What Are We Doing Again?

Palo Alto makes many things, most of which are built on their custom linux distribution dubbed PAN-OS. One of the things they build is a VPN product that lets users remotely access internal company resources. It's had quite the number of pretty horrible problems of late.

Folks contracted to assess security defenses (colloquially dubbed "pen-testers" tho no pens are usually involved) and practitioners within an organization who want to gain an independent view of what their internet perimeter looks like often assemble tools to perform
ad-hoc assessments. Sure, there are commercial tools for performing these assessments ($DAYJOB makes some!), but these open source tools make it possible for folks to learn from each other and for makers of products (like PAN-OS) to do a better job securing their creations.

In this case, the creation is a script that lets the caller figure out what version of PAN-OS is running on a given GlobalProtect box.

To follow along at home, you'll need access to a PAN-OS system, as I'm not providing an IP address of one for you. It's *really not hard to find one* (just google a bit or stand up a trial one from the vendor). Throughout the examples I'll be using {glue} to replace `ip` and `port` in various function calls, so let's get some setup bits out of the way:

```
library(httr)
library(tidyverse) # for read_fwf() (et al), pluck(), filter(), and %>%

gg <- glue::glue # no need to bring in the entire namespace just for this
```

Assuming you have a valid `ip` and `port`, let's try making a request against your PAN-OS GlobalProtect (hereafter using "GP" so save keystrokes) system:

```
httr::HEAD(
  url = gg("https://{ip}:{port}/global-protect/login.esp")
) -> res
## Error in curl::curl_fetch_memory(url, handle = handle) :
##  SSL certificate problem: self signed certificate
```

We're using a `HEAD` request as we really don't need the contents of the remote file (unless you need to verify it truly is a PAN-OS GP server), just the metadata about it. You can use a traditional `GET` request if you like, though.

We immediately run into a snag since these boxes tend to use a self-signed SSL/TLS certificate which web clients aren't thrilled about dealing with unless explicitly configured to. We can circumvent this with some configuration options, but you should not use the following incantations haphazardly. SSL/TLS no longer really means what it used to (thanks, Let's Encrypt!) but you have no guarantees of what's being delivered to you is legitimate if you hit a plaintext web site or one with an invalid certificate. Enough with the soapbox, let's make the request:

```
httr::HEAD(
  url = gg("https://{ip}:{port}/global-protect/login.esp"),
  config = httr::config(
    ssl_verifyhost =FALSE,
    ssl_verifypeer = FALSE
  )
) -> res

httr::status_code(res)
## [1] 200
```

In that request, we've told the underlying {curl} library calls to not verify the validity of the host or peer

certificates associated with the service. Again, don't do this haphazardly to get around generic SSL/TLS problems when making normal API calls or scraping sites.

Since we only made a `HEAD` request, we're just getting back headers, so let's take a look at them:

```
str(httr::headers(res), 1)
## List of 18
##  $ date                    : chr "Fri, 10 Jul 2020 15:02:32 GMT"
##  $ content-type            : chr "text/html; charset=UTF-8"
##  $ content-length          : chr "11749"
##  $ connection              : chr "keep-alive"
##  $ etag                    : chr "\"7e0d5e2b6add\""
##  $ pragma                  : chr "no-cache"
##  $ cache-control           : chr "no-store, no-cache, must-revalidate, post-
check=0, pre-check=0"
##  $ expires                 : chr "Thu, 19 Nov 1981 08:52:00 GMT"
##  $ x-frame-options         : chr "DENY"
##  $ set-cookie              : chr "PHPSESSID=bde5668131c14b765e3e75f8ed5514
a0; path=/; secure; HttpOnly"
##  $ set-cookie              : chr "PHPSESSID=bde5668131c14b765e3e75f8ed5514
a0; path=/; secure; HttpOnly"
##  $ set-cookie              : chr "PHPSESSID=bde5668131c14b765e3e75f8ed5514
a0; path=/; secure; HttpOnly"
##  $ set-cookie              : chr "PHPSESSID=bde5668131c14b765e3e75f8ed5514
a0; path=/; secure; HttpOnly"
##  $ set-cookie              : chr "PHPSESSID=bde5668131c14b765e3e75f8ed5514
a0; path=/; samesite=lax; secure; httponly"
##  $ strict-transport-security: chr "max-age=31536000;"
##  $ x-xss-protection        : chr "1; mode=block;"
##  $ x-content-type-options  : chr "nosniff"
##  $ content-security-policy : chr "default-src 'self'; script-src 'self'
'unsafe-inline'; img-src * data:; style-src 'self' 'unsafe-inline';"
##  - attr(*, "class")= chr [1:2] "insensitive" "list"
```

As an aside, I've always found the use of PHP code in security products quite, er, *fascinating*.

