Chapter 2: Learning The “Hello World” Of Security Data Analysis

“from one thing, know ten thousand things”

― Miyamoto Musashi, The Book of Five Rings: Miyamoto Musashi

If you’ve ever tried to learn a new programming language there’s a good chance you started of with a “Hello World” example that quickly introduces basic language structure and code execution. The immediate sense of accomplishment as the syntax is verified by the compiler/interpreter and the familiar two-word output is displayed becomes a catalyst for the notion that, soon, you shall have the ability to bend this new language to your will.

This chapter takes the “Hello World” concept and expands it to a walk-through of a self-contained, introductory security data analysis use case that you will be able to follow along with, execute and take concepts from as you start to perform your own analyses. There are parallel examples in *Python* and *R* to give you an idea of the similarities, strengths and differences between both languages in a real life example context. If you’re not familiar with one or both of those languages you should read Chapter 2 first and at least skim some of the external resources referenced there. This is a good place to reinforce the recommendation to use *IPython Notebooks* or *RStudio* for your analyses and exploration as they provide very robust and forgiving environments and each will be far more optimal then saving and executing scripts. Remember, all the source code, sample data and visualizations are on the book’s web site, so no need for transcription, just cut/paste and focus on the flow of and concepts presented in the examples.

Preparing For Analysis

Before jumping into data retrieval and analysis, we need to setup an area where we can organize all our input data, analysis scripts, output (visualizations, reports and/or data) and any supporting documentation. For the purposes of this chapter, we’ll be using the following directory structure:

/book/chapter3/reputation

|-*R*

|-data

|-docs

|-output

|-python

|-support

|-tmp

Like most elements of programming, there is no one, true way to setup this structure, but you should strive to find one that works for you and stick with it. A great way to do that is to take a lesson from modern web framework builders and use a simple setup script that builds the structure for you:

Sample analysis preparation script

#!/bin/sh

#

# prep: prep analytics directory structure

#

# usage: prep DIRNAME

#

DIR=$1

if [ ! -d "${DIR}" ]; then

mkdir -p ${DIR}/*R* \

${DIR}/data \

${DIR}/docs \

${DIR}/output \

${DIR}/python \

${DIR}/support \

${DIR}/tmp

> ${DIR}/readme.md

ls -lR ${DIR}

else

echo "Directory "${DIR}" already exists"

fi

You now only need to type “prep NAME” whenver you want to start a new project (so, for this project, “prep reputation”). As you develop your own styles and patterns, you can expand this script to include generation of script templates and initialization of source code repositories. Once the structure is in place, it’s time to retrieve, explore and analyze some data.

Getting Data

We are living in a silver age of data in information security. The challenge is no longer where to get data from, but what to do with it. Figure 3-1 lists many of the common internal and external sources and types of available data, and—as you’ll see in the rest of the book—the kind of information in each will drive the type of research you perform.

Table 3-1 Common "Security" Data Sources

|  |  |
| --- | --- |
| Internal Data Sources | External Data Sources |
| Windows Event Logs  Linux/UNIX syslogs  Mainframe Logs  Network Device Logs  Proxy Server Logs  Firewall Logs  Anti-malware Management Event Databases/Alerts  Vulnerability Management Databases  Patch Management Databases  System Configuration Logs/Databases  Identity & Access Management/RBAC Records  NetFlow Data  PCAP Data  HR Data Feeds  Application Logs  Web Application Firewall Logs  E-mail Gateway/Spam Filter Logs  Business Transactional Data Logs  Database Audit Logs  Asset Management Databases  Physical Security Event Logs  IDS/IDP Alerts  Help Desk/Non-security Incident Tickets  Risk Assessments  Penetration Testing Results  Application Security Scans  Firewall Port Requests | Threat Intelligence Reports  Indicators of Compromise  Industry Reports  Open Source Reputation Data  Malware Signatures  IDS Signatures  Botnet Details  Spam Corpi |

For this use case, we’ll be working with AlienVault’s IP Reputation Database (<http://labs.alienvault.com/labs/index.php/projects/open-source-ip-reputation-portal/download-ip-reputation-database/>). AlienVault develops OSSIM—an open source security information manager—and a proprietary unified threat management (UTM) product, both of which make use of this freely available data set that contains information on various types of “badness” across the internet. AlienVault provides this data in numerous formats and the version we’ll be working with is the OSSIM Format (<http://reputation.alienvault.com/reputation.data>) as it provides the richest information of the ones available.

type="tip"

AlienVault updates their IP reputation data set hourly and produces a companion “revision” file (<http://reputation.alienvault.com/reputation.rev>), enabling you to ensure you are working with the latest data set or keep a history of data sets. There is additional code on the companion web site that shows how to perform this check to see if it’s time to download a new one.

