Security Now! #855 - 01-25-22

Inside the NetUSB Hack

This week on Security Now!

This week we briefly touch on the ongoing Log4j background noise. We look at the result of the insurance industry's pushback against ransomware coverage and at the resulting changing cyber-insurance landscape. We look at another WordPress add-on problem and a supply-chain attack on a very popular add-on provider. We also wonder whether WordPress still makes sense in 2022? We cover the EU's quite welcome major bug bounty funding, and Kaspersky's discovery of a very difficult to root out UEFI bootkit. We'll share some interesting questions and topics suggested by our listeners, then we're going to take another of our recent technical deep dives to examine the precise cause of that pervasive NetUSB flaw — it's really fun and completely understandable!

Log4J News

I didn't want to skip over anything important regarding Log4j this week. And the good news is that there were no clear Log4j-related disasters reported in the past week. There was plenty of Log4j stirring. Microsoft reported that the SolarWinds Serv-U servers were under Log4j attack in a failed attempt at leveraging their support for the LDAP protocol. Though the attacks failed, SolarWinds has updated their Serv-U servers. And Akamai reported seeing attacks aimed at Zyxel networking gear. In response, Zyxel has updated one of their products and posted: *"Zyxel is aware of remote code execution (RCE) vulnerabilities in Apache Log4j and confirms that among all its product lines, ONLY NetAtlas Element Management System (EMS) is affected. Users are advised to install the applicable updates for optimal protection."* So it was mostly just like that. No huge drama. Just stirrings and rumblings and general indications of ongoing, largely behind the scenes work by many, to remediate the surprising discovery of such a widespread vulnerability. The way "The Record" summed things up was to write:

> *"While news cycles move fast from topic to topic, the situation around the Log4Shell exploit has not changed since last month, and the vulnerability is still heavily targeted and abused by threat actors seeking to enter corporate networks.*
>
> *At the time of writing, there have been reports about threat actors such as ransomware gangs, nation-state cyber-espionage groups, crypto-mining gangs, initial access brokers, and DDoS botnets, all of which are using the vulnerability in their operations."*

So, all of the usual suspects jumped on this quickly and all of their various operations acquired yet another means for remotely penetrating networks.

Security News

Who pays for RansomWare attack recovery?

I find the topic quite interesting since one of the targeting parameters used by the attackers has been their victim's ability to pay. And in the case of many public and private targets, their cyber-insurance has been the actual source of ransomware payment and post-attack recovery funding.

Four years ago, back in June of 2017, the pharmaceutical giant, Merck, was hit by the NotPetya ransomware in a devastating cyberattack. The attack was pretty devastating and Merck claimed that the data on more than 40,000 Merck computers was destroyed. At the time, Merck estimated the damage to it at $1.4 billion, a quite sizeable loss resulting from production outage, costs to hire IT experts, and costs of buying new equipment to replace all affected systems. As we discussed at the time — and I remember you, Leo, wondering about the size of that number — that seems a bit steep, and we questioned whether replacing 40,000 affected systems was warranted.

But in any event, Merck was quite well insured, carrying a $1.75 billion "all-risk" insurance policy, which included coverage for software-related data loss events. Merck's insurer, "Ace American" wasn't happy about the idea of paying out $1.4 billion and so refused to cover the losses, citing that the NotPetya attack was part of Russian hostilities against Ukraine and, as a

result, was subject to the "Acts of War" exclusion clause which is a standard boilerplate exclusion present in most insurance contracts.

So, Merck sued Ace American in November 2019, arguing in court that the attack was not "an official state action," hence the Acts of War clause should not apply. Merck's attorneys argued that the exclusion clause contained language that limited the Acts of War to official government agencies and did not specifically mention cyber-related events; and, as a result, the "Act of War" exclusion clause should not apply to their customer. This popped back into the news last week after a court in New Jersey agreed with what seemed rather clear cut given the language of the contract, ruling in favor of Merck.

Judge Thomas J. Walsh wrote in an opinion justifying the ruling: "Given the plain meaning of the language in the exclusion, together with the foregoing examination of the applicable case law, the court unhesitatingly finds that the exclusion does not apply." The judge argued that despite knowing that cyber-attacks can be acts of war, Ace American did not move to update the language in its exclusion clauses. He said: "Certainly they had the ability to do so. Having failed to change the policy language, Merck had every right to anticipate that the exclusion policy applied only to traditional forms of warfare."

