Universiteit Leiden
The Netherlands

Department of Computer Science

Exploratory data Analysis and Visualization

for Cyber Security log files

Pim Claessens

Supervisors:
Dr. Wojtek Kowalczyk
Bas van Stein

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

06/08/2017

Abstract

Cyber security systems generate a huge amount of log files. To detect intrusions and unusual patterns in the log files, cyber security experts must gain insight in the data stored in the log files. But there are a lot difficulties to overcome. The data is unlabeled, current tools are primitive and visualization does not yet meet the complex needs.

In this research we created a program to overcome some of these difficulties. The goal is to develop a visualization and exploration tool which allows a cyber security expert to gain insight in the data represented in the cyber security system log files.

We tested our newly developed system on a data set provided by Los Alamos. During testing we found that our program is fast in exploring and visualizing the data. The program is only lacking in efficient RAM usage, it uses about 4 times as much RAM as the file is big. Also the usability of the program, especially the wrapper part, can be improved.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General information

Modern day societies thrive on the wealth generated by information and communication technologies (ICT). ICT provides governments and companies to grow further and harder than before, but it has a downside. ICT also presents opportunities to those with criminal intentions, leaving our society exposed to cyber-attacks. Those attacks come in all sorts and places and are hard to detect [6]. To detect and prevent these attacks from happening, we created cyber security systems. These cyber security systems monitor all data in an organization and store that information into log files. To detect attacks, cyber security experts need to analyze the information represented in the log files, but therein lies the problem.

1.2 Problem statement

When you think about cyber security system log files, you think about data and a lot of it. So, it should be natural that all the data represented in the log files is used when it comes to cyber security. But the opposite is true, a huge amount of cyber security related data remains untouched [5]. According to this study it has two reasons [5]:

- Most of the IT data sources are not intended and collected for cyber security purposes. The data is collected for other purposes than data analysis for cyber security.
- The collection of cyber security related data analysis is significantly under-represented in current research. The elements of data analytics for cyber security require substantial efforts of research. Unfortunately, this is not happening.

These factors contribute to the problem that data analytics is hardly used to assess cyber security log files.

But even when data is collected for cyber security purposes, there are a lot of difficulties to overcome.

But even when data is collected for cyber security purposes, there are a lot of difficulties to overcome. The problem with cyber security data is that it is unlabeled. Making it hard to detect attacks and unusual patterns (anomalies) [15]. So, to detect those unusual data flows you need to perform exploratory data analysis to get an insight in what is happening in specific cases [15]. So, you want to visualize a user's behavior over time for example. But the tools that are currently being used are primitive [9]. They are ineffective for the data it is being used for and are not using visualization because it does not yet meet their complex needs [9]. This is because the environment of visualizing and exploration of cyber security log files have not yet been researched enough. [9].

The goal of this project is to develop a visualization and exploration tool which allows a cyber security expert to gain insight in the contents of the cyber security log files and detect unusual patterns.

1.3 Approach

To create a program which allows easier visualization and exploration of cyber security log files, we will create a prototype system which allows visualization and exploration with the help of simple commands. To create this system, we will make use of Python and its packages Numpy [3], Matplotlib [4] and Pandas [2]. The aim of our study is to create an easy to use system to explore and visualize cyber security log files. We will especially look at:

- Time needed to explore or visualize cyber security log files
- The RAM usage needed to explore or visualize cyber security log files

To test these requirements of our program, we will make use of a data set provided by Los Alamos nation security, see 1.4. We will visualize some parts of the data set and keep track of the time it took to generate the graphs and the RAM needed to create the graph or find the anomalies. Witch this approach, we hope to create a prototype system which allows easy visualization and exploration of cyber security log files.

1.4 Data Set

To test our algorithm we use the data set obtained from Los Alamos National security [1]. The data contains information from authorization events, Dns lookups and network flow events. To see the data more specifically read Appendix a 7.

All this data has been normalized by Los Alamos to secure privacy and to make sure that all relevant cyber security data can be extracted [5]. There is however a problem with this data, and especially with network flow events. Due to a misconfiguration in the router, no data was collected after day 29, as can been seen in figure 1.1 [5].

