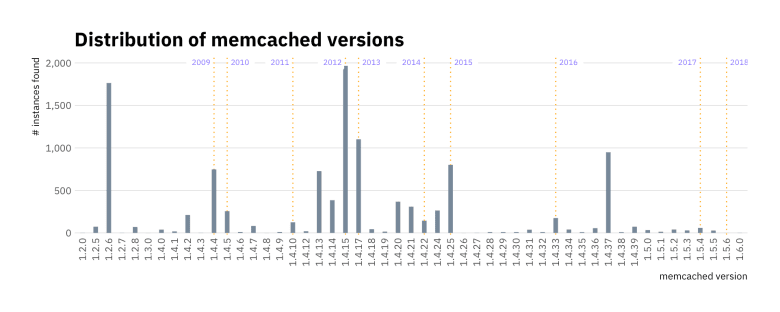
We work with internet-scale data and do my fair share of macro-analyses on vulnerabilities. We use the R [semver](https://cran.r-project.org/package=semver) package for most of my work and wanted to blather on a bit about it since it’s super-helpful for this work and doesn’t get the attention it deserves. semver makes it possible to create charts like this:

[](https://i0.wp.com/rud.is/b/wp-content/uploads/2018/03/unnamed-chunk-6-1.png?ssl=1)

which are very helpful in when conducting exposure analytics.

We’ll need a few packages to help us along the way:

library(here) # file mgmt

library(semver) # the whole purpose of the blog post

library(rvest) # we'll need this to get version->year mappings

library(stringi) # b/c I'm still too lazy to switch to ore

library(hrbrthemes) # pretty graphs

library(tidyverse) # sane data processing idioms

By issuing a stats command to a memcached instance you can get a full list of statistics for the server. The recent newsmaking DDoS used this feature in conjunction with address spoofing to create 30 minutes of chaos for GitHub.

I sent a stats command (followed by a newline) to a vanilla memcached installation and it returned 53 lines (1108 bytes) of STAT results that look something like this:

STAT pid 7646

STAT uptime 141

STAT time 1520447469

STAT version 1.4.25 Ubuntu

STAT libevent 2.0.21-stable

...

The version bit is what we’re after, but there are plenty of other variables you could just as easily focus on if you use memcached in any production capacity.

I extracted raw version response data from our most recent scan for open memcached servers on the internet. For ethical reasons, I cannot blindly share the entire raw data set but hit up [research@rapid7.com](mailto:research@rapid7.com) if you have a need or desire to work with this data.

Let’s read it in and take a look:

version\_strings <- read\_lines(here("data", "versions.txt"))

set.seed(2018-03-07)

sample(version\_strings, 50)

## [1] "STAT version 1.4.5" "STAT version 1.4.17"

## [3] "STAT version 1.4.25" "STAT version 1.4.31"

## [5] "STAT version 1.4.25" "STAT version 1.2.6"

## [7] "STAT version 1.2.6" "STAT version 1.4.15"

## [9] "STAT version 1.4.17" "STAT version 1.4.4"

## [11] "STAT version 1.4.5" "STAT version 1.2.6"

## [13] "STAT version 1.4.2" "STAT version 1.4.14 (Ubuntu)"

## [15] "STAT version 1.4.7" "STAT version 1.4.39"

## [17] "STAT version 1.4.4-14-g9c660c0" "STAT version 1.2.6"

## [19] "STAT version 1.2.6" "STAT version 1.4.14"

## [21] "STAT version 1.4.4-14-g9c660c0" "STAT version 1.4.37"

## [23] "STAT version 1.4.13" "STAT version 1.4.4"

## [25] "STAT version 1.4.17" "STAT version 1.2.6"

## [27] "STAT version 1.4.37" "STAT version 1.4.13"

## [29] "STAT version 1.4.25" "STAT version 1.4.15"

## [31] "STAT version 1.4.25" "STAT version 1.2.6"

## [33] "STAT version 1.4.10" "STAT version 1.4.25"

## [35] "STAT version 1.4.25" "STAT version 1.4.9"

## [37] "STAT version 1.4.30" "STAT version 1.4.21"

## [39] "STAT version 1.4.15" "STAT version 1.4.31"

## [41] "STAT version 1.4.13" "STAT version 1.2.6"

## [43] "STAT version 1.4.13" "STAT version 1.4.15"

## [45] "STAT version 1.4.19" "STAT version 1.4.25 Ubuntu"

## [47] "STAT version 1.4.37" "STAT version 1.4.4-14-g9c660c0"

## [49] "STAT version 1.2.6" "STAT version 1.4.25 Ubuntu"

It’s in decent shape, but it needs some work if we’re going to do a version analysis with it. Let’s clean it up a bit:

data\_frame(

string = stri\_match\_first\_regex(version\_strings, "STAT version (.\*)$")[,2]