Although the case wasn't mainstream news, the insurance industry has been watching it closely, and it has had a large impact on the cyber-insurance business with several insurers updating the language of their Acts of War exclusion clauses, the latest being Lloyd's, which updated their language just days before the court's ruling.

The Merck case is attention grabbing due to the amount of money involved. But there have been a great many lower-profile lawsuits brought by ransomware victims in recent years and those cases have also been won. The insurance companies are not happy and they are responding.

The rising cost of cyber-insurance.

The Suburban School Cooperative Insurance Program (SSCIP) is an insurance pool designed to allow school districts to join together to negotiate better insurance rates and lower management fees. One of the school districts participating in the Co-op program is the Bloomington School District 87 in Chicago, Illinois. That school district recently published its cyber-insurance renewal details, reporting that its cost jumped in one year from $6,661 in 2021 to $22,229 for 2022, 334% of the previous year. And, in fact, the Cooperative was apparently fortunate to even get that. In their memo, the district noted that: "In light of events that have negatively impacted the Cyber Insurance market, SSCIP was unable to initially find the required coverage for the group. After a small delay, the Cooperative was ultimately able to secure an insurer willing to accept the risks of the pool."

EmsiSoft recently published a summary of ransomware attacks in 2021. They wrote: "Predictably, 2021 was largely a replay of the previous two years, with the US public sector again experiencing a barrage of financially-motivated ransomware attacks. The attacks impacted a total of 2,323 local governments, schools and healthcare providers."

- 77 state and municipal governments and agencies.
- 1,043 schools.
- 1,203 healthcare providers.

An interesting bit of the memo published by District 87 indicated that the districts participating in the Cooperative insurance policy would be required to do more than just pay their premiums. To qualify for this insurance, the districts would also be required to fully implement multi-factor authentication. And, interestingly, until that is done the insurance coverage limits will remain low. The districts estimate that they can get MFA fully implemented by the end of March.

As we know, in a Log4j world, adding multi-factor authentication to logins won't help, since the bad guys are crawling in through a back door, rather than guessing the authentication protecting the front door. But adding multi-factor authentication where it's been absent before — which was probably everywhere — would certainly remove the lowest hanging fruit.

Overall, though, the cost of cyber-insurance coverage is predictably increasing significantly across the board for anyone who needs it.

Another very dangerous WordPress add-on

The guys at WordFence have uncovered another very critical (CVSS of 8.3) flaw in a WordPress add-on named "WP HTML Mail." WP HTML Mail is an Email template designer used for designing custom eMails. Unfortunately, its use is currently exposing more than 20,000 WordPress sites that use it to malicious code injection, phishing scams and more.

The root of the problem is a faulty configuration in the REST-API routes used to update the template and change settings. There is no authentication required to access the REST-API endpoint. Consequently, any user has access to execute the REST-API endpoint to save or retrieve an email's theme settings. This would allow the injection of malicious JavaScript into the mail template that would execute anytime a site administrator accessed the HTML mail editor.

So, threat actors could add new users with administrative credentials, inject backdoors, implement site redirects, and use legitimate site templates to send phishing emails, among many other things — even site takeovers.

WordFence said that: "Combined with the fact that the vulnerability can be exploited by attackers with no privileges on a vulnerable site, this means that there is a high chance that unauthenticated attackers could gain administrative user access on sites running the vulnerable version of the plugin when successfully exploited."

As I said, the plugin is installed at more than 20,000 sites and is compatible with other plugins run by WordPress sites with large followings like eCommerce platform WooCommerce, online form builder Ninja Forms and community builder plugin BuddyPress.

WordFence wrote in their disclosure: "We recommend that WordPress site owners immediately verify that their site has been updated to the latest patched version available, which is version 3.1 at the time of this publication."

And, for what it's worth, this latest disclosure comes just a week after the firm "Risk Based Security" published their findings that the number of WordPress plugin vulnerabilities exploded by triple digits in 2021. As we mentioned last week, three different WordPress plugins, all written by the same author, were reported with the same bug, thus exposing 84,000 sites running eCommerce add-ons to full site takeovers.