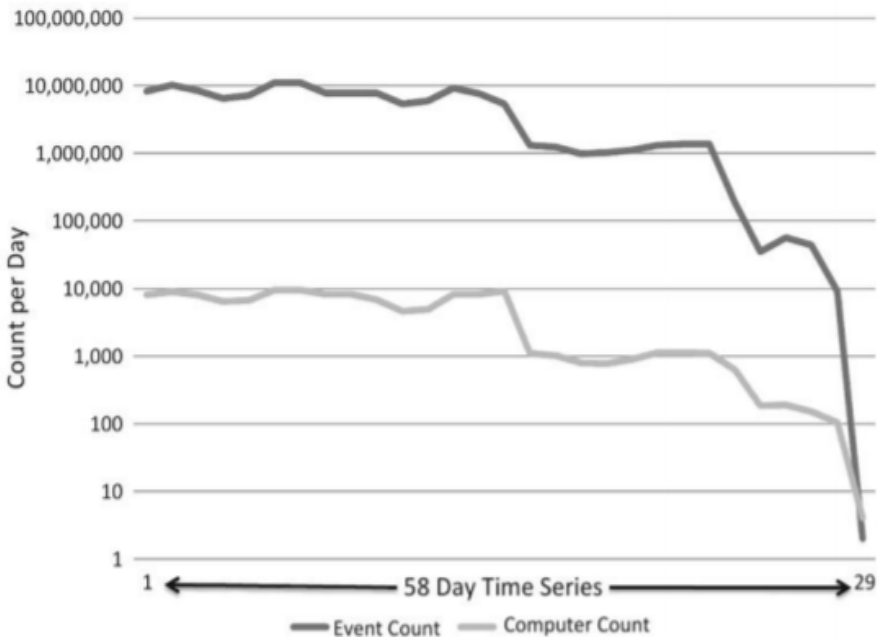

Figure 1.1: Time series and computer counts per day. Due to a misconfiguration in the router no information was gathered after day 29 [5].

1.5 Structure of the Thesis

Chapter 1 provides an introduces the problem. Chapter 2 provides background information about data visualization, finding anomalies and data exploration. Chapter 3 shows the requirements, target user, usage models, implementation details and technical details of our program. Chapter 4 provides a case study where we use our program to visualize cyber security log files. –todo–

Chapter 2

Related Work

Analyzing cyber security log files involves a lot of different techniques, like data analysis, finding anomalies, data visualization. We will provide a brief overview of related work that addresses these aspects.

2.1 Visualization

According to this study [7] visual analytics can and should be used for cyber security. Visual analytics improves cyber security by:

- Recognizing risks and protection against cyber threats. This allows more effective attack prevention and faster isolation and softening of attacks.
- Enable key aspects of the digital forensic process, including data collection, discovery, investigation, examination, analysis and reporting capabilities for information discovery.
- Allow information discovery, processing and visualization

Also a cyber (visual analytics) tool should also meet three important criteria:

- A cyber security defense tool should allow easy input and easy output.
- A cyber security defense should allow the support of report building.
- A cyber security defense should allow the management and analysis of cyber security risks

To meet these requirements, cyber analysts are better of using a toolbox with different visualization as shown in Figure 2.1.

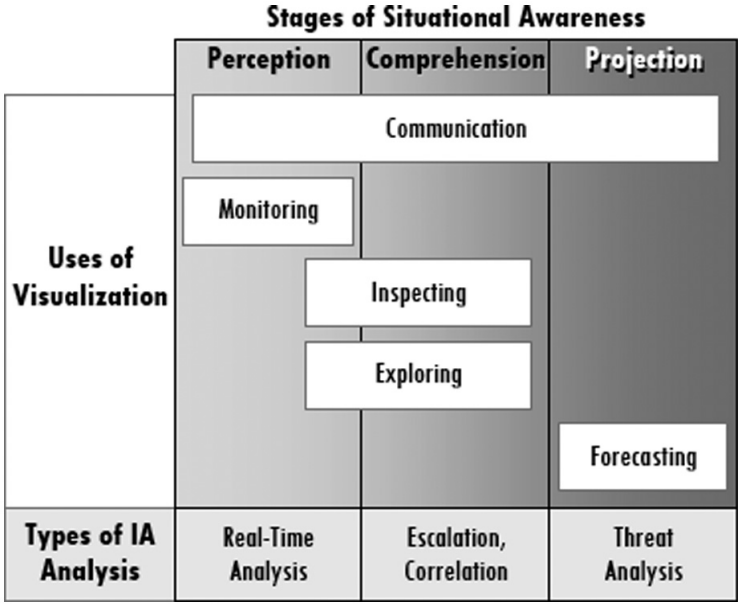

Figure 2.1: This figure shows the relation between the stages of awareness, use of visualization and the types of analysis performed (IA = information assurance) [8]

