAltName	*.facebook.com messenger.com *.fbcdn.net *.fb.com *.m.facebook.com fb.com *.facebook.net *.xx.fbcdn.net *.xz.fbcdn.net *.messenger.com *.fbsbx.com *.xy.fbcdn.net facebook.com
CN	DigiCert SHA2 High Assurance Server CA	CN	Security Certificate
O	DigiCert Inc	O	No data
C	US	C	KZ
Not Before	2013-10-22T12:00:00Z	Not Before	2018-02-12T06:36:56Z
Not After	2028-10-22T12:00:00Z	Not After	2021-02-12T06:36:56Z
SHA1	A0:31:C4:67:82:E6:E6:C6:62:C2:C8:7C:76:DA:9A:A6:2C:C	SHA1	07:2B:83:FF:A7:78:E0:A5:CB:AB:87:4E:3D:22:7C:BD:E7:41:0F:94
MD5	AA:EE:5C:F8:B0:D8:59:6D:2E:0C:BE:67:42:1C:F7:DB	MD5	34:14:E9:22:2A:F8:45:98:90:89:6F:25:7F:78:A3:05

Security News

Welcome to Kazakhstan! - Please check your privacy at the border.
We'll start off with one from our "Unfortunately we did see this coming" department. I received a significant number of tweets and private messages containing variations on the theme of "This is what you've been predicting, Steve" and such ... though it was less a prediction than a worry about something that COULD happen, which would be bad. And now it has.

So, what have I been worrying about? Turing to our picture of the week, on the left we see the authentic Facebook certificate with the Common Name "*.facebook.com" and a long list of affiliated domains listed in its Subject Alternative Name (SAN) field. And that authentic certificate was issued by and signed by my favored certificate authority, DigiCert.

And, sadly, on the right we see a certificate that is in every way a clone-copy of Facebook's authentic certificate containing the same "*.facebook.com" common name and the identical list of affiliated domains. But the Common Name of the signer of THIS fraudulent certificate reads "Security Certificate".

And what we now know is that the government of Kazakhstan has begun officially requiring that its own CA root certificate be installed into the web browsers of its citizens. And, of course, we know why. It's so that it can perform man-in-the-middle interception, decryption and perhaps alteration of the HTTPS-encrypted traffic moving within its borders. That fraudulent Facebook certificate was discovered being used to impersonate Facebook.

The first indication of this was a "bugzilla" report #1567114 titled: "MITM on all HTTPS traffic in Kazakhstan"
https://bugzilla.mozilla.org/show_bug.cgi?id=1567114

> Since today all Internet providers in Kazakhstan started MITM on all encrypted HTTPS traffic. They asked end-users to install government-issued certificate authority on all devices in every browser: http://qca.kz/
>
> Actual results: MITM attack: https://i.imgur.com/rFEjXKw.jpg
> Message from Internet provider, requires to install this CA: https://i.imgur.com/WyKjOug.jpg

https://groups.google.com/forum/#!msg/mozilla.dev.security.policy/wnuKAhACo3E/cpsvHgcuDwAJ

> Matthew Hardeman / 18 Jul
> Other recipients: wth...@mozilla.com, mozilla-dev-s...@lists.mozilla.org If the government of Kazakhstan requires interception of TLS as a condition of access, the real question being asked is whether or not Mozilla products will tolerate being used in these circumstances.
>
> Your options are to block the certificate, in which case Mozilla products simply become unusable to those subject to this interception, or not block the certificate.
>
> I certainly think that Mozilla should not distribute the MiTM root or do anything special to aid in its installation. I believe policy already makes clear that NO included root (commercial or

government) is allowed for use in MiTM scenarios and I believe that policy should be held firm. I do believe that as it is manually installed rather than distributed as a default that it should continue to override pinning policy.

This is an accepted corporate use case today in certain managed environments. The dynamic is quite different for an entire people under an entire government, but the result is the same:

One has to choose whether to continue serving the user despite the adverse and anti-privacy scenario, or if one simply won't have their products be any part of that.

Much has been said about the TLS 1.3 design hoping to discourage use cases like this, but the reality is what I always suspected: some enterprises or governments actually will spend the money to do full active MiTM interception.

Let's posit what might happen if Mozilla made their products intentionally break for this use case.

Further, let's stipulate that every other major browser follows course and they all blacklist this or any other nation-state interception certificate, even if manually installed.

Isn't the logical outcome that the nation-state forks one of the open-source browser projects, patches in their MiTM certificate, and un-does the blacklisting? I think that's exactly what would happen. The trouble is, there's no reason to expect that the fork will be maintained or updated as security issues are discovered and upstream patches are issued. We wind up with an infrequent release cycle browser being used by all these users, who in turn get no privacy AND get their machines rooted disproportionate to the global population.

I do definitely support a persistent UI indicator for MiTM scenarios that emphasizes on-screen at all times that the session is being protected by a non-standard certificate and some sort of link to explain MiTM and the risks.

The Kazakh government first tried to have all its citizens install a root certificate three and a half years ago in December 2015. And that sounded familiar, so I imagine it caught our attention and we talked about it at the time. Back then, the government ruled that all Kazakh Internet users had to install the government’s root certificate by January 1st, 2016.