) -> versions

count(versions, string, sort = TRUE) %>%

knitr::kable(format="markdown")

| **string** | **n** |
| --- | --- |
| 1.4.15 | 1966 |
| 1.2.6 | 1764 |
| 1.4.17 | 1101 |
| 1.4.37 | 949 |
| 1.4.13 | 725 |
| 1.4.4 | 531 |
| 1.4.25 | 511 |
| 1.4.20 | 368 |
| 1.4.14 (Ubuntu) | 334 |
| 1.4.21 | 309 |
| 1.4.25 Ubuntu | 290 |
| 1.4.24 | 259 |
| … | … |

Much better! However, we really only need the major parts of the [semantic version string](https://semver.org/) for a macro view, so let’s remove non-version strings completely and extract just the *major*, *minor* and *patch* bits:

filter(versions, !stri\_detect\_fixed(string, "UNKNOWN")) %>% # get rid of things we can't use

mutate(string = stri\_match\_first\_regex(

string, "([[:digit:]]+\\.[[:digit:]]+\\.[[:digit:]]+)")[,2] # for a macro-view, the discrete sub-versions aren't important

) -> versions

count(versions, string, sort = TRUE) %>%

knitr::kable(format="markdown")

| **string** | **n** |
| --- | --- |
| 1.4.15 | 1966 |
| 1.2.6 | 1764 |
| 1.4.17 | 1101 |
| 1.4.37 | 949 |
| 1.4.25 | 801 |
| 1.4.4 | 747 |
| 1.4.13 | 727 |
| 1.4.14 | 385 |
| 1.4.20 | 368 |
| 1.4.21 | 309 |
| 1.4.24 | 264 |
| … | … |

Much, *much* better! Now, let’s dig into the versions a bit. Using semver is dirt-simple. Just use parse\_version() to get the usable bits out:

ex\_ver <- semver::parse\_version(head(versions$string[1]))

ex\_ver

## [1] Maj: 1 Min: 4 Pat: 25

str(ex\_ver)

## List of 1

## $ :Class 'svptr'

## - attr(\*, "class")= chr "svlist"

It’s a special class, referencing an external pointer (the package relies on an underling C++ library and wraps everything up in a bow for us).

These objects can be compared, ordered, sorted, etc but I tend to just turn the parsed versions into a data frame that can be associated back with the main strings. That way we keep things pretty tidy and have tons of flexibility.

bind\_cols(

versions,

pull(versions, string) %>%

semver::parse\_version() %>%

as.data.frame()

) %>%

arrange(major, minor, patch) %>%

mutate(string = factor(string, levels = unique(string))) -> versions

versions

## # A tibble: 11,157 x 6

## string major minor patch prerelease build

##

## 1 1.2.0 1 2 0 "" ""

## 2 1.2.0 1 2 0 "" ""

## 3 1.2.5 1 2 5 "" ""

## 4 1.2.5 1 2 5 "" ""

## 5 1.2.5 1 2 5 "" ""

## 6 1.2.5 1 2 5 "" ""

## 7 1.2.5 1 2 5 "" ""

## 8 1.2.5 1 2 5 "" ""

## 9 1.2.5 1 2 5 "" ""

## 10 1.2.5 1 2 5 "" ""

## # ... with 11,147 more rows

Now we have a tidy data frame and I did the extra step of creating an ordered factor out of the version strings since they are ordinal values. With just this step, we have everything we need to do a basic plot shoing the version counts in-order:

count(versions, string) %>%

ggplot() +

geom\_segment(

aes(string, n, xend = string, yend = 0),

size = 2, color = "lightslategray"

) +

scale\_y\_comma() +

labs(

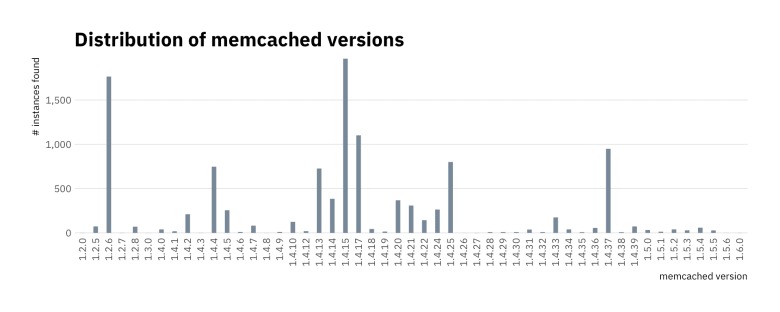
x = "memcached version", y = "# instances found",

title = "Distribution of memcached versions"

) +

theme\_ipsum\_ps(grid = "Y") +

theme(axis.text.x = element\_text(hjust = 1, vjust = 0.5, angle = 90))

[](https://i1.wp.com/rud.is/b/wp-content/uploads/2018/03/unnamed-chunk-2-1.png?ssl=1)

That chart is informative on its own since we get the perspective that there are some really old versions exposed. But, how old are they? Projects like Chrome or Firefox churn through versions regularly/quickly (on purpose). To make more sense out of this we’ll need more info on releases.

