Au: Please check the following:
*chapter cross-refs (highlighted)

4

Performing Exploratory Security Data Analysis

"Sometimes, bad is *bad."*

Huey Lewis and the News, Sports, Chrysalis Records, 1983

What constitutes "security data" is often in the eye of the beholder. Malware analysts gravitate towards process, memory and system binary dumps. Vulnerability researchers dissect new patch releases, and network security professionals tap wired and wireless networks to see what secrets can be sifted from the packets as they make their way from node to node.

This chapter focuses on exploring IP addresses by starting with further analyses on the AlienVault IP reputation database first seen in Chapter 3. You'll examine aspects of the ZeuS botnet (a fairly nasty bit of malware) from an IP address perspective and then perform some basic analyses on real firewall data. To fully understand the examples in this chapter, you should be familiar with the description of the AlienVault data set and have at least followed along with all previous, preliminary analyses. The other major goal of the chapter is to help you get more proficient in R by walking you through a diversity of examples that bring into play many core programming idioms of the language.

IP addresses—along with domain names and routing concepts—are the building blocks of the Internet. They are defined in RFC 791, the "Internet Protocol / DARPA Internet Program / Protocol Specification" (`http://tools.ietf.org/html/rfc791`), which has an elegant and succinct way of describing them:

> ***A* name *indicates what we seek. An* address *indicates where it is. A* route *indicates how to get there.***

Global entities slice and dice IP address space for public and private use; devices, systems, and applications log IP addresses for reference; network management systems test, group, display, and report on IP addresses; and security tools often make critical decisions based on IP addresses. But, what—exactly—***is*** an IP address? What can you learn from them and what part do they play in the quest for finding and mitigating malicious activity?

Note

We make no attempt to incorporate consideration of or conduct analyses on Internet Protocol (IP) version 6 (IPv6) addresses. Likewise, all the examples in this chapter are based on IPv4. Given the slow adoption and migration to IPv6, the plethora of malicious activity still found on IPv4 networks, and the fact that it's fairly straightforward to extrapolate IPv4 concepts to IPv6, this should not be a practical limitation.

If you plan on typing the code from the chapter versus executing each snippet from the `ch04.R` source file you will need to download the data files in the `ch04/data` directory from the repository on the book's website (`www.wiley.com/go/datadrivensecurity`) for many of the listings to work correctly. You will also need to run the code in Listing 4-0 to set up your R environment for the code examples in this chapter.

LISTING 4-0

```
# Listing 4-0
# This code sets up the R environemnt for the chapter
# set working directory to chapter location
```

toward (style sheet)

AR: VERIFIED REF

Cap (style sheet)

Do we need a source line here?

AR: I can only offer my opine in that it's fully referenced in the preceding txt

comma

```
# (change for where you set up files in ch 2)
setwd("~/book/ch04")
# make sure the packages for this chapter
# are installed, install if necessary
pkg <- c("bitops","ggplot2", "maps", "maptools",
         "sp", "maps", "grid", "car" )
new.pkg <- pkg[!(pkg %in% installed.packages())]
if (length(new.pkg)) {
  install.packages(new.pkg)
}
```

Dissecting the IP Address

Some information security practitioners may think of IP addresses as simply the strings used with a `ping`, `nessus`, `nmap`, or other commands. But to perform security-oriented analyses of your system and network data, you must fully understand as much as you can about security domain data elements, just as those who perform data analyses in financial, agricultural, or bio-medial disciplines must understand the underpinnings of the data elements in those fields. IP addresses are, perhaps, the most fundamental of security domain data elements. In this section you'll dig a bit deeper into them so you can fully integrate them into your own analytics endeavors.

Representing IP Addresses

IPv4 addresses comprise four bytes, which are known as ***octets***, and you'll usually come across them in a form called ***dotted-decimal notation*** (such as 192.168.1.1). Practically everyone reading this book understands this representation, if only by sight. This method of representation was briefly introduced in the IETF RFC 1123 in 1989 when they denoted it as # . # . # . #, but it was more clearly defined in the IETF's uniform resource identifier (URI) generic syntax draft (RFC 3986, `http://tools.ietf.org/html/rfc3986`) in 2005.

> ***Note***
>
> *When you come across other security domain elements, you'll want to do plenty of similar digging to ensure you have all information you need to process them or create complete regular expressions to locate them in unstructured data.*

that / the

Since you know an 8-bit byte can range in value from 0 to 255, you also know the dotted-decimal range is `0.0.0.0` through `255.255.255.255`, which is 32 bits. If you count the possible address space, you have a total of 4,294,967,296 possible addresses (the maximum value of a 32-bit integer). This brings up another point of storing and handling IP addresses: ***any IP address can be converted to/from a 32-bit integer value***. This is important because the integer representation saves both space and time and you can calculate some things a bit easier with that representation than with the dotted-decimal form. If you are writing or using a tool that perceives an IP address only as a character string or as a set of character

Cap

strings, you are potentially wasting space by trading a 4-byte, 32-bit representation for a 15-byte, 120-bit representation (worst case). Furthermore, you are also choosing to use less efficient string comparison code versus integer arithmetic and comparison plus bitwise operations to accomplish the same tasks. Although this may have little to no impact in some scenarios, the repercussions grow significant when you're dealing with large volumes of IP addresses (and become worse in the IPv6 world) and repeated operations.

Converting IPv4 Addresses to/from 32-Bit Integers

To take advantage of integer operations for IPv4 addresses, you need to have some method of converting them to and from dotted-decimal notation. IEEE Standard 1003.1 defines the common low-level (for example, C) method of performing this conversion via the `inet_addr()` and `inet_ntoa()` functions (`http://pubs.opengroup.org/onlinepubs/009695399/functions/inet_addr.html`). However, these functions are not exposed to R. Although it would be possible to write a C library and corresponding R glue module, it's easier to write the functions in pure R with some help from the ***bitops*** package. In Listing 4-1 you will find R functions that convert IPv4 address strings to/from 32-bit integer format.

LISTING 4-1

```
# Listing 4-1
# requires packages: bitops

library(bitops) # load the bitops functions

# Define functions for converting IP addresses to/from integers
# take an IP address string in dotted octets (e.g.
"192.168.0.1")
# take an IP address string in dotted octets (e.g.
"192.168.0.1")
# and convert it to a 32-bit long integer (e.g. 3232235521)
ip2long <- function(ip) {
  # convert string into vector of characters
  ips <- unlist(strsplit(ip, '.', fixed=TRUE))
  # set up a function to bit-shift, then "OR" the octets
  octet <- function(x,y) bitOr(bitShiftL(x, 8), y)
  # Reduce applys a function cumulatively left to right
  Reduce(octet, as.integer(ips))
}

# take an 32-bit integer IP address (e.g. 3232235521)
# and convert it to a (e.g. "192.168.0.1").
long2ip <- function(longip) {
  # set up reversing bit manipulation
  octet <- function(nbits) bitAnd(bitShiftR(longip, nbits),
0xFF)
```

Au: One line of code x2. OK as broken?

AR: If they are pre-pended with "#" (comment char) yes. thx.

Au: One line of code. OK as broken?

AR: yes. broken at the comma is cool. thx.

delete: (Continued)

~~(Continued)~~

```
    # Map applys a function to each element of the argument
    # paste converts arguments to character and concatenates them
    paste(Map(octet, c(24,16,8,0)), sep="", collapse=".")
}
```

You can test the functionality by reviewing the output from the following test code:

```
long2ip(ip2long("192.168.0.0"))
## [1] "192.168.0.0"
long2ip(ip2long("192.168.100.6"))
## [1] "192.168.100.6"
```

Note: Python coders can use the preexisting ***ipaddr*** package (`https://code.google.com/p/ipaddr-py/`), which has been incorporated into the Python 3 code base as the ***ipaddress*** module.

Segmenting and Grouping IP Addresses

There are a few different reasons you'd want to divide and group IP addresses. Internally, you might separate hosts by functionality or sensitivity, which means the routing tables would be overwhelmed if they needed to track each individual IP address. Due to the way TCP/IP was designed and how IPv4 networks are implemented, there are numerous ways to segment or group them so that it's easier to manage individual networks (subnets) and interoperate in the global Internet. The original specification identified top-level ***classes*** (`A` through `E`), which were nothing more than a list of corresponding bitmasks for consuming consecutive octets. This limited the usable range of addresses in each class and put some structure around the suggested use of each class.

rebreak

A more generalized, ***classless*** method of segmentation was established in RFC 4632 (`http://tools.ietf.org/html/rfc4632`). Rather than segment on whole octets, you can now specify address ranges in the CIDR (Classless Inter-Domain Routing) prefix format by appending the number of bits in the mask to a specified IP address. So, `172.16.0.0` (which has a mask of `255.255.0.0`) now becomes `172.16.0.0/16`. These CIDR ***blocks*** themselves can be used to look for "bad neighborhoods" (that is, identifying network packets coming from or going to groups of malicious nodes).

It's important to understand these points because you'll want to leverage the groupings to dig into the data and relationships to pull out meaning. Once you understand the CIDR prefix format, you can see how those prefixes are grouped and defined as an autonomous system (AS) that are all assigned a numerical identifier known as the autonomous system number (ASN). ASNs have many uses (and associated data); for example, they are used by the border gateway protocol (BGP) for efficient routing of packets across the Internet. Because of the relationship between ASN and BGP, it's also possible to know the adjacent "neighbors" of each ASN. If one ASN "neighborhood" is rife with malicious nodes,it might be a leading indicator that ASNs around it are also harboring malicious traffic. You'll use this relationship later in the chapter to get an ASN view of malicious activity.

comma

There are many more details regarding autonomous systems that you should investigate even if you only occasionally work with IP addresses in your analyses. To get a feel for the global make-up of autonomous

systems, you can explore public ASN information at the CIDR Report (`http://www.cidr-report.org/as2.0/`). But keep reading, as you'll be looking at malicious traffic through an ASN lens later in the chapter.

