Global Information Assurance Certification Paper

Applying Data Analytics on Vulnerability Data

GIAC GCIH Gold Certification

Author: Yogesh Dhinwa, yogeshdhinwa@gmail.com
Advisor: Robert Vandenbrink
Accepted: December 21, 2015

Abstract

Organizations, by law, should exercise due care and due diligence in securing data at rest, in transit, and in use. Regardless of the whereabouts of data, an organization needs to thwart adversaries and secure its data properly. One of the key methods of thwarting external attackers is to lock down public-facing networks. To secure public-facing networks, a prudent organization often conducts vulnerability assessments. It may take a month or more for tens of thousands of IP addresses because of the time and effort required in collating and analyzing overwhelming vulnerability data. A common penetration testing proverb "Nine hours of fun and ninety hours of writing" accurately states the ratio of time between performing vulnerability scans and analyzing vulnerability data, which may be further extrapolated to estimate the number of hours required to analyze the vulnerability data of tens of thousands of hosts. To increase the fun aspect in assessment, we can utilize data analysis techniques and tools, which would eventually help save the time taken to analyze vulnerability data, and hence, produce effective reports quickly. Data analytics techniques using Splunk and Pandas can be leveraged to quickly and efficiently analyze network vulnerability reports from a scanner, for example Nessus. Data analytics tools and techniques help in reducing the time required to analyze vulnerability data as a part of vulnerability assessment.

1. Introduction

An organization with services spread across the globe depends on information technology and information systems. Adoption and compliance of information security standards have become mandatory for many organizations, especially those working under government regulations**. "**It is crucial to monitor for compliance in a manner as close to real time as possible to ensure that the organization does not drift out of compliance over time" (Gula, 2014). Organizations are legally obligated to secure their sensitive information by implementing various security controls and vulnerability management programs.

A vulnerability assessment can be conducted to identify, quantify and prioritize vulnerabilities in a system. Moreover, large organizations use automated vulnerability scanning tools to identify and report vulnerabilities in order to secure their public network. For the budget-conscious, Nessus is an appropriate vulnerability scanning tool because of its notable detection capabilities powered by the extensive plugin family. Nessus is capable of generating multi-format scan reports for the hosts being scanned, including a CSV file. The Nessus scan report captures a plethora of information about the target system, which include Common Vulnerability Exposure (CVE) identifiers, Common Vulnerability Scoring System (CVSS), Risk, and Vulnerability Name among other useful information in the report. The CVSS is a risk measurement system used for calculating and assessing the severity of system's vulnerabilities (Ou, Singhal, 2011).

Although analyzing a single host's vulnerability scan report is straightforward and doesn't require much time, it is difficult to analyze vulnerability scan reports of tens of thousands of hosts. We still don't have cost-effective and efficient techniques to analyze the overwhelming amount of vulnerability data generated by scanners. Fiscus (2014) mentioned that "The problem, thus, is one of data overload from any vulnerability scanner. Particularly when performing internal, credentialed scans against network resources, the amount of data generated can be overwhelming."

The Nessus vulnerability scan report is human-readable and can also be considered to contain machine-generated data; hence, an interesting approach would be to use Splunk for analyzing and visualizing vulnerability data. Splunk performs data

indexing, in which the collected data is broken down into events for easy search and analysis using the Splunk processing language (Zadrozny & Kodali, 2013). It also provides strong data correlation that enriches the data and fosters decision making.

The vulnerability scan data thus obtained can be mixed with other sets of data in order to add more value. To enrich the vulnerability scan data, we can use the vulnerability correlation DB (vFeed) to add more contextual information to vulnerability scan reports generated by Nessus. "vFeed.db is a detective and preventive security information repository used for gathering vulnerability and mitigation data from scattered internet sources into a unified database" (Ouchn, 2015). Since the Nessus vulnerability scan report contains limited information about the target, mixing it with vFeed data would provide further insight and different views of the vulnerability data.

Python is increasingly becoming popular for data analysis and can be used for vulnerability data analysis as well. A powerful data analytic library available in Python is Pandas. "The Python's Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures, and data analysis tools for the Python programming language" (Python Data Analysis, 2015). Pandas introduced two new advanced data structures known as Series and DataFrame to Python. These advanced data structures are very helpful in data analysis and provide a wide range of methods that can be performed on a data set. As per McKinney (2013), "The Pandas provides rich data structures and functions designed to make working with structured data fast, easy, and expressive."

The Nessus' network vulnerability scan report can be uploaded to Splunk or read into the Pandas DataFrame. With Splunk or Pandas, we can quickly explore, summarize and visualize the data. Data visualization provides a big picture of the available data. An analyst can examine the big picture view and look for areas of particular interests and patterns (Conti, 2007).

The vulnerability data analysis covered in this paper is for didactic purposes and would consist of two stages. During the first stage, we will analyze Nessus vulnerability scan reports alone and in the second stage, we will analyze vulnerability scan data in correlation with vFeed data. During both stages, we would explore how Pandas and Splunk are helpful and what we can achieve in a quick-and-efficient manner.

Yogesh Dhinwa, yogeshdhinwa@gmail.com

2. Understanding Vulnerability Data in the Nessus' CSV

Nessus supports the export or download of vulnerability reports in the CSV file format. A vulnerability scan report in CSV format has valuable information about the vulnerabilities detected in the target system. A Nessus vulnerability report in CSV format that has the following header fields as shown in Figure 1.

Plugin ID	CVE	CVSS	Risk	Host	Protocol	Port	Name	Synopsis	Description	Solution	See Also	Plugin Output

Figure 1. Nessus's CSV File Header in the Vulnerability Report

A Nessus vulnerability report presents structured data in a CSV file and contains data about the vulnerabilities detected during the scan. The most interesting field with respect to correlating the Nessus output to vFeed data is the CVE column in a scan report. Interestingly, some plugins like OS detection, traceroute, etc., which do not have a real port associated with them would return NULL (0) in the Port field. Since the port number is a numeric value, numeric 0 is used as a placeholder. One should not get confused when the port value is set to 0 in the Port field. In the Nessus vulnerability scan report, not all rows necessarily contain a CVE number or vulnerability, and it may also contain data with "None" as the risk rating.

3. Prepare Vulnerability Data for Analysis

In order to begin the data analysis, the vulnerability data has to be prepared by merging the Nessus scan reports and preparing the vFeed files to be used during analysis.

3.1. Merge Vulnerability Scan Data

There may be different vulnerabilities mentioned across several scan reports. All these scan reports should be merged into a CSV file, thus creating a bigger scan file. In Windows, the COPY command can be used to merge two or more CSV files into one CSV file, but it would add the header from each CSV file as row data into the final file. To refine the output file generated by the COPY command, remove the file headers inserted as rows from the final output file.

It is easy to avoid the occurrences of multiple file header fields that are added as rows, while combining the files in UNIX. The following UNIX shell commands merge all the available CSV files available into a directory and save the data in a new CSV file.

```
~/merging_vulnerability_data> head -1 vul_report1.csv > vul_data_10.csv
~/merging_vulnerability_data> tail –n +2 *.csv >> vul_data_10.csv
```

The above shell commands prevent the CSV header or the first record from each file from being repetitively added as rows in the resulting file. They simply merge all the CSV files in the directory into a single file named vul_data10.csv.