However, the ruling was never implemented because the government was sued by several organizations, including ISPs, banks, and even foreign governments, who feared this would weaken the security of all internet traffic (and the related business) originating from the country. And, of course, they were, and still are, completely correct. At that time, in December 2015, the Kazakh government also applied with Mozilla to have its root certificate included in Firefox by default, but Mozilla declined.

But now, starting last Wednesday, July 17, the Kazakhstan government has started intercepting all HTTPS internet traffic inside its borders. Local internet service providers (ISPs) have been instructed by the local government to force their respective users into installing a government-

issued certificate on all devices, and in every browser. And, as we know, once installed, the certificate will allow local government agencies to decrypt users' HTTPS traffic, look at its content, encrypt it again with their certificate, and forward it its destination.

To make this happen, local ISPs have started forcing their customers to install the government's root certificate by redirecting them to intercept web pages containing instructions on how to install the government's cert in their respective browsers, whether desktop or mobile device.

In a statement posted on its website, the Kazakh Ministry of Digital Development, Innovation and Aerospace said only internet users in Kazakhstan's capital of Nur-Sultan will have to install the certificate; however, users from all across the country reported being blocked from accessing the internet until they installed the government's certificate. Some users also received SMS messages on their smartphones about having to install the certificates, according to local media.

Ministry officials said the measure was "aimed at enhancing the protection of citizens, government bodies and private companies from hacker attacks, Internet fraudsters and other types of cyber threats."

It's true, as we know, that this sort of interception could be used benignly to filter traffic for malware. But as we know, it's equally possible to scan and surveil traffic for whatever other content the government or other MITM agency chooses.

Google, Microsoft, and Mozilla are discussing a plan of action about dealing with websites that have been (re-)encrypted by the Kazakh government's root certificate. So it’s going to be very interesting to see how this develops. What seems most likely, is that browsers and other devices will adopt some form of prominent visible warning. Since the industry's best browsers are now all open source, private labeling a "Kazakhstani" browser containing only Kazakhstan's root certificate would not be an insurmountable task. Though maintaining it and keeping it current would be a big job. The fact is, most users will not know by themselves what's going on. But their press will likely quickly inform them. On the other hand, what choice will they have?

Lose your privacy, or lose the Internet.

And it's certainly likely the other repressive and authoritarian governments are watching this experiment with significant interest.

Some later follow-up reporting has somewhat muddied the water. Caleb Chen, writing for the PrivateInternetAccess blog, notes that: “A Kazakh official clarified on July 19th, 2019 that the installation of the certificate was voluntary and not a prerequisite to accessing the internet.” Unfortunately, Caleb provided a link to the page https://rus.azattyq.org/a/30064788.html which is written in either Kazakh and Russian, neither of which I can read.

What do tech-savvy Kazakhs think about this? The Mozilla bug thread contains a posting from one: “I am a citizen of Kazakhstan. If Mozilla/Google Chrome developers see this message, I kindly ask you to consider blocking the above mentioned certificate and any access to your browsers for the certificate holders. If this certificate didn’t pass Web trust audit, it can be the same as presented in 2016. So blocking it from the major world browsers is the only chance for

Kazakhs to avoid MITM attacks and keep at least some privacy rights (meaning that if blocked/ blacklisted, the government will have to call back the certificate as it was done in 2016). […] If the certificate is not blacklisted, but only the visual message will pop up warning users about untrusted certificate – it will not help since majority of citizens (especially elderly ones) simply will not pay enough attention to such [a] message.

And, our illustrious crypto professor, Mattthew Green weighed-in via Twitter:

And speaking of indicators and browser security & privacy…
Starting in October with Firefox #70, Mozilla will be prominently marking all HTTP, non-HTTPS pages as "not secure" as Chrome does now.

What Mozilla has been doing with Firefox was only showing "not secure" indicators on HTTP pages where it probably mattered more, meaning pages which accepted user content containing forms or login fields. However today the percentage of sites serving HTTPS has surpassed 80%. So their thinking is that users don't need good news for what has rapidly become the default -- being secure -- but rather a warning when, for some reason, a site is not secured by HTTPS.

Firefox developer Johann Hofmann wrote: "In desktop Firefox 70, we intend to show an icon in the 'identity block' (the left hand side of the URL bar which is used to display security / privacy information) that marks all sites served over HTTP (as well as FTP and certificate errors) as insecure."

Firefox has had configurable behavior flags viewable and configurable on its about:config page since December of 2017. Those flags are still present in the current stable version of Firefox, and users can enable them to preview this forthcoming indicator:

The flags are:

- **security.insecure_connection_icon.enabled** - show a broken lock on HTTP sites
- **security.insecure_connection_text.enabled** - show the "not secure" text on HTTP sites
- **security.insecure_connection_icon.pbmode.enabled** - show a broken lock on HTTP sites in Private Browsing

- **security.insecure_connection_text.pbmode.enabled** - show the "not secure" text on HTTP sites in Private Browsing

Since it can now be difficult to find a non-HTTPS site for testing, I have one. Although my new GRC link-shortener service of course supports HTTPS, and it always redirects users to secure pages, I didn't see any reason not to allow it to accept an incoming link over HTTP. So you can always check to see what you browser shows for non-secured HTTP sites by going to the root page of GRC.SC… http://grc.sc/

And speaking of Firefox's about:config page...
Mozilla's Dan Callahan, writing about the changes in the latest Firefox 68, explained about some of the subtleties of browser behavior which were required to allow Firefox to operate on websites that were apparently written to expect some specific quirks of some non-Firefox browser...

https://hacks.mozilla.org/2019/07/firefox-68-bigints-contrast-checks-and-the-quantumbar/

Dan writes: "Keeping the Web open is hard work. Sometimes browsers disagree on how to interpret web standards. Other times, browsers implement and ship their own ideas without going through the standards process. Even worse, some developers intentionally block certain browsers from their sites, regardless of whether or not those browsers would have worked.