The main task of a data exploration and visualization tool should be to enable analysts to uncover potential security risks in data. The data gathered by systems is usually big and high-dimensional. The traditional methods of exploring such data are limited and have proven to be ineffective handling this kind of data [10]. Visualization proves to be more effective than the traditional methods. It is more effective in helping to understand high dimensional large data [10], like for instance cyber security log files.

Visual data analysis is an iterative process. Each iteration provides more insight in the data and helps understanding it. Patterns in the visualization direct what the user wants to look at in more detail. The process of progressing to a more detailed level creates a feedback loop as seen in figure 2.2.

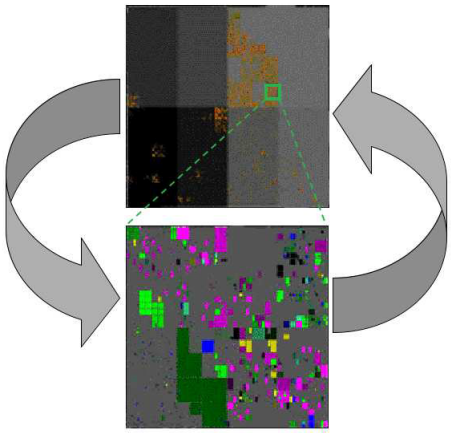

Figure 2.2: This figure shows the feedback loop, beginning at a high semantic level progressing to a more detailed level [10]

Visualizing data is powerful method of exploring large data sets because it makes use of the human input device with the highest bandwidth, our eyes and our perceptual capabilities [11]. There are six ways information

visualization can amplify cognition [12]:

- Increased memory and process resources,
- Reduced information search,
- Using visual representations to enhance the detection of patterns,
- Enabling perceptual inference,
- Using perceptual inference operations,
- Encoding information in manipulable form

Visualization shifts the burden of processing information from the cognitive side, to the human perceptual system. This expands the working memory of the human brain and increases the memory. Visualization reduces searching by grouping information together [11].
Visualization also introduces another key element, visual pattern recognition. Pattern recognition allows the user to detect anomaly's using our perceptual inference and monitoring abilities. Pattern recognition makes use of recognition instead of recall, which increases the working memory of our brain [11].
Another way visualization helps exploring the data is that visualization shows the data in a manipulable form, which allows easy exploration of the data by the user [11].

2.2 Log files visualization techniques

There are many types of visualization techniques, but they can generally be mapped in the following categories [16]:

- *Scalar*: Producing a number to be used in a quantitative analysis. It is useful for activities such as hypothesis testing but lacks the capabilities to capture the kind of data needed determining usage patterns.
- *1-D*: This used to visualize 2 events against each other, for example a timeline.
- *1-D Color and Dynamic*: Using color or displaying in a dynamic fashion will aid the human perception in noticing patterns.
- *2-D Abstract*: A lot of study has been done to visualize log files in abstract 2-D representations. These ranges from a simple frequency table to a more comples markov network chart.
- *2-D Color and Dynamic*: Color can be used to add additional dimensionality to an abstract 2-D representation.
- *3-D*: Modern techniques allow for complex, three-dimensional images. These are not common for visualizing log files.

2.3 Anomaly detection

Abstractly said, anomalies are defined as patterns that does not meet the expected normal behavior [13]. A simple anomaly detection system will define the 'normal' behavior and report the data that does not follow this 'normal' behavior. An example of that can be seen in Figure 2.3. This data set has two regions representing normal behavior, namely regions $N1$ and $N2$. We know this are normal regions because most of the data points lie in these two regions. The anomalies are represented by the regions $O1$, $O2$ and $O3$. These regions are anomalies because the data points in these regions do not lie in the normal regions $N1$ and $N2$.