When performing a one-off, exploratory analysis or getting a first look at a data set, it’s acceptable to just do a quick download via browser. If we do that for the AlienVault IP reputation database and examine the first few data elements we can get an idea of the contents and format, which will come in handy when we start to read in and work with the data. Here, we use some simple Linux/UNIX commands to inspect the download:

Performing a quick review of the downloaded data set

$ **head -10 reputation.data** *# look at the first few lines in the file*

222.76.212.189#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.185#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.186#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

5.34.246.67#6#3#Spamming#US##38.0,-97.0#12

178.94.97.176#4#5#Scanning Host#UA#Merefa#49.823001861,36.0507011414#11

66.2.49.232#4#2#Scanning Host#US#Union City#37.59629821,-122.0656966#11

222.76.212.173#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.172#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.171#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

174.142.46.19#6#3#Spamming###24.4797992706,118.08190155#12

$ **wc –l reputation.data** *# see how many total records there are*

258626 reputation.data

For most projects it’s better to get into the habit of retrieving the data source directly from your analysis scripts. If you still prefer to download files manually you should provide some type of comment in your programs that provides details on where the source data comes from and when you retrieved the data for your current analysis to make it easier to repeat the analyses at a later date.

The following snippets show how to perform the data retrieval in both *R* and *Python*. If you are following along with *RStudio* or *IPython*, both sets of snippets assume a working directory of the top level of the project structure (e.g. executing from “reputation” directory). Most snippets rely on the execution of the previous ones, so it’s important to run them all in the same session. If you need to stop and come back later, you will need to re-run all the code from the start of the chapter to where you left off to ensure proper results.

R code to download the AlienVault data

# URL for the AlienVault IP Reputation Database (OSSIM format)

# storing the URL in a variable makes it easier to modify later

# if it changes

avURL <- "http://reputation.alienvault.com/reputation.data"

# relative path for the downloaded data

avRep <- "data/reputation.data"

# using an if{}-wrapped test with download.file() vs read.xxx() avoids

# having to re-download a 16MB file every time we run the script

if (file.access(avRep)) {

download.file(avURL,avRep)

}

Python code to download the AlienVault data

#!/usr/bin/python

#

# reputation.py

#

# sample analysis script for AlienVault IP Reputation Database data

#

# URL for the AlienVault IP Reputation Database (OSSIM format)

# storing the URL in a variable makes it easier to modify later

# if it changes

import urllib

import os.path

avURL = "http://reputation.alienvault.com/reputation.data"

# relative path for the downloaded data

avRep = "data/reputation.data"

# using an if-wrapped test with urllib.urlretrieve() vs direct read

# via panads avoids having to re-download a 16MB file every time we

# run the script

if not os.path.isfile(avRep):

urllib.urlretrieve(avURL, filename=avRep)

The *R* and *Python* code look very similar and follow the same basic structure: using variables whenever possible for URL and filenames plus testing for the existence of the data file before downloading it again. These are good habits to get into and we’ll be underscoring other suggested good practices throughout the rest of the book.

With the IP reputation data in hand, it’s now time to read in the data so we can begin to work with it.

Reading In Data

*R* and *Python* (especially with *pandas*) abstract quite a bit of complexity when it comes to reading and parsing data into structures for processing. R’s read.table(), read.csv() and read.delim() and *pandas* read\_csv() will cover nearly all your delimited file reading needs and provide robust configuration options for even the most gnarly input file. Both tools, as we’ll see in later chapters, also provide ways to retrieve data from SQL and “NoSQL” databases, HDFS “big data” setups and even handle unstructured data quite well.

From our cursory examination of the downloaded file, we can see the AlienVault data has a fairly straightforward record format with eight primary fields using a “#” as the field separator.

222.76.212.189#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

The consistency in the record format makes the consumption of the data equally as straightforward in each language.

R code to read in the AlienVault data

# read in the IP reputation db into a data frame

av <- read.csv(avRep,sep="#",stringsAsFactors=FALSE)

**str(av)** *# take a quick look at the data frame*

'data.frame': 258625 obs. of 8 variables:

$ X222.76.212.189 : chr "222.76.212.185" "222.76.212.186"

"5.34.246.67" "178.94.97.176" ...

$ X4 : int 4 4 6 4 4 4 4 4 6 4 ...

$ X2 : int 2 2 3 5 2 2 2 2 3 2 ...

$ Scanning.Host : chr "Scanning Host" "Scanning Host"

"Spamming" "Scanning Host" ...

$ CN : chr "CN" "CN" "US" "UA" ...

$ Xiamen : chr "Xiamen" "Xiamen" "" "Merefa" .°°

$ X24.4797992706.118.08190155: chr "24.4797992706,118.08190155"

"24.4797992706,118.08190155" "38.0,-97.0" "49.82300116,36.05070114"

...

$ X11 : chr "11" "11" "12" "11" ...