Many organizations maintain different types of repositories, e.g. Asset Register, IP Allocation Register (IPs to office locations or datacenters), Application Register, etc. For our analysis, we will use a repository that captures the IP address and the associated department in a CSV file called the Host-to-Department file. The Host-to-Department file contains information about IP addresses and the department that they belong. Besides, the Host-to-Department mapping file (i.e. Host-to-Dept.csv) can also be used to derive security indicators specific to departments like the most and the least secure department. A few lines of the content of Host-to-Dept.csv is available in Appendix A.

3.2. Prepare vFeed Data

The vFeed license allows the non-commercial use of the vFeed database and it is downloadable as an SQLite DB file. At the time of writing this paper, there were 39 tables that store additional useful information about CVEs – CVEs to various exploits and testing scripts like Metasploit (MSF), NMAP (i.e. a port scanner), exploit-db (the exploit database repository), and others; CVEs to vendor patches like SUSE (an enterprise Linux flavor) and Microsoft (MS); CVEs to Common Weakness Enumeration (CWE) numbers mapping, etc. It amasses a lot of extra information about CVEs in one place. The information in the vulnerability report is useful, but not complete from many perspectives. An analyst or consultant, while analyzing the vulnerability report would have to frequently check whether a given CVE is exploitable or not, what vendor patches are available, a reference to other vulnerability databases, etc. The analyst would have to manually figure out other relevant information about the vulnerabilities as the Nessus report has limited information about them. Using vFeed would shed more light on the vulnerabilities identified during a vulnerability scan.

3.2.1. Convert vfeed Tables into CSV Files

The vFeed DB can be used as such or can be converted into various CSV files before use. These converted CSV files can be used as a lookup table or a reference table in Splunk or Pandas. A Python script named vfeed2csv.py has been written to convert all the tables in the vFeed DB to the corresponding CSV files. Information regarding the vFeed DB download and conversion of vFeed tables into their respective CSV files is available in Appendix B. The Python script vfeed2csv.py is also available in Appendix B.

3.2.2. Categorize and Group vFeed CSV Files

During aggregation, the resulting CSVs can be combined into one CSV based on their type and purpose. Thirty-nine CSV files have been converted from vFeed tables and these CSV files can be grouped together based on their usability and application in data correlation. For example, there are files that map CVEs to various vendors' patches, which can be placed into one directory. Based on the requirement, all the CSVs have been grouped into eight categories. Details about these eight categories are available in Appendix C.

3.2.3. Merge vFeed Files within Categories

As seen earlier, each category contains certain files that have additional useful information about CVEs. All the files have CVEID as a common column name; hence, files in that directory or category can be merged with CVEID values. The Pandas' Outer Join can merge two files based on a common key "CVEID" without losing information, which is similar to Full Outer Join in SQL; however, it is a CPU resource-intensive job. For such big datasets, an outer join is expensive in terms of computing resources and power, so among other alternatives, the Pandas' Append operation is a merging method that is often better suited. All files can be appended into a single file, which can work as a lookup table during the vulnerability data analysis. CVEID can thus be used to correlate records from the vFeed combined files and scan reports.

For example, files in the vendor_security_patch directory map CVEs to their respective security patches from affected vendors. All these files can be combined into a single file, containing all the CVEs and security patch information from all affected

vendors in one place. It would reduce the time taken to locate information about a CVE in multiple files. As shown below, Pandas reads CSV files, appends them one after another, and then saves the output in the final CSV. All files in the directory and all column names in the output CSV named combined_vendor_security_patch.csv are shown below.

```
>>> import os
>>> import pandas as pd
>>> lst_dir = [ file for file in os.listdir('.') if
os.path.isfile(file) ]
>>> lst_dir
['map_cve_mandriva.csv', 'map_cve_suse.csv', 'map_cve_redhat.csv',
'map_cve_debian.csv', 'map_cve_mskb.csv', 'map_cve_ubuntu.csv',
'map_cve_vmware.csv', 'map_redhat_bugzilla.csv', 'map_cve_gentoo.csv',
'map_cve_cisco.csv', 'map_cve_fedora.csv', 'map_cve_ms.csv',
'map_cve_aixapar.csv', 'map_cve_hp.csv']
>>> vendor_patch_data = pd.read_csv(lst_dir[0])
>>> for file in lst_dir[1:]:
...     df = pd.read_csv(file)
...     vendor_patch_data =
vendor_patch_data.append(df,ignore_index=True)
...
>>> list(vendor_patch_data.columns.values)
['advisory_dateissue', 'aixaparid', 'bugzillaid', 'bugzillatitle',
'ciscoid', 'cveid', 'debianid', 'fedoraid', 'gentooid', 'hpid',
'hplink', 'mandrivaid', 'msid', 'mskbid', 'mskbtitle', 'mstitle',
'redhatid', 'redhatovalid', 'redhatupdatedesc', 'suseid', 'ubuntuid',
'vmwareid']
>>>
>>> vendor_patch_data.to_csv('combined_vendor_security_patch.csv',
index=False)
```

The above Python code needs to be run in all the eight directories, which would then combine all the files in each directory into a single file. Pandas has a built-in function to read and write a CSV file; and also keep and drop indexes.

4. Vulnerability Data Analysis with Python Pandas

Python can be used to analyze vulnerability data using its powerful data analytics library named Pandas. Python Pandas can ingest and index various file types into the memory and provide a powerful way to search, filter, group, and apply methods on the data.

4.1. Pandas for Vulnerability Data Analysis

A large volume of vulnerability data, which is a result of combining different Nessus scan reports, would be loaded into the Pandas DataFrame. For didactic purposes, we will try to obtain the answers to the most common questions asked during the vulnerability data analysis by using Pandas. The Python's interactive shell is used to execute the Python code and show results.

4.1.1. The Number of Scanned IP Addresses in the Scan Report

Pandas and NumPy packages are imported into the Python's shell and the read_csv method is used to load the vulnerability file into Pandas' DataFrame named df. The Pandas' info() method is applied to the df and the results are shown below.

```
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.read_csv('vul_data_10.csv')
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 121610 entries, 0 to 121609
Data columns (total 13 columns):
Plugin ID        121610 non-null object
CVE              17774 non-null object
CVSS             19948 non-null float64
Risk             121609 non-null object
Host             121528 non-null object
Protocol         121609 non-null object
Port             121609 non-null float64
Name             121609 non-null object
Synopsis         121609 non-null object
Description      121609 non-null object
Solution         121242 non-null object
See Also         24662 non-null object
Plugin Output    115633 non-null object
dtypes: float64(2), object(11)
```

All the columns in the DataFrame df will not be used during the vulnerability data analysis, hence only the required columns are copied into another DataFrame named df1. The new DataFrame named df1, which contains only selected columns and the data slice, is shown below.

```
>>> df1 = df[['Plugin ID',
'CVE','CVSS','Risk','Host','Protocol','Port','Name']]
>>> df1[99:100]
   Plugin ID  CVE  CVSS  Risk          Host Protocol  Port  \
99     70544  NaN   NaN  None  10.92.22.112      tcp   443

                                          Name
99  SSL Cipher Block Chaining Cipher Suites Supported
```

With Pandas, it is very easy to obtain the total number of IP addresses using the Pandas' describe command. The unique field in the output shows the number of IP addresses scanned in the vulnerability scan report.

```
>>> df1['Host'].describe()
count           121528
unique           11243
top       10.92.22.139
freq              3084
dtype: object]
```