Testing IPv4 Address Membership in a CIDR Block

When you're performing ASN- and CIDR-based analyses, one task that comes up regularly is the need to determine whether an address falls within a given CIDR range. To do this in R, you just expand on the previously defined IPv4 address operations, convert both the IP address in question and the network block address to integers, and then perform the necessary bitwise operations to see if they do, indeed, line up. Listing 4-2 defines a new R function for testing membership of an IPv4 address in a specified CIDR block.

Listing 4-2

```
# Listing 4-2
# requires packages: bitops
# requires all objects from 4-1
# Define function to test for IP CIDR membership
# take an IP address (string) and a CIDR (string) and
# return whether the given IP address is in the CIDR range
ip.is.in.cidr <- function(ip, cidr) {
  long.ip <- ip2long(ip)
  cidr.parts <- unlist(strsplit(cidr, "/"))
  cidr.range <- ip2long(cidr.parts[1])
  cidr.mask <- bitShiftL(bitFlip(0), (32-as.integer(cidr.
parts[2])))
  return(bitAnd(long.ip, cidr.mask) == bitAnd(cidr.range, cidr.
mask))
}

ip.is.in.cidr("10.0.1.15","10.0.1.3/24")
## TRUE
ip.is.in.cidr("10.0.1.15","10.0.2.255/24")
## FALSE
```

Au: One line of code x2. OK as broken?

AR: pls break at the space so "(32…" is on the next line.

AR: pls break at the space so "cidr.…" is on the next line.

thx.

Your organization most likely uses CIDRs and ASNs internally as well, but these are not the only logical grouping mechanisms of sets of IP addresses. For example, you might have "workstations" as a high-level grouping that covers all the user-endpoint DHCP-assigned address space or "printers" to logically associate all statically assigned single- or multi-function output devices. Servers can be grouped according to function or operating system type (or both). The concept of "internal" and "external" groupings for nodes may apply even if you use publicly routable addresses across your entire network. When looking for malicious activity, do not discount the power of these logical groupings, since you may be able to tie characteristics of them (data!) to various indicators you may be looking for. For example, it's reasonable to expect the typical end-user workstation to make attempts to access nodes on the Internet. However, the same is

comma

probably not true for printers. Therefore, one of the keys to learning about malicious activity is this type of metadata and relationships.

Locating IP Addresses

Going down in detail, IP addresses map to individual devices that (usually) have unique media access control (MAC) addresses. It's a fairly straightforward process to identify the switch and port of a node on your local network. With the proper metadata, you can create logical groupings based on this physical information and tie additional attributes to it, such as where the node lives—organizationally-geographically speaking. By tying this information to an IP address, you won't have to wait until a barrage of help desk calls come in to discover that there is something amiss in a particular department.

On a broader scale, there are also ways to tie an IP address that lives on the Internet to a geographical location, with varying degrees of accuracy. One of the most popular ways to do so is with the Maxmind GeoIP database and APIs (`http://dev.maxmind.com/geoip/`), which are also used by the freegeoip project (`http://freegeoip.net/`). The Maxmind data has varying degrees of precision depending on whether you use the free databases or their commercial offerings. For country-level identification, using the freely available databases should provide sufficient precision for most organizations. The freegeoip project provides an online query interface to the free Maxmind data set, but it also provides all the code for the service that you can then clone and set up internally to avoid working directly with Maxmind's low-level APIs and avoid rate-limiting and other restrictions from the free service. Chapter 5 goes into more detail on working with geolocated data.

AR: VERIFIED REF

Once you know where a malicious node physically is, it's a fairly straightforward process to visualize it on a map. The AlienVault data provides over 250,000 pre-geolocated addresses, but you'll need to extract the pairs from the ***coords*** field first. You'll find example code for performing this extraction in Listing 4-3. Note that if you have a slightly slower machine, it may take 30–60 seconds to read and parse the data.

Listing 4-3

AR: VERIFIED REF

```
# Listing 4-3
# R code to extract longitude/latitude pairs from AlienVault data
# read in the AlienVault reputation data (see Chapter 3)
avRep <- "data/reputation.data"
av.df <- read.csv(avRep, sep="#", header=FALSE)
colnames(av.df) <- c("IP", "Reliability", "Risk", "Type",
                     "Country", "Locale", "Coords", "x")

# create a vector of lat/long data by splitting on ","
av.coords.vec <- unlist(strsplit(as.character(av.df$Coords), ","))
# convert the vector in a 2-column matrix
av.coords.mat <- matrix(av.coords.vec, ncol=2, byrow=TRUE)
# project into a data frame
av.coords.df <- as.data.frame(av.coords.mat)
# name the columns
colnames(av.coords.df) <- c("lat","long")
# convert the characters to numeric values
av.coords.df$long <- as.double(as.character(av.coords.df$long))
av.coords.df$lat <- as.double(as.character(av.coords.df$lat))
```

With the latitude and longitude coordinates in hand, you could avoid R or Python code altogether and use Google Maps to visualize the locations—if you felt like grabbing a caffeinated beverage while it uploads to Google Fusion Tables (`http://tables.googlelabs.com/`) and renders. Even though Google has considerably sped up their online mapping API, the resultant—albeit, handsome—map would end up being partially obscured with map markers. For example, mapping the AlienVault data set with Google Maps (see Figure 4-1) produces a result that makes it seem like malicious hosts have consumed Japan. Rather than rely solely on Google, you can use the mapping functions in R (see Listing 4-4) to accomplish a similar task (see Figure 4-2) with far greater precision.

> ***Note***
>
> *There are more examples of geographical mapping and analysis in Chapter 5.*

AR: VERIFIED REF

FIGURE 4-1 *Google Fusion Table + Google Maps chart of AlienVault malicious nodes*

LISTING 4-4

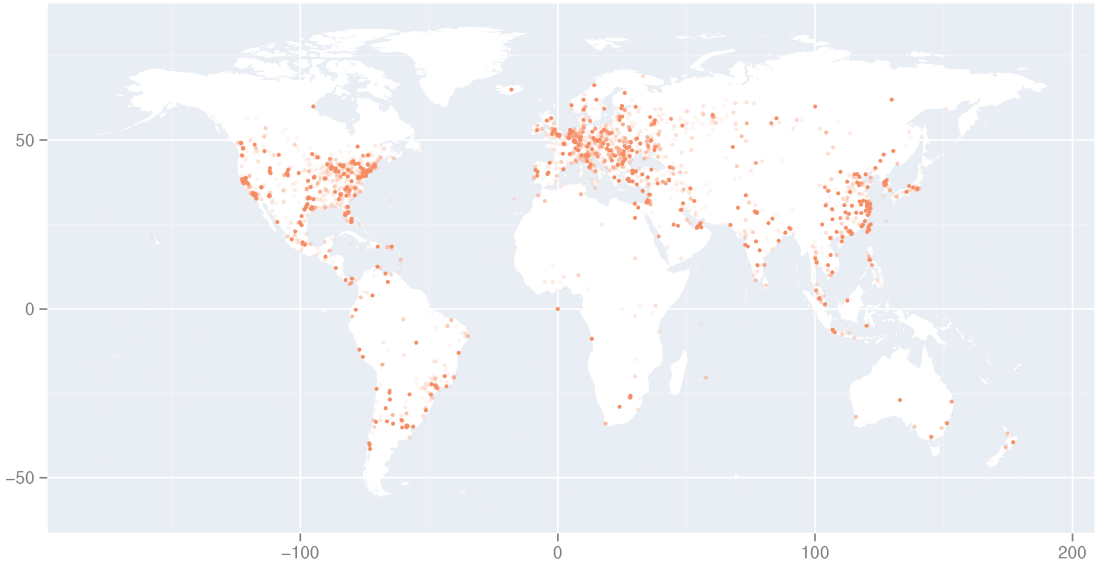
```
# Listing 4-4
# requires packages: ggplot2, maps, RColorBrewer
# requires object: av.coords.df (4-3)
# generates Figure 4-2
# R code to extract longitude/latitude pairs from AlienVault data
```

```
# need plotting and mapping functions
library(ggplot2)
library(maps)

# extract the polygon information for the world map, minus Antarctica
world <- map_data('world')
world <- subset(world, region != "Antarctica")

# plot the map with the points marking lat/lon of the geocoded entries
# Chapter 5 examples explain mapping in greater detail
gg <- ggplot()
gg <- gg + geom_polygon(data=world, aes(long, lat, group=group),
                        fill="white")
gg <- gg + geom_point(data=av.coords.df, aes(x=long, y=lat), col="red",
                      size=0.5, alpha=0.1)
gg <- gg + labs(x="", y="")
gg <- gg + theme(panel.background = element_rect(fill='#A6BDDB',
                                                 colour='white'))
gg
```

AR: VERIFIED REF



Au: figure too light?

AR: No. IT looks good & communicates what I the section intends. thx.

FIGURE 4-2 *R ggplot/maps package dot-plot map of AlienVault malicious nodes*

The ability to associate an IP address with a physical location and display it on a map has inherent utility (which will become even more apparent in Chapter 5). It's one thing to read the destinations of your Internet users and quite another to "see" them on a map, especially when you're trying to communicate the groupings versus just analyze them. Yet, you do not necessarily need to generate a pretty picture to looking at malicious activities geographically.