#make smarter column names

colnames(av) <- c("IP","Reliability","Risk",

"Type","Country","Locale","Coords","x")

**head(av)** *# take a look at the first few rows*

IP Reliability Risk Type Country Locale

1 222.76.212.185 4 2 Scanning Host CN Xiamen

2 222.76.212.186 4 2 Scanning Host CN Xiamen

3 5.34.246.67 6 3 Spamming US

4 178.94.97.176 4 5 Scanning Host UA Merefa

5 66.2.49.232 4 2 Scanning Host US Union City

6 222.76.212.173 4 2 Scanning Host CN Xiamen

Coords x

1 24.4797992706,118.08190155 11

2 24.4797992706,118.08190155 11

3 38.0,-97.0 12

4 49.8230018616,36.0507011414 11

5 37.5962982178,-122.065696716 11

6 24.4797992706,118.08190155 11

Python code to read in the AlienVault data

import *pandas* as pd

# read in the data into a *pandas* data frame

av = pd.read\_csv(avRep,sep="#")

**print(av)** *# take a quick look at the data structure*

<class '*pandas*.core.frame.DataFrame'>

Int64Index: 258625 entries, 0 to 258624

Data columns (total 8 columns):

IP 258625 non-null values

Reliability 258625 non-null values

Risk 258625 non-null values

Type 258625 non-null values

Country 248570 non-null values

Locale 184555 non-null values

Coords 258625 non-null values

x 258625 non-null values

dtypes: int64(2), object(6)

# make smarter column names

av.columns = ["IP","Reliability","Risk","Type","Country",

"Locale","Coords","x"]

**av.head()** *# take a look at the first few rows*

IP Reliability Risk Type Country Locale

0 222.76.212.185 4 2 Scanning Host CN Xiamen

1 222.76.212.186 4 2 Scanning Host CN Xiamen

2 5.34.246.67 6 3 Spamming US NaN

3 178.94.97.176 4 5 Scanning Host UA Merefa

4 66.2.49.232 4 2 Scanning Host US Union City

5 222.76.212.173 4 2 Scanning Host CN Xiamen

6 222.76.212.172 4 2 Scanning Host CN Xiamen

7 222.76.212.171 4 2 Scanning Host CN Xiamen

8 174.142.46.19 6 3 Spamming NaN NaN

9 66.2.49.244 4 2 Scanning Host US Union City

Coords x

0 24.4797992706,118.08190155 11

1 24.4797992706,118.08190155 11

2 38.0,-97.0 12

3 49.8230018616,36.0507011414 11

4 37.5962982178,-122.065696716 11

5 24.4797992706,118.08190155 11

6 24.4797992706,118.08190155 11

7 24.4797992706,118.08190155 11

8 24.4797992706,118.08190155 12

9 37.5962982178,-122.065696716 11

*IPython* *Notebook*s also have a useful set of functions to output data to a more viewer-friendly HTML format:

IPython code to display head() as an HTML table

from *IPython*.display import HTML

HTML(av.head(10).to\_html())

Figure 3-1 *IPython* HTML head() Output [f0301.png]

Since the reputation data file lacks a header, each example code segment assigns more meaningful column names manually. This is a completely optional step, but it will help avoid confusion as you expand your analyses and, as we’ll see further in this chapter, help build consistency across data frames if you bring in additional data sets.

Exploring Data

It’s now time to bring your security domain expertise into the discussion to explore and discover what is interesting about the data and enable us to form good questions to ask and answer. Despite having almost 260,000 records, we have many tools at our disposal to help get a feel for what it contains.

There are some tidbits of information we know about the data even before we take a more programmatic look:

* each record is associated with a unique IP address, so there are 258,626 IP addresses (in this download) and we won’t be able to glean much (if any) insights from just the IP addresses contained in this file;
* some attempt has been made to discern how reliable the IP address classification is;
* some attempt has been made to discern the level of “risk” associated with each IP address;
* each IP address has been pre-geo-located for us (so we won’t have to do that if our exploration leads us down that path).

A good next step perform is to take a look at the *central* *tendency* of all or parts of the data set. We’ll perform this part of the examination on the *Reliability* and *Risk* columns, as they are the only numeric columns we have at our disposal. For the purposes of this introductory chapter, the elements we’ll pick out are the *mean* (average value), *median* (midpoint value) and *mode* (most common value) along with the range (minimum and maximum values). Note that for *R* we need to whip up a quick helper function to calculate the mode since it’s not part of the standard library, and we need to use the mode function from *Python*’s *scipy* package.

R code to look at the central tendency of Reliabilty and Risk

**summary(av$Reliability)**

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.000 2.000 2.000 2.798 4.000 10.000

**summary(av$Risk)**

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.000 2.000 2.000 2.221 2.000 7.000

colmode <- function(n) {

siftedUnique <- unique(n)

siftedUnique[which.max(tabulate(match(n, siftedUnique)))]

}

**colmode(av$Reliability)**

2

# use the fact that we can index into an R

# table() to mimic Python’s mode() function output

**table(av$Reliability)[colMode(av$Reliability)]**

149117

**colmode(av$Risk)**

2

**table(av$Risk)[colmode(av$Risk)]**

213851

Python code to look at the central tendency of Reliabilty and Risk

**av['Reliability'].describe()**

count 258625.000000

*mean 2.798036*

std 1.130419

*min 1.000000*

25% 2.000000

50% 2.000000

75% 4.000000

*max 10.000000*

**av['Risk'].describe()**

count 258625.000000

*mean 2.221363*

std 0.531572

*min 1.000000*

25% 2.000000

50% 2.000000

75% 2.000000

*max 7.000000*

from scipy.stats import mode

**mode(av[Reliability])**

(array([ 2.]), array([ 149117.]))