At Mozilla, we call these "Web Compatibility" problems, or "webcompat" for short.

Each release of Firefox contains fixes for webcompat issues. For example, Firefox 68 implements:

Internet Explorer's addRule() and removeRule() CSS methods.
Safari's -webkit-line-clamp CSS property.

In the latter case, even with a standard line-clamp property in the works, we have to support the -webkit- version to ensure that existing sites work in Firefox.

Unfortunately, not all webcompat issues are as simple as implementing non-standard APIs from other browsers. Some problems can only be fixed by modifying how Firefox works on a specific site, or even telling Firefox to pretend to be something else [via a lie in the User-Agent header] in order to evade browser sniffing.

We deliver these targeted fixes as part of the webcompat system add-on that's bundled with Firefox. This makes it easier to update our webcompat interventions as sites change, without needing to bake those fixes directly into Firefox itself. And as of Firefox 68, you can view (and disable) these interventions by visiting about:compat and toggling the relevant switches.

Our first preference is always to help developers ensure their sites work on all modern browsers, but we can only address the problems that we're aware of. If you run into a web compatibility issue, please report it at webcompat.com.

Being a bit more Incognito

We've probably all encountered teaser paywall websites that allow a limited number of articles to be viewed per month by non-subscribers. This sort of feels like at workable compromise between fully open and fully closed sites. They are clearly hoping that you'll find value there and finally be annoyed when that one article you really sent to read fades out into a notice that the site requires a monthly fee for you to keep reading.

This, of course, begs the question: How are they counting the number of pages you've visited? The answer could be some form of fancy fingerprinting. But we know that browser fingerprints are not unique. Many other web users will coincidentally have a browser with the same fingerprint. And these sites are not attempting to block rocket scientists from having access, only casual visitors... Who have their browser's 1st-party domain cookies set in the universal "enabled" state.

So it wasn't long before people who want to read that one additional article, but didn't want to join up, discovered that switching to their favorite browser's "Incognito" or "Private Browsing" mode would make this possible. Since private browsing deliberately does not record 1st-party domain cookies, a user of an Incognito browser is suddenly unknown and always starts out with a zeroed "prior articles" counter since they appear as an unknown and "un-cookied" visitor.

And in this cat and mouse world, just as didn't take long for the use of the private browsing trick to become widespread through independent discovery and word of mouth, neither did it take the web developers long to figure out a way, at least in the case of the Internet's #1 web browser, Chrome, to detect when someone was visiting their paywalled site "Incognito."

For example, if you visit any article on "The Washington Post" news site using Chrome's Incognito mode, you'll be greeted with the message: *"We noticed you're browsing in private mode. Private browsing is permitted exclusively for our subscribers. Turn off private browsing to keep reading this story, or subscribe to use this feature, plus get unlimited digital access."*

But, wait... what?!?! The idea of being IDENTIFIED as someone who is visiting "Incognito" sort of puts the lie to the whole Incognito thing, right? It's like "Wait a sec... You're not supposed to know **anything** about me, including, and perhaps even importantly, that that I have deliberately chosen not to have you know anything about me, including that."

This occurred to the engineers at "The Goog" so they decided to do something about it. It turns out that Chrome's Incognito mode disables Chrome's FileSystem API as part of Chrome's effort to prevent the user's activities from leaving any lasting traces. Websites figured out that this could be easily checked with a bit of JavaScript running on the page, so that's what generates that "please disable your Incognito mode to receive a limited amount of free stuff" message.

Consequently, at the end of this month, on July 30th, we're going to be getting Chrome 76 with a FileSystem API that no longer gives away the browser's Incognition. And just in case publishers then go searching around for some next method to detect private browsing, since Google really does wish to avoid having its Incognito visitors flagged, "The Goog" have stated that: "Chrome will likewise work to remedy any other current or future means of Incognito Mode detection."

And speaking of Private Browsing...
Firefox might be getting "Super Private Browsing" in the not-to-distant future, a la, TOR!

Due to the fact that the TOR browser is a descendant of Firefox, there have been various projects and grants awarded to look into the possibility of adding a "TOR mode" to Firefox. This is exciting because setting up a TOR session take some doing, and also because wrapping all outbound traffic in a multi-layered successively-encrypted "onion" and then having that traffic bounce three or four times around TOR nodes while each successive layer is stripped and decrypted and then forwarded on to the next TOR node means that "surfing the web through TOR" is not for the impatient. But... The idea of being able to just "flip a switch" the way we currently do when we turn on private browsing would be VERY appealing. It would allow Firefox users to briefly and quickly drop a cone of silence over their online activities when they need some real anonymity, then just as easily lift the cone to return to regular high-speed browsing.