4.1.2. The Number of Live IP Addresses

In most cases, a remote host is considered to be live if that host listens to at least one TCP/UDP port or replies to any ICMP query. To minimize the chances of missing live hosts, DataFrame df1 can be filtered for both open ports, where the port number is not zero, as well as for the hosts that send replies for ICMP requests. In the results below, the unique field shows the count of unique live IP addresses as per the scanning results.

```
>>> print df1[(df1.Port != 0) | ( df1.Protocol ==
'icmp')]['Host'].describe()
count            85900
unique            1454
top       10.92.22.139
freq              3075
dtype: object
```

4.1.3. The Top 10 CVEs by Occurrence

Pandas provides a powerful method of grouping fields together in order to obtain certain interesting statistics. The Pandas GroupBy method is similar to SQL's GroupBy operation. The Pandas command given below shows the most occurring top 10 CVE values in the vulnerability scan report.

```
>>> df1.groupby('CVE').size().order(ascending = False)[:10]
```

4.1.4. The Top 10 CVEs by the Highest Mean CVSS Score

The previous analysis of the top 10 CVEs by occurrence doesn't rely on the CVSS score of the vulnerabilities, but depends only on the frequency of their occurrence. To create another view of vulnerability data, analyze the CVEs based on their CVSS

scores to find out the top 10 High Risk CVEs affecting the organization. The query below shows the usage of the NumPy package to calculate the size and average CVSS score for CVEs in the report. The vulnerability data is first grouped by CVE values, after which the NumPy's size and mean methods are applied on the grouped data. Since there are many CVEs that have a CVSS score of 10, it would be a good idea to select CVEs that have the most occurrence in the report as well as the highest CVSS score. In this case, after CVEs with more than 60 events or occurrences have been selected, the results are sorted based on the mean CVSS score to find the top 10 CVEs based on their CVSS scores. The top first three rows are shown in the results.

```
>>> CVEs = df1.groupby('CVE').agg({'CVSS': [ np.size, np.mean ]})
>>> CVEs_size_gt_60 = CVEs['CVSS'].size >= 60
>>> CVEs[CVEs_size_gt_60].sort([('CVSS','mean')], ascending =
False)[:10]
               CVSS
               size  mean
CVE
CVE-2011-1471    75    10
CVE-2012-0135    75    10
CVE-2010-2791    75    10
```

4.1.5. Top 10 Most Vulnerable IP Addresses

Similarly, Python's Pandas can be used to find the top 10 most vulnerable hosts. There are many ways to search for the most vulnerable hosts based on one's requirement; for example, the top 10 hosts based on the average CVSS score. The Pandas' GroupBy method is applied on the Host column with the minimum number of events (size) set to 8. The results are then sorted by the average CVSS score to obtain the top 10 most vulnerable hosts. The Pandas operations and the first three rows are shown below.

```
>>> Hosts = df1.groupby('Host').agg({'CVSS': [ np.size, np.mean]})
>>> Hosts_size_gt_8 =  Hosts['CVSS'].size > 8
>>> Hosts[Hosts_size_gt_8].sort([('CVSS','mean')], ascending =
False)[:10]
                CVSS
                size       mean
Host
10.92.22.211      32  10.000000
10.92.22.70       17  10.000000
10.15.126.7      236   9.680769
```

4.1.6. The Number of Affected Hosts and Vulnerabilities per Department

It is easy to obtain the number of Critical or High Risk vulnerabilities or hosts affected by those vulnerabilities. To make the analysis a bit interesting, we will use the Host-to-Department mapping file for data correlation in order to obtain the number of hosts affected in each department. In the Nessus report, some of the fields will have null values, e.g. the CVE and CVSS fields. Pandas provides the fillna method to handle the null value, which fills the null value with a value of choice such as nocve.

```
>>> df1['CVE'].fillna('nocve', inplace = True)
>>> df2 = df1[( df1.Risk == 'Critical') | ( df1.Risk == 'High') | (
df1.Risk == 'Medium')]
>>> ip_dept = pd.read_csv('Network_Address_Book.csv')
>>> ip_dept.columns = ['Host','Dept']
```

To initiate a mapping between the affected hosts and their departments, first load the IP-to-Dept file into the Pandas DataFrame and rename the column for consistency purposes. As seen before, Network_Address_Book.csv contains the mapping of IP addresses to departments and this file can be used to correlate the affected hosts to the departments in the organization.

The Pandas Left Join can be used to merge the df2 and ip_dept DataFrame. The ip_dept DataFrame would be used as a lookup table for the DataFrame df2. The Host field is used as a key to perform the left join operation. The resultant DataFrame from the join operation is saved in a new DataFrame named df3. The GroupBy operation is then performed on the Dept field along with other operations to obtain the number of affected hosts from each department. As per the results shown below, the nunique column shows the number of affected hosts for each department.

```
>>> df3 = pd.merge(df2, ip_dept, on='Host', how='left')
>>> df4 = df3.groupby('Dept').agg({'Host': ( np.size,
pd.Series.nunique)})
>>> df4.sort([('Host', 'nunique')], ascending = False)
         Host
         size  nunique
Dept
m-site  10684      569
t-site   3785      300
y-site   1525       73
e-site    942       19
b-site   1578       16
```

The number of Critical, High and Medium Risk vulnerabilities affecting each department can also be derived as shown below. We can make a copy of df3 into the DataFrame dfx. There may be a few rows having duplicate values for fields like Plugin ID, Host, CVE, Port, etc. Such rows can be deleted using the Pandas' drop_duplicates method to retain only unique rows – a data reduction combination, which is similar to the dedup command in Splunk. The resultant DataFrame can be grouped by Dept and Host, saved in the DataFrame dfplt, and then the Risk items in each department are counted.

```
>>> dfx = df3.copy()
>>> dfx.drop_duplicates(['Plugin ID', 'Host','CVE','Port'], take_last =
True, inplace = True)
>>> dfplt = dfx.groupby(['Dept','Risk']).Risk.value_counts().unstack(0)
>>> dfplt
Dept               b-site  e-site  m-site  t-site  y-site
Risk
Critical Critical     690     500    1430     597     653
High     High         510     353    2612    1156     498
Medium   Medium       204      89    5713    1865     261
```

4.1.7. The Most Vulnerable Department Based on the Mean CVSS Score

With Pandas, the average CVSS score for a department can be calculated and based on that score, the most and least vulnerable department can also be deduced – the query is shown below. A high mean value reflects the existence of more High Risk vulnerabilities.

```
>>> df5 = df3.groupby('Dept').agg({'CVSS': ( np.sum, np.mean) })
>>> df5.sort([('CVSS', 'sum')], ascending = False)
           CVSS
            sum      mean
Dept
m-site  70332.3  6.582956
t-site  26450.8  6.988322
b-site  14075.6  8.919899
y-site  13174.1  8.638754
e-site   8808.7  9.351062
```

4.2. Analyze Vulnerabilities in Correlation with vFeed

As seen in the vFeed section, we have created files based on common characteristics that can be used to correlate the data present in the scan reports. Pandas provides a way to join tables, which would come handy during such scenarios.