**mode(av['Risk'])**

(array([ 2.]), array([ 213851.]))

From an examination of the above results, we can make two notes for further follow up:

* The *Reliability* column spreads across the documented potential range [1…10] but the *Risk* column—which also has a documented potential range of [1…10] only has a spread of [1…7];
* Both columns heavily gravitate towards a value of ‘2’.

We can now dig a bit deeper and use the fact that the “*Reliability*”, “*Risk*”, “*Type*” and “*Country*” fields are the equivalent of categorical data: i.e. they enable dividing the data set into groups. We will use the summary() and factor() functions in *R* to see counts of some these groupings and use an equivalent set of functions from *pandas* to convert a data frame column (which is just an array) into an a very appropriately named Categorical object which we can tweak a bit to give us similar helpful output.

R exploratory code for AlienVault data

**summary(factor(av$Reliability))**

1 2 3 4 5 6 7 8 9 10

5612 149117 10892 87039 7 4758 297 21 686 196

**summary(factor(av$Risk))**

1 2 3 4 5 6 7

39 213851 33719 9588 1328 90 10

**head(summary(factor(av$Type)),20)**

APT;Malware Domain

1

C&C

610

C&C;Malware Domain

31

C&C;Malware IP

20

C&C;Scanning Host

7

Malicious Host

3770

Malicious Host;Malware Domain

4

Malicious Host;Malware IP

2

Malicious Host;Scanning Host

163

Malware distribution

1

Malware distribution;Malicious Host

1

Malware distribution;Malware IP

4

Malware Domain

9274

Malware Domain;C&C

25

Malware Domain;Malicious Host

4

Malware Domain;Malware IP

173

Malware Domain;Scanning Host

39

Malware Domain;Spamming

2

Malware IP

6470

Malware IP;C&C

2

**summary(factor(av$Country))**

CN US TR DE NL RU GB

68582 50387 13958 10055 9953 7931 6346 6293

IN FR TW BR UA RO KR CA

5480 5449 4399 3811 3443 3274 3101 3051

AR MX TH IT HK ES CL AE

3046 3039 2572 2448 2361 1929 1896 1827

JP HU PL VE EG ID RS PK

1811 1636 1610 1589 1452 1378 1323 1309

VN LV NO CZ BG SG IR IL

1203 1056 958 928 871 868 866 854

PT BE MD MY SA ZA GR PA

847 834 788 664 582 573 557 554

PH BD LB IS UY CH KZ CY

552 535 517 516 516 333 313 295

PE FI LU EC EE NZ KW A1

295 286 283 278 274 272 269 267

GT KH DO PY AW AO NI IE

261 261 259 259 257 256 256 201

AU SE EU LT VG DK AT BY

155 130 129 65 59 54 51 35

GE CO SK HR PS SI JO BA

34 33 31 25 23 20 16 15

MK MA BO TN BZ AZ MN PR

14 13 10 10 8 7 7 7

AM LK MQ (Other)

6 5 5 105

Python exploratory code for AlienVault data

# summary\_col(col)

#

# helper function to mimic *R*'s "summary()" function

# for *pandas* "columns" (which are really just *Python*

# arrays)

#

def factor\_col(col):

factor = pd.Categorical.from\_array(col)

return pd.value\_counts(factor,sort=True).reindex(factor.levels)

rel\_ct = pd.value\_counts(av['Reliability'])

risk\_ct = pd.value\_counts(av['Risk'])

type\_ct = pd.value\_counts(av['Type'])

country\_ct = pd.value\_counts(av['Country'])