But due to the following factors, this simple approach becomes very difficult [13]:

- Defining the normal region is difficult. The boundary between the normal region and an abnormal region is most of the time not precise, making it difficult to keep the normal region and abnormal region apart.
- Anomalies that are the result of malicious actions like hacking can are often concealed as normal behavior, making it difficult to detect them.
- Due to the ever-evolving nature of some domains, normal behavior now might not be normal behavior in the future.
- The notion of an anomaly is different for different domains.
- The availability of labeled data for training and validation is a major issue
- The data can contain noise which tends to be similar to anomalies and is therefore difficult to detect and remove.

To still be able to detect anomalies, researchers have combined concepts from several research areas such as statistics, machine learning and data mining.

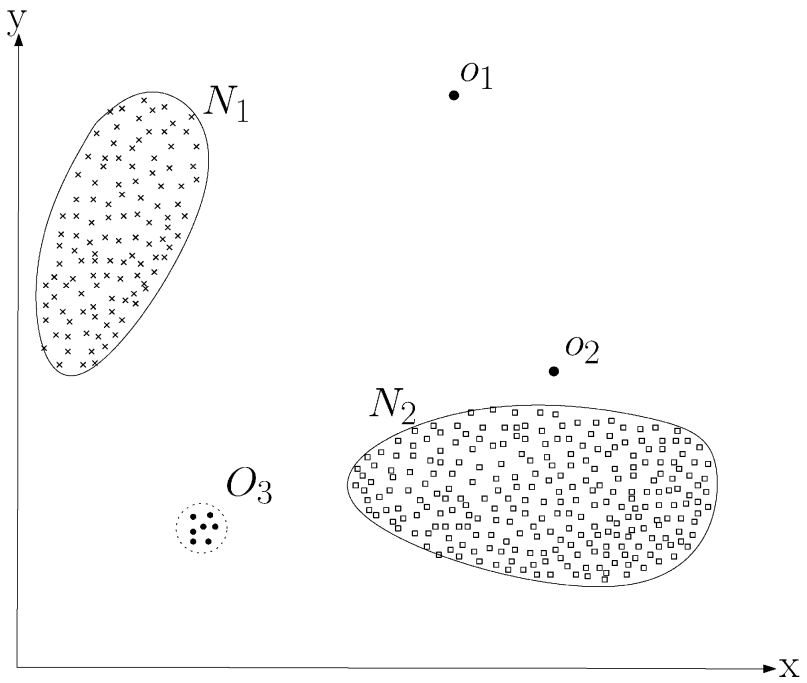

Figure 2.3: This figures shows the anomalies in a population of points [13]

2.4 Anomaly detection techniques

As mentioned before, there are many different techniques for anomaly detection. Here we will give some methods that have been used for anomaly detection [14].

2.4.1 Statistical anomaly detection

This technique fits a statistical model to the given data to detect the normal region. To test if an unseen part of the data belongs to the normal region or not it uses statistical inference [13].

These types of systems have several advantages [13]:

- If the assumptions about the data hold, statistical methods provide a statistically justifiable solution for detecting anomalies.
- The score of the anomaly is given as a confidence interval. This additional information can be used to test whether it really is an anomaly.
- Statistical methods can be used on unlabeled data in an unsupervised setting.

But statistical methods for anomaly detection also have some drawbacks [13]:

- Statistical techniques rely on the assumption that the data has a particular distribution. This assumption often does not hold with high dimensional real data sets.

- It is difficult to choose the best test statistic, especially when you are dealing with high dimensional data sets.
- Histogram based techniques are easy to implement, but they cannot capture the interaction between attributes.