AR: VERIFIED REF

Augmenting IP Address Data

In an analyst's dream world, every data set you are asked to crunch through would be error-free and have all the attributes necessary for thorough and robust analyses. Sadly, information security is no different from other disciplines when it comes to imperfect data sets and highly distributed referential data or just a plethora of potential metadata sources. This less than perfect data ***can*** pose challenges to effective data analyses, but it is usually possible to find and use the data you need.

Even though you have geographic information in the AlienVault data set, the Internet has, as indicated, physical and logical groupings. It might be interesting to see how this data looks through a different lens. For this example, the data set (see Listing 4-3) is augmented with additional data from the IANA IPv4 Address Space Registry (`https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml`). This data represents a very high-level grouping of IPv4 address space registry allocations, and it should be emphasized that most of the registrants are not responsible for the malicious activity of individual nodes. So, although you cannot use this information to cast blame, it will give you one view of where malicious nodes are clustered, setting up possible, additional investigations.

rebreak

> ***Note***
>
> *On their web page under the heading "Alternative Formats," the IANA provides a handy link to the CSV version of the IPv4 address space allocations as well as a link to the traditional annotated text file. If you run the example code, you may see some strange behavior at times due to the CSV file being incomplete. It seems there is an automated process that converts a source of the IP table into the various formats and stops processing when the first octet hits three digits. You can either practice your data-munging skills and convert the fixed-width version in the text file to CSV or use the version of the CSV that's on the companion website if you encounter any issues.*

The data frame foundational data structure in R and pandas makes it very straightforward to reference and incorporate new data into your analyses, and your own projects will follow something close to this basic data-analysis workflow pattern:

1. Downloading (if necessary) new data
2. Parsing/munging and converting the new data into a data frame
3. Validating the contents and structure of the new data
4. Extracting or computing relevant information from the new data source
5. Creating one or more new columns in the existing data frame
6. Running new analyses

For the example Listing 4-5, you'll process the IANA data to see which registry allocations have the most malicious nodes. Note that the `sapply()` function call may take some time to execute depending on the speed of your machine.

Au: One line of code. OK as broken?

AR: yep. there's no good way to break these long URLs and most readers will prbly use the source files as the reference.

LISTING 4-5

```
# Listing 4-5
# requires object: av.df (4-3)
# R code to incporporate IANA IPv4 allocations
# retrieve IANA prefix list
ianaURL <- "http://www.iana.org/assignments/ipv4-address-space/ipv4-
address-space.csv"
ianaData <- "data/ipv4-address-space.csv"
if (file.access(ianaData)) {
  download.file(ianaURL, ianaData)
}

# read in the IANA table
iana <- read.csv(ianaData)

# clean up the iana prefix since it uses the old/BSD-
# number formatting (i.e. allows leading zeroes and
# we do not need to know the CIDR component.
iana$Prefix <- sub("^(00|0)", "", iana$Prefix, perl=TRUE)
iana$Prefix <- sub("/8$", "", iana$Prefix, perl=TRUE)

# define function to strip 'n' characters from a string
# (character vector) and return the shortened string.
# note that this function is 'vectorized' (you can pass it a single
# string or a vector of them)
rstrip <- function(x, n){
  substr(x, 1, nchar(x)-n)
}

# extract just the prefix from the AlienVault list
av.IP.prefix <- rstrip(str_extract(as.character(av.df$IP),
                                   "^([0-9]+)\\."), 1)

# there are faster ways than 'sapply()' but we wanted you to
# see the general "apply" pattern in action as you will use it
# quite a bit throughout your work in R
av.df$Designation <- sapply(av.IP.prefix, function(ip) {
  iana[iana$Prefix == ip, ]$Designation
})

##       Administered by AFRINIC         Administered by APNIC
##                           322                          2615
##          Administered by ARIN     Administered by RIPE NCC
##                         17974                          5893
##                       AFRINIC                        APNIC
##                          1896                         93776
##                          ARIN       AT&T Bell Laboratories
##                         42358                           24
## Digital Equipment Corporation      Hewlett-Packard Company
```

(continues)

Listing 4-5 *(continued)*

```
##                             1                               3
##                         LACNIC  Level 3 Communications, Inc.
##                          18914                              31
##                  PSINet, Inc.                        RIPE NCC
##                             30                           74789
```

You can do a quick check against the main IANA allocation table to see if this matches overall block assignments. The code in Listing 4-6 makes a data frame from the `table()` summary of the `iana$Designation` column and merges that data with the AlienVault data.

LISTING 4-6

```
# Listing 4-6
# requires packages: ggplot2, maps, RColorBrewer
# requires object: av.coords.df (4-3), iana (4-5)
# Code to extract IANA block assignments & compare w/AlienVault groups
# create a new data frame from the iana designation factors
iana.df <- data.frame(table(iana$Designation))
colnames(iana.df) <- c("Registry", "IANA.Block.Count")

# make a data frame of the counts of the av iana
# designation factor
tmp.df <- data.frame(table(factor(av.df$Designation)))
colnames(tmp.df) <- c("Registry", "AlienVault.IANA.Count")

# merge (join) the data frames on the "reg" column
combined.df <- merge(iana.df, tmp.df)
print(combined.df[with(combined.df, order(-IANA.Block.Count)),],
      row.names=FALSE)
```

Au: Grey shading as wanted?

AR: yes. looks good. thx.

```
##                      Registry IANA.Block.Count AlienVault.IANA.Count
##                         APNIC               45                 93776
##         Administered by ARIN                44                 17974
##                          ARIN               36                 42358
##                     RIPE NCC                35                 74789
##                        LACNIC                9                 18914
##           Administered by APNIC               6                  2615
##         Administered by RIPE NCC              4                  5893
##                       AFRINIC                 4                  1896
##          Administered by AFRINIC              2                   322
##    Level 3 Communications, Inc.               2                    31
##           AT&T Bell Laboratories              1                    24
##  Digital Equipment Corporation                1                     1
##         Hewlett-Packard Company               1                     3
##                    PSINet, Inc.               1                    30
```

Then you plot the data (see Listing 4-7) to generate the chart in Figure 4-3.

LISTING 4-7

```
# Listing 4-7
# requires packages: reshape, grid, gridExtra, ggplot2, RColorBrewer
# requires object: combined.df (4-6), set2 (4-4)
# generates Figure 4-3
# plot charts from IANA data
# flatten the data frame by making one entry per "count" type
# versus having the counts in individual columns
# need the 'melt()' function from the reshape package
# to transform the data frame shape
library(reshape)
library(grid)
library(gridExtra)

# normalize the IANA and AV values to % so bar chart scales
# match and make it easier to compare
combined.df$IANA.pct <- 100 * (combined.df$IANA.Block.Count /
                                 sum(combined.df$IANA.Block.Count))
combined.df$AV.pct <- 100 * (combined.df$AlienVault.IANA.Count /
                               sum(combined.df$AlienVault.IANA.Count))

combined.df$IANA.vs.AV.pct <- combined.df$IANA.pct - combined.df$AV.pct

melted.df <- melt(combined.df)
# plot the new melted data frame values
gg1 <- ggplot(data=melted.df[melted.df$variable=="IANA.pct",],
              aes(x=reorder(Registry, -value), y=value))
# set min/max for axis so scale is same for both charts
gg1 <- gg1 + ylim(0,40)
gg1 <- gg1 +  geom_bar(stat="identity", fill=set2[3]) # using bars
# make a better label for the y axis
gg1 <- gg1 + labs(x="Registry", y="%", title="IANA %")
# make bar chart horizontal
gg1 <- gg1 + coord_flip()
# rotate the x-axis labels and remove the legend
gg1 <- gg1 + theme(axis.text.x = element_text(angle = 90, hjust = 1),
                   panel.background = element_blank(),
                   legend.position = "none")

gg2 <- ggplot(data=melted.df[melted.df$variable=="AV.pct",],
              aes(x=reorder(Registry,-value), y=value))
gg2 <- gg2 + ylim(0,40)
gg2 <- gg2 + geom_bar(stat="identity", fill=set2[4]) # using bars
gg2 <- gg2 + labs(x="Registry", y="%", title="AlienVault IANA %")
gg2 <- gg2 + coord_flip()
gg2 <- gg2 + theme(axis.text.x = element_text(angle = 90, hjust = 1),
                   panel.background = element_blank(),
                   legend.position = "none")
```

move to previous page

```
# grid.arrange makes it possible to do very precise placement of
# multiple ggplot objects
grid.arrange(gg1, gg2, ncol=1, nrow=2)
```

There is some variation, but overall the larger blocks contribute the majority of malicious hosts. We've highlighted `RIPE NCC`, `Administered by ARIN`, and `LACNIC` in the text/console output in Listing 4-6 since `RIPE NCC` has a significantly larger number of malicious hosts than its allocation block count might imply (nearly double that of its very close neighbor `ARIN`). `LACNIC` and `Administered by ARIN` both have a similar number of malicious hosts yet have different allocation block counts. Even with these discrepancies, can you make a more confident statement regarding the comparison between

Au: Rotate nos. 90 degrees cloackwise?

AR: submitted new image

793725c04f03.eps

Au: Rotate nos. 90 degrees cloackwise?