This is where things can get ugly for folks who do not have commercial software management databases handy (or are analyzing a piece of software that hasn’t made it to one of those databases yet). The memcached project maintains a [wiki page](https://github.com/memcached/memcached/wiki/ReleaseNotes) of version history that’s mostly complete, and definitely complete enough for this exercise. It *will* some processing before we can associate a version to a year.

GitHub does not allow scraping of their site and — off the top of my head — I do not know if there is a “wiki” API endpoint, but I *do* know that you can tack on .wiki.git to the end of a GitHub repo to clone the wiki pages, so we’ll use that knowledge and the git2r package to gain access to the  
ReleaseNotes.md file that has the data we need:

td <- tempfile("wiki", fileext="git") # temporary "directory"

dir.create(td)

git2r::clone(

url = "git@github.com:memcached/memcached.wiki.git",

local\_path = td,

credentials = git2r::cred\_ssh\_key() # need GH ssh keys setup!

) -> repo

## cloning into '/var/folders/1w/2d82v7ts3gs98tc6v772h8s40000gp/T//Rtmpb209Sk/wiki180eb3c6addcbgit'...

## Receiving objects: 1% (5/481), 8 kb

## Receiving objects: 11% (53/481), 8 kb

## Receiving objects: 21% (102/481), 49 kb

## Receiving objects: 31% (150/481), 81 kb

## Receiving objects: 41% (198/481), 113 kb

## Receiving objects: 51% (246/481), 177 kb

## Receiving objects: 61% (294/481), 177 kb

## Receiving objects: 71% (342/481), 192 kb

## Receiving objects: 81% (390/481), 192 kb

## Receiving objects: 91% (438/481), 192 kb

## Receiving objects: 100% (481/481), 192 kb, done.

read\_lines(file.path(repo@path, "ReleaseNotes.md")) %>%

keep(stri\_detect\_fixed, "[[ReleaseNotes") %>%

stri\_replace\_first\_regex(" \\\* \\[\\[.\*]] ", "") %>%

stri\_split\_fixed(" ", 2, simplify = TRUE) %>%

as\_data\_frame() %>%

set\_names(c("string", "release\_year")) %>%

mutate(string = stri\_trim\_both(string)) %>%

mutate(release\_year = stri\_replace\_first\_fixed(release\_year, "(", "")) %>% # remove leading parens

mutate(release\_year = stri\_replace\_all\_regex(release\_year, "\\-.\*$", "")) %>% # we only want year so remove remaining date bits from easy ones

mutate(release\_year = stri\_replace\_all\_regex(release\_year, "^.\*, ", "")) %>% # take care of most of the rest of the ugly ones

mutate(release\_year = stri\_replace\_all\_regex(release\_year, "^[[:alpha:]].\* ", "")) %>% # take care of the straggler

mutate(release\_year = stri\_replace\_last\_fixed(release\_year, ")", "")) %>% # remove any trailing parens

mutate(release\_year = as.numeric(release\_year)) -> memcached\_releases # make it numeric

unlink(td, recursive = TRUE) # cleanup the git repo we downloaded

memcached\_releases

## # A tibble: 49 x 2

## string release\_year

##

## 1 1.5.6 2018

## 2 1.5.5 2018

## 3 1.5.4 2017

## 4 1.5.3 2017

## 5 1.5.2 2017

## 6 1.5.1 2017

## 7 1.5.0 2017

## 8 1.4.39 2017

## 9 1.4.38 2017

## 10 1.4.37 2017

## # ... with 39 more rows

We have more versions in our internet-scraped memcached versions data  
set than this wiki page has on it, so we need to restrict the official  
release history to what we have. Then, we only want a single instance of  
each year for the annotations, so we’ll have to do some further processing:

filter(memcached\_releases, string %in% unique(versions$string)) %>%

mutate(string = factor(string, levels = levels(versions$string))) %>%

group\_by(release\_year) %>%

arrange(desc(string)) %>%

slice(1) %>%

ungroup() -> annotation\_df

knitr::kable(annotation\_df, "markdown")

| **string** | **release\_year** |
| --- | --- |
| 1.4.4 | 2009 |
| 1.4.5 | 2010 |
| 1.4.10 | 2011 |
| 1.4.15 | 2012 |
| 1.4.17 | 2013 |
| 1.4.22 | 2014 |
| 1.4.25 | 2015 |
| 1.4.33 | 2016 |
| 1.5.4 | 2017 |
| 1.5.6 | 2018 |