**print factor\_col(av['Reliability'])**

1 5612

2 149117

3 10892

4 87039

5 7

6 4758

7 297

8 21

9 686

10 196

dtype: int64

**print factor\_col(av['Risk'])**

1 39

2 213851

3 33719

4 9588

5 1328

6 90

7 10

dtype: int64

**print factor\_col(av['Type'])**

APT;Malware Domain 1

C&C 610

C&C;Malware Domain 31

C&C;Malware IP 20

C&C;Scanning Host 7

Malicious Host 3770

Malicious Host;Malware Domain 4

Malicious Host;Malware IP 2

Malicious Host;Scanning Host 163

Malware Domain 9274

Malware Domain;C&C 25

Malware Domain;Malicious Host 4

Malware Domain;Malware IP 173

Malware Domain;Scanning Host 39

Malware Domain;Spamming 2

Malware IP 6470

Malware IP;C&C 2

Malware IP;Malicious Host 1

Malware IP;Malware Domain 57

Malware IP;Scanning Host 8

Malware IP;Spamming 7

Malware distribution 1

Malware distribution;Malicious Host 1

Malware distribution;Malware IP 4

Scanning Host 234179

Scanning Host;C&C 2

Scanning Host;Malicious Host 215

Scanning Host;Malware Domain 19

Scanning Host;Malware IP 7

Scanning Host;Spamming 7

Spamming 3487

Spamming;Malware Domain 5

Spamming;Malware IP 4

Spamming;Scanning Host 24

dtype: int64

**print factor\_col(av['Country'])**

A1 267

A2 2

AE 1827

AL 4

AM 6

AN 3

AO 256

AR 3046

AT 51

AU 155

AW 257

AX 1

AZ 7

BA 15

BD 535

...

TW 4399

TZ 1

UA 3443

UG 1

US 50387

UY 516

VC 1

VE 1589

VG 59

VI 1

VN 1203

YE 2

ZA 573

ZM 1

ZW 3

Length: 152, dtype: int64

These numerical tables can help us get some understanding of the makeup of the data, but a picture has the potential to provide a whole new perspective, often times giving insights that numbers alone cannot reveal. For this chapter, we are using the “base graphics” package in *R* and the standard matplotlib package in *pandas*, but there are many ways to produce appealing and informative visualizations in both environments and future chapters will expand upon these options.

We’ll use a simple bar chart to get a very quick visual overview of the *Country*, *Reliability* and *Risk* factors.

R code for visualizing portions of AlienVault data

barcol = "#762A83AA"

# save the current (default) graphic parameters

opar <- par()

# setup a 3x1 grid for plotting multiple charts

par(mfrow=c(3,1))

barplot(head(summary(factor(av$Country)),20),

col=barcol,

main="Summary By Country",xlab="Country")

barplot(summary(factor(av$Risk)),

col=barcol,

main="Summary by 'Risk'",xlab="Host 'Risk' Level")

barplot(summary(factor(av$Reliability)),

col=barcol,

main="Summary by Rating Reliability",xlab="Reliability")

# restore the original graphics parameters

par(opar)

Figure 3-2 Bar Charts of Reliability, Risk, Type and Country Factors (*R*) [f0302.png]

Python code for visualizing portions of AlienVault data

# we want the country counts sorted

country\_ct = pd.value\_counts(av['Country'])

country\_ct[:20].plot(kind='bar', rot=0,

title="Summary By Country")

factor\_col(av['Reliability']).plot(kind='bar',

rot=0,title="Summary By 'Reliability'")

factor\_col(av['Risk']).plot(kind='bar', rot=0,

title="Summary By 'Risk'")

factor\_col(av['Type']).plot(kind='bar', rot=0)

Figure 3-3 Bar Charts of Reliability, Risk, Type and Country Factors (*pandas*) [f0303.png]

Those traditional visualizations have helped us quickly glean that:

* China and the United States are the countries that seem to have the majority of malicious nodes. We’ve limited the bar chart stops to the first twenty countries for visual brevity, but you can modify the code to see that there’s a long tail for the rest of the world. This tracks consistently with other group’s public data and industry reports, so we won’t be focusing on this column much more.
* The level of risk of most of the nodes is, well, *negligible*. There are other elements that stand out with this factor, foremost being that practically no endpoints are in categories 1 (very low), 5, 6 or 7 (heading towards very high). We should make another note to dig a bit deeper.
* The reliability of the node ratings also appears to be a bit skewed. There aren’t many “solid” ratings (i.e. greater than 5 or 6) and there are overt clusters in levels 2 and 4. We can make one more note here for follow up.

We now have some leads to pursue on and a much better idea of the makeup of the key components of the data, which should be plenty of fodder for formulating some practical questions.

Asking A Question

Consider the primary use-case for the AlienVault reputation data: importing it into a SEIM or IDP/IDS environment to alert incident response team members or just block malicious activity. How would this quick overview of the reputation data influence the configuration of your security technologies to ensure the least number of false positives? Or, possibly more importantly, how valuable is this data set to you given the reliability and risk levels?

We’ll take a slightly more deterministic view of those questions by asking how many IP addresses from the reputation database represent a real threat? There *is* a reason AlienVault included both *Risk* and *Reliability* fields, and each has a range of [1…10] making them dividable into categories of *very low*, *low*, *medium*, *high* and *very high*. The definition of “real threat” can be somewhat subjective, but for the purposes of this example we’ll focus on a medium or higher reliability and risk ratings and review the results. How many nodes fall into a the “truly risky” category (*Reliability* > 4 and *Risk > 3*)?