FIGURE 4-3 *R bar charts comparing IANA block allocations*

Please keep paras together. Move back to previous page. (above figure)

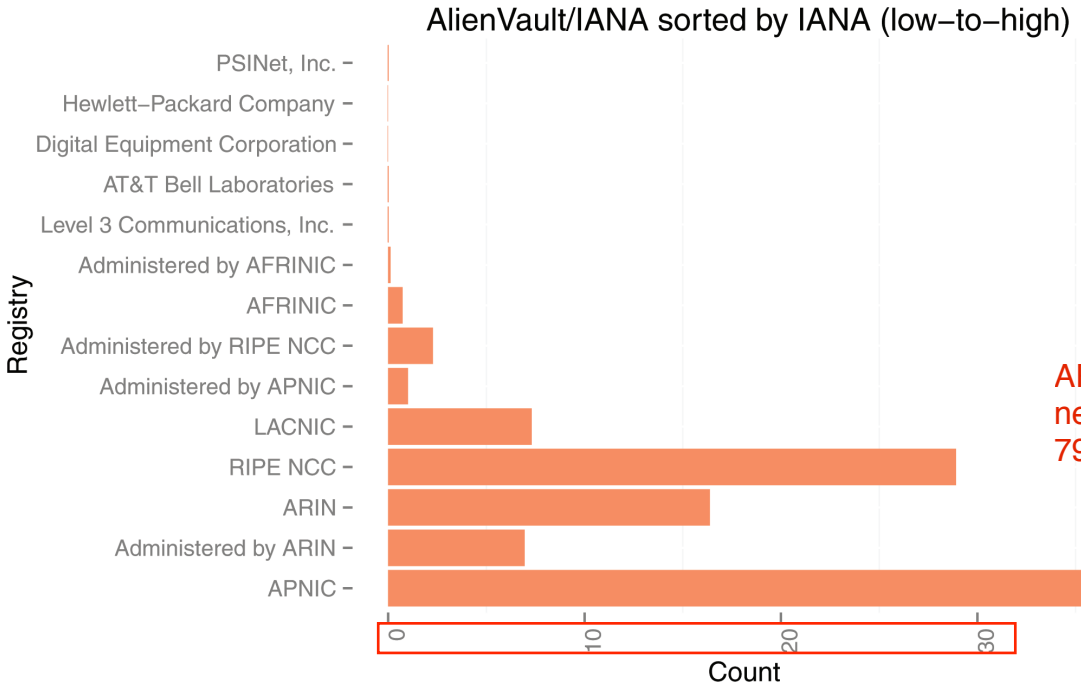
the number of malicious hosts in the /8s managed by a registrar and the number of /8s managed by a registrar? You can if you do one more visualization (see Listing 4-8), displaying the number of AlienVault malicious nodes per IANA block, sorted by IANA block (lowest to highest), as seen in Figure 4-4.

LISTING 4-8

```
# Listing 4-8
# requires packages: ggplot2
# requires object: combined.df (4-7), set2 (4-4)
gg <- ggplot(data=combined.df,
             aes(x=reorder(Registry, -IANA.Block.Count), y=AV.pct ))
gg <- gg + geom_bar(stat="identity", fill=set2[2])
gg <- gg + labs(x="Registry", y="Count",
                title="AlienVault/IANA sorted by IANA (low-to-high")
gg <- gg + coord_flip()
gg <- gg + theme(axis.text.x = element_text(angle = 90, hjust = 1),
                 panel.background = element_blank(),
                 legend.position = "none")
gg
```

toward

The AlienVault population does gravitate ~~towards~~ the IANA blocks with the most allocations, but we can do better by introducing some basic statistics into the mix in the next section.



AR: submitted new figure 793725c04f04.eps

Au: Rotate nos. 90 degrees cloackwise?

FIGURE 4-4 *Plotting AlienVault population per IANA block sorted by IANA block size*

Association/Correlation, Causation, and Security Operations Center Analysts Gone Rogue

Since this chapter contains the first examples where you group data elements (variables) and compare them to other variables, this is a good place to mention the concept of ***association*** or, as you'll see it referred to more often, ***correlation***. Correlation is simply a measurement of the linear relationship between two or more variables.

- A ***positive*** correlation is a relationship between two or more variables whereby their values increase or decrease together.
- Similarly, a ***negative*** correlation is a negative relationship, whereby when one variable increases, the other will decrease, and vice versa.

If there is no consistent linear pattern in the change between variables, they are said to be uncorrelated. When you calculate the correlation value (stats nerds call it the ***r*** value or correlation coefficient), you get a value between 1 (perfect positive correlation) and -1 (perfect negative correlation). As ***r*** gets closer to zero, the linear correlation decreases. At zero, you say there is no correlation between the two values.

It's important to remember that a simple correlation like this is a ***linear*** comparison. By contrast, look at the scatterplot in Figure 4-5 that has the parabola (upside down U shape). Obviously there is a pattern and some type of relationship, but it's not a linear correlation, so the calculated ***r*** value is very close to zero. Like most elements of statistics (or any complex discipline), there are many methods available to perform various tasks. This is also true when calculating correlation between two variables. Chapter 5 looks at a topic called ***linear regression***, which provides more detailed insight into correlation. Linear regression is also the basis for one type of predictive modeling. For the purposes of this chapter, you'll use a basic form of correlation.

AR: VERIFIED REF

Correlation Caveats

Believe it or not, there are parallels between statistics and information security. Statisticians use strange symbols and tools to perform their dark art much like malware researchers and network security specialists stare at rows of hexadecimal, octal, and binary data to derive meaning. Security researchers also understand which tool to use for the job at hand (you wouldn't use NetFlow data to try to understand detailed payload information in a communication session between two nodes). The same holds true for data scientists. There are, unfortunately, further considerations to take into account when working with even basic correlation techniques.

This chapter describes the ***Pearson*** correlation method, which is widely used given that it can work with data on an interval or ratio scale, with no restrictions placed on both variables being the same type. If you have ordinal or ranked data, you should use two other algorithms—Spearman or Kendall's Tau—instead. We aren't delving into the correlation algorithm subtleties in this book, but you should have a solid understanding of the uses and limits of each before applying correlation in your own analyses. You can find out more details about these correlation coefficients at `http://www.statisticssolutions.com/academic-solutions/resources/directory-of-statistical-analyses/correlation-pearson-kendall-spearman/`.

rebreak

Finally, correlation is a descriptive statistical measure versus an inferential one, meaning that you can only describe the population you are studying and cannot use the outcome to generalize a statement about a larger group or make predictions based on the outcome.

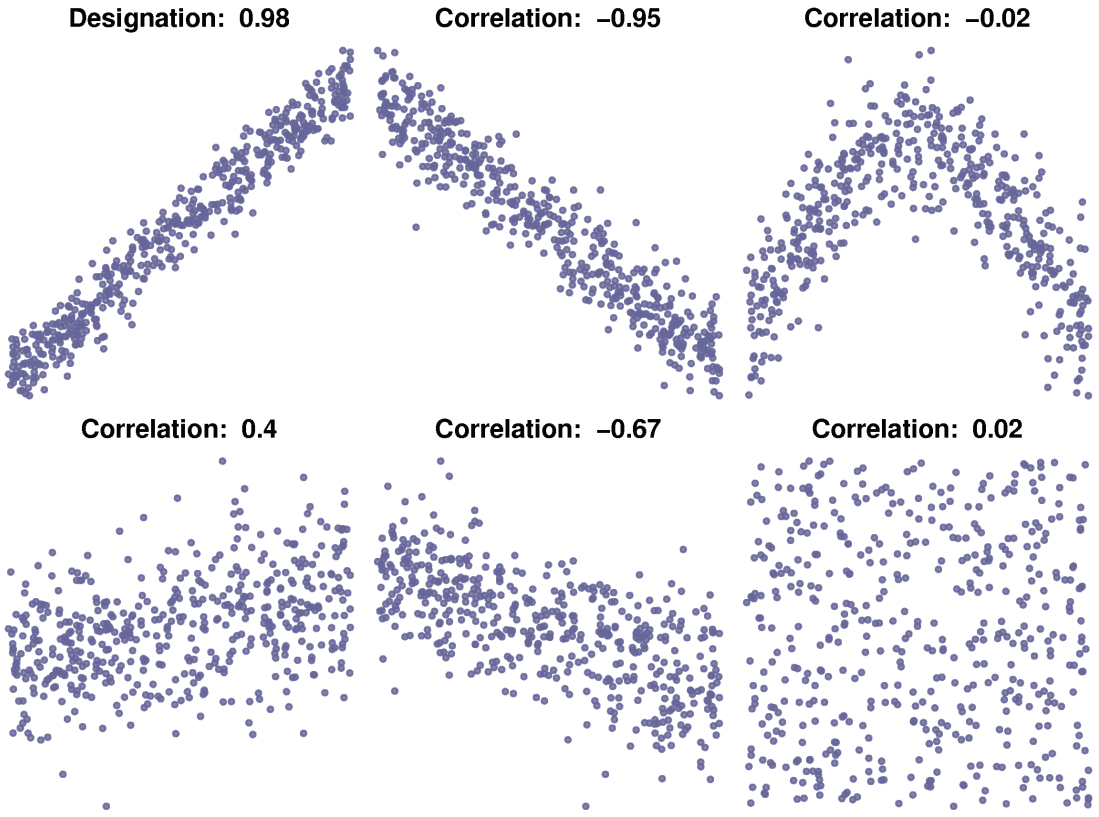


FIGURE 4-5 *Scatterplots showing correlations*

It's also important to remember that correlation is just showing some existence of a relationship between variables, with no implication of ***causation***. For example, imagine that a hypothetical analyst looked at the relationship between security incidents and the number of security operations staff, and reported, "There is a strong positive correlation between the number of SOC analysts in an organization and the number of incidents reported." This could be misunderstood to imply that SOC analysts cause security incidents. In reality, the patterns of data have similar trends with nothing else implied. Perhaps organizations with more incidents hire more SOC analysts, or after hiring more analysts, organizations discover more incidents. Perhaps the two are both a product of something else completely, such as larger organizations are targeted more and have both more incidents and analysts. When you calculate relationships like correlation, you have to be careful to keep it in context. People (and especially those looking for a headline) put a lot of faith in mathematically derived answers and in "having a number." That overconfidence may cause an analyst to take the results out of context and into some really weird places, such as "Researchers suggest we fire SOC analysts to reduce breaches!" You have to be careful of how you position your work and be sure to present the results with an appropriate communication of confidence in the techniques.