During a recent meeting of the core Tor team, developers, volunteers and invited guests in Stockholm, to discuss plans, milestones, deadlines, and other important matters... The idea of a TorMode Add-on for Firefox was proposed, discussed and considered:

https://trac.torproject.org/projects/tor/wiki/org/meetings/2019Stockholm/Notes/FirefoxTorUpliftAndTorModeAddOn#TorModeAdd-onproposal

"There is an idea to, in the future, have Firefox use Tor in private browsing mode, or an a new extra-private mode. That will take a lot of engineering work and buy-in. To help smooth the path, there is a proposal for a "Tor mode" addon. This would not be packaged with the browser by default, but would be something that users could download from addons.mozilla.org to give them a "Tor mode" button or similar. It would allow users to experience what an eventual full integration with Tor could look like. It could also help gauge interest by counting downloads, etc.

acat (Alex Catarineu) has demonstrated how to compile tor to WASM. This would allow packaging all the necessary tor code inside the addon itself, without a dependency on external binaries. The addon would still need to be a privileged addon.

What's a privileged add-on? A privileged addon is one with elevated privileges compared to a standard WebExtension. It can call XPCOM functions, for example. A privileged addon needs to be signed by Mozilla, or something, but the idea for this proposal is to have it produced and distributed by Mozilla anyway, so that's not a problem.

The addon would configure the browser to use tor as a proxy, as well as setting various prefs to prevent proxy bypasses and resist fingerprinting, much like those set by Tor Browser.

Discussion of visual options for UI. Clicking the Tor-mode button would probably open a new window that uses a dedicated profile. This is because some of the prefs that the addon has to set are global to a profile, not to a window or a tab.

What to do about HTTP? The feeling is that it's dangerous to pass unauthenticated HTTP through exit nodes. Packaging NoScript does not provide the best experience either. The easiest solution is to enforce (require) HTTPS when in Tor mode.

So... This is well in advance of any timeline or release number target... But it's neat that the Tor gurus are thinking along these lines. As for myself, beyond experimenting with TOR just for the experience, I've never used it seriously. But I, for one, would add the Tor add-on to my browser for use during those times when being truly stealthy would be useful.

"Rustls" (Rust-TLS) outperforms OpenSSL in nearly every way

A tiny and relatively unknown TLS library written in Rust (which is a language we'll be talking about a bit more in a minute) outperformed the industry-standard OpenSSL library in almost every major category.

The benchmarks were performed by Joseph Birr-Pixton, the author of the RUSTLS library. Joseph's findings showed that Rustls was 10% faster when setting up and negotiating a new server connection, and between 20 and 40% faster when setting up a client connection. These days most TLS traffic relies on resuming previously negotiated handshakes. But here, too, Rustls outperformed the aging and creaky OpenSSL, being between 10% and 20% faster in resuming a connection on the server-side, and being between 30 and 70% quicker to resume a client connection. Moreover, Rustls fared better in bulk data transfer performance. Joseph's measurements showed that Rustls could send data 15% faster than OpenSSL, and receive it 5% faster as well. And just to put a cherry on top, Rustls uses only half of the memory consumed by OpenSSL.

Since it has been a core of the Internet, we have spoken of OpenSSL often. It **IS** old and creaky. And we're seeing more and more alternatives appearing. OpenSSL has been the old workhorse where new ideas were first developed and proven as SSL evolved and acquired features through the years, and then made the straddle from SSL to TLS. And we all know that anyone using a brand new TLS stack should initially do so with extra suspenders because TLS is a complex protocol and you don't want to find your pants down around your ankles.

That said, Rust is probably well suited to this sort of application. Unlike C and C++, Rust has security designed into the language itself to avoid memory-related security bugs. And it's appearing that tasks coded in Rust are now generally outperforming C and C++.

Rust was designed by Mozilla, from the ground up to prevent memory management-related bugs, which are usually at the heart of most security flaws in C and C++ applications.

Although the Rust project was initially ridiculed, it is now being adopted at a rapid pace, and the once-controversial decisions behind its safety-first design have served to make the language a proven success. The Firefox and Brave browsers rely on Rust components, and large companies like Cloudflare, Dropbox, Yelp, and npm, have adopted it for production systems. The Tor Project is experimenting with Rust, and Facebook's recently launched Libra cryptocurrency will be using it. And last but not least, Microsoft has just, this past week, announced plans to explore using Rust as a replacement for C and C++.

The language is also popular with developers. Rust has come out on top as the most popular programming language in StackOverflow's developer survey for the past five years, in 2016, 2017, 2018, and 2019.

https://www.rust-lang.org

Performance: Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability: Rust's rich type system and ownership model guarantee memory-safety and thread-safety — and enable you to eliminate many classes of bugs at compile-time.

Productivity: Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

I wanted to mention all this, first to put Joseph's RUSTLS library on everyone's radar in case there might be some interest and need. And also because computer languages are fun. I plan to accelerate my development of SpinRite v7 -- to which I plan to add a large array of features -- by using the low-level driver code I will first create for the SpinRite v6.x series. But I plan to implement the next UI and its underlying technology in something that's faster for me to prototype and experiment an work with. I was thinking of Python for that, but I'll definitely be taking a look at Rust. (And, no, Leo... I won't be using LISP.)

Microsoft announces "ElectionGuard" during last week's Aspen Security Forum
Last week, Microsoft demonstrated a new open source technology named ElectionGuard, which it said can be used to secure modern electronic voting machines.