Now, we’re ready to add the annotation layers! We’ll take a blind stab at it before adding in further aesthetic customization:

version\_counts <- count(versions, string) # no piping this time

ggplot() +

geom\_blank(data = version\_counts,aes(string, n)) + # prime the scales

geom\_vline(

data = annotation\_df, aes(xintercept = as.numeric(string)),

size = 0.5, linetype = "dotted", color = "orange"

) +

geom\_segment(

data = version\_counts,

aes(string, n, xend = string, yend = 0),

size = 2, color = "lightslategray"

) +

geom\_label(

data = annotation\_df, aes(string, Inf, label=release\_year),

family = font\_ps, size = 2.5, color = "lightslateblue",

hjust = 0, vjust = 1, label.size = 0

) +

scale\_y\_comma() +

labs(

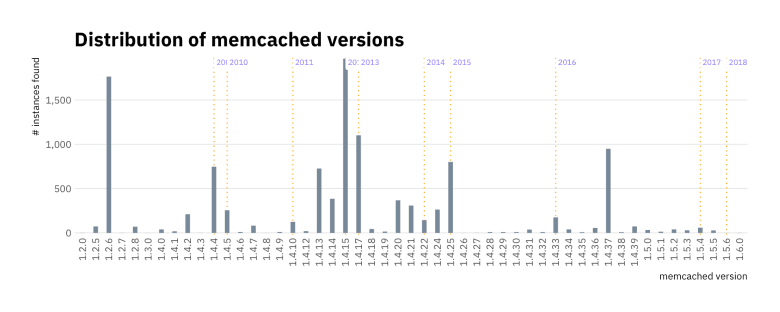
x = "memcached version", y = "# instances found",

title = "Distribution of memcached versions"

) +

theme\_ipsum\_ps(grid = "Y") +

theme(axis.text.x = element\_text(hjust = 1, vjust = 0.5, angle = 90))

[](https://i2.wp.com/rud.is/b/wp-content/uploads/2018/03/unnamed-chunk-5-1.png?ssl=1)

*Almost* got it in ggpar 1! We need to tweak this so that the labels do not overlap each other and do not obstruct the segment bars. We can do most of this work in geom\_segment() itself, plus add a bit of a tweak to the Y axis scale:

ggplot() +

geom\_blank(data = version\_counts,aes(string, n)) + # prime the scales

geom\_vline(

data = annotation\_df, aes(xintercept = as.numeric(string)),

size = 0.5, linetype = "dotted", color = "orange"

) +

geom\_segment(

data = version\_counts,

aes(string, n, xend = string, yend = 0),

size = 2, color = "lightslategray"

) +

geom\_label(

data = annotation\_df, aes(string, Inf, label=release\_year), vjust = 1,

family = font\_ps, size = 2.5, color = "lightslateblue", label.size = 0,

hjust = c(1, 0, 1, 1, 0, 1, 0, 0, 1, 0),

nudge\_x = c(-0.1, 0.1, -0.1, -0.1, 0.1, -0.1, 0.1, 0.1, -0.1, 0.1)

) +

scale\_y\_comma(limits = c(0, 2050)) +

labs(

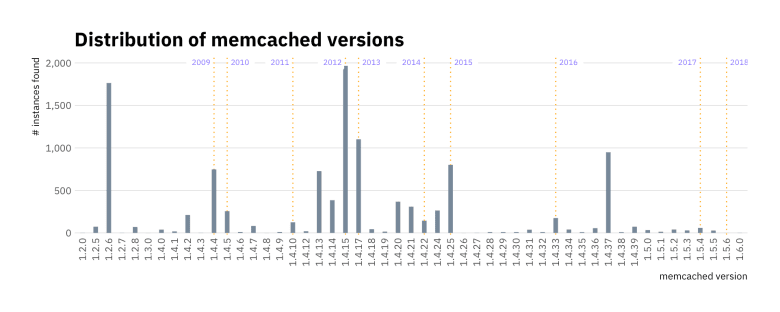
x = "memcached version", y = "# instances found",

title = "Distribution of memcached versions"

) +

theme\_ipsum\_ps(grid = "Y") +

theme(axis.text.x = element\_text(hjust = 1, vjust = 0.5, angle = 90))

[](https://i0.wp.com/rud.is/b/wp-content/uploads/2018/03/unnamed-chunk-6-1.png?ssl=1)

Now, we have version and year info to we can get a better idea of the scope of exposure (and, just how much technical debt many organizations have accrued).

With the ordinal version inforamtion we can also perform other statistical operations as well. All due to the semver package.