4.2.1. Map Vulnerabilities to Exploits or Testing Scripts

The Pandas merge operation and the vFeed file as a lookup table can help generate required information. Pandas can also save the results in a CSV file. To map CVEs to exploits, DataFrame df3 can again be used as it contains the details for Critical, High and Medium Risk vulnerabilities. As shown in the query below, first load the combined scripts and tools file to the DataFrame df_vtools. In the DataFrame df_vtools, the column name cveid is renamed to CVE, so that the left join on both the DataFrames can be performed using CVE as a key. The resultant DataFrame from the left join is named dfy. It can be further trimmed to only Metasploit Modules and saved in the DataFrame dfm. The DataFrame dfy also contains data for other exploit providers. It is also possible that all the vulnerabilities in the DataFrame dfm might not have an exploit in the Metasploit; the rows with empty fields can be dropped as a part of the DataFrame size reduction. The DataFrame dfm can be grouped on Dept and Risk columns using the Pandas GroupBy method. On the GroupBy object, we can apply the value_counts method that would yield the number of risk items in the scan report, which can be exploited or tested using Metasploit modules. There can be more than one Metasploit module to exploit or test a CVE that eventually would increase the count of exploitable vulnerabilities in the results.

```
>>> df_vtools = pd.read_csv('combined_cve_va_scripts_tools.csv')
>>> df_vtools.rename(columns={'cveid':'CVE'}, inplace = True)
>>> dfy = pd.merge( df3, df_vtools, on='CVE', how='left')
>>> dfm =
dfy[['Host','CVE','Risk','Port','Dept','msf_script_name','msfid']]
>>> dfm.dropna(inplace = True)
>>> dfm.to_csv('vuls_having_metasploit_modules')
>>> dfm.groupby(['Dept','Risk']).Risk.value_counts().unstack(0)
```

4.3. Pandas to Visualize Vulnerability Data

The data in the Pandas' DataFrame can also be visualized using the matplotlib package, e.g. visualizing the number of vulnerable hosts in each department. The DataFrame df3 can be used to obtain the number of unique vulnerable hosts. The methods are shown below and the chart is drawn in Figure 2.

```
>>> import matplotlib.pyplot as plt
>>> df4 = df3.groupby('Dept').agg({'Host': ( pd.Series.nunique)})
>>> df4.plot(kind="pie", subplots= True)
<matplotlib.axes.AxesSubplot object at 0x10aeb5650>
```

```
>>> plt.show()
```

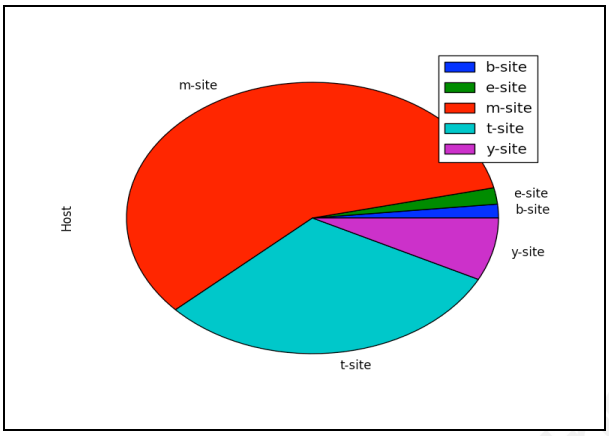

Figure 2. The Number of Unique Vulnerable Hosts in Each Department

As seen before, the DataFrame dfplt contains information about the number of Critical, High, and Medium Risk vulnerabilities affecting each department, which can be drawn in a bar chart as shown in Figure 3.

```
>>> dfplt.plot(kind="bar")
<matplotlib.axes.AxesSubplot object at 0x117475690>
>>> plt.show()
```

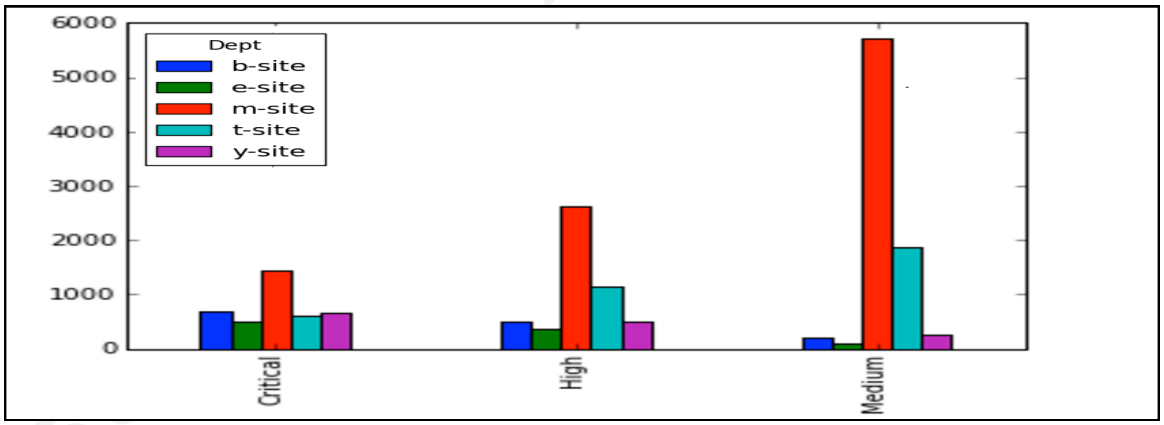

Figure 3. The Number of Vulnerabilities in Each Department

5. Vulnerability Data Analysis with Splunk

The Splunk Enterprise can be installed from the Splunk website as either a free or paid version. The free version is quite suitable for vulnerability data analysis. Moreover, Splunk converts its free license into a perpetual free license when the initial free version expires, which allows the continued usage of Splunk's powerful indexing and searching feature forever for analyzing the data.

Yogesh Dhinwa, yogeshdhinwa@gmail.com

5.1. Uploading Data to Splunk

The combined CSV file can be uploaded to Splunk for analysis. Although multiple files can also be uploaded, it is a good idea to combine and upload one single file in order to save time. Also, the Splunk Enterprise provides a nice GUI to upload, index, search, analyze, and visualize data. Adding data to the Splunk is straightforward – click Add Data icon at the Home screen, click Upload, select the file to be uploaded, e.g. combined vulnerability scan report, set the source type to CSV, set options for input settings, review the options and start searching. All steps to upload a file to the Splunk are shown in Figure 4. In our analysis, the timestamp is set to the current time and indexing relies on the default Splunk index. The guidance and instructions regarding the Splunk installation, data upload, etc. can be found on its site.

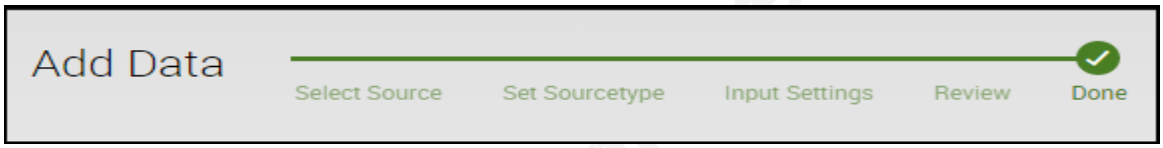

Figure 4. Steps to Add Data in Splunk

5.2. Vulnerability Data Analysis

Splunk provides multiple views of data, e.g. tables, raw data, graphs, and charts. As an introduction to vulnerability data analysis using Splunk, a few interesting analyses would be performed on the vulnerability data present in the combined scan report.