This is not, thank God, the beginning of Microsoft balloting machines based on Windows 10.

Microsoft stated that it had no plans to release commercial voting machines. Instead, they said they planned to release the ElectionGuard software on GitHub, under an open-source license, later this year.

Microsoft described the technology behind ElectionGuard as being relatively simple and centering around a few core principles (which sounds good so far)...

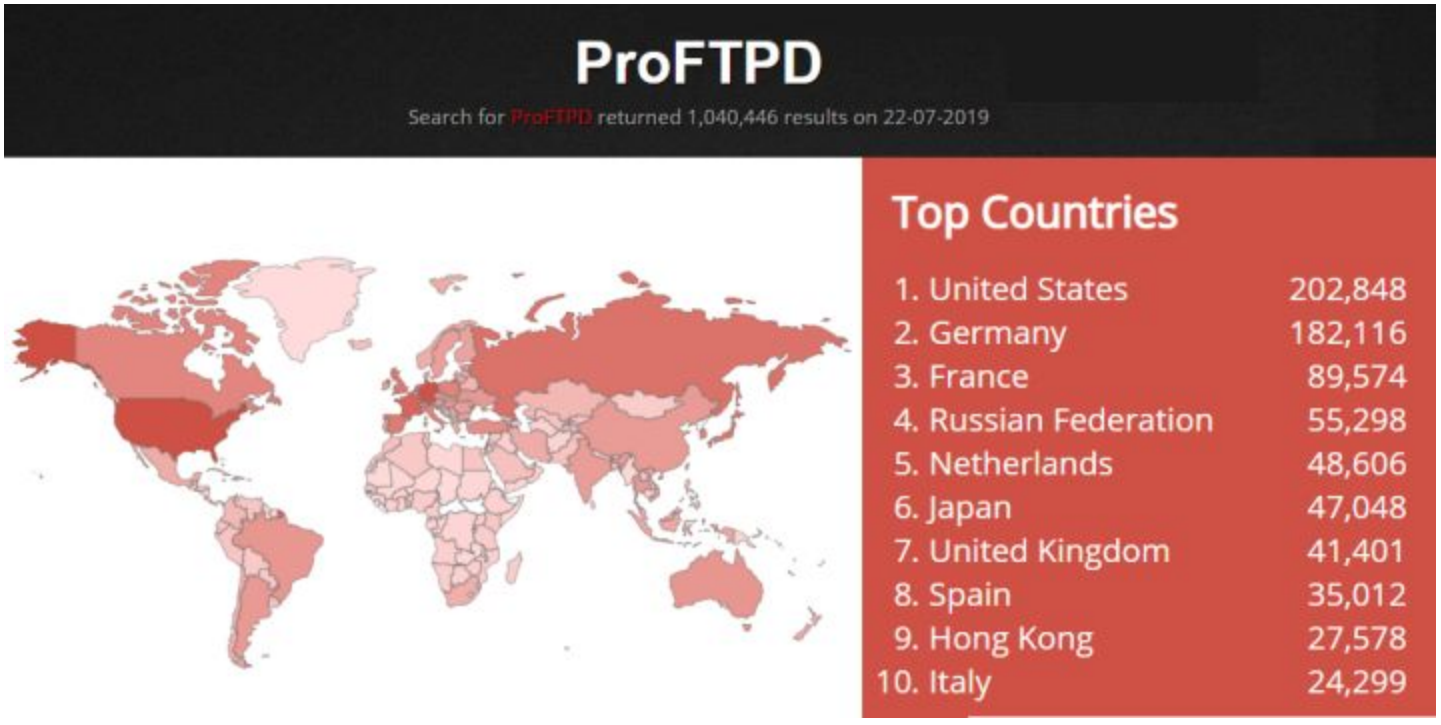
- Voters receive a tracking code.
- The tracking code on an election website to verify that their vote has been counted and that the vote has not been altered.
- But the tracking code reveals nothing about the voting, so it will not allow third-parties to see who voted for whom.
- ElectionGuard uses a "homomorphic encryption" encryption system developed in-house at Microsoft under Senior Cryptographer Josh Benaloh.
- Homomorphic encryption is a very cool technology that allows the counting of votes while keeping the votes encrypted.
- The ElectionGuard SDK also supports third-party "verifier" apps to independently check if encrypted votes have been counted properly and not altered.
- Verifier apps were created for voting officials, the media, or any third party interested in the voting process.

- ElectionGuard machines can also produce paper ballots, as a printed record of their vote, which voters can place inside voting boxes, like old-fashioned votes.
- ElectionGuard also supports voting through accessibility hardware, such as Microsoft Surface or the Xbox Adaptive Controller.

And ElectionGuard has already attracted interest, as well it should. Microsoft hopes voting machine makers will use its new ElectionGuard software for their products. According to Microsoft, the SDK has been warmly welcomed by some voting machine vendors already. Tom Burt, Microsoft Corporate Vice President, Customer Security & Trust said: "We previously announced that we have partnerships with suppliers that build and sell more than half of the voting systems used in the United States. Today, we're excited to announce that we're also now partnering with Smartmatic and Clear Ballot, two of the leading voting technology vendors, and Dominion Voting Systems is actively exploring the inclusion of ElectionGuard in their offerings."

