Security Now! #767 - 05-19-20

WiFi 6

This week on Security Now!

We begin this week as we often do on the third Tuesday with a look at the previous week's Patch Tuesday and, in this case, a troubling new trend is emerging. We look at the DoH support coming soon to Windows 10, and at a little known packet capture utility that was quietly added to Windows 10 with the October 2018 feature update. We'll spend a bit of time on yesterday's DOJ/FBI press conference and then take a look at a problem that Microsoft appears to be having a surprising time resolving. We'll take a look at face masks thwarting automated public facial recognition, and Utah's decision to roll their own contact tracing and locating app. And we'll wind up with what I hope will be an interesting walk through the history of Ethernet, from the beginning of wired to the evolution of the many confusing wireless protocols.

Our "Painful Pun" Picture of the Week

Would you say... he's wearing a Subnet Mask?

Security News

Patch Tuesday Redux

Last Tuesday's patching round was the not the biggest ever, but it was the 3rd largest in Microsoft's history, weighing in with a whopping 111 CVE-tracked bug fixes, 16 of which were rated CRITICAL and all but one of which enabled Remote Code Execution by an attacker. In a refreshing change of pace, none were 0-day flaws. But think about this... If things have been seeming worse recently, it's not our imagination, because the first and second place patched bug counts occurred with 115 bugs fixed in March and 113 in April. So these past three months -- March, April and May -- were the first, second and third most patched bugs in Microsoft's entire history. And although this month's bugs span 12 different Microsoft products, from Edge to Windows, and from Visual Studio to the .NET Framework... nearly half were problems within Windows itself.

I'm glad that these oversights are being fixed. And I'm glad that whatever it is that is wrong, doesn't appear to be directly affecting my own work when I'm using Windows 10. So perhaps these are all just exploitable edge cases. Since I was wondering exactly that, I took the time to read each and every one of this month's 111 descriptions, and I made a note of what it was that was wrong and was fixed. Every one of the 111 problems was exactly one of the following:

- Remote Code Execution Vulnerability
- Denial of Service Vulnerability
- Elevation of Privilege Vulnerability
- Cross-Site Scripting Vulnerability
- Memory Corruption Vulnerability
- Spoofing Vulnerability
- Information Disclosure Vulnerability

In other words, among those 111 problems there was not a single one that was NOT a vulnerability. So this suggests that it's not that Windows 10 is not working. As we all know, for the most part it works. I'm sure that many of us still recall those early days using Windows when it would just lock up at any time without apparent cause and always without warning. Some of the losses that resulted were so traumatic that to this day, several decades later, I'm still hitting "Ctrl-S" for "Save" before I ever switch away from anything I'm doing. I'm still not willing to trust that I'll be able to come back. In that sense it's like today's "Ctrl-C" for "Copy" that doesn't always seem to "take" the first time. So I hit Ctrl-C three or four times when I want to be SURE that I've copied something to the clipboard.

In the case of these myriad vulnerabilities, and those that WERE 0-days recently, this suggests that while Windows 10 works, you really don't want to continually expose it to too much incoming. If possible, get something to filter your eMail externally, use a well-curated web browser, and if you are going to go anywhere sketchy on the 'Net, do that in a VM. Pass anything you might download through VirusTotal before you move it out of the VM, and discard or revert any changes made to the VM while you're using it. It's probably not completely necessary, but consider that in just the past three months, 339 vulnerabilities were patched.

DoH test drive appears for Windows Insiders
Without indicating when it would receive a wider release, Microsoft is letting Windows Insiders test-drive DNS-over-HTTPS protocol in the Insider Preview Build 19628. Last Wednesday they wrote:

> *"If you have been waiting to try DNS-Over-HTTPS (DoH) on Windows 10, you're in luck: the first testable version is now available to Windows Insiders. If you haven't been waiting for it, and are wondering what DoH is all about, then be aware this feature will change how your device connects to the internet and is in an early testing stage, so only proceed if you're sure you're ready."*

Microsoft first indicated their interest and intent last November, and being at the OS level, unlike all existing browser support for DoH -- which we've seen is already quite comprehensive -- but limited in scope to the use of the browser, THIS would be native for all of Windows DNS.