5.2.1. The Number of Unique IP Addresses in the Scan Report

The Splunk query shown below can be used to obtain the unique count of IP addresses in the vulnerability report. Splunk indexed Host field data as extracted_Host; hence, the extracted_Host field would be used to extract IP addresses in the Splunk query.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" | stats
dc(extracted_Host) as "Total Host"
```

5.2.2. The Number of Scanned IP Addresses for Each Department

The IP address to Department mapping file, i.e. Network_Address_Book.csv can be used to obtain the number of IP addresses scanned for each department in the scan report. Firstly, create a lookup table using the IP address to Department file, which can

later find the number of IP addresses from each department in the scan report. Splunk's outputlookup command can be used to create a lookup table as shown below. This lookup table has two columns, namely extracted_Host and Dept. The IP_Address field is renamed to the extracted_Host field in order to maintain consistency with the Nessus scan data.

```
source="Network_Address_Book.csv" host="ip_to_dept" sourcetype="csv" |
table IP_Address Dept | rename IP_Address as extracted_Host |
outputlookup ip2dept.csv
```

The Splunk query given below finds the number of unique IP addresses in the vulnerability scan report, i.e. vul_data_10.csv, from each department. This query also uses the lookup table – ip2dept.csv, which would map IP addresses from the scan report to the respective department.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" | lookup
ip2dept.csv extracted_Host | stats dc(extracted_Host) as Hosts by Dept
```

The results of the above Splunk query are shown in Figure 5. It shows the number of unique IP addresses from each department in the scan report.

Dept	Hosts
b-site	1895
e-site	3943
m-site	2867
t-site	1070
y-site	1468

Figure 5. The Number of Scanned IP Addresses for Each Department

The IP address distribution can also be visualized using the Splunk visualization feature. The results obtained in Figure 5 can be quickly fit into a pie chart, as shown in Figure 6, to see the allocations of the IP addresses per department, which would quickly show the biggest and smallest department based on the IP address allocation.

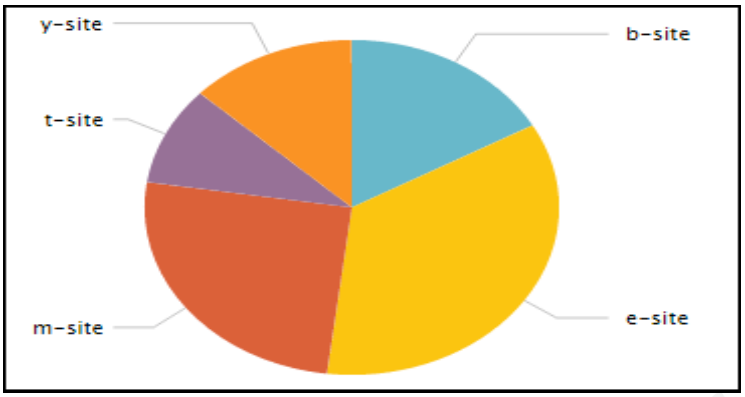

Figure 6. IP Address Allocation to the Departments in the Pie Chart

5.2.3. The Average Vulnerability CVSS Score

The Splunk query below can be used to calculate the average CVSS score for the vulnerabilities identified during the scans.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" | stats
mean(CVSS) as mean_cvss | eval mean_cvss = round(mean_cvss,2)
```

The average CVSS score result is shown in Figure 7 as a radial gauge where the color ranges have been adjusted between 0 and 10 for scaling purposes. A few manual selections need to be performed in Splunk before creating the visualization.

Figure 7. The Average CVSS Score

5.2.4. The Most and Least Vulnerable Departments

The sum of CVSS scores for all the vulnerabilities in the scan report can be used to see the most and least vulnerable department as per the combined scan results. The Splunk stats operation can sum CVSS scores for all the vulnerabilities across the departments in the organization. The results can then be sorted to find the most and least vulnerable department. The Splunk query for it is shown below.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" | lookup
ip2dept.csv extracted_Host | stats sum(CVSS) as SUM_CVSS by Dept | sort
- SUM_CVSS | head 1
```

The above query would return a single value, thus enabling, Splunk's visualization feature for a single value to display results in nicely formatted font. Similarly, instead of the head command, the tail command can be used to display the least vulnerable department in the organization.

5.2.5. The Number of Live IP Addresses for Each Department

With Splunk, the number of live IP addresses for each department as per scan results across the organization can be found and analyzed. Vulnerability data uploaded onto Splunk can be enquired to show IP addresses having at least one TCP or UDP port. It is also possible that there can be hosts that do not have a TCP or UDP port, but only reply to ICMP requests. In order to obtain maximum live hosts, first make a list of all the IP addresses running at least one TCP/UDP port, and then search for hosts that reply to certain types of ICMP requests, but have no open ports. The following Splunk query searches for IP addresses having at least one open port and saves the results in a lookup table.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" Risk="*"
Port!="0" | dedup extracted_Host | lookup ip2dept.csv extracted_Host |
table extracted_Host Dept | outputlookup hosts_with_open_ports.csv
```

Now, search for IP addresses that respond only to the ICMP queries and are not in the hosts_with_open_ports.csv lookup table. Splunk's dedup can delete duplicate extracted_Host values. The result of the following query is stored in another lookup table.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" Risk="*"
Protocol="icmp" NOT [| inputlookup hosts_with_open_ports.csv | fields +
extracted_Host ] | dedup extracted_Host | lookup ip2dept.csv
extracted_Host | table extracted_Host  Dept | outputlookup
hosts_ICMP.csv
```

The above two lookup tables can be used on their own to detect IP addresses with at least an open port or with only the ICMP. These two files can also be combined into one single file to obtain the total number of live IP addresses from each department in the scan report.

The query below joins two lookup tables and generates a new lookup table.

```
| inputlookup hosts_with_open_ports.csv | append [| inputlookup
hosts_ICMP.csv] | outputlookup live_ip.csv
```

The live_ip lookup can map the number of live IP addresses from the departments as depicted below.

```
source="vul_data_10.csv" host="vul_data" sourcetype="csv" Risk="*" |
dedup extracted_Host | lookup live_ip.csv extracted_Host | chart
count(extracted_Host) by Dept
```

5.2.6. The Number of Affected Hosts per Department

Now, let's see how many IP addresses in each department are affected with Critical, High, and Medium Risk vulnerabilities. This can be achieved by using the Splunk query given below.

```
source="vul_data_10.csv" host="vul_data" extracted_Host="*"
sourcetype="csv" Risk="Critical" OR Risk="High" OR Risk="Medium"
Port!="0"  | fillnull value=nocve CVE | dedup extracted_Host CVE Port |
lookup ip2dept.csv extracted_Host | chart count over Dept by Risk
```

In the above query, Low and Informational Risk-rated vulnerabilities have been excluded and rows having the same IP address, CVE, and Port number have also been deleted using the dedup command in order to keep only unique records. There can be different field name combinations in the data reduction query using the dedup command, e.g. delete rows with the same PluginID, CVE, and Port or CVE, Name, and Port. Each data reduction query with different combinations would produce different results. As such, it should be used based on requirement. In the sub-search, the IP addresses are looked up for their respective departments and then using the chart command, the number of affected hosts are counted for each department. Moreover, there are vulnerabilities that don't have a CVE number associated with them and instead have a null value in the report. The fillnull command fills vulnerabilities that have no CVE number with a user-supplied value like nocve in this case. The result of the above query is shown in a bar chart, which is drawn using Splunk's visualization tool, as shown in Figure 8 below.