This is **exactly** the correct direction for our future voting systems. As I've said before, it's fine for the likes of Diebold (dee-bold) to manufacture and sell the hardware, but they way they work and what they do **cannot** be allowed to remain proprietary secrets any longer. The entire notion that some random hardware manufacturer has some kind of secret sauce to protect our votes is just utter nonsense. So it takes someone with a solid reputation -- exactly like Microsoft Research -- to offer an answer at no cost to the hardware vendors. Yay!

And what would a weekly installment of Security Now! be without the announcement of yet-another remotely exploitable vulnerability in a widely-used, high-profile and easily discoverable Internet server??
Yes… More than one million currently-online instances of the popular open-source ProFTPD server, present in distributions of Debian, SUSE, Ubuntu and many other Linux and UNIX-like systems, is wide open to remote compromise, including remote code execution, as a result of an arbitrary file copy vulnerability.



ALL ProFTPd versions up to and including 1.3.6 (the latest version which was released back in 2017) are impacted by the vulnerability which enables remote attackers to execute arbitrary code without the need to authenticate and with the user rights of the ProFTPD service.

The problem arises from ProFTPD's "mod_copy" module which implements additional commands. The ProFTPD site explains: "The mod_copy module implements SITE CPFR and SITE CPTO commands (analogous to RNFR and RNTO), which can be used to copy files/directories from one place to another on the server without having to transfer the data to the client and back." So, in other words, SITE CPFR and SITE CPTO are commands that any anonymous, non-authenticated FTP client which connects to a ProFTPD server, can use to command the server to perform server-side file copy operations.

http://bugs.proftpd.org/show_bug.cgi?id=4372

A ProFTPD user reported the following via email: the mod_copy module's custom SITE CPFR and SITE CPTO commands do not honor <Limit READ> and <Limit WRITE> configurations as expected.

To reproduce, just enable the anonymous user example that is configured in the Debian default proftpd.conf:

```
<Anonymous ~ftp>
    User                ftp
    Group               nogroup
    UserAlias           anonymous ftp
    RequireValidShell   off

    MaxClients          10
    DisplayLogin        welcome.msg
    DisplayChdir        .message

#Limit WRITE everywhere in the anonymous chroot
<Directory *>
   <Limit WRITE>
      DenyAll
   </Limit>
</Directory>

</Anonymous>
```

Command: ftp proftptest.domain.org (on linux, using a regular ftp client)

Login as anonymous here. You normally can't upload files, because of the DenyAll
site cpfr welcome.msg
site cpto malicious.php

We've created a file named "malicious.php" with the contents of welcome.msg. That example is probably not very useful, but it can be easily extended for RCE and similar things on many setups. For example, many FTP setups allow file submissions by anonymous users for various purposes. But that's ALL they can do. They can transfer a file into an "Uploads" directory but that's it. Now, however, after uploading their own cleverly-crafted code, the remote user is able to instruct the FTP server to copy that file to somewhere else in the system... which is really not what you want remote unknown and unauthenticated FTP client users to be able to do.

Miscellany

From: Patrick Delahanty / Web Engineer - TWiT.tv
Hi Steve,

I've been making some simple Apple TV apps for some TWiT shows that have evergreen content people like to watch even after the content has been rotated out of our podcast feeds. I've released a Security Now app for Apple TV that uses the TWiT API to access content and lets people watch or listen to every episode of Security Now going all the way back to the beginning.

http://twit.to/sntv

I hope your fans will enjoy it!

SQRL

Many projects are showing wonderful progress on Github...

- Scott White has "sqrl-ssp" working, a very nice pluggable implementation of the SQRL SSP API written in Go.

- Paul Farrer has a reference implementation of the SSP API in 'C', which he closely translated from mine in assembly language, so it's feature complete and supports several different TLS stacks.

- Daniel Persson, who created the Android client and the Wordpress plug-in, both which then evolved into very effective team efforts, is now working on "sqrl-libpam" a PAM (pluggable authentication module) for Linux.

- Adam Lenda, working with PHP and SQRL, is currently working to add SQRL support to the PHP Symfony application development framework.

- Jurgen Haas has SQRL for Drupal 8 in the works.
- Meanwhile, the Android, iOS and web browser extensions are rapidly nearing completion.

- I have finished the second of four SQRL documents. The first one was "SQRL Explained" and this next one is titled "SQRL Operating Details." Before making it widely available I notified those in GRC's NNTP newsgroup who have passed a very fine tooth comb through it for typos

and grammatical mistakes. So I'll be revising the document starting tonight and I'll post it widely. I'm now at work on the "SQRL Cryptography" document which will detail all of crypto-glue that holds SQRL together, and the final document will be "SQRL On The Wire" which will document the communications protocol. When these are finished we'll have both top-down and bottom-up views of SQRL.

- And, last but certainly not least, I'm very excited to have been invited to present SQRL, in late September, to two OWASP groups which will be meeting in Dublin, Ireland and Gothenburg, Sweden. Both have graciously offered to cover the direct travel and lodging expenses for Lorrie and me. (Lorrie would never forgive me if I went without her.) So I am very much looking forward to that two months from now. I'll be at the OWASP meeting in Dublin, Ireland on September 24th and then at the OWASP meeting in Gothenburg, Sweden two days later on September 26th. They are Security Now listeners, and if they have room I'd love to meet more of our listeners at those meetings. So, anyone interested should contact the organizers of those OWASP chapters and inquire about attending.

Hide Your RDP Now!