Whereas DNS is ubiquitously available from every ISP using UDP protocol over port 53, DoH is not yet universal. So the first question anyone has is... What DoH provider does the system use? There =IS= a proposal in the works for added a DoH type to DHCP so that just as our systems are able to auto-configure to our provider's traditional DNS, they would be able to similarly auto-configure to our provider's DoH server once all of the pieces are in place.

But until that time, it's up to the user to decide which external DoH provider to use and, in Microsoft's case, three servers are currently supported to be used as DoH resolvers – Cloudflare, Google and Quad9. Windows needs to be configured to use one of these as a DNS server for DoH.

DoH will be disabled by default in the preview build. Naturally, anyone wishing to play with this will need to make sure that their Microsoft account is part of the Windows Insider program and that they are in the Fast Ring to receive the super-early builds for the next feature update of Windows 10. Once you have Build 19628 or higher you'll be able to activate DoH with a Registry tweak:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Dnscache\Parameters
Create a new DWORD value named "EnableAutoDoh"
Set its value to 2

This enables DoH. So as long as you have your Windows machine to use the DNS of Cloudflare, Google or Quad9, when Windows is restarted, rather than sending its queried to one of those three providers over DNS/UDP/53 as it was before, it will instead send its DNS queries over DoH to the same provider. That's sort of an elegant solution.

And... If you want to setup a DoH server that isn't already on what Microsoft calls their "auto-promotion list" -- meaning that DNS queries to that IP are auto-promoted to DoH -- because, for example, your ISP already supports DoH and you want to use it, additional auto-promotions can be created using the command line. You need to identify your ISP's DNS IP and their DoH URI template (which would be part of the ISP's DoH configuration information).

Then issue the following command:

netsh dns add encryption server={your-server's-IP-address} dohtemplate={your-server's-DoH-URI-template}

The "dohtemplate" is just the HTTPS URL that's associated with each DoH service. The template's active presence can be verified with this command:

netsh dns show encryption server={your-server's-IP-address}

As before, whenever Windows gets ready to use that IP for DNS, so long as DoH has been enabled with the registry tweak, it will get a template match on the IP and use DoH to the designated HTTPS URL instead of classic DNS.

Now you might ask... how do you know it's working?
Good question. The standard answer would be to use Microsoft's Network Monitor or Wireshark. But an intriguing option is to use Windows 10's little known, built-in and undocumetned "Packet Monitor" application named "pktmon.exe"

A built-in packet monitor you ask? That's right. It was quietly added with zero fanfare in the October 2018 Windows 10 feature update. And with this month's Win10 2004 feature update, it will be gaining the ability to dump directly to the console in real time, not just to a log file.

So... open a command prompt and type: **pktmon** to being playing.

For example, if you were to type:

pktmon filter remove - any existing filters would be cancelled.

pktmon filter add -t UDP -p 53 - would capture all traditional DNS over UDP queries.

pktmon start - will begin running counters for all matching packets.

pktmon start --etw - will capture the packets from a pktmon.etw file.

pktmon stop will stop the capture and dump a list of everything that happened...

```
Intel(R) Dual Band Wireless-AC 8260
 Id Name                    Counter  Direction Packets  Bytes | Direction Packets  Bytes
 -- ----                    -------  --------- -------  ----- | --------- -------  -----
 37 Native WiFi Filter Dr... Upper    Rx              43  5,523 | Tx              55  4,388
 36 Npcap Packet Driver (... Upper    Rx              43  5,523 | Tx              55  4,388
 35 VirtualBox NDIS Light... Upper    Rx              43  5,523 | Tx              55  4,388
 34 QoS Packet Scheduler     Upper    Rx               0      0 | Tx              55  4,388
                                                               |
 74 TCPIP (NDIS)             Lower    Rx              43  5,523 | Tx              55  4,388
 70 LLTDIO                   Lower    Rx              43  5,523 | Tx               0      0
 69 NDISUIO                  Lower    Rx              43  5,523 | Tx               0      0
```

`pktmon comp list` - will list the registered components, your system's network driver stacks.

The pktmon utility is a terrific built-in tool for performing quick checks on network traffic. Although there is ZERO documentation about it anywhere, it contains complete built-in help which is accessed by adding "help" to the end of any unfinished command.