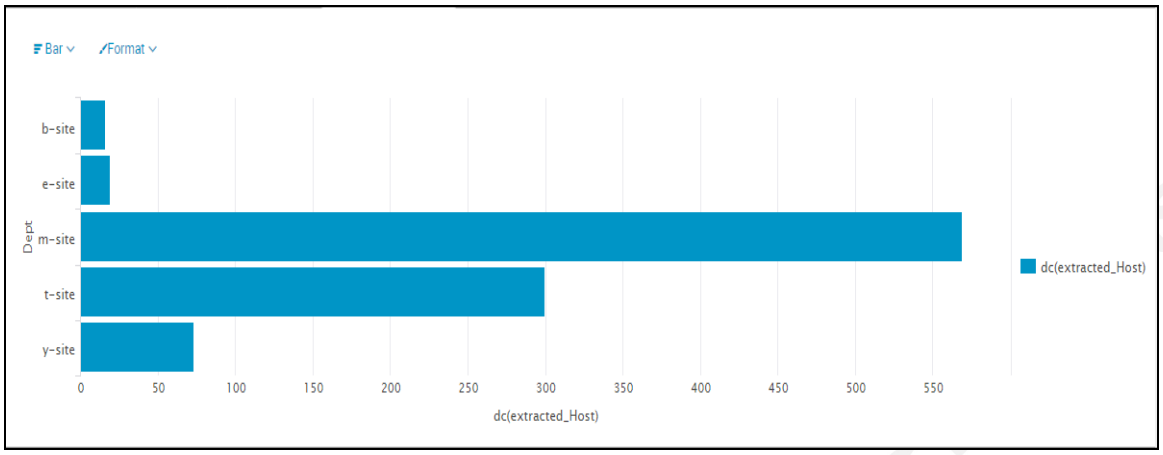

Figure 8. Count of Affected Hosts for Each Department

In a Splunk search, vulnerabilities with no CVE number would be difficult to search, and at times, different vulnerabilities with no CVE value would be deleted by the dedup. Null CVE values for two different vulnerabilities would be considered to be duplicate on the same Host and Port in the dedup combination as seen in the query above.

5.2.7. The Top 10 Vulnerabilities and Hosts

There are certain vulnerabilities that affect the organization the most and knowing them can expedite the vulnerability management process. Therefore, focusing on the top 10 vulnerabilities would drastically reduce the risk of the organization. In a Nessus scan report, the Name field contains the name of a vulnerability that may be the same for many CVEs. It would thus be a good idea to find the top 10 vulnerabilities by Name. The following query shows the top 10 vulnerabilities by Name. The results are shown in Figure 9.

```
source="vul_data_10.csv" host="vul_data" extracted_Host="*"
sourcetype="csv" Risk="Critical" OR Risk="High" OR Risk="Medium"
Port!="0"  | dedup extracted_Host CVE | chart count over Name by Risk |
addtotals fieldname=total Critical,High,Medium | sort - total limit=10
| fields - total Risk None Low
```

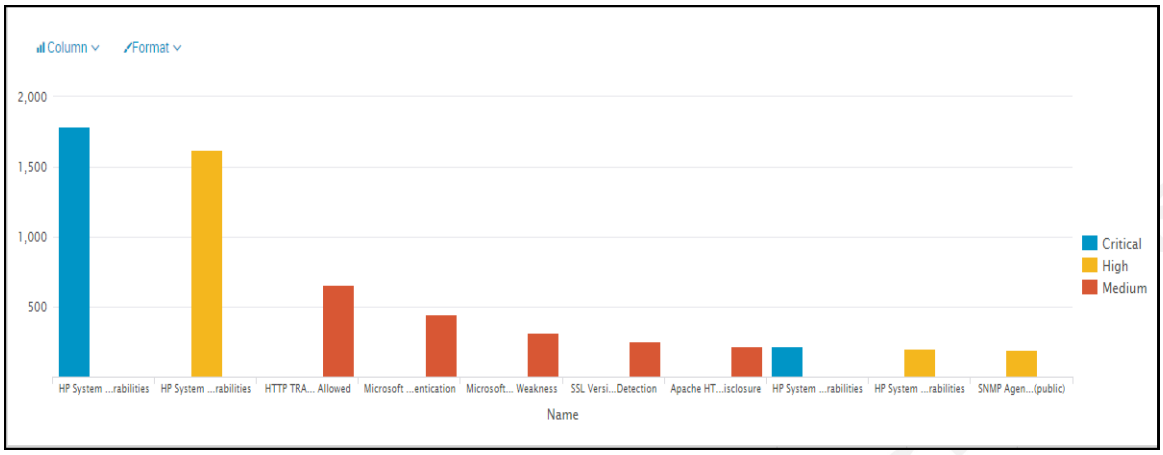

Figure 9. The Top 10 Vulnerabilities Detected During the Scan

The Splunk query below finds out the top 10 most vulnerable hosts by risk across the organization. The output of this query would comprise the hosts with the most number of vulnerabilities associated with them.

```
source="vul_data_10.csv" host="vul_data" extracted_Host="*"
sourcetype="csv" Risk="*" Port!="0"  | fillnull value=nocve CVE | dedup
extracted_Host CVE Port | chart count over extracted_Host by Risk |
addtotals fieldname=total Critical,High,Medium | sort - total limit=10
| fields - total Risk None Low
```

5.3. Analyze Vulnerabilities in Correlation with vFeed

We can leverage Splunk to correlate data using vFeed files. Using Splunk, vfeed data would be mixed with Nessus vulnerability data to perform a few interesting analyses.

5.3.1. The Most Vulnerable Application and Operating System Vendors

This information cannot be directly obtained from the Nessus vulnerability data. We will have to correlate information from the vulnerability data with the vFeed files. For this analysis, the CVE to CPE mapping file comes handy. Before we can use the CVE to CPE mapping file, we should perform certain operations on it to make it fit for analysis. vFeed's CVE-to-CPE mapping file will be uploaded onto Splunk and then the CPEID will be split at the delimiter ":" into new fields, namely type, vendor, and product. These new fields, which are derived by splitting the CPEID field, can be saved in a lookup table for future use, as shown below:

```
source="cve_cpe.csv" host="cve_to_cpe" sourcetype="csv" | eval temp =
split(cpeid,":") | eval type = mvindex(temp,1)  | eval vendor =
mvindex(temp,2) | eval product = mvindex(temp,3) | table cveid type
vendor product  | rename cveid as CVE | dedup CVE vendor type product |
outputlookup cve2cpe.csv
```

Query the vulnerability data and correlate it with the cve2cpe lookup table for the most affected vulnerable applications and operating systems. Based on the scanned report, it wouldn't have been easy to obtain this information. However, with data correlation, it is easily obtainable, e.g. the top 5 most vulnerable application vendors in the organization are shown in Figure 10. As a side note, the certainty and reliability of results depend upon the quality of the scan report and the logical mapping of CVEID to CPEID. There are CVEs that affect many different platforms, wherein eventually one CVE may be mapped to many CPEIDs; hence, the results should be cross-checked with other analyses. It might happen that CVEs in the scan report may refer to CPEIDs that do not exist in our environments, in which case, a lookup table with contextual data may resolve the problem. The Splunk query shown below would produce the top five most vulnerable application vendors in the organization.

```
source="vul_data_10.csv" host="vul_data" extracted_Host="*"
sourcetype="csv" Risk="Critical" OR Risk="High" OR Risk="Medium" |
dedup extracted_Host CVE Port | where isnotnull(CVE) | lookup
cve2cpe.csv CVE | where type = "/a" OR type !="/o" OR type !="/h" |
chart count over vendor by Risk | addtotals fieldname=total
Critical,High,Medium | sort - total limit=5 | fields - total Risk None
Low
```