The value we're really looking for here is `etag` (which really looks like `ETag` in the raw response).

Bishop Fox (and others) figured out that that header value contains a timestamp in the last 8 characters. That timestamp maps to the release date of the particular PAN-OS version. Since Palo Alto maintains multiple, supported versions of PAN-OS and generally releases patches for them all at the same time, the mapping to an exact version is not super precise, but it's sufficient to get an idea of whether that system is at a current, supported patch level.

The last 8 characters of `7e0d5e2b6add` are `5e2b6add`, which — as Bishop Fox notes in their repo — is just a hexadecimal encoding of the POSIX timestamp, in this case, `1579903709` or `2020-01-24 22:08:29 GMT` (we only care about the date, so really `2020-01-24`).

We can compute that with R, but first we need to note that the value is surrounded by `"` quotes, so we'll have to deal with that during the processing:

```
httr::headers(res) %>%
  pluck("etag") %>%
  gsub('"', '', .) %>%
  substr(5, 12) %>%
  as.hexmode() %>%
  as.integer() %>%
  anytime::anytime(tz = "GMT") %>%
  as.Date() -> version_date
```

```
version_date
## [1] "2020-01-24"
```

To get the associated version(s), we need to look the date up in their table, which is in a fixed-width format that we can read via:

```
read_fwf(
  file = "https://raw.githubusercontent.com/noperator/panos-scanner/master/version-table.txt",
  col_positions = fwf_widths(c(10, 14), c("version", "date")),
  col_types = "cc",
  trim_ws = TRUE
) %>%
  mutate(
    date = lubridate::mdy(date)
  ) -> panos_trans

panos_trans
## # A tibble: 153 x 2
##    version  date
##
##  1 6.0.0    2013-12-23
##  2 6.0.1    2014-02-26
##  3 6.0.2    2014-04-18
##  4 6.0.3    2014-05-29
##  5 6.0.4    2014-07-30
##  6 6.0.5    2014-09-04
##  7 6.0.5-h3 2014-10-07
##  8 6.0.6    2014-10-07
##  9 6.0.7    2014-11-18
## 10 6.0.8    2015-01-13
## # … with 143 more rows
```

Now, let's see what version or versions this might be:

```
filter(panos_trans, date == version_date)
## # A tibble: 2 x 2
##   version date
##
## 1 9.0.6   2020-01-24
## 2 9.1.1   2020-01-24
```

Putting It All Together

We can make a command line script for this (example) scanner:

```
#!env Rscript
library(purrr)

gg <- glue::glue

# we also use {httr}, {readr}, {lubridate}, {anytime}, and {jsonlite}

args <- commandArgs(trailingOnly = TRUE)

stopifnot(
  c(
    "Must supply both IP address and port" = length(args) == 2
  )
```

```
)

ip <- args[1]
port <-  args[2]

httr::HEAD(
  url = gg("https://{ip}:{port}/global-protect/login.esp"),
  config = httr::config(
    ssl_verifyhost =FALSE,
    ssl_verifypeer = FALSE
  )
) -> res

httr::headers(res) %>%
  pluck("etag") %>%
  gsub('"', '', .) %>%
  substr(5, 12) %>%
  as.hexmode() %>%
  as.integer() %>%
  anytime::anytime(tz = "GMT") %>%
  as.Date() -> version_date

panos_trans <- readr::read_csv("panos-versions.txt", col_types = "cD")

res <- panos_trans[panos_trans[["date"]] == version_date,]

if (nrow(res) == 0) {
  cat(gg('{{"ip":"{ip}","port":"{port}","version"=null,"date"=null}}\n'))
} else {
  res$ip <- ip
  res$port <- port
  jsonlite::stream_out(res[,c("ip", "port", "version", "date")], verbose =
FALSE)
}
```

Save that as `panos-scanner.R` and make it executable (provided you're on a non-legacy operating system that doesn't support such modern capabilities). Save `panos_trans` as a CSV file in the same directory and try it against another (sanitized IP/port) system:

```
./panos-scanner.R 10.20.30.40 5678
1
{"ip":"10.20.30.40","port":"5678","version":"9.1.2","date":"2020-03-30"}
```

FIN

To be complete, the script should test all the URLs the ones in the script from Bishop Fox does and stand up many more guard rails to handle errors associated with unreachable hosts, getting headers from a system that is not a PAN-OS GP host, and ensuring the `ETag` is a valid date.