If PKTMON captures your imagination, Bleeping Computer's Lawrence Abrams has reverse-engineered a few additional tips and tricks. The link to his coverage is here: https://www.bleepingcomputer.com/news/microsoft/windows-10-quietly-got-a-built-in-network-sniffer-how-to-use/

The DOJ and FBI again criticize Apple over encryption

I didn't want to fail to touch on the fact that in a press conference yesterday, William Barr, the head of the US Department of Justice, announced that FBI technicians had finally, after four months and <quote> "much expenditure of taxpayer dollars", managed to crack and gain access to the two locked iPhones belonging to last December 6th's Pensacola naval airbase shooter, Mohammed Saeed Alshamrani.

During the press conference, FBI's Director Christopher Wray criticized Apple for not helping its investigators unlock the two phones. Wray said the entire process of cracking the terrorist's two iPhones took four months and <quote> "large sums of taxpayer dollars."

Since actual cracking itself typically doesn't cost anything more than time and skill, either the FBI techs were extremely well paid during the past four months, or more likely, the FBI went outside to purchase the golden keys for those two iPhones. (And remember that one of the two phones had been shot by Alshamrani.)

In any event, the DOJ said that following the FBI's success, they were able to link Alshamrani to an Al Qaeda branch active in the Arabian Peninsula. William Barr said: "We now have a clearer understanding of Alshamrani's associations and activities in the years, months, and days leading up to the attack."

However, FBI Director Wray said the investigation could have advanced sooner if Apple had helped the FBI's technicians. Wray said that despite public pleas from both President Trump and the Attorney General Barr, Apple did not cooperate in the investigation.

But then, in an immediate contradiction, Christopher Wray continued: "Apple made a business and marketing decision to design its phones in such a way that only the user can unlock the contents no matter the circumstances. [...] Apple's desire to provide privacy for its customers is understandable, but not at all costs."

So... First he's complaining that Apple refused to cooperate with this specific case despite pleas from top administration officials, and with the next breath he's explaining that Apple chose to design their iPhones in such a way that it was impossible for them to comply. So I think we need to read this nonsense as the continuing drumbeat toward the inevitable collision of consumer encryption technology and legislation to criminalize the use of "subpoena proof" encryption.

We might imagine this might take the form of some stiff penalty legislation, where any manufacturer of a consumer electronics device produced for sale within the United States will have 30 days to comply with a lawfully issued subpoena to decrypt and provide all data contained within the device... after which a significant fine, perhaps some percentage of that company's annual revenue, would be levied against the company for each additional day beyond the initial 30 days that the device's decrypted contents are not provided to law enforcement.

The EFF weighed in, noting that the very fact that the FBI was able to crack the iPhones to obtain all of its information argued against the need for any change in the existing status quo.

When is a fix not a fix?
All too often it's when path traversal attacks are involved.

Leo, we were just talking recently about the original sin of hierarchical directory design with its inherent ..\..\.. path traversal vulnerability. From a security perspective it's soft of the file system equivalent of using the application's execution stack to temporarily buffer communication's data. What could possibly go wrong.

In this week's installment of "what did actually go wrong", we learn that when Microsoft fixed their reverse RDP attack problem in February, it wasn't really fixed... At least not for everyone.

Backing up a bit, recall that last summer several attacks against Microsoft's RDP protocol came to light. There was an authentication bypass against the server which led to last summer's spate of attacks against vulnerable RDP servers, and also a path traversal vulnerability that could compromise an RDP client which remotely accessed a malicious RDP server. I noted at the time that this one seemed significantly less worrisome since someone using RDP to access a server other than their own seemed unlikely.

This all came to light after Microsoft patched the vulnerability (CVE-2019-0887) as part of its July 2019 Patch Tuesday update. But it turned out that Microsoft only thought that they had resolved the problem. Researchers were able to bypass that patch simply by replacing the backward slashes in paths with forward slashes.

The researchers explained that the July patch can be bypassed because of a problem that lies in its path canonicalization function "PathCchCanonicalize," which is used to sanitize file paths. This allows a malicious attackers to exploit the clipboard synchronization between a client and a server to drop arbitrary files in arbitrary paths on the client machine. Thus, the clipboard redirection feature, while connected to a compromised RDP server, allows the server to use the shared RDP clipboard to send files to the client's computer to achieve remote code execution of arbitrary code.