2.4.2 Machine learning based anomaly detection

Machine learning is the ability of a program or system to improve on their task over time. Machine learning based techniques will try to answer the same questions as statistical based techniques. Machine learning however builds a system that improves on its performance based on previous results instead of statistical approaches which try to understand the process that generated the data. So, machine learning based techniques have to ability to change their strategy based on newly acquired data [14]. Some examples of machine learning based techniques are [14]:

- **System call based sequence analysis**: This technique learns the normal behavior of a program which allows it to detect significant deviations from the normal behavior [14].
- **Bayesian networks**: This technique works by encoding probabilistic relationships between variables of interest [14]. The advantages are that Bayesian networks can handle missing data because it encodes inter-dependencies. Bayesian networks can also predict the outcome of an action because it can represent causal relations. Because Bayesian network have causal and probabilistic dependencies, it can be used to model problems where there is a need to combine prior knowledge with data [14].
- **Principal component analysis**: Cyber security related data sets are usually large and multidimensional, this complex data is hard to analyze and understand [14]. To tackle this problem, researchers have developed a way to tackle the dimensionality of such datasets to make it possible to extract relevant information.

2.4.3 Data Mining based anomaly detection

Data mining algorithms work by taking data as its input and pull from it relevant information like patterns and associations [14].

- **Classification-based intrusion detection**: A classification based intrusion detection works by first identifying class attributes and classes from training data. It then will identify attributes for classification and create a model based on the data. This model is then applied on the new data samples [14].
- **Clustering and outlier detection**: Clustering is the identification of patterns in unlabeled data with many dimensions [14]. This allows it to detect outliers without having any previous knowledge of the system. The detection of outliers using clustering is explained in figure 2.3.

- **Association rule discovery**: This describes events that tend to occur together. So, an outlier is detected when a certain sequel of events is unfolding [14].

Chapter 3

Cyber Explorer: Design and Key Functionalities

To help a cyber security expert gaining insight in cyber security log files, we have created a tool. In this chapter, we will provide the target user, how to use, requirements, usage models and technical information of our tool.

3.1 Target user

This tool is created for human experts in the field of cyber security. Modern day cyber security systems generate a lot of log files which are hard to analyses. This tool helps cyber security experts in getting insight in the data by easing the exploration and visualization of cyber security log files. The users however must have some basic understanding of programming, in particular Python programming, and some basic knowledge in data exploration and visualization.

3.2 How to use

This tool can be used in two different ways. You can use the functions that are described here later (see todo) directly in a custom-made script, or you can use the wrapper to directly work from the command line. To call the function directly, the user first must download and install the Python package. To do this, you can make use of *easy_install* by typing the following line in the command line *easy_install cyberexplorer*. This will download and install the cyberexplorer package. To call this package use the following:

```
import cyberexplorer
```

To use the wrapper, the user must download the code from `https://github.com/wiedenkje/cyberexplorer`. To run the wrapper, go to the folder containing the code and run 'Main.py'. This will start the command line

section of this tool.

3.3 Requirements

To use a data set containing cyber security log files in our program, it must meet certain requirements. The data must first be prepared to meet our requirements. First, every data set file must contain comma separated values (CSV). Second, every column containing the time information must be preprocessed to seconds. So, every column containing the time must start at 0 (meaning 0 seconds past since start of period contained in data set) and end with the number of seconds past since the start of the period at the end of the period. So, if a data set contains the period from 12:00 1-1-2016 to 12:00 1-1-2017, the time column starts with 0 and ends with 31556926. Also, the columns containing the Time information must be called 'Time'. Third, all the data must be normalized across all files in the data set. Data entries that contain the same information (for example user) must have the same entry in all of the files.

3.4 The required functionality

The tool was designed to help a human expert gain insight in cyber security log files without having much programming skills. The tool must be easy to use, fast and create clear data visualization/explorations. To satisfy these goals we made a command line tool which makes use of some basic operators which allows for a whole variety of different visualizations and explorations. The user of this tool can use the command line section of this tool to create graphs using simple queries, or can use the functions directly to make reusable scripts. To visualize/explore a cyber security log file data set you need to have preprocessed that particular data set to the extent that it contains usable information. So, a data set must have been preprocessed outside of this tool to create a data set containing only the use full information.

3.5 Usage models

As said before, this tool can be used in two ways. The user can create custom made reusable scripts using the functions described in this section, or the user can use the wrapper to work directly from the command line. Both ways of working are described in this section.