Is anyone else here noticing a worrisome trend with WordPress? I'm certain that the developers of these plug-in have the best of intentions. But, as we're going to see in detail at the end of today's podcast, writing secure code is surprisingly difficult. Most developers stop their work the moment their code starts to work. That often means that the security of that code, as a vital aspect, is barely if ever considered.

And a supply-chain attack on a popular WordPress add-on provider

It's bad enough when plug-in authors who are untrained in security coding make mistakes. But when deliberately malicious actors are able to get their malicious code into widespread and popular add-ons, things get even worse. In this case, threat actors were able to compromise 40 themes and 53 plugins all belonging to AccessPress, a developer of popular WordPress add-ons which are in use by over 360,000 active WordPress websites.

The guys at Jetpack, who are professional Wordpress developers explained their discovery:

While investigating a compromised site, we discovered some suspicious code in a theme by AccessPress Themes (aka Access Keys), a vendor with a large number of popular themes and plugins. Upon further investigation, we found that all the themes and most plugins from the vendor contained this suspicious code, but only if downloaded from their own website. The same extensions were fine if downloaded or installed directly from the WordPress.org directory.

Due to the way the extensions were compromised, we suspected an external attacker had breached the website of AccessPress Themes in an attempt to use their extensions to infect further sites.

We contacted the vendor immediately, but at first we did not receive a response. After escalating it to the WordPress.org plugin team, our suspicions were confirmed. AccessPress Themes websites were breached in the first half of September 2021, and every one of their extensions available for download on their site was injected with a backdoor.

Once we had established a channel for communicating with the vendor, we shared our detailed findings with them. They immediately removed the offending extensions from their website.

Most of the plugins have since been updated. I won't take up any more of our time on this, specifically. But I think that our constant Extinction Level Event reports about WordPress ought to give us some pause...

Does WordPress make sense anymore?

I think its becoming clear that WordPress, with its add-on ecosystem made a lot more sense when it was initially released 19 years ago, back in 2003, than it does today. There is zero

control over the design of WordPress add-ons. Anyone can make one. And WordPress site admins love to add tasty bits to their sides. This means that it's really not possible to care about online security while running a WordPress site with a bunch of add-ons.

Yet WordPress remains incredibly popular, powering a whopping 39.5% of all Internet websites. 39.5%!! Considered a content management system, or CMS, it commands 62% of all CMS with Sappily in a distant second place at 3.2%.

I have not looked around the CMS marketplace, so I don't have anything to recommend. But it's very clear that this 19 year old model of anyone producing add-on plug-ins for WordPress is both super-functional and super-insecure.

The European Union plans to fund some bug bounty programs

In a welcome bit of news, we learn that the European Union will be funding a bug bounty program for five open source projects that are heavily used by public services across the EU.

The five programs include:

- LibreOffice, the very good free alternative to Microsoft Office;
- Mastodon, a web-based system for hosting private social networks;
- Odoo, an enterprise resource planning (ERP) application;
- Cryptopad, an app for exchanging encrypted messages; and
- LEOS, software designed to help with drafting legislation.

The bug bounty program will run throughout 2022 on the Intigriti bug bounty platform, and the EU will provide a rewards pool of up to €200,000 ($225,000).

Intigriti, whom we've never mentioned before, describes itself as *"Europe's #1 ethical hacking and bug bounty platform"* https://www.intigriti.com/ So they're like HackerOne. They have more than 300 active programs, more than 40,000 researchers hunting bugs, and more than 3 million Euros paid to date.

Bug hunters will be eligible to earn as much as €5,000 ($5,600) for finding and reporting "exceptional vulnerabilities," with a 20% bonus if they provide a fix within their reports.

And, you know, having covered the recent problems with Microsoft patching against proofs of concept, rather than repairing the underlying problems, I think that this idea of also getting the fix from the researcher who discovers the problem makes a great deal of sense. Who better to understand the problem that they found and suggest its proper and complete fix? Now, admittedly, that's more feasible with Open Source projects than it is within a close-source environment like Windows. But it's a very sane perk to offer, and who wouldn't go for it?