JAY THOUGHT:

first: <http://www.statmethods.net/stats/frequencies.html>

> mytable <- xtabs(~Risk+Reliability, data=av)

> ftable(mytable) # print table

Reliability 1 2 3 4 5 6 7 8 9 10

Risk

1 0 0 16 7 0 8 8 0 0 0

2 804 149114 3670 57652 4 2084 85 11 345 82

3 2225 3 6668 22168 2 2151 156 7 260 79

4 2129 0 481 6447 0 404 43 2 58 24

5 432 0 55 700 1 103 5 1 20 11

6 19 0 2 60 0 8 0 0 1 0

7 3 0 0 5 0 0 0 0 2 0

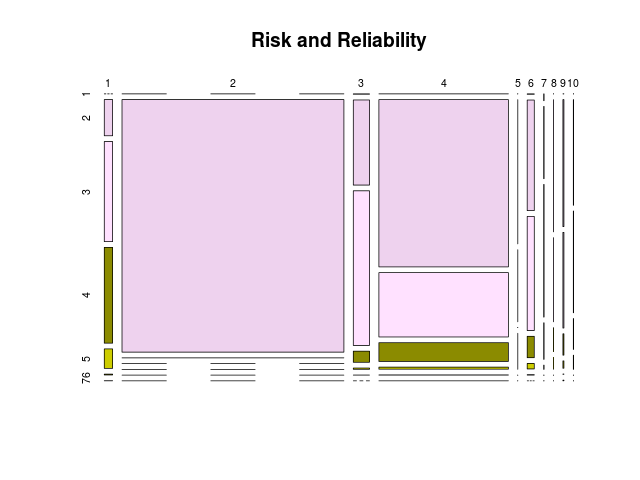
Or we can do:

mycolors <- c("yellow1","yellow2","yellow3","yellow4",

"thistle1","thistle2","thistle3")

cont.table <- table(av$Reliability, av$Risk)

plot(cont.table, col=mycolors, main="Risk and Reliability")



(didn’t look like this picture pasted at all?)

R code for filtering out lower risk/reliable elements

nrow(av[(av$Reliability>4) & (av$Risk>3),])

683

Python code for filtering out lower risk/reliable elements

len(av[(av['Reliability']>4) & (av['Risk']>3)])

683

The *R* and *Python* code are virtually identical as is the underlying functionality. The expressions test for the conditions across the entire column(s) and return a list of TRUE/FALSE values which are are then used to extract only the salient values—the ones that meet the condition of the expression—from the actual data frame.

There’s quite a bit of difference between 683 and 258,626. What are the characteristics of these 683 hosts that make them more risky? We can dig a bit deeper by looking at the *Type* of malicious activity these “truly risky” nodes are involved in by creating a new data frame from the subset we’ve identified and look at *Type*s we are left with:

R code for filtering out lower risk/reliable elements

# what makes these nodes "risky"?

risky <- av[(av$Reliability>4) & (av$Risk>4),]

risky.types <- summary(factor(risky$Type))

risky.types <- risky.types[order(-risky.types)]

risky.types

table(risky$Type)

C&C Malware IP

226 205

Spamming Malware Domain

150 37

Scanning Host Malicious Host

31 12

C&C;Malware Domain C&C;Malware IP

11 6

C&C;Scanning Host Malware IP;Malware Domain

3 2

Python code for filtering out lower risk/reliable elements

# what makes these nodes "risky"?

risky = av[(av['Reliability']>4) & (av['Risk']>3)]

print pd.value\_counts(risky['Type'])

C&C 226

Malware IP 205

Spamming 150

Malware Domain 37

Scanning Host 31

Malicious Host 12

C&C;Malware Domain 11

C&C;Malware IP 6

C&C;Scanning Host 3

Malware IP;Malware Domain 2

dtype: int64

This is helpful, but the *Type* field in the AlienVault data is a bit overloaded as there are often multiple values assigned in it. This makes sense since there’s nothing inherently limiting a node to one type of “badness”, but it is preventing us from easily getting a complete picture of each primary category of malicious activity. There are multiple ways to derive this information and one method is to allow for duplicate IP addresses (since it has been a “primary key” up until this point) in our this frame and split the *Type* field whenever it has multiple values, then add a new row to the data frame with the separated value, keeping the rest of the columns the same. Thankfully, *R* and *Python* make it easier to do than describe:

R code to “split” data based on Type field

# strsplit will split the Type column field into multiple

# values whenever it comes across a ";" and will return a

# list of the same length as the original column, but each

# entry will be a vector of "splits".

tmp <- strsplit(risky$Type,";")

# this will run the "length()" function on every element

# of our new vector and return it as a list

times <- sapply(tmp,length)

# we "re-make" the data frame by using cbind() to

# combine our individual columns, but with the added

# step of using rep() to expand each column but the

# size number of split elements on the Type field.

risky <- data.frame(cbind(IP=rep(risky$IP, times),

Reliability=rep(risky$Reliability, times),

Risk=rep(risky$Risk, times),

Type=unlist(tmp),

Locale=rep(risky$Locale, times),

Coords=rep(risky$Coords, times)))

nrow(risky)

risky.types <- summary(factor(risky$Type))

risky.types <- risky.types[order(-risky.types)]

risky.types

C&C Malware IP Spamming Malware Domain

246 213 150 50

Scanning Host Malicious Host

34 12

Python code to “split” data based on Type field

# make each entry an array

tmp = risky['Type'].str.split(';')