Although Check Point researchers had originally confirmed that <quote> "the fix matches our initial expectations," it appears that it didn't actually fix the problem. The patch can be bypassed by replacing backward slashes (e.g., file\to\location) in paths with forward slashes (e.g., file/to/location), which are what, as we know, serve as path separators in Unix-based systems.

The researchers explained: “It seems that PathCchCanonicalize, the function that is mentioned in Windows's best practice guide on how to canonicalize a hostile path, simply ignored the forward-slash characters. We verified this behavior by reverse-engineering Microsoft's implementation of the function, seeing that it splits the path to parts by searching only for '\' and ignoring '/.'”

After this was pointed out, Microsoft acknowledged the incomplete fix and re-patched the flaw three months ago in its February 2020 Patch Tuesday update. But, in the latest chapter of this ongoing saga, a CheckPoint researcher observed that Microsoft addressed the issue by adding a separate workaround in Windows while not actually repairing the underlying cause of the bypass issue in the "PathCchCanonicalize" function. The upshot of this is that while the workaround apparently works fine for the built-in RDP client in Windows operating systems, the patch will not protect other third-party RDP clients against the same attack that relies upon the vulnerable Path Canonicalize sanitization function developed by Microsoft.

Check Point wrote: “We found that not only can an attacker bypass Microsoft's patch, but they can bypass any canonicalization check that was done according to Microsoft's best practices. As a result, a remote malware-infected server computer could take over any client that tries to connect to it.”

As I was doing the research into this latest bit of annoyance I had the sense that someone was not really paying attention over at Microsoft. We heard last year that they were really taking a much-needed long hard look at RDP and their other exposed server protocols. But the persistent incomplete “solving” (in quotes) of this problem made me aware that as buggy, vulnerable and patch-needy as Windows has become, at least we have always had the sense that the developers at Microsoft were quite highly skilled. And, frankly, they need to be to keep Windows moving in a more or less straight line. If that should ever change, Windows is done for.

Face masks have thwarted the London police's LFR rollout

“LFR” is, unfortunately, an abbreviation we're probably going to need to learn. It stands for “Live Facial Recognition.” Its initial trial-program use in London has been controversial, with the biggest problem being, aside from just the creepy “Big Brother” aspect of being monitored and surveilled without your explicit consent, is that in the best of times, even before everyone was wearing masks, the system was causing more trouble that it was worth due to extremely high false positive rates as high as 90%. In two recent LFR deployments, in which over 13,000 faces were scanned, six individuals were stopped as a result of a match, five of whom had been misidentified by the system.

And civil rights groups say that there is no clear legal basis for scanning the faces of potentially millions of citizens in the hope of catching a few people with fears that a wholesale rollout could contribute to a shift towards a surveillance state model adopted in countries such as China.

My own well-trained iPhone looks back at me with a great deal of puzzlement when I hold it up to my masked face, so the London Police's automated facial recognition system has been rendered utterly useless, at least for the short term. And as long as wearing face masks is considered polite, bad guys wishing to avoid any chance of automated recognition can appear to be acting with full social responsibility.

Utah chooses to roll their own contact tracing app
https://coronavirus.utah.gov/healthy-together-app/

... And my guess is that we can expect to see more of this since local governments are very likely to want to have more than just the super-anonymized and location-free prior contact data provided by the Apple/Google system.

Utah's "Healthy Together" app was created by a startup "Twenty Holdings" who is best known for their "Twenty - Hang Out With Friends" social app which allows users to "See who's around. See who's down. Hang out." So the company already has a platform for enabling physical, in-person connections. No doubt, when this all happened, they thought "Hey! We already have a contact dating app, we can easily convert it into contact tracing!" So their "Healthy Together" app uses everything it can get its hands on, including physical location data including GPS, WiFi access point proximity, cellular phone tower triangulation and Bluetooth. Its goal is to pinpoint users locations and location history and ID coronavirus breakouts and hotspots.

And I've got to say, for those who are civic minded and who do not mind the privacy implications of explicit historical tracking -- since that's clearly what's going to be necessary -- for the duration of this pandemic I think it makes a lot of sense. There's nothing that I'm doing in my life that's the least bit controversial. So I, personally, would not hesitate to add that to my phone for the duration. If I were to become infected, and my location and time history could be played back to determine where that occurred, and everyone else who was also present in the same group at the some time could then be interviewed and checked... that would prove to be of vital value.

https://coronavirus.utah.gov/healthy-together-app/

Utah's site explains: Healthy Together Beta App

Protect yourself and your family. Utahns are working to slow the spread of COVID-19. We can work together to protect our family members, friends, health workers, and our communities. The Healthy Together app helps you assess your symptoms, find the nearest testing center, view test results, and learn what to do after you've been tested for COVID-19.