The new program was announced last week and is sponsored by the European Commission Open Source Programme Office (EC-OSPO) which was founded in 2020. It succeeds the EU-FOSSA (EU-Free and Open Source Software Auditing) project, through which the EU had previously funded two other bug bounty program initiatives for open-source software in 2017 and 2018, respectively.

In 2017 the EU funded bug reports for VLC Player — my own preferred standalone video player — and in 2018 the EU-FOSSA sponsored bug reports for 14 projects, such as 7-zip, Apache Kafka, Apache Tomcat, Digital Signature Services (DSS), Drupal, Filezilla, FLUX TL, the GNU C Library (glibc), KeePass, midPoint, Notepad++, PuTTY, the Symfony PHP framework, again the VLC Media Player, and WSO2.

The same program also funded security audits for the Apache HTTPD web server and the Keepass password manager.

So, a huge YAY! to all of this! It would be GREAT to see more of this in this and coming years. As I said last week, the industry is already doing many things that make a great deal of sense. We just need more of what we're already doing. This is definitely that.

The "MoonBounce" EFI Bootkit

UEFI was designed to be a secure and securable boot platform firmware foundation. Everyday it's looking less and less so.

Last Thursday, researchers at Kaspersky Labs disclosed their discovery of yet another novel UEFI bootkit which can infect a computer’s UEFI firmware.

What makes “MoonBounce” — that's the name they gave it — special is that unlike some previous pre-boot malware, this one doesn’t hide inside the hard drive's EFI System Partition where UEFI malware typically resides. Instead, it infects the SPI memory that is found on the motherboard. (SPI stands for Serial Peripheral Interface.) Consequently, unlike other bootkits, defenders cannot shake this one by reinstalling the operating system and/or replacing the hard drive. This pernicious bootkit will continue to remain on the infected device until the motherboard's SPI memory is re-flashed or the motherboard is replaced.

And, according to Kaspersky, MoonBounce is not even the first UEFI bootkit they've seen that can infect and live inside the SPI memory. It's the third, following LoJax and MosaicRegressor.

And, sadly, UEFI bootkits are proliferating. Recent months have uncovered ESPectre, FinSpy’s UEFI bootkit and others. Kaspersky commented that what was once considered unachievable following the rollout of the UEFI standard has gradually become the norm.

Oh, and I almost forgot to mention that MoonBound has been directly attributed to the Chinese government’s state sponsored hacking group, APT41.

Closing the Loop

miwo86 / @Miwo86

Morning Steve. What do you use to run pfsense on? I'm looking at switching to that from a UniFi security gateway.

- NetGate SG-1100 — https://shop.netgate.com/products/1100-pfsense
  https://protectli.com/product-comparison/
  Three Ports: WAN, LAN and OPT.
  Turnkey out of the box running pfSense.
  Routing: 880 Mbps / Firewall: 656 Mbps / IPsec VPN: 74.2 Mbps
  $189

- Protectli: https://protectli.com/
  2-, 4- or 6-port models. Pick the processor, RAM, mass storage.
  It's a tough little industrial-feeling solution.

Peter Crocker / @brndpete

Filling CO2. The big tanks need to be tested every 5 years in a hydrostatic test. Basically a requirement of any pressure cylinder like welding gas, scuba tank or CO2. Then they stamp it with a new date. Most places send it away for the test. In terms of filling the big tanks, they need to be cold and filled slowly, so often not a service you wait for, hence some exchanging if you want it right away.

Paul Walker / @PaulWalkerUK

Hi Steve, have you considered updating ShieldsUp so it can scan the full port range at once? Just been listening to episode 854 where you talk about port 20005 - if we could scan the entire range, it might help preempt things like this.

Dylan Anthony / @TBDylan

According to the author of cURL "it's not" an RCE. So you're not the only one who couldn't figure out why Microsoft categorized it as such 😜
https://curl.se/docs/CVE-2021-22947.html

Inside the NetUSB Hack

I did some more examination of the NetUSB hack which reportedly affects many millions of Internet-connected routers. Last week when we first talked about it, I skipped over the techie details because there was so much to talk about other than the nitty gritty of the attack's actual mechanics. But after spending some time looking into what went wrong with the code, I realized that not only would a full explanation of the flaw be well within our listeners' ability to grasp, but that it would serve as another very valuable example of precisely the way things go wrong with software and Internet security.