# expand the series

tmp = tmp.apply(lambda x: pd.Series(x)).unstack()

# join expanded series with original daa frame

r2 = risky.join(pd.DataFrame(s.reset\_index(level=0, drop=True)));

# remove the NaN's

risky = r2[pd.notnull(r2[0])]

C&C 246

Malware IP 213

Spamming 150

Malware Domain 50

Scanning Host 34

Malicious Host 12

dtype: int64

We can perform this analysis on the original list to show not only a reduction size has occurred, but a definitely change in what types of hosts seem to generate more risk. We’re only showing the *R* code here for brevity and since we’re not introducing any new code.

R code to compare malicious host Types

tmp <- strsplit(av$Type,";")

times <- sapply(tmp,length)

av.expanded <- data.frame(

cbind(IP=rep(av$IP, times),

Reliability=rep(av$Reliability, times),

Risk=rep(av$Risk, times),

Type=unlist(tmp),

Country=rep(av$Country, times),

Locale=rep(av$Locale, times),

Coords=rep(av$Coords, times),stringsAsFactors=FALSE),

stringsAsFactors=FALSE)

opar <- par()

# adjust plotting margins and setup 2x1 grid

par(mar=c(5.1,10,4.1,2.1),mfrow=c(2,1))

av.x.types <- summary(factor(av.expanded$Type))

av.x.types <- av.x.types[order(av.x.types)]

# draw the boxplot horizontally and adjust labels

bp <- barplot(av.x.types, horiz=TRUE, yaxt='n',

col=barcol, main="All Nodes Broken Down By Type")

axis(2, at=bp, labels=names(av.x.types), tick=FALSE, las=2)

risky.types <- summary(factor(risky$Type))

risky.types <- risky.types[order(risky.types)]

risky.types

bp2 <- barplot(risky.types, horiz=TRUE, yaxt='n',

col=barcol, main="Higher Risk Nodes Broken Down By Type")

axis(2, at=bp2, labels=names(risky.types), tick=FALSE, las=2)

par(opar)

Figure 3-2 Charts Comparing Makeup Of Malicious Hosts By Risk (*R*) [f0304.png]

We can also see how the filtering down by risk impacted which countries are now “on top”. Again, for brevity, we’re only showing the *Python* code this time:

Python code to examine new country breakdown

pd.value\_counts(risky.Country)

US 208

RU 57

DE 50

CN 30

TR 26

UA 25

IN 24

IT 16

NL 16

FR 15

GB 15

KR 14

AR 13

CA 13

PL 9

...

RS 1

...

The United States has now taken first place with China falling to fifth, and we can take a deeper look there to enumerate the top ten locales (in this case, cities) generating “higher risk” malicious traffic, noting that it is pure coincidence the birthplace of one of the book’s authors is in this list:

R code see the ten “most wanted” locales in the United States

us.locales <- summary(factor(risky[(risky$Country == "US"),]$Locale))

head(us.locales[order(-us.locales)],10)

Scottsdale Turlock Dallas Boulder San Antonio

26 15 13 12 11 8

Scranton Lansing Atlanta Chicago

8 7 6 6

Python code see the ten “most wanted” locales in the United States

us.locales = summary(factor(risky[(risky$Country == "US"),]$Locale))

Note the continued presence of a blank country and now a blank locale. These are nodes that could not be identified at all or identified precisely enough based upon their IP address. We’ll be covering more about various issues with IP address geo-location in the chapter on “Mapping Badness”.

There are additional questions we could ask of this standalone data, but you should have a good idea on how to perform more exploration on your own. Let’s bring in some additional data for one final exploration.

Augmenting 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 (i.e. “we aren’t special”) when it comes to imperfect data sets and highly distributed referential data or more metadata sources. This *can* pose challenges to effective data analyses, but it is usually possible to find and use the data you need.

Even though we have geographic information in our AlienVault data set, the internet has both physical and logical groupings, which we will cover more in the next chapter. It might be interesting to see how this data looks through a different lens, and for this example we’ll augment our data set with additional data from the IANA IPv4 Address Space Registry (https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml). This data is a very high level grouping of IPv4 address space registry allocations and most of the registrants are not responsible for the malicious activity of individual nodes. So, while we cannot use this information to cast blame, it will give us one view of where badness is clustered, enabling us to perform additional investigations which we’ll cover in Chapter 4.

type="tip"

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. 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 our companion web site 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 our analyses and your own projects will follow something close to this basic pattern:

* downloading (if necessary) of new data
* parsing and converting the new data into a data frame
* validating the contents and structure of the new data
* performing any necessary munging of the new data to make it easier to process/incorporate
* performing any necessary munging of the existing data to make it easier to incorporate the new data
* extracting or computing relevant information from the new data source
* creating one or more new columns in our existing data frame
* running new analyses

For this example, we process the IANA data to see which registry allocations have the most malicious nodes.