- Assess your symptoms: Use the symptom checker to see if you need to be tested.
- Find the nearest COVID-19 testing center: Testing centersare located across the state.
- Learn what to do after you get tested: Get your test results and instructionsfor care.
- Location data: Find COVID-19 hot spots to focus public health efforts.

Q: *How does the Healthy Together beta app help protect me and my family and slow the spread of COVID-19?*

A: The Healthy Together beta app helps you assess your symptoms, find the nearest testing center, view test results, and learn what to do after you've been tested for COVID-19. We can work together to slow the spread of COVID-19 and protect our family members, friends, health workers, and our communities. If authorized by the user, the app can also provide location data to public health workers, providing them with a faster and more accurate picture of where and how the virus is spreading in our community to focus public health efforts.

***Q**: What happens to my data?*

***A**:* Protecting your data is of utmost concern to the State of Utah and the developer, Twenty. To ensure the privacy and security of the data will follow these principles and limitations:

- Using the app is strictly opt-in and voluntary.
- You own your data and can delete it at any time.
- Only data required to combat COVID-19 will be shared with public health officials.
- Location data is automatically deleted after 30 days.
- Symptom data is automatically de-identified after 30 days.
- The developer will comply with State requirements for data security and encryption.
- You can decide what data you would like to share – for example, bluetooth data, location data, or contact lists.

***Q**: Who has access to my data? Is the data shared with any third parties? What details are shared?*

A: Your data is secure and you are in full control of what you choose to share. Only data that is useful to combat COVID-19 will be shared with public health officials.

While the State will have access to your symptom data, location and bluetooth data will only be released to the state should you test positive for COVID-19.

***Q:** Which public health officials will have access?*

Utah has trained a team of contact tracers under the Utah Department of Health who reach out to people who have tested positive for COVID-19 and have been potentially exposed to the disease. When you grant access to your location or GPS and bluetooth data, members of this team will be able to access your data to help in the contact tracing process.

The app will help these professionals identify transmission zones, contact patterns, and other vital information to inform their research.

***A:** Why aren't you using just bluetooth like Apple/Google is? Or utilizing the API that Apple/Google built?*

Bluetooth on its own gives a less accurate picture than bluetooth and GPS location data. The goal of Healthy Together is to allow public health officials to understand how the disease spreads through the vector of people and places, and both location and bluetooth data are needed to accomplish that.

Bluetooth helps us understand person-to-person transmission, while location/GPS data helps us understand transmission zones — having both of these important data points provides a more effective picture of how COVID-19 spreads. This data helps policy makers make the best possible decisions about how and where we begin to relax and modify restrictions as our community and economy begin to reactivate.

SpinRite

Work is continuing quite well on the SpinRite project. Yesterday, before switching to work on this podcast, I posted my just-completed FAT partition formatting code for the gang in the newsgroup to pound on and it's working surprisingly well so far. I need SpinRite's thumb drive boot installer to be able to work on any old or new thumb drive the user might have around, regardless of that drive's history. SpinRite's current installer, which I wrote back in 2004, was also showing its age and needed reworking.

That work will be finished shortly, at which point everyone will be able to more easily boot the testing code on their machines. Then, the new AHCI driver code will be merged with the other new drivers to produce an interesting benchmark that we'll all be able to use to really pound on and test the suite of drivers before they are integrated into SpinRite itself.