One thing I should note is that this flaw is incredibly widespread. The tiny list of router vendors that I shared last week from the guys who found this flaw should not provide anyone with any solace. If your router has a USB connection port, you really do need to make absolutely certain that it's not listening on its WAN interface port 20005.

This is not the first time that Kcodes Technology, the Taiwanese developer and licensor of the NetUSB technology has had trouble. This Security Now podcast episode #509, recorded May 26th, 2015 — nearly seven years ago — was titled "The NetUSB Bug." That bug was a biggie too, and perhaps because we were all seven years less jaded by the endless parade of vulnerabilities than we are today, there was more attention paid to the breadth and scope of the trouble that its exploitation might cause. It feels to me as though today's tech press got one headlined page out of the story then went back to wondering about NFTs.

That earlier vulnerability, which was found in the same NetUSB kernel module, was discovered by a group called SEC Consult. At the time, they wrote:

> *NetUSB suffers from a remotely exploitable kernel stack buffer overflow. Because of insufficient input validation, an overly long computer name can be used to overflow the "computer name" kernel stack buffer. This results in memory corruption which can be turned into arbitrary remote code execution. Furthermore, a more detailed summary of this advisory has been published at our blog:* https://blog.sec-consult.com.

And they provided a benign proof-of-concept which simply crashed the targeted router. In their disclosure under vulnerable and tested versions they wrote: *"The vulnerability has been verified to exist in most recent firmware versions of the following devices: TP-Link TL-WDR4300 V1, TP-Link WR1043ND v2, NETGEAR WNDR4500."*

But then they added that they had identified NetUSB as being present in the most recent firmware version of the following products, noting that the list they were providing was not necessarily complete. And you'll be thanking me for not reading the list because there are truly too many for me to read. So I shortened the list for the podcast. But I want to give everyone a sober feel for it: In the list was the D-Link DIR-615C, then **42** different NetGear router models, **40** different TP-LINK models, **14** TrendNet models, and **4** Zyxel routers.

And they added that *"Based on information embedded in KCodes drivers we believe the following vendors are affected: Allnet, Ambir Technology, AMIT, Asante, Atlantis, Corega, Digitus, D-Link,*

EDIMAX, Encore Electronics, Engenius, Etop, Hardlink, Hawking, IOGEAR, LevelOne, Longshine, NETGEAR, PCI, PROLiNK, Sitecom, Taifa, TP-LINK, TRENDnet, Western Digital, ZyXEL."

This one company appears to have cornered the market on, and they do claim to have patents on, extending USB links across consumer WiFI to router USB ports. Although that was 7 years ago, there is no reason to believe that any router using USB extension today is NOT using Kcode's troublesome technology.

SEC Consult's original write up provided a full responsible disclosure timeline, and it would be charitable to say that Kcodes Technology was responsive. SEC Consult finally disclosed what they had found to the CERT Coordination Center (CERT/CC) and directly to several of the major and most seriously affected router vendors. After speaking with some of the gazillion vendors, they wrote: "Sometimes NetUSB can be disabled via the web interface, but at least on NETGEAR devices this does not mitigate the vulnerability. NETGEAR told us, that there is no workaround available, the TCP port can't be firewalled nor is there a way to disable the service on their devices."

And, of course finally, a fully weaponized exploit was published and it's been archived on GitHub: https://github.com/blackorbird/exploit-database/blob/master/exploits/multiple/remote/38454.py