The tool consists out of five main functions. These functions are used to process, visualize, or explore the data. The five main functions are:

- READ: The READ operator is used to process the data of a new data set. (see 3.5.1)
- SELECT: The SELECT operator is used for loading an already processed data set into RAM for analysis. (see 3.5.2)

- PLOT: The Plot operator is used for visualizing a certain part of the data specified by the user. (see 3.5.3)
- DETECT: The DETECT operator is used to explore the data in search for anomalies. (see 3.5.4)
- ADD: The ADD operator is used to add a new data batch to an already processed data set (see 3.5.5.

3.5.1 READ

The read operator is used when an user wants to load a new data set. The data set is loaded and all of its inputs are transformed to integers. We transform all data to integers because integers allow us to make rapid calculations since integers are easier to process. To keep a link to the real data we store 2 dictionaries for each column: one dictionary to store the data from integer to strings (the real data) and one dictionary to store the data from strings (the real data) to integers. The transformation to integers is done using pandas 'factorize function'. This function will rapidly transform the data entries in each column (except the column containing the time) of the data set to integers.

The read operator requires 2 arguments:

- *new*: This contains the location of the data set the user wants to process. You must give the folder in which data set is located, so if the data set is located in '/home/user/dataset/':

  ```
  new='/home/user/dataset/'
  ```

  Per file the user will be asked how many columns the file contains and if the file contains a header or not. If the file does not contain the header, the use will be asked to specify the names of the columns. The user must make sure the folder containing the new data set only contains the files that are part of the data set and nothing else. The READ function will detect the files, but to avoid any problem it is best use to keep the folder clear of any noise.

- *to*: This specifies the location the processed data set will be saved in. If the user wants to save the data set in '/home/user/save/' use:

  ```
  to='/home/user/save/'
  ```

  It is best use to save it to an empty folder to avoid any noise in the folder.

Usage

To use this function when writing a script and the data set is located in '/home/user/dataset/' and you want to save it at '/home/user/save/' use it as follows:

```
import cyberexplorer as ce

ce.read(new='/home/user/dataset/', to='/home/user/save/')
```

To use this function in the wrapper, use:

```
READ(new:/home/user/dataset, to:/home/user/save/)
```

3.5.2 SELECT

The SELECT operator is used to read in a processed data set. The data is loaded in the RAM of the machine and is used in the further exploration and visualization of the data set. SELECT requires one argument:

- *Location*: This specifies the location of the processed data set (as given in 'to' at the READ operator). So, if the processed data is located in '/home/user/save/' use:

```
Location='/home/user/save/'
```

Usage

To use this function when writing a script and the data set is located in '/home/user/save/', use it as follows:

```
import cyberexplorer as ce

dataset, names = ce.select(location='/home/user/save/')
```

'dataset' Is a dictionary containing all of the files that form the data set. 'names' Is a dictionary containing all the names of the columns in the data set.
To use this function in the wrapper, use:

```
SELECT('/home/user/save/')
```

3.5.3 PLOT

The PLOT function is used for visualizing a certain part of the data set specified by the user. The user specifies a part of a data set and the PLOT operator will automatically visualize this part for the user in a plot type specified by the user. The PLOT operator makes use of 7 arguments (3 are required, 4 are optional)

- *X* (required): This specifies the column name used for the x-axis of the resulting plot. This column name is used to group the data and to count the resulting grouping. The behavior is like SQL's (Structured Query Language) 'GROUP BY'. Only one column entry is allowed by the system. If the users wants the column 'Source Computer' to be on the x-axis, use:

```
x="Source Computer'
```

If the column contains the time information, the user can transform the seconds to minutes, hours, days, weeks or months by making use of the following sub functions:

- *MIN()*: MIN() will transform the seconds to minutes and is used as follows:

```
x='MIN(Time)'
```

- *HOUR()*: HOUR() will transform the seconds to hours and is used as follows:

```
x='HOUR(Time)'
```

- *DAY(Time)*: DAY() will transform the seconds todays and is used as follows:

```
x='DAY(Time)'
```

- *WEEK(Time)*: WEEK() will transform the seconds to weeks:

```
x='WEEK(Time)'
```

- *(MONTH(Time)*: MONTH() will transform the seconds to months:

```
x='MONTH(Time)'
```

- *Y* (required): This specifies the column names used for the values on the y-axis. The column entries will be grouped on the column specified by the 'X' attribute. For every entry, the result is counted on every X-entry and visualize by the specified plot type. There are three ways to count the entries in a column:

  - *Y='Column-name'*: This is used when the user wants to count the occurrence of every unique value in the column. So, if a y-column contains two unique values like "Success" and "Fail". Success has 4 occurrences and Fail has 1 occurrence and the X-column has only 1 entry, the PLOT function will visualize $Success == 4$ and $Fail == 1$.