WiFi 6

Since we're going to do a bit of historical framing to place WiFi 6 into context, and since the **47th anniversary** of the invention of **Ethernet** is just three days away, I thought I'd start by describing how that happened:

Ethernet was invented by a guy named Bob Metcalfe. We've talked a lot on this podcast about packet switching. Bob is the person who built some of the first hardware. In 1970, while I was rapidly falling in love with assembly language on a PDP-8 and Bill Gates was playing with a newly installed teletype at his high school which was hooked to a remote time sharing system, Bob Metcalfe was building an interface, known as the IMP, for Interface Message Processor, which linked a PDP-10 at MIT to the ARPAnet. Three years later in 1973, after building a second IMP-host interface at Xerox Parc, he was assigned the task of somehow extending the ARPAnet into buildings full of PARC's personal computers -- which, of course at the time, was the only place in the world where it had occurred to anyone that computers might be personal. And on May 22, 1973, Bob wrote the memo inventing Ethernet. Today, 47 years later, more than 1.2 **BILLION** new Ethernet ports are shipped each year, 1/3rd are wired, 2/3rds are WiFi.

So let's follow the path that leads from there to today's WiFi 6...

Bob's wired Ethernet first appeared commercially in 1980 and was standardized by the IEEE (Institute of Electrical and Electronics Engineering) as IEEE 802.3, what would become a growing family of electrically connected Ethernet adapters. In 1983 we had 10 megabits/sec with 10BASE5 which used a thick coax cable. In '85 the much more popular 10BASE2 switched to a much more manageable thin coax. In 1990 we got 10BASE-T where the 'T' stood for twisted pair, which initially ran at the same 10 megabits. '95 saw the jump to 100BASE-T, or the so-called Fast Ethernet which gave us a 10-fold jump to 100 megabits/sec. In '98 we got 1000BASE-X for 1 gigabits over fiber optic cabling and in 1999 the engineers figured out how to deliver the same speed over the much more convenient multiple twisted pairs.

In 1997, when our 100BASE-T was then a couple of years old, engineers began looking at wireless. Whereas the family of wired Ethernet were all 802.3, the IEEE assigned 802.11 for their work on taking the same time-proven Ethernet technology wireless.

The very first 802.11 was mostly experimental, being quickly followed up two years later in 1999 by 802.11a. Whereas the first 802.11 operated in the 2.4 GHz band and was only able to deliver around 1 or 2 megabits per second, 802.11a moved into the 5 GHz band with a physical bit rate of 54 megabits/second, but it also relied upon a lot of forward error correction so that it delivered an effective data rate in the mid-20 megabits/second range.

The 2.4 GHz band is occupied to the point of being crowded with microwave ovens, Bluetooth, baby monitors, cordless telephones, and some amateur radio equipment. So moving 802.11a into the relatively unused 5 GHz band gave it a significant advantage. But the higher carrier radio frequency brings some trade offs since a higher frequency means a shorter wavelength which increases the signal's absorption by walls and other solid objects in their path.

As a consequence, the next move was to work to improve the performance of the inherently more robust 2.4 GHz lower frequency band, which gave us 802.11b products starting 20 years ago in the year 2000. 802.11b used a more advanced carrier modulation scheme to deliver a maximum effective bit rate of 11 megabits/second. The products were inexpensive and plentiful and WiFi really began to take off.

And since 802.11a operated at 5 GHz and 802.11b operated at 2.4 GHz, it was feasible to create dual-band WiFi, known as 802.11a/b.

Three years later, in 2003, 802.11g delivered a third carrier modulation standard for the lower band 2.4 GHz. It managed to deliver the same 22 megabits/second throughput of the much more finicky 5 GHz 802.11a while operating in the inherently longer-range (though more congested) 2.4 GHz band.

And as with 802.11a/b, all three modes were often combined to yield 802.11a/b/g.

At this point, things had become a bit of a mess, so work was undertaken to pull all of the various amendments to 802.11 together and to figure out where to go next. The result that emerged by the end of 2009 was 802.11n -- even though (and we talked about this on the podcast at the time) the industry didn't actually wait for the publication of the formal standard, jumping-the-gun by a couple of years by following a draft specification. There was just too much pent up need. 802.11n combined everything, and it would later come to be retroactively labeled "WiFi 4" by the WiFi Alliance. It introduced the three-antenna "MIMO" -- Multiple Input / Multiple Output -- system and operates on both the 2.4 and 5 GHz bands, although support for 5 GHz is considered optional in the spec.