It's author explains:

```
# CVE-2015-3036 - NetUSB Remote Code Execution exploit (Linux/MIPS)
# ==========================================
# This is a weaponized exploit for the NetUSB kernel vulnerability discovered by SEC Consult
# Vulnerability Lab. [1]
#
# I don't like lazy vendors, I've seen some DoS PoC's floating around for this bug.. and it's been
# almost five(!) months. So let's kick it up a notch with an actual proof of concept that yields
# code exec.  So anyway.. a remotely exploitable kernel vulnerability, exciting eh. ;-)
#
# Smash stack, ROP, decode, stage, spawn userland process. woo!
#
# Currently this is weaponized for one target device (the one I own, I was planning on porting
# OpenWRT but got sidetracked by the NetUSB stuff in the default firmware image. 0oops. ;-D).
#
# This python script is horrible, but its not about the glue, its about the tech contained therein.
# Some things *may* be (intentionally?) botched.. lets see if "the community" cares enough to
# develop this any further, I need to move on with life. ;-D
#
# Shoutouts to all my boys & girls around the world, you know who you are!
#
# Peace,
# -- blasty <peter@haxx.in> // 20151013
#
# References:
https://www.sec-consult.com/fxdata/seccons/prod/temedia/advisories_txt/20150519-0_KCodes_NetUSB_Kernel_Stack_Buffer_Overflow_v10.txt
```

Okay. So all that was then. But it provides some important context for today. I wanted to be certain that everyone understood that many many more than five or six router vendors were involved and that Kcodes Technology has, in the past, been anything but helpful and responsible.

The guys who discovered today's problem understood that the only possible way to get all of the routers whose manufacturers had licensed the common NetUSB code updated would be to contact Kcodes Technology. But in the past that hasn't turned out so well. Although the guys at Sentinel Labs deliberately stopped short of providing anything beyond a simple DoS (crash and reboot the router) proof of concept, there's every reason — in fact, much more in today's climate than seven years ago — to expect that someone is going to weaponize this exploit, if that hasn't been done already.

The Asus router I use at home does not offer NetUSB functions. Asus never has. And even if it did, it's daisy-chained behind another little router running pfSense. So no ports that the Asus might have open would have been exposed to the Internet anyway. I use pfSense to perform port-NAT and IP filtering which is beyond what the Asus can do.

So, now that I hope I have everyone's attention, let's take a look inside Kcode Technology's actual code to see what they messed up...

To initiate a connection with the router, any PC located on the LAN initiates a TCP connection to the router's port 20005. The router's kernel service and server listening for those incoming connections should only be listening on the router's LAN interface. But as we know from last week, the worst aspect of the flaw is that the NetUSB service is bound to 0.0.0.0, giving it a presence on both the router's LAN and WAN interfaces, thus exposing it to the public Internet. It would still be a problem to have this vulnerable service only on the LAN, since anyone on the LAN could potentially take over the router without authorization. But allowing that to happen with anyone, anywhere, definitely takes it up a notch.

Okay. So once the standard 3-way TCP connection handshake has occurred, the PC wishing to have access to the USB devices connected to the router sends a signature plus 16 bytes (128-bits) of what they termed "Verify Data." It appears that these 16 bytes of so-called "Verify Data" serves as a connection nonce to prevent replay attacks. Upon receiving the PC's signature with its "Verify Data", the router AES-encrypts its client's "Verify Data" and returns along with its own 16 bytes of "Random Data." The PC probably decrypts the AES-encrypted "Verify Data" that the router returned to verify that it matches what it sent. Assuming that it does, it similarly AES-encrypts the router's "Random Data" and sends that along with the computer's name. The name is formatted as a length-of-name integer followed by that number of name characters.

At this point, the endpoints have exchanged and verified random nonce data and have established a connection. The router's code then drops into a command-parsing while-loop to await the PC client's commands. The while-loop waits for and receives a 16-bit command word which causes it to jump to a function which then further handles the needs of each specific command. The researchers at Sentinel Labs found a problem in the function whose 16-bit command is hexadecimal 805F. The receipt of that command word branches the code to a routine named: "SoftwareBus_dispatch Normal EP Msg Out."

Whatever this command does, we don't need to know or care. The client first sends 4 bytes, which is 32-bits, as the maximum number of bytes, to follow, which will be read from the client. That's clean, since the client is declaring up front the number of bytes that it will be sending. This allows the receiving server to request an allocation of memory from the underlying operating system — a buffer — which will be used to receive and hold up to that much data received from the client.

But whatever this function is doing, 17 bytes of additional “scratchpad” working memory space is apparently required to process the command. So, to the size of the memory allocation being requested, this function adds 17 so that an extra 17 bytes of memory will be obtained from the server for the function's use. That all seems great.