  - *Y='UNIQUE(Column-name)*: This is used when the user wants to count the number of unique occurrences. So, if a column contains two unique values like "Success" and "Fail" and the X-column has only 1 entry, the PLOT function will visualize the number 2 since this column contains 2 unique values ("Success" and "Fails").

  - *Y="TOTAL(Column-name)*: This is used when you want to visualize the total number of rows. So, if a column contains two unique values like "Success" and "Fail". Success has 4 occurrences and Fail has 1 occurrence and the X-column has only 1 entry, the PLOT function will return the number 5 since the total number of rows equals 5 ($4 + 1 = 5$)

If you want to visualize multiple columns at once, use the ';' as a separator. So, if the user wants to visualize the columns 'Source Computer' and 'Resolved Computer':

```
Y='Source Computer;Resolved Computer'
```

If you want to visualize columns from multiple files, use '.' to distinguish file and column. So, if the user wants to visualize the column 'Source Computer' from file 'Authorization':

```
Y='Authorization.Source Computer'
```

This is only useful if you want to visualize multiple files at once. When you are visualizing multiple files, make sure that the x-column is present in all files.

- *FROM* (required): This specifies which file you want to visualize from the data set. So, if the user wants to visualize the file 'Authorization', the syntax will be as follows:

```
FROM='Authorization'
```

If you want to visualize multiple columns from a data set at once, separate the file names using ';'. So, if the user wants to visualize the files 'Authorization' and 'Dns', use the following syntax:

```
FROM='Authorization;Dns'
```

- *WHERE* (optional, default None): This is used to query the data before visualization. The allowed operators are:
  - ==: Equal to.
  - >: Greater than.
  - <: Less than.
  - >=: Greater than or equal to.
  - <=: Less than or equal to.
  - != : Not equal to

The structure of a query should look like this:

```
Column operator value
```

Make sure that there are spaces surrounding the operators and inputs,

```
column == value
```

instead of

```
column==value
```

If you are visualizing columns from multiple columns separate the query per query using ';' and '.' to separate file from column. So, if the user wants to visualize the column 'Source Computer' from the file

'Authorization and the column 'Resolved Computer' from the file 'Dns' and the user wants to query both files, he should use:

```
WHERE="Authorization.Time > 10" ; Dns.Source Computer == Computer"
```

- *USING* (optional, default 'line'): This is used to specify the type of visualization the user wants to user. To supported types of visualization are:
  - *line*: This will produce a line plot, default value.
  - *bar*: This will produce a vertical bar plot.
  - *barh*: This will produce a horizontal bar plot.
  - *box*: This will produce a box plot.
  - *scatter*: This will produce a scatter plot.
  - *hexbin*: This will produce a hexbin plot.
  - *pie*: This will produce a pie chart.

  it is used as follows:

  ```
  USING="visualization_type"
  ```

- *SAVEAS* (optional, default None): This is used to specify to save location of the generated plot. If this is not given, the user will be asked to specify it later on the algorithm. If the user wants to save the plot at '/home/user/saveplots/' and call it plot.png, he uses it as follows:

  ```
  SAVEAS='/home/user/saveplots/plot.png'
  ```

- *average* (optional, default None): This is used when to user wants to compare some values in a column against the average count for the values in the column. The resulting plot will show the count for the average value of the selected column and the count for the value in the column you wanted to visualize. So if we want to compare source computer C1250 against the average of all source computers and save it at '/home/user/saveplots/', the plot function will look like this:

  ```
  PLOT(x='Time', y='Source computer', FROM='Authorization',
  WHERE='Source Computer == C1250', SAVEAS='/home/user/saveplots/',
  Average='True')
  ```

  The PLOT function will now calculate the count for Source Computer C1250 and the count for all source computers and visualize both.