Sophos: "RDP exposed: the wolves already at your door"
For the last two months the infosec world has been waiting to see if and when criminals will successfully exploit CVE-2019-0708, the remote, wormable vulnerability in Microsoft's RDP (Remote Desktop Protocol), better known as BlueKeep.

The expectation is that sooner or later a BlueKeep exploit will be used to power some self-replicating malware that spreads around the world (and through the networks it penetrates) in a flash, using vulnerable RDP servers.

In other words, everyone is expecting something spectacular, in the worst possible way.
But while companies race to ensure they're patched, criminals around the world are already abusing RDP successfully every day, in a different, no less devastating but much less spectacular way.

Many of the millions of RDP servers connected to the internet are protected by no more than a username and password, and many of those passwords are bad enough to be guessed, with a little (sometimes very little) persistence.

Correctly guess a password on one of those millions of computers and you're in to somebody's network.

It isn't a new technique, and it sounds almost too simple to work, yet it's popular enough to support criminal markets selling both stolen RDP credentials and compromised computers. The technique is so successful that the criminals crippling city administrations, hospitals, utilities and enterprises with targeted ransomware attacks, and demanding five- or six-figure ransoms, seem to like nothing more.

	SamSam	Dharma	Matrix	BitPaymer	Ryuk
First appeared	2015	2016	2016	2017	2018
Active	No	Yes	Yes	Yes	Yes
Infection vector	RDP	RDP	RDP	RDP	RDP

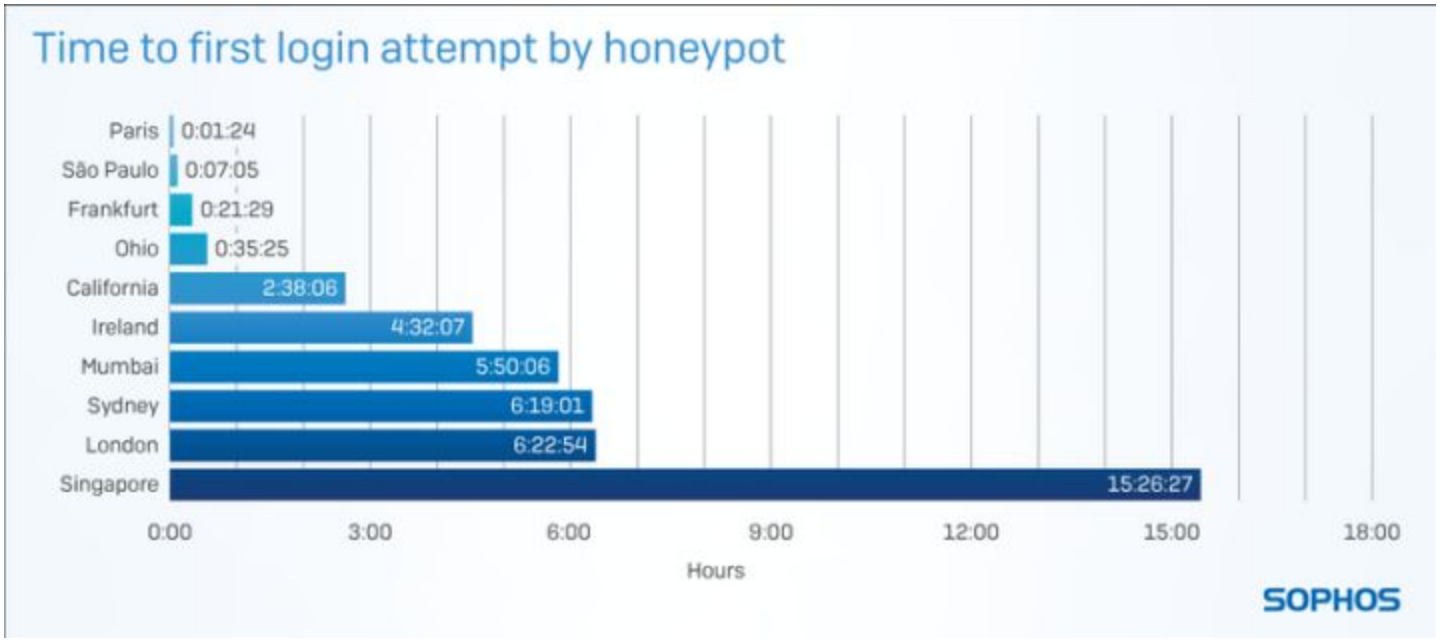
All of which might make you think – there must be a lot of RDP password guessing going on. Well, there is, and thanks to new research published by Sophos today, we can take a stab at saying just how much.

Noting the popularity of RDP password guessing in targeted ransomware attacks, Sophos's Matt Boddy and Ben Jones and I, set out to measure how quickly an RDP-enabled computer would be discovered, and just how many password guessing attacks it would have to deal with every day.