So, to recap: The client sent the hex command code 805F, which was then followed by 4 bytes to indicate the amount of follow-on data that the client would be sending. Upon receiving those four bytes, to allocate memory to serve as a buffer to then receive the client's data, the server asks the operating system for that much plus 17 bytes more, so that it will have some additional memory to use.

The code then calls a data receive function, giving that function a pointer to the buffer that the operating system has allocated and the number of bytes that the client said would be forthcoming to fill that buffer.

The data receive function will receive data from the client, placing that data into the buffer that has been pre-allocated to contain it, until the specified number of bytes of data has been received. Once the expected count of data is received, the data receive function will return to its caller with the expected data in the allocated buffer, and those 17 extra bytes at the end of the buffer for the command processor's use.

This is all perfectly reasonable code. You could stare at it all day and find nothing to criticize. There's no way for the sender to overflow the receiving buffer, since the sender pre-declares the number of bytes that it will be sending, and that's all that the data receive function will accept. Upon receiving the 4-byte data size, an allocation of that amount plus 17 extra bytes is requested from the operating system. Upon the successful allocation of a buffer to hold the incoming data, the data receive function is told exactly how many bytes to accept from the sender and to place them into the properly sized receive buffer.

So what's the problem that the clever researchers at Sentinel Labs found?

As we know, the sending client specifies its byte count as a 4-byte 32-bit integer. So it's reasonable to store that byte count in a 4-byte 32-bit integer. The authors correctly declare the integer as unsigned because a byte count should logically be unsigned (how can you have a negative byte count?). So that's fine.

The problem arises when those 17 extra bytes are added to the memory allocation. The largest value that a 32-bit unsigned integer can represent is that very familiar (just shy of) 4.3 billion, the total number of IPv4 addresses, and so on. It's 2^32-1.

So, what happens if a malicious client connects to this router's port 20005 and properly negotiates a handshake. Then it sends that vulnerable command 805F and for the count of the bytes it's going to send, it declares that maximum of 4.3 billion. In binary, it's all 1's — the largest number that can be represented in 32-bits.

So, okay, the command processing code needs to allocate a buffer from the operating system to contain that many bytes of data. As before, it adds 17 to accommodate its small bit of extra scratchpad working memory. But now, adding 17 to a 32-bit value that is already as large as it can be will cause the addition to overflow and the value to wrap back around. If we were to add 1 to that 32-bit value of all 1's, the count would wrap around to 0. But we add 17, so the count will wrap around to 16…

And THAT 16 is the number of bytes we ask the operating system to allocate for us to hold the data that the client is about to send. So the operating system does that. We receive a pointer, to a 16-byte buffer which we then hand to the data receive function, asking it to please receive the original byte count, of 4.3 billion bytes of client data... which it dutifully begins to do.

And what we have is a textbook perfect classic buffer overflow, where the client has absolute and total control over the contents that fills and then overflows the tiny 16 bytes of data buffer.

Without any additional work, that will immediately crash the router, probably causing it to reboot. But with the addition of some skilled hacking, it's quite clear that many many millions of consumer routers are exposed to a very critical and extremely exploitable remote code execution attack where the attacker is readily able to supply the code in what they send.

There are several points of failure evidenced in the design of this code.

For one thing, though we don't know what command 805F does, it seems quite unlikely that telling it that it's about to receive 4.3 billion bytes of data would be reasonable. For all we know, the most that would ever be sent would be a few Kbytes at a time. Yet the command performs no sanity checking of any kind of that forthcoming-bytes declaration.

We've often talked about this problem which affects the designers of interpreters. They assume, as the designers of this code must have, that only a valid client would ever be connecting to their server — after all, they wrote and provided the OS clients running at the other end. So they never bothered to place any limits — even high-end sanity checking limits — on anything that a client might send.

If the machine had been 64 bits rather than 32 bits, adding 17 to a maximum 4-byte value would not have wrapped around and the operating system would have been asked to allocate 4.3 gig of RAM, which it would have surely balked at, failed the request, and thus protected the router. But our little consumer routers are using inexpensive 32-bit MIPS chips, so 32-bits is going to wrap around.

As a consequence of this pervasive bug, we can expect that a lot of damage will be done to users whose routers have aged out of their service life, or who have manufacturers or users who are not paying attention to what's going on.

Fortunately, everyone here... is.