For the IANA data, it makes sense that there would be more malicious nodes in larger groups of assigned network blocks. This is an expert opinion that's based on a cursory observation of data and an intuitive feel for the "right" answer. To make a more statistically backed statement, use the code in Listing 4-9 to

generate the plot in Figure 4-6 of the relationship between the two variables *IANA.Block.Count* and *AlienVault.IANA.Count*.

LISTING 4-9

```
# Listing 4-9
# requires packages: ggplot2
# requires object: combined.df (4-7), set2 (4-4)
# generates figure 4-6
gg <- ggplot(data=combined.df)
gg <- gg + geom_point(aes(x=IANA.Block.Count,
                          y=AlienVault.IANA.Count),
                      color=set2[1], size=4)
gg <- gg + labs(x="IANA Block Count", y="AlienVault IANA Count",
                title="IANA ~ AlienVault")
gg <- gg + theme(axis.text.x = element_text(angle = 90, hjust = 1),
                 panel.background = element_blank(),
                 legend.position = "none")
gg
```

The scatterplot in Figure 4-6 appears to show a positive correlation, but to be sure, you should move from eyeballs to keyboards to run a statistical comparison. There are a number of methods available to perform basic pairwise correlation. R provides access to three fundamental algorithms via the built-in `cor()` function:

```
cor(combined.df$IANA.Block.Count,
    combined.df$AlienVault.IANA.Count, method="spearman")
## [1] 0.9488598
```

The value returned by `cor()` is known as the ***correlation coefficient*** and, as pointed out earlier, if it falls close to +1, this indicates there is a strong positive linear relationship between the two variables. As previously noted, R's built-in `cor()` function offers three methods of correlation:

- `pearson` (which is the default if no parameter is specified) refers to the Pearson product moment correlation function and was designed to be most effective when run on continuous data sets with a normal distribution (see later in this chapter) that is free of outliers;
- `spearman` refers to the Spearman rank-order correlation coefficient and—as the name implies—works on rank-ordered data. In other words, if you have two columns in an R data frame that have either pre-ranked data (for example, ordered "top 10" elements) or can be put into rank order (especially to avoid the outlier problems that reduce the efficacy of the Pearson algorithm). Algorithmically, the Spearman calculation is actually the Pearson correlation coefficient calculated from ranked variables;
- `kendall` refers to Kendall's ***tau*** scorrelation coefficient and was specifically designed to work on ranked, ordinal variables. It represents the difference between the probabilities of the outcome variable increasing and decreasing with respect to an input variable.

comma / lc

Keep para together. Move to next page beneath Figure 4-6.

For this example, we applied the Spearman correlation, as it produces a rank correlation coefficient and is generally more suited to variables that do not have a normal distribution (you can execute the `hist()`

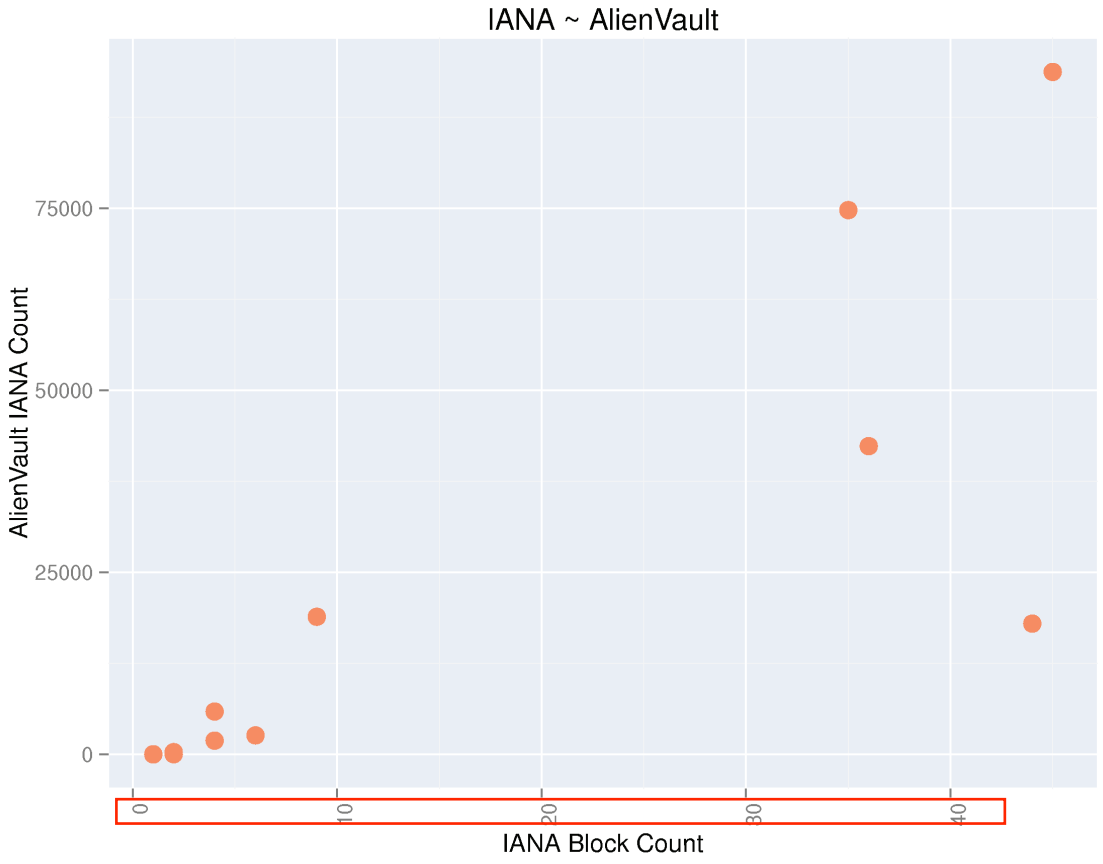


FIGURE 4-6 *Scatterplot of malicious node counts to number of /8 blocks managed by a registrar*

Au: Rotate nos. 90 degrees cloackwise?

AR: submitted new fig.

function on each list to see how far removed each data set is from the normal distribution) and are better compared by rank. For readers who are unfamiliar with what a normal distribution is, a simplified explanation is that a data set is normally distributed if:

- It has an equal mean and median.
- 68% of the values lie within one standard deviation of the mean.
- 95% of the values lie within two standard deviations of the mean.
- 99.7% (or more) lie within three standard deviations of the mean.

period

percent / period

percent / period

percent / period

You now have some statistical backing to help validate the visual pattern and logical (common sense) view that larger blocks of networks will contain more malicious hosts. You could run a similar analysis of your own internal data and, say, determine if there's a relationship between the number of employees in a department and the number of viruses detected.

> ***Note***
>
> *Chapter 5 goes into more detailed methods for determining relationships between variables.*

Mapping Outside the Continents

Calculating and graphing information about malicious nodes is highly useful and vital to the operation of most, if not all, security technologies deployed in today's organizations. However, as a security data scientist, it's a good idea to get into the habit of visualizing data to pick up structures or patterns you might not see otherwise. The classic example of this is Anscombe's quartet, illustrated in Figure 4-7.

Here (Listing 4-10) are the (x,y) pairs that make up the plots in Figure 4-7:

LISTING 4-10

```
# Listing 4-10
# anscombe is a data set that comes with R
# Use the column indexing feature of the
# data frame to show them as pairs
anscombe[,c(1,5,2,6,3,7,4,8)]
```

Au: Grey shading as wanted?

AR: looks good. thx.

```
##    x1    y1 x2    y2 x3    y3 x4    y4
## 1  10  8.04 10  9.14 10  7.46  8  6.58
## 2   8  6.95  8  8.14  8  6.77  8  5.76
## 3  13  7.58 13  8.74 13 12.74  8  7.71
## 4   9  8.81  9  8.77  9  7.11  8  8.84
## 5  11  8.33 11  9.26 11  7.81  8  8.47
## 6  14  9.96 14  8.10 14  8.84  8  7.04
## 7   6  7.24  6  6.13  6  6.08  8  5.25
## 8   4  4.26  4  3.10  4  5.39 19 12.50
## 9  12 10.84 12  9.13 12  8.15  8  5.56
## 10  7  4.82  7  7.26  7  6.42  8  7.91
## 11  5  5.68  5  4.74  5  5.73  8  6.89

# calculate mean and standard deviation for each column
sapply(anscombe,mean)
## x1       x2       x3       x4       y1       y2       y3
## 9.000000 9.000000 9.000000 9.000000 7.500909 7.500909 7.500000
## y4
## 7.500909
sapply(anscombe,sd)
## x1       x2       x3       x4       y1       y2       y3
## 3.316625 3.316625 3.316625 3.316625 2.031568 2.031657 2.030424
## y4
## 2.030579
sapply(anscombe,var)
##       x1        x2        x3        x4        y1        y2
```

Au: Columns aligned as wanted?