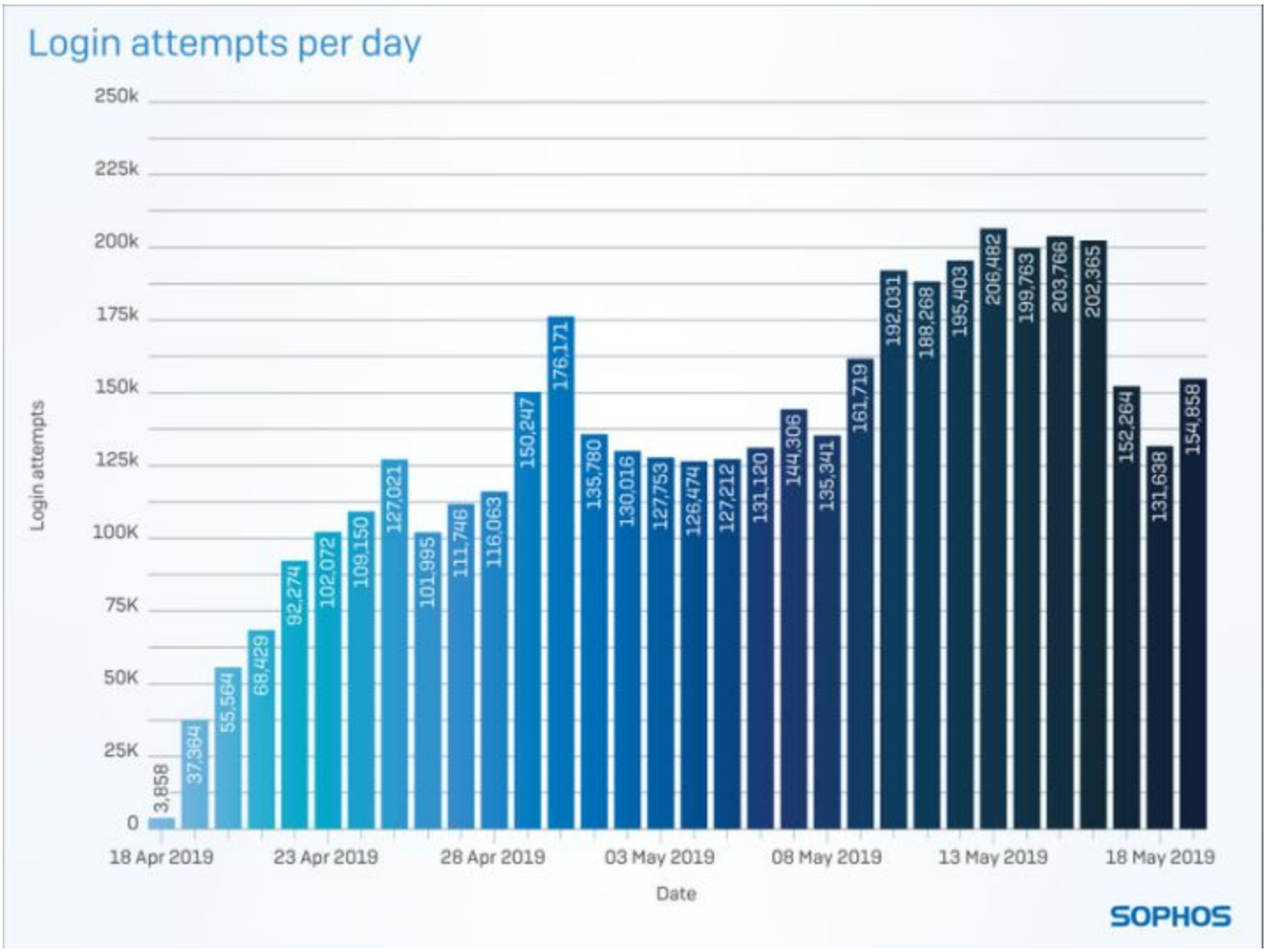
We set up ten geographically dispersed RDP honeypots and sat back to observe. One month and over four million password guesses later we they switched off the honeypots, just as CVE-2019-0708 was announced.

The low interaction honeypots were Windows machines in a default configuration, hosted on Amazon's AWS cloud infrastructure. They were set up to log login attempts while ensuring attackers could never get in, giving us an unhindered view of how many attackers came knocking, and for how long, and how their tactics evolved over the 30-day research period.

The first honeypot to be discovered was found just one minute and twenty four seconds after it was switched on. The last was found in just a little over 15 hours.



Between them, the honeypots received 4.3 million login attempts at a rate that steadily increased through the 30-day research period as new attackers joined the melee.



While the majority of attacks were quick and simple attempts to dig out an administrator password with a very short password list, some attackers employed more sophisticated tactics.

The researchers classified three different password guessing techniques used by some of the more persistent attackers and you can read more about them – the Ram, the Swarm and the Hedgehog – in the whitepaper.

What to do?

RDP password guessing shouldn't be a problem – it isn't new, and it isn't particularly sophisticated – and yet it underpins an entire criminal ecosystem.

In theory, all it takes to solve the RDP problem is for all users to avoid really bad passwords. But the evidence is they won't, and it isn't reasonable to expect they will. The number of RDP servers vulnerable to brute force attacks isn't going to be reduced by a sudden and dramatic improvement in users' password choices, so it's up to sysadmins to fix the problem.

While there are a number of things that administrators can do to harden RDP servers, most notably two-factor authentication, the best protection against the dual threat of password guessing and vulnerabilities like BlueKeep is simply to take RDP off the internet. Switch off RDP where it isn't absolutely necessary, or make it accessible only via a VPN (Virtual Private Network) if it is.

https://grc.sc/sn724

https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophos-rdp-exposed-the-threats-thats-already-at-your-door-wp.pdf

~30~