Then, seven years ago in 2013, the 802.11n "WiFi 4" standard had its 5 GHz transmission channel bandwidth significantly widened from 40 MHz to 80 or 160 MHz, the number of allowable spatial streams was doubled from 4 to 8 and the modulation scheme jumped to a higher order from 64-QAM (quadrature amplitude modulation) to 256-QAM --so from 6 bits to 8 bits. And since it's all just silicon, they also defined multi-user MIMO, or MU-MIMO... All of which results in 802.11ac which was also later retroactively labeled "WiFi 5"

We should pause our history here for a moment to talk about this. The original Ethernet used a system known as Carrier Sense multiple access with collision detection. It was essentially a party line. The idea was that someone wanted to send something to someone else would listen until the shared party line was quiet, then would transmit their data. But it was entirely possible, especially on a busy shared network with many nodes, that multiple parties might start transmitting at the same time. Back when Ethernet used coax, such a collision would actually create a higher voltage swing on the coax which was readily detected by everyone on the line. So the parties who were responsible for the jam-up would wait a random length of time before retrying. The system was elegant, simple and clever. Later, when twisted pair wiring was used, there was not an over-voltage event. So the transmitting parties would listen to the line while transmitting to see whether they were able to reliably receive the message they sent. If not, that meant that someone else was transmitting and interfering with them, so, again, backoff a random interval and retry. The point was... Ethernet has always been a shared medium technology.

When semi-intelligent Ethernet switches replaced simple repeater hubs, things got better, since these semi-intelligent switches could dynamically learn the network topology by building a table of which Ethernet MAC addresses were connected to which of their ports. So now, rather than simply sending anything incoming out of all ports, the incoming data would automatically be routed out of the one port where the destination MAC address had previously been seen. This hugely reduced packet collisions within large networks.

Multi-user MIMO aims (of you'll pardon the pun) to do the same sort of thing for radio, which is otherwise a single massively shared mess of a medium. Multi-User MIMO is a spatial multiplexing technique. It's literally radio beam forming. With MIMO, access points send out "sounding frames" with varying antenna phasing and clients reply with the received signal strength they obtain from each. Over time access points “learn” how to best send information to specific fixed-location clients to effectively form beams individually aimed at each Client. This reduces the system's overall radio interference. As with a wired Ethernet hub versus a switch, this technology reduces the interference between clients with the result that the overall network throughput is increased by allowing multiple clients to receive data at the same time.

So... 802.11ac, also known as WiFI 5 introduced this MIMO technology. It was was actually defined and equipment was certified in two rounds, with the second round in 2016 adding the higher bandwidth capabilities of multi-user MIMO, the widest 160 MHz channel bandwidth, more 5 Ghz channels and the doubling of the number of spatial streams with four antennas versus three in the first round.

Which brings us now to 802.11ax, also known as WiFi 6.

802.11ax aims to quadruple a single access point's overall data exchange capacity in heavily used multi-client environments while delivering about 30% more performance to each client compared to WiFi 5's 802.11ac. Multi-user MIMO could be thought of as "spatial domain multiplexing" to reduce inter-client interference. To that, 802.11ax significantly enhances its frequency domain multiplexing with MU-OFDMA -- multi-user orthogonal frequency-division multiple access -- which is a fancy way of saying that the entire available spectrum allocation is dynamically divided into a very large number of much smaller subcarriers. And by a very large number I mean 2048 individual subcarriers.

This much higher level of individual attention to each client, which requires WiFi 6 compliance at both ends, promises to cut individual client latency by 75%, which should be quite noticeable to all WiFi 6 capable users of a WiFi 6 access point. Anyone who has ever attempted to use satellite-based Internet knows what a thrill killer network latency can be!

There are also significant power savings possible for the client, thanks to individual "Target Wake Time" (TWT) negotiation introduced in 802.11ax.

And just last month the FCC delivered some very good news to the WiFi industry by agreeing to open and make available a significant amount of new bandwidth in the 6 GHz band. This additional 1200 MHz worth of WiFi spectrum will add 14 80 MHz channels and 7 160 MHz channels. So we can expect to see reduced interference, even lower latency, gigabit speeds, and higher capacity for simultaneously managing more devices.

I was getting ready to make the move to WiFi 6, but since I don't yet have many other WiFi 6 client devices, and since last month's chunk of new unlicensed bandwidth means that another generation of access points will be coming along, I think I may wait until the newer access points, which add the 6 GHz band, hit the market. But I don't expect those devices soon since it'll likely require tooling up new silicon and radio design from scratch. So, depending upon your need for WiFi 6 today, you choose rationally choose not to wait.