AR: looks good. thx.

```
## 11.000000 11.000000 11.000000 11.000000  4.127269  4.127629
##       y3        y4
## 4.122620  4.123249
for (i in 1:4) cat(cor(anscombe[,i], anscombe[,i+4]), "\n")
## 0.8164205
## 0.8162365
## 0.8162867
## 0.8165214
```

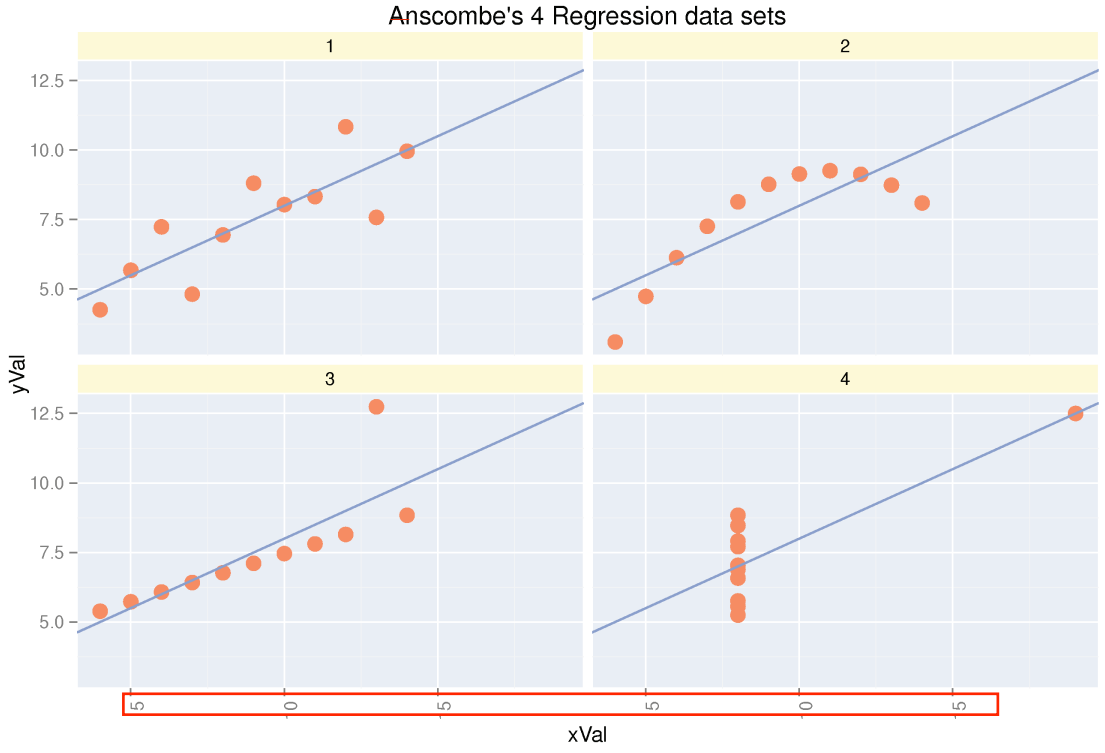


FIGURE 4-7 *Anscombe's quartet*

Au: Rotate nos. 90 degrees cloackwise?

AR: fixed and submitted new figure.

delete parenthesis (no open parenthesis)

All four data sets have the same statistical description (mean, standard deviation, variance, correlation), and they even fit the same linear regression—the blue diagonal line—in the charts in Figure 4-7). Yet, when visualized, patterns emerge. Panel 1 in Figure 4-7 shows a basic linear relationship of a data set that is distributed fairly normally. Panel 2 is definitely not exhibiting a linear relationship, but there is clearly ***some*** relationship between them. Panel 3 has a mostly linear relationship but with an obvious outlier. Finally, Panel 4 shows the power outliers hold, as the extreme point in the upper right is strong enough to have the data still show a strong correlation despite the lack of a linear relationship. Visualizations like this are often key to gaining a much better understanding of the data.

As you learned earlier in the chapter, maps can also be powerful tools for communicating information visually. However, there are other logical and physical visual representations of IP addresses available, especially when you want to see the interconnectedness of nodes. One very versatile representation is the ***graph*** structure since it provides statistical data and has a myriad of options for visual presentation. Do not confuse the term "graph structure" here with producing a graphic or chart. A ***graph structure*** is nothing more than a collection of nodes (vertices) and links between nodes (edges). Nodes and edges have inherent attributes, such as a name/label, but also have attributes that are calculated, such as the number of links going into and coming from the node (the degree). In a traditional graph structure, the direction of an edge (in or out) can be specified as well. In fact, as you'll see in Chapter 8, graphs are becoming so generally useful that there are extremely popular, custom databases that make it very straightforward to store, modify, and analyze large graph structures.

AR: verified ref

Visualizing the ZeuS Botnet

In this section, you combine the metadata you can pull from IP addresses and apply the graph structure to that data to visualize relationships in IP addresses that have been affiliated with malicious behavior. You'll mostly be focusing on building and visualizing graph structures and touching on graph-based analytics for the remainder of this chapter. Previous examples have worked with the AlienVault IP reputation database, but it's time to switch things up a bit and look at one particularly pervasive bit of maliciousness on the Internet: the ZeuS botnet. Most security professionals have heard of ZeuS before, but just in case, here's the description from the `abuse.ch` ZeuS tracker site (`https://zeustracker.abuse.ch/`):

Cap

> ***ZeuS (also known as* Zbot/WSNPoem*) is a crimeware kit that steals credentials from various online services like social networks, online banking accounts, FTP accounts, email, accounts and other (phishing).***

Au: Check extract punctuation

AR: verified. thx.

Despite some prominent attempts at taking down this botnet, it continues to hum along siphoning credentials. The `abuse.ch` site provides a handy blocklist (`https://zeustracker.abuse.ch/blocklist.php?download=badips`) of IP addresses that organizations can use to both identify ZeuS infected nodes and prevent infected systems from communicating with ZeuS command and control (C&C) servers, which orchestrate all operations within the botnet. To work with the blocklist, you need to use the code in Listing 4-11 to get the data file into R (a task you're hopefully getting very familiar with by now).

rebreak

LISTING 4-11

```
# Listing 4-11
# Retrieve and read ZeuS blocklist data into R
zeusURL <- "https://zeustracker.abuse.ch/blocklist.php?\
download=ipblocklist"
zeusData <- "data/zeus.csv"
if (file.access(zeusData)) {
  # need to change download method for universal "https" compatibility
  download.file(zeusURL, zeusData, method="curl")
}
# read in the ZeuS table; skip junk; no header; assign colnames
zeus <- read.table(zeusData, skip=5, header=FALSE, col.names=c("IP"))
```

We've switched to `read.table()` (`read.csv()` is a variant of that function) in Listing 4-11 since there's only one column. This particular data file has no header but it does have five lines at the beginning of the file that are comments and of no use to you programmatically. We also use some shorthand by avoiding a separate call to `colnames()` and embedding the column names right in the `read.table()` function call.

Let's start by determining which countries host ZeuS bots. You could use a geolocation service to get this data, but we'll take a different approach here since you will also require some additional information for the next part of your analysis. The Team Cymru firm provides a number of IP-based lookup services (`http://www.team-cymru.org/Services/ip-to-asn.html`), including an IP to ASN mapping service that supports bulk queries over port 43 and returns quite a bit of handy information:

- AS number
- BGP prefix
- Country code
- Registry
- When it was allocated
- AS organization name

The Team Cymru site clearly states that the country code data is only as accurate as the regional registry databases, but we've run some comparisons against geolocation databases and for the purposes of the examples in this chapter the data is accurate enough. To use this data, you need some helper functions that can be found in the `ch04.R` file in `ch04/R` directory provided on the book's website (`www.wiley.com/go/datadrivensecurity`):

- `trim(`***c***`)`: Takes a character string and returns the same string with leading and trailing spaces removed
- `BulkOrigin(`***ips***`)`: Takes a list of IPv4 addresses and returns a detailed list of ASN origins
- `BulkPeer(`***ips***`)`: Takes a list of IPv4 addresses and returns a detailed list of ASN peers

To build the graph structure, you'll perform the following steps:

- Look up the ASN data.
- Turn the IP addresses into graph vertices.
- Turn the AS origin countries into graph vertices.
- Create edges from each IP address to its corresponding AS origin country.

Surprisingly, it's simple R code, as shown in Listing 4-12.

LISTING 4-12

```
# Listing 4-12
# Building ZeuS blocklist in a graph structure by country
# requires packages: igraph, plyr, RColorBrewer, colorspace
# requires object: set2 (4-4)
```

continues

Listine 4-12 (continued)

```
library(igraph)
library(plyr)
library(colorspace)
# load the zeus botnet data used to perform the
# remainder of the analyses in the chapter
zeus <- read.table("data/zeus-book.csv", skip=5, header=FALSE,
                   col.names=c("IP"))
ips <- as.character(zeus$IP)
# get BGP origin data & peer data;
origin <- BulkOrigin(ips)
g <- graph.empty() # start graphing
# Make IP vertices; IP endpoints are red
g <- g + vertices(ips, size=4, color=set2[4], group=1)
# Make BGP vertices
g <- g + vertices(origin$CC, size=4, color=set2[2], group=2)
# for each IP address, get the origin AS CC and return
# them as a pair to create the IP->CC edge list
ip.cc.edges <- lapply(ips, function(x) {
  iCC <- origin[origin$IP==x, ]$CC
  lapply(iCC, function(y){
    c(x, y)
  })
})
g <- g + edges(unlist(ip.cc.edges)) # build CC->IP edges
# simplify the graph by combining commmon edges
g <- simplify(g, edge.attr.comb=list(weight="sum"))
# delete any standalone vertices (lone wolf ASNs). In "graph" terms
# delete any vertex with a degree of 0
g <- delete.vertices(g, which(degree(g) < 1))
E(g)$arrow.size <- 0 # we hate arrows
# blank out all the IP addresses to focus on ASNs
V(g)[grep("\\.", V(g)$name)]$name <- ""
```

Fig. 4-8 is set 3 pages forward. OK?