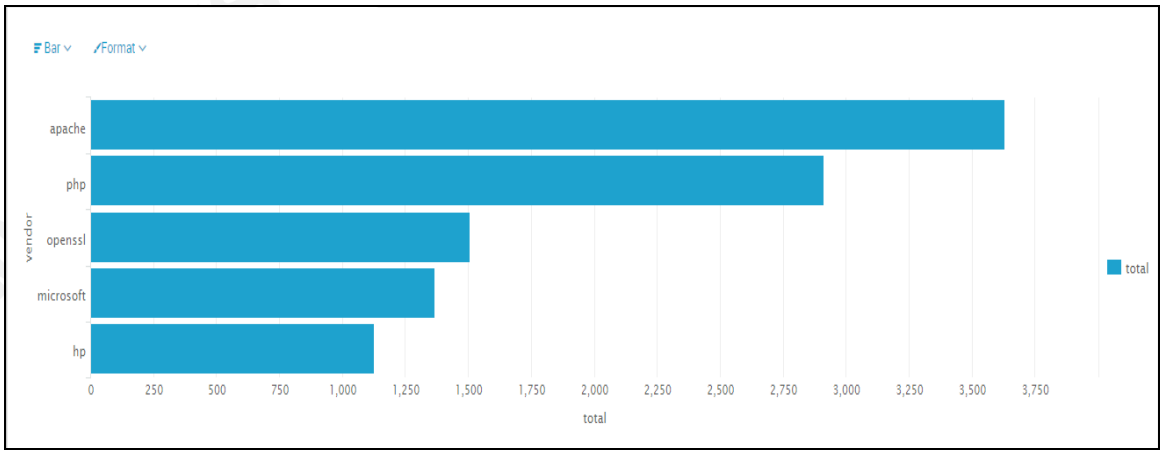

Figure 10. The Top Five Most Vulnerable Application Vendors

Similarly, another Splunk query can obtain results for the top five most vulnerable operating system vendors in the organization. The only change in this query would be to

replace the type field from Application to Operating System (OS). The type field "/a" denotes Application whereas "/o" denotes OS in a CPEID.

5.3.2. Map Vulnerabilities to Exploits or Testing Scripts

Nessus is a pretty reliable vulnerability scanner and it can detect vulnerabilities affecting a target system. Sometimes, there are requirements to confirm or exploit the discovered vulnerabilities with other tools and methods. It is a cumbersome job to find out which NMAP script, Metasploit module or exploit-db code, etc. can be used to test a vulnerability again, especially when there are thousands of unique vulnerabilities. In such scenarios, vFeed can help as there are several useful files in vFeed that map a CVE to an NMAP-script, Metasploit module or many other exploit kits. In the Data Preparation phase, such files have been merged into a single file and kept in the vulnerability_assessment_tools directory.

As in the previous analysis, a lookup table with all the required fields can be created to correlate data in the scan report. This lookup table can be used to analyze data from the scan report. The Splunk query, as shown below, uses the lookup table to list the available exploits for CVEs in the scan report. The results of the query are shown in Figure 11.

```
source="vul_data_10.csv" host="vul_data" extracted_Host="*" Risk="*"  |
dedup extracted_Host CVE Port | lookup cve2exploit.csv CVE | table
extracted_Host CVE Port Name nmap_script_id exploitdbscript
msf_script_file d2_script_name saintexploitid | where
isnotnull(nmap_script_id) OR isnotnull(exploitdbscript) OR
isnotnull(msf_script_file) OR isnotnull(d2_script_name) OR
isnotnull(saintexploitid)
```

extracted_Host	CVE	Port	Name	nmap_script_id	exploitdbscript	msf_script_file
10.169.159.118	CVE-2011-3192	8880	IBM WebSphere Application Server 6.1 < 6.1.0.41 Multiple Vulnerabilities	http-vuln-cve2011-3192.nse	platforms/linux/dos/18221.c http://www.exploit-db.com/download/17696	metasploit-framework/modules/auxiliary/dos/http/apache_range_dos.rb
10.169.158.102	CVE-1999-0651	514	rsh Service Detection			metasploit-framework/modules/auxiliary/scanner/rservices/rsh_login.rb metasploit-framework/modules/auxiliary/scanner/rservices/rlogin_login.rb metasploit-framework/modules/auxiliary/scanner/rservices/rexec_login.rb
10.169.157.204	CVE-1999-0651	513	rlogin Service Detection			metasploit-framework/modules/auxiliary/scanner/rservices/rsh_login.rb metasploit-framework/modules/auxiliary/scanner/rservices/rlogin_login.rb metasploit-framework/modules/auxiliary/scanner/rservices/rexec_login.rb
10.15.228.104	CVE-2012-1823	2381	HP System Management Homepage < 7.1.1 Multiple Vulnerabilities	http-vuln-cve2012-1823.nse	platforms/linux/remote/29290.c	metasploit-framework/modules/exploits/multi/http/php_cgi_arg_injection.rb

Figure 11. Reported CVEs to Various Exploits and Testing Scripts

Similarly, other useful results can be obtained by first creating lookup tables for CVEs to vendor security patches, IDS signatures, etc. and then using these lookup tables to map the respective fields.

6. Conclusion

The methodology of vulnerability data analysis is as important as vulnerability scanning. Using data analytical techniques, one can solve the vulnerability data overload problem faced by big organizations. Panda and Splunk are powerful and useful tools for performing data analytics and can be utilized in vulnerability data analysis. Using Pandas or Splunk, we can look for interesting results and insights in the piles of vulnerability data. A manual analysis of the vulnerability data of tens of thousands of hosts is a laborious and cumbersome job. Nevertheless, using Splunk or Pandas, we can quickly explore data and generate desired results without any risk of missing important results or trends present in the vulnerability data. Additionally, we can slice and dice data, perform conditional searches, GroupBy and Join operations, besides the other useful operations on the data in order to summarize, visualize, and derive results. The output obtained by vulnerability data analysis can also be fed into other tools to test the existence of vulnerabilities and store the test results for correlation, thus creating more scope for automation.

7. References

Conti, G. (2007). *Security data visualization: Graphical techniques for network analysis*. San Francisco: No Starch Press.

Fiscus. (2014, April 29). *SANS Penetration Testing | Data, Data, Everywhere What to do with Volumes of Nessus Output | SANS Institute*. Retrieved from https://pen-testing.sans.org/blog/pen-testing/2014/04/29/data-data-everywhere-what-to-do-with-volumes-of-nessus-output

Gula. (2014). *Real-Time Compliance Monitoring*. Retrieved from tenable Nessus website: http://www.tenable.com/sites/drupal.dmz.tenablesecurity.com/files/uploads/documents/whitepapers/tenable_compliance.pdf

McKinney, W. (2013). *Python for data analysis*. Sebastopol, CA: O'Reilly.

Ou, X., & Singhal, A. (2011). *Quantitative Security Risk Assessment of Enterprise Networks*. New York, NY: The Author(s).