R code to incorporate 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)

}

iana <- read.csv(ianaData,stringsAsFactors=FALSE)

str(iana) # examine it(now shown below)

# clean up the iana prefix

iana$Prefix <- sub("^(00|0)","",iana$Prefix,perl=TRUE)

iana$Prefix <- sub("/8$","",iana$Prefix,perl=TRUE)

head(iana$Prefix) # not shown

# extract just the prefix from the AlienVault list

av.IP.prefix <- sapply(strsplit(av$IP,'.',fixed=TRUE),"[",1)

av$Designation <- sapply(av.IP.prefix,function(ip) {

iana[iana$Prefix == ip,]$Designation

})

desig <- summary(factor(av$Designation))

desig <- desig[order(-desig)]

desig

APNIC RIPE NCC

93775 74789

ARIN LACNIC

42358 18914

Administered by ARIN Administered by RIPE NCC

17974 5893

Administered by APNIC AFRINIC

2615 1896

Administered by AFRINIC Level 3 Communications, Inc.

322 31

PSINet, Inc. AT&T Bell Laboratories

30 24

Hewlett-Packard Company Digital Equipment Corporation

3 1

Python code to incorporate 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 not os.path.isfile(ianaData):

urllib.urlretrieve(ianaURL, filename=ianaData)

iana = pd.read\_csv(ianaData)

iana # examine it (not shown below)

# clean up the iana prefix

iana['Prefix'] = iana['Prefix'].map(lambda x:

str(int(x.rstrip("8").rstrip("/"))))

iana['Prefix'] # (not shown)

# extract just the prefix from the AlienVault list

avPrefix = [ octet[0] for octet in av['IP'].str.split('.') ]

av['Designation'] = [ iana[(iana['Prefix'] == prefix)].Designation

for prefix in avPrefix ]

pd.value\_counts(av['Designation'])

[APNIC] 93775

**[RIPE NCC] 74789**

[ARIN] 42358

**[LACNIC] 18914**

**[Administered by ARIN] 17974**

[Administered by RIPE NCC] 5893

[Administered by APNIC] 2615

[AFRINIC] 1896

[Administered by AFRINIC] 322

[Level 3 Communications, Inc.] 31

[PSINet, Inc.] 30

[AT&T Bell Laboratories] 24

[Hewlett-Packard Company] 3

[Digital Equipment Corporation] 1

dtype: int64

We can do a quick check against the main IANA allocation table to see if this matches overall block assignments:

R code (no Python example) to extract IANA block assignments

df <- data.frame(table(iana$Designation),stringsAsFactors=FALSE)

colnames(df) <- c("reg","ct")

av.reg <- df[df$reg %in% names(desig),]

av.reg[with(av.reg, order(-ct)),]

reg ct

8 APNIC 45

**3 Administered by ARIN 44**

10 ARIN 36

**39 RIPE NCC 35**

**33 LACNIC 9**

2 Administered by APNIC 6

5 Administered by RIPE NCC 4

6 AFRINIC 4

1 Administered by AFRINIC 2

34 Level 3 Communications, Inc. 2

12 AT&T Bell Laboratories 1

18 Digital Equipment Corporation 1

28 Hewlett-Packard Company 1

38 PSINet, Inc. 1

There is some variation, but overall—as expected—the larger blocks contribute the majority of malicious hosts. We’ve highlighted “RIPE NCC”, “Administered by ARIN” and “LACNIC” in the output since “RIPE NCC” has a significantly larger number of malicious hosts than it’s assignment count might imply (nearly double that of it’s very close neighbor “ARIN”) and “LACNIC” and “Administered by ARIN” both have a similar number of malicious hosts yet have very different allocation counts. Delving into why might be an interesting exercise (for the intrepid reader).

In Summary

This chapter introduced the core structure and concepts of data analyses in *Python* and *R*. We incorporated basic statistics, foundational scripting/analysis patterns and introductory visualizations to help us both ask and answer questions and also brought in additional data sets to produce additional views of our data. In addition, each example has demonstrated the similarity of *Python* (with *pandas*) and *R* coding techniques and generated output.

In future chapters we will focus mainly on *R* code, with some *Python* sprinkled in on occasion. If you are already a familiar with *Python* and/or *pandas* the previous examples should help you translate between the two languages. If you are new to both *R* and *Python* the standardization of future examples in one language should both help you follow along with less confusion and learn *R* a bit better.

Recommended Reading

*Statistics and Data with R: An applied approach through examples* by Yosef Cohen and Jeremiah Y. Cohen (John Wiley & Sons, Ltd. ISBN: 9780470758052)

*Python for Data Analysis* by Wes McKinney (O’Reilly Media, Inc. ISBN: 9798-1-4493-1979-3)