Now that you have a graph structure, it's equally as straightforward to make the graph visualization in Figure 4-8 just by passing the graph structure to the `plot()` function with some layout and label parameters as seen in Listing 4-13.

AR: re:3pgs => not a problem.

LISTING 4-13

```
# Listing 4-13
# Visualizing the ZeuS blocklist country cluster graph793725c04f07
# requires packages: igraph, plyr
# requires all objects from Listing 4-11
# this is a great layout for moderately sized networks. you can
# tweak the "n=10000" if this runs too slowly for you. The more
# iterations, the cleaner the graph will look
```

Au: CodeHighlight for # / ?

AR: no. it was a good idea to leave it like the others. thx.

Au; Grey shading as wanted?

AR: yes. looks good. thx

```
L <- layout.fruchterman.reingold(g, niter=10000, area=30*vcount(g)^2)
# plot the graph
par(bg = 'white', mfrow=c(1,1))
plot(g, margin=0, layout=L, vertex.label.dist=0.5,
     vertex.label.cex=0.75,
     vertex.label.color="black",
     vertex.label.family="sans",
     vertex.label.font=2,
     main="ZeuS botnet nodes clustered by country")
```

If your country code memory is a bit rusty, you can use the R code in Listing 4-14 to provide a lookup table.

LISTING 4-14

```
# Listing 4-14
# require package: igraph (4-11)
# requires object: V() (4-11), g (4-11)
# read in country code to name translation table
zeus.cc <- grep("[A-Z]", V(g)$name, value=TRUE)
zeus.cc <- zeus.cc[order(zeus.cc)]
# read in the country codes data frame
cc.df <- read.csv("data/countrycode_data.csv")
# display cc & name for just the ones from our data set
print(head(cc.df[cc.df$iso2c %in% zeus.cc, c(7,1)], n=10),
      row.names=FALSE)
## iso2c   country.name
##    AR      ARGENTINA
##    AU      AUSTRALIA
##    AT        AUSTRIA
##    AZ     AZERBAIJAN
##    BG       BULGARIA
##    CA         CANADA
##    CL          CHILE
##    CN          CHINA
##    CZ CZECH REPUBLIC
##    DE        GERMANY
```

As stated earlier, simple bar charts and tables make it easier to understand quantities, but the graph tends to add a visual impact that traditional presentation techniques lack. Therefore, depending on the consumers, it may be useful/helpful to put them both together when presenting your output.

From your previous work with ASNs, you know that IPs live in both physical and logical space. Now that you have a graph view of the physical world, you can use the code in Listing 4-15 to create the graph network in Figure 4-9 and take look at the ZeuS IP addresses in relation to their ASNs of origin and include ASN peers to truly start to see it as a network.

Fig.4-9 is set three pages forward. OK?

AR: re:3pgs => no problem. thx.

LISTING 4-15

```
# Listing 4-15
# requires objects: BulkOrigin() & BulkPeer() from book's web site
# require package: igraph (4-11)
# create connected network of ZeuS IPs, ASNs, and ASN peers
# generates Figure 4-9
g <- graph.empty()
g <- g + vertices(ips, size=3, color=set2[4], group=1)
origin <- BulkOrigin(ips)
peers <- BulkPeer(ips)
# add ASN origin & peer vertices
g <- g + vertices(unique(c(peers$Peer.AS, origin$AS)),
                  size=3, color=set2[2], group=2)
# build IP->BGP edge list
ip.edges <- lapply(ips, function(x) {
  iAS <- origin[origin$IP==x, ]$AS
  lapply(iAS,function(y){
    c(x, y)
  })
})

bgp.edges <- lapply(
  grep("NA",unique(origin$BGP.Prefix),value=TRUE,invert=TRUE),
  function(x) {
    startAS <- unique(origin[origin$BGP.Prefix==x,]$AS)
    lapply(startAS,function(z) {
      pAS <- peers[peers$BGP.Prefix==x,]$Peer.AS
      lapply(pAS,function(y) {
        c(z,y)
      })
    })
  })
g <- g + edges(unlist(ip.edges))
g <- g + edges(unlist(bgp.edges))
g <- delete.vertices(g, which(degree(g) < 1))
g <- simplify(g, edge.attr.comb=list(weight="sum"))
E(g)$arrow.size <- 0
V(g)[grep("\\.", V(g)$name)]$name = ""
L <- layout.fruchterman.reingold(g, niter=10000, area=30*vcount(g)^2)
par(bg = 'white')
plot(g, margin=0, layout=L, vertex.label.dist=0.5,
     vertex.label=NA,
     main="ZeuS botnet ASN+Peer Network")
```

By expanding the network with the ASN peers, you can see a cluster of interconnected ASNs that might be worth exploring further, but you'll need to reference the resources in the "Recommended Reading" section to take that next step.

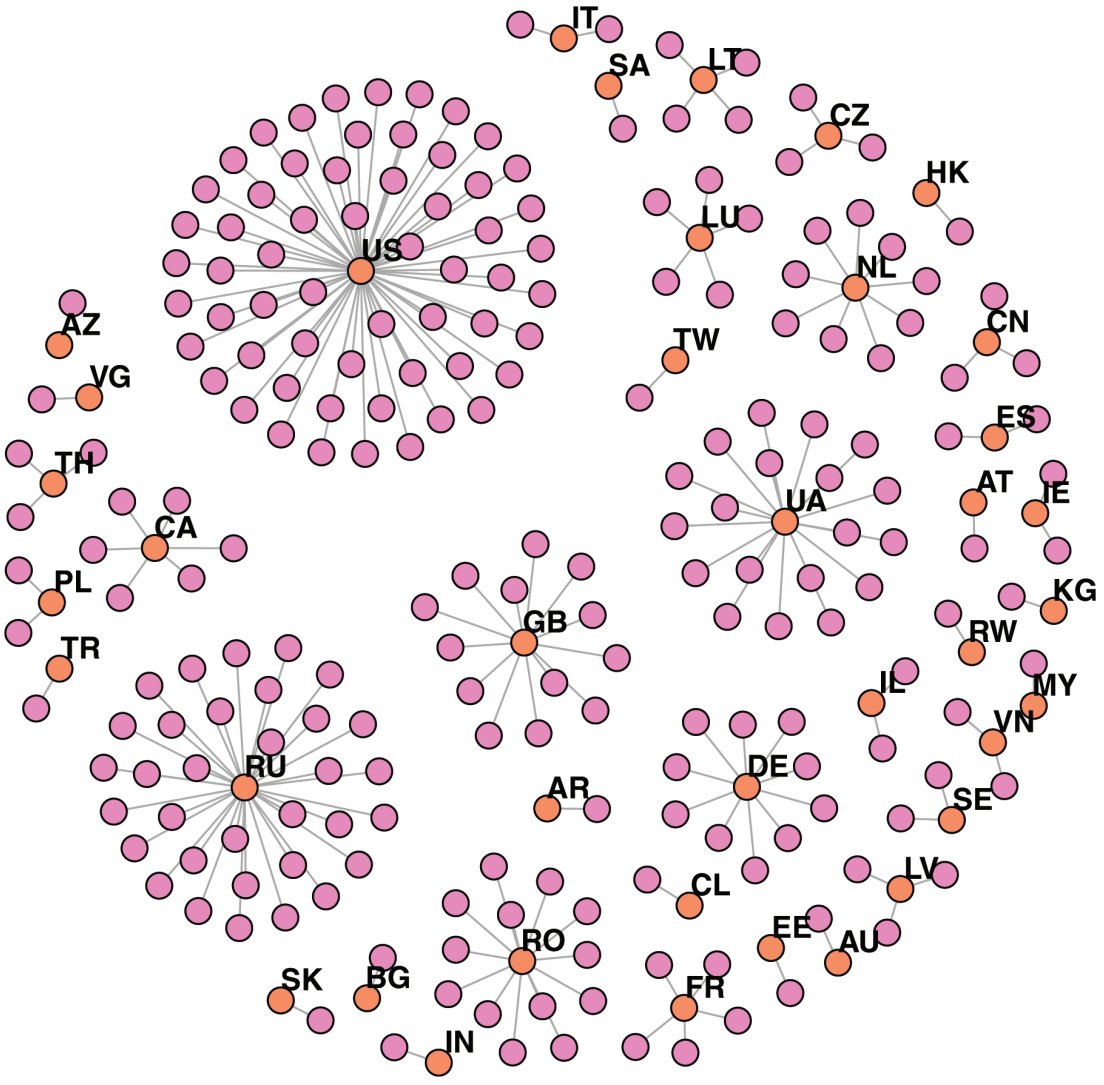


FIGURE 4-8 *ZeuS nodes clustered by origin country*

With basic graph network concepts well in hand, you can turn your attention to a more practical application of these functions—visualizing malicious activity on ***your*** network using actual data from a real environment and attempting to ***visualize*** the answer to the question, "What potentially malicious nodes are attempting to come into/get out of my network?"