Ouchn. (2015, August 11). ToolsWatch.org – The Hackers Arsenal Tools Portal » vFeed Correlated Vulnerability Database API major update 0.6 released. Retrieved from http://www.toolswatch.org/2015/08/vfeed-correlated-vulnerability-database-api-major-update-0-6-released/

Python Data Analysis Library — pandas: Python Data Analysis Library. (2015). Retrieved from http://pandas.pydata.org/

Zadrozny P. & Kodali R. R. (2013). *Big data analytics using Splunk*.

Yogesh Dhinwa, yogeshdhinwa@gmail.com

Appendix A - Sample Network_Address_Book.csv or Host-to-Dept.csv file

The IP_Address field contains an IP address for a host and the Dept field contains the department name to which a corresponding IP address has been allocated.

IP_Address	Dept
10.15.6.198	t-site
10.15.42.222	e-site
10.15.16.101	y-site
10.15.42.185	m-site
10.15.126.103	t-site
10.15.16.72	m-site
10.15.16.46	m-site

Appendix B - Downloading and Converting vFeed Tables

The vFeed DB can be downloaded with wget, decompressed using untar, and then, the vFeed tables are, converted into CSVs using the Python script written for this purpose, as shown below.

```
~/vul_data_analysis> wget http://www.toolswatch.org/vfeed/vfeed.db.tgz
~/vul_data_analysis> tar -xvzf vfeed.db.tgz
~/vul_data_analysis> ./vfeed2csv.py
```

The output of the vfeed2csv.py Python script:

```
 ~/vul_data_analysis/vfeed> ls
cve_cpe.csv              map_cve_gentoo.csv     map_cve_saint.csv
cve_cwe.csv              map_cve_hp.csv         map_cve_scip.csv
cve_reference.csv        map_cve_iavm.csv       map_cve_snort.csv
cwe_capec.csv            map_cve_mandriva.csv   map_cve_suricata.csv
cwe_category.csv         map_cve_milw0rm.csv    map_cve_suse.csv
cwe_db.csv               map_cve_ms.csv         map_cve_ubuntu.csv
map_cve_aixapar.csv      map_cve_msf.csv        map_cve_vmware.csv
map_cve_bid.csv          map_cve_mskb.csv       map_redhat_bugzilla.csv
map_cve_certvn.csv       map_cve_nessus.csv     nvd_db.csv
map_cve_cisco.csv        map_cve_nmap.csv       stat_new_cve.csv
map_cve_d2.csv           map_cve_openvas.csv    stat_vfeed_kpi.csv
map_cve_debian.csv       map_cve_osvdb.csv      vfeed2csv.py
map_cve_exploitdb.csv    map_cve_oval.csv       vfeed.db
map_cve_fedora.csv       map_cve_redhat.csv     vfeed.db.tgz
 ~/vul_data_analysis/vfeed>
```

Yogesh Dhinwa, yogeshdhinwa@gmail.com

The Python script vfeed2csv.py code is illustrated below.

```
#!/usr/bin/python
import sqlite3
import csv
con = sqlite3.connect('vfeed.db')
con.text_factory = str
table_list = []
query = "select name from sqlite_master where type = 'table'"
tlist = con.execute(query)
for table in tlist:
        table_list.append(table)
table_list = [ x[0] for x in table_list ]

for table_name in table_list:
        t_query = "select * from " + str(table_name)
        cursor = con.execute(t_query)
        col_names = [ description[0] for description in
cursor.description ]
        filename = table_name + ".csv"
        with open(filename,'wb') as fn:
                writer = csv.writer(fn)
                writer.writerow(col_names)
                writer.writerows(cursor)
```

In the above python code, the text_factory option instructs SQLite3 to return byte strings in order to avoid Unicode encoding related errors. The variable named table_list contains all the table names in vFeed, which are later used to assign names to the resulting files.

Appendix C - vFeed Categories and Details

Eight vfeed categories are shown below:

```
[D]:vfeed_Cat/
[D]:common_weakness_category_capec/
                    [f1]: cve_cwe.csv
                    [f2]: cwe_capec.csv
                    [f3]: cwe_category.csv
                    [f4]: cwe_db.csv
[D]:cve_additional/
                    [f1]: cve_reference.csv
                    [f2]: map_cve_bid.csv
                    [f3]: map_cve_oval.csv
                    [f4]: nvd_db.csv
[D]:ids/
                    [f1]: map_cve_snort.csv
                    [f2]: map_cve_suricata.csv
[D]:misc/
                    [f1]: stat_new_cve.csv
                    [f2]: stat_vfeed_kpi.csv
[D]:platform/
                    [f1]: cve_cpe.csv
[D]:vendor_security_patch/
                    [f1]: map_cve_aixapar.csv
                    [f2]: map_cve_cisco.csv
                    [f3]: map_cve_debian.csv
                    [f4]: map_cve_fedora.csv
                    [f5]: map_cve_gentoo.csv
                    [f6]: map_cve_hp.csv
                    [f7]: map_cve_mandriva.csv
                    [f8]: map_cve_ms.csv
                    [f9]: map_cve_mskb.csv
                    [f10]: map_cve_redhat.csv
                    [f11]: map_cve_suse.csv
                    [f12]: map_cve_ubuntu.csv
                    [f13]: map_cve_vmware.csv
                    [f14]: map_redhat_bugzilla.csv
[D]:vul_db/
                    [f1]: map_cve_certvn.csv
                    [f2]: map_cve_iavm.csv
                    [f3]: map_cve_osvdb.csv
                    [f4]: map_cve_scip.csv
[D]:vulnerability_assessment/
                    [f1]: map_cve_d2.csv
                    [f2]: map_cve_exploitdb.csv
                    [f3]: map_cve_milw0rm.csv
                    [f4]: map_cve_msf.csv
                    [f5]: map_cve_nessus.csv
                    [f6]: map_cve_nmap.csv
                    [f7]: map_cve_openvas.csv
                    [f8]: map_cve_saint.csv
```

Prefix "D" indicates a directory name and prefix "f" indicates a filename. All these categorized files have been placed inside a directory.

As shown in the figure above, all these files contain additional information about the CVEs and have been put into eight different directories based on their common characteristics. A short description of the above displayed categories are as follows:

1. **common_weakness_category_capec:** There are four files in this directory. Three files, as the names suggest, map the CVE to its CWE number, the CWE to its CAPEC, and the CWE to its Category. The last file in the directory, i.e. CWE_DB, contains more details about a particular CWE. A CVE number can

first be mapped to a CWE and then, for more information on a particular CWE, other CWE-related files should be referred, for example, CWE_DB.

2. **cve_additional:** There are four files in this directory. The two files map the CVEs to bid and oval id, respectively, and the remaining two files contain more details about the CVEs.
3. **Ids:** This directory contains files having data related to the IDS signature for a CVE. As the name suggests, one file contains information for Snort and the other for Suricata.
4. **Misc:** This directory contains vFeed-specific data.
5. **Platform:** This is the only file in this directory that maps the CVE to its Common Platform Enumeration (CPE).
6. **vendor_security_patch:** Files in this directory contain security-related patch information for various vendors that map CVEs to various vendor patches.
7. **vul_db:** This directory contains four files and these files map CVEs to CERT vulnerability data, IAVM DoD vulnerability data, OSVDB, and SCIP.
8. **vulnerability assessment:** The files in this directory are used to map CVEs to vulnerability assessment and penetration testing tools and scripts. These files contain identifiers for testing scripts or exploits for CVEs, which are helpful during quick searches for tools or scripts to be used for testing or exploiting a vulnerability discovered during a scan.