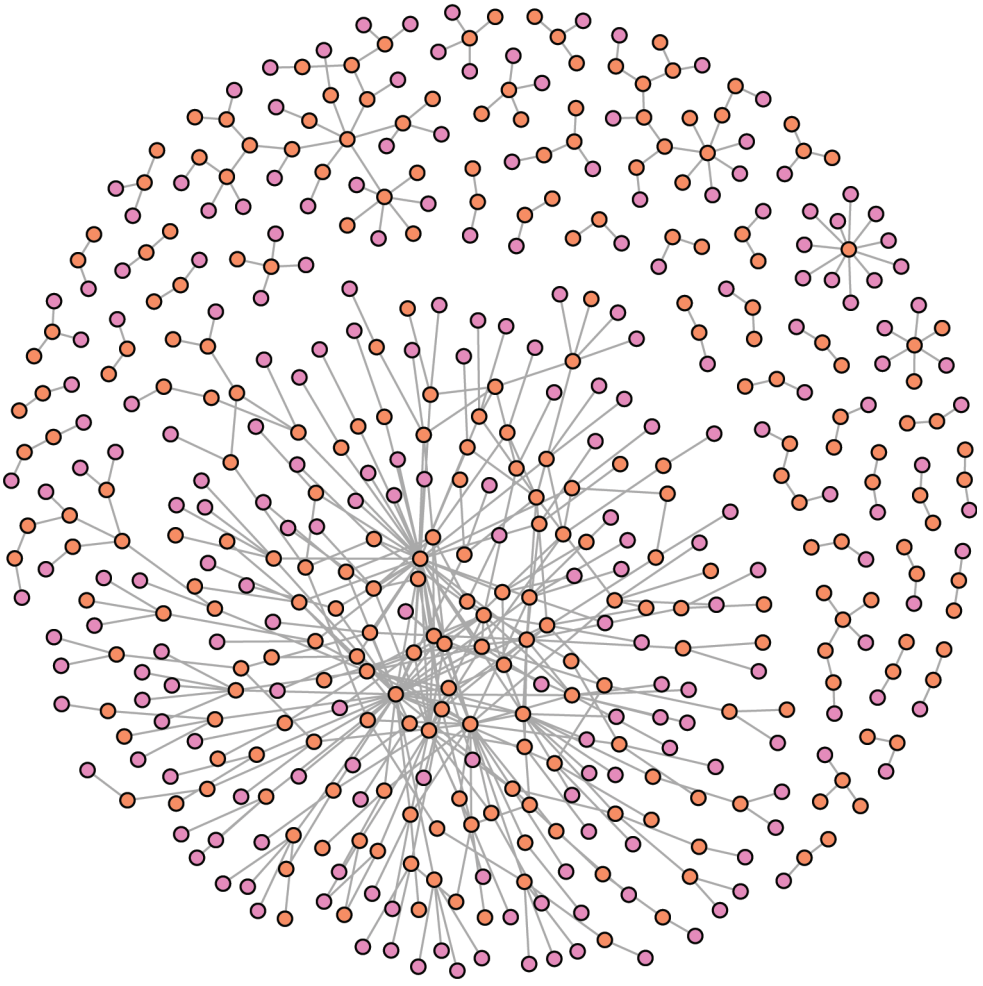


FIGURE 4-9 *ZeuS nodes graph with ASNs and peers*

Visualizing Your Firewall Data

Examining generic data about malicious nodes has some merit, but it's more helpful to apply these analysis and visualization techniques to your own organization. To that end, this last example provides a way to use both the AlienVault IP Reputation database and the graphing techniques presented in this chapter to examine what's happening on a perimeter firewall. Rather than generate some artificial data, we obtained a 24 hours' worth of Internet-bound IP addresses from volunteers, which can be found in the file `dest.ips` in the `ch04/data` directory on the book's website (`www.wiley.com/go/datadrivensecurity`).

This example has also created two new functions, which can be found on the website in the `ch04.R` file in the `ch04/R` directory:

- `graph.cc(ips,av.df)`: Takes in a list of IPv4 addresses and an AlienVault data frame and returns a complete graph network structure of nodes clustered by country code. Also (optionally) plots the graph with a summary of malicious traffic types.
- `graph.asn(ips,av.df)`: Takes in a list of IPv4 addresses and an AlienVault data frame and returns a complete graph network structure of nodes clustered by ASN. Also (optionally) plots the graph with a summary of malicious traffic types.

em dash
It also / ?
em dash
It also / ?

You can start by loading the destination IP addresses and filtering out everything that isn't in the AlienVault database. You then assess the result and try to get a feel for what type of malicious activity to hone in on. Even with the potential bias in the data (as described in Chapter 3), a higher reliability rating should still mean there is a better chance the node is actually "bad." Therefore, you can focus on entries with reliability greater than 6, which will give you 127 nodes to send to `graph.cc()` to process and plot. See Listing 4-16.

AR: VERIFIED REF

LISTING 4-16

```
# Listing 4-16
# requires objects: BulkOrigin() & BulkPeer(), graph.cc(), graph.asn()
#   from book's web site & set2 (4-4)
# working with Real Data
# create connected network of ZeuS IPs, ASNs, and ASN peers
# generates Figure 4-10
# require package: igraph, RColorBrewer
avRep <- "data/reputation.data"
av.df <- read.csv(avRep, sep="#", header=FALSE)
colnames(av.df) <- c("IP", "Reliability", "Risk", "Type",
                     "Country", "Locale", "Coords", "x")

# read in list of destination IP addresses siphoned from firewall logs
dest.ips <- read.csv("data/dest.ips", col.names= c("IP"))

# take a look at the reliability of the IP address entries
# (you could also plot a histogram)
table(av.df[av.df$IP %in% dest.ips$IP, ]$Reliability)
##  1   2   3   4   5   6   7   8   9  10
## 16 828 831 170   1 266  92   2  23  24

# extract only the "bad" ones, designated by presence in alienvault
# database with a reliability greater than 6 since there seems to
# be a trailing off at that point
ips <- as.character(av.df[(av.df$IP %in% dest.ips$IP) &
                          (av.df$Reliability > 6), ]$IP)
# graph it
g.cc <- graph.cc(ips, av.df)
```

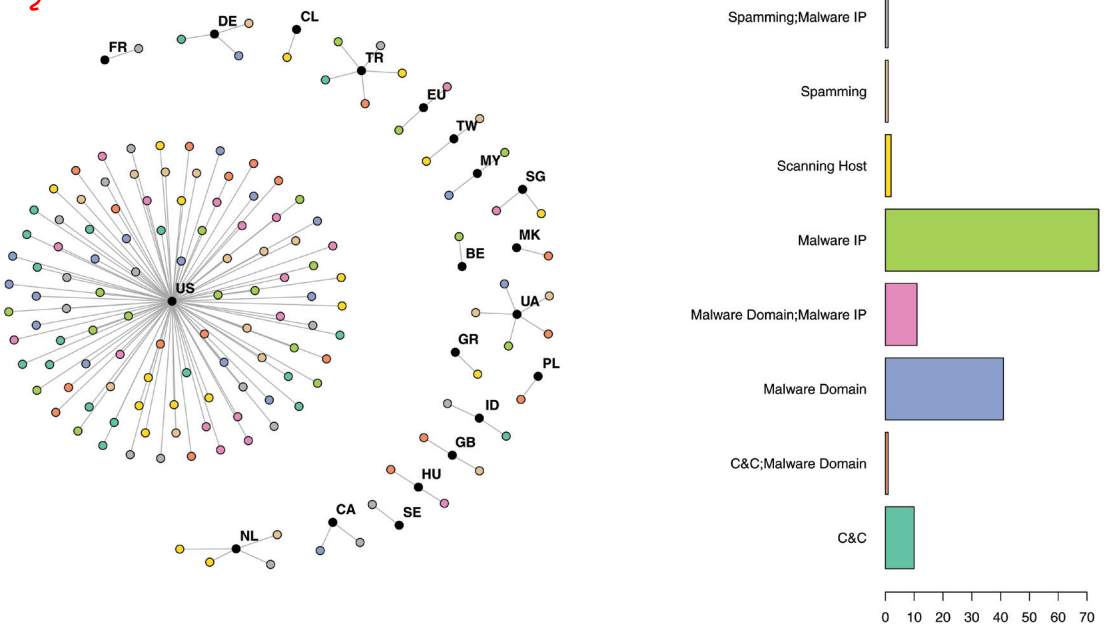


FIGURE 4-10 *Graph of malicious destination traffic by country*

The bar chart on the right serves as a legend for the colors of the graph nodes and also provides a summary of the totals of each classification type. In Figure 4-10, you can see there is some potential C&C traffic and that the United States has the highest number of possible malicious destinations. With `graph.cc()`'s ASN cousin and the slicing and dicing example in Listing 4-16, you should have enough tools to generate your own views in order to look at different aspects of the malicious traffic.

Move para above Fig.. 4-10 /

Summary

The goal of this chapter was to show you the importance of fully understanding the data elements you want to analyze and visualize as well as the need to start with a question and iterate through computations and visualizations to work toward an answer. There are plenty of other similar data sets available on the Internet to substitute for the ones provided in the most of the examples. Hunting those down (or just using your own firewall data in the last example), working through the sample analyses, and formulating your own questions will help to ingrain the pattern of the data analysis workflow in your mind.

comma

delete: the

There are many ways to look at IP-based malicious activity and this chapter was by no means comprehensive. Furthermore, R was not entirely necessary for anything but the visualizations and statistical analyses. Much of the sorting, slicing, and dicing could have been performed in a database and—as you'll see in Chapter 8—that is definitely the place to start when working with larger data sets.

The next chapter expands on these analyses and should give you a new "out of this world" perspective on botnet data.

AR: VERIFIED REFS

Recommended Reading

The following are some recommended readings that can further your understanding on some of the topics we touch on in this chapter. For full information on these recommendations and for the sources we cite in the chapter, please see Appendix B. AR: VERIFIED REF

Mining Graph Data **by Diane J. Cook and Lawrence B. Holder**

Graphical Models with R **by Søren Højsgaard, David Edwards, and Steffen Lauritzen**