Usage

If we want to visualize column 'Start or Stop' against Time in days of file 'Process' as a line plot in a script: Use the following:

```
import cyberexplorer as ce

dataset, names = select('/home/user/save/')

plot(x="DAY(Time)", y="Start or Stop", FROM="Process", USING="line",
    dataset=dataset, names=names)
```

When you use this function in a script, you have to specify the dataset and the names of the columns in the data set. This data was gathered using the SELECT function. When you want to visualize the same information using the wrapper, use:

```
PLOT(x:DAY(Time), y:Start or Stop", FROM:Process, USING=line)
```

3.5.4 DETECT

The DETECT function can be used to detect anomalies in a pre-specified part of the data set. The user specifies the part of dataset and a list with anomalies together with a distribution of the anomalies is returned. The DETECT function makes use of 7 arguments (3 required, 4 optional):

- *X* (required): The X is used to specify the column in which to find anomalies. So, if you want to find anomalies in the column 'Source Computer', the entries which are anomalies are listed and returned. It is used as follows:

  ```
  x='column_name'
  ```

  Only one column allowed If the column contains the time information, the user can transform the seconds to minutes, hours, days, weeks or months by making use of the following sub functions:

  - *MIN()*: MIN() will transform the seconds to minutes and is used as follows:

    ```
    x='MIN(Time)'
    ```

  - *HOUR()*: HOUR() will transform the seconds to hours and is used as follows:

    ```
    x='HOUR(Time)'
    ```

  - *DAY(Time)*: DAY() will transform the seconds todays and is used as follows:

    ```
    x='DAY(Time)'
    ```

- *WEEK(Time)*: WEEK() will transform the seconds to weeks:

  ```
  x='WEEK(Time)'
  ```

- *(MONTH(Time)*: MONTH() will transform the seconds to months:

  ```
  x='MONTH(Time)'
  ```

- *Y* (required): The Y is used to base the anomalies on. So, if the user wants to find anomalies in the column 'Source Computer' based on the column 'Start or Stops', the entries (grouped on X) in Y are counted and analyzed by an algorithm (see later) to find the anomalies. It is used as follows:

  ```
  y="column-name"
  ```

  Only one column allowed

- *FROM* (required): The FROM is used to specify the file you want to detect anomalies in. It is used as follows:

  ```
  FROM="file"
  ```

  Only one file is allowed

- *WHERE* (optional, default None): This is used to query the data before visualization. The allowed operators are:

  - ==: Equal to.
  - >: Greater than.
  - <: Less than.
  - >=: Greater than or equal to.
  - <=: Less than or equal to.
  - !=: Not equal to

  The structure of a query should look like this:

  ```
  Column operator value
  ```

  Make sure that there are spaces surrounding the operators and inputs,

  ```
  column == value
  ```

  instead of

  ```
  column==value
  ```

- *USING* (optional, default iqr): Here the user defines which type of anomaly detection he wants to use. There are currently two types of anomaly detection implemented:
  - *z-score*: This algorithm works by calculating the mean of all the counts of the y-column and report the entries in the X-column that are too much (default is 3 standard deviations away from the mean. If that threshold is positive, the calculation looks like this:

    ```
    anomalies = entries > (mean + var * standard deviation)
    ```

    If that threshold is negative, the calculation looks like this:

    ```
    anomalies = entries < (mean + var * standard deviation)
    ```

    Note that var will be negative so the + will effectively be a minus. It returns a table containing the anomaly, the observed counts and the z-score of all anomalies. For an example table see Table 8.2.
  - *Interquartile range* (iqr): This algorithm works by calculating the difference between the first and third quartile.

    ```
    difference = quartile 3 - quartile 1
    ```

    Then it will return the X-column entries that are more than a threshold (default 1.5) of that difference away. If that threshold is positive, the calculations looks like this:

    ```
    anomalies = entries > (quartile 3 + var * difference)
    ```

    For var being the threshold, see later on (default value is 1,5) If that threshold is negative, the calculation looks like this:

    ```
    anomalies = entries < (quartile 1 + var * difference)
    ```

    Note that 'var' will be negative so the + will effectively be a minus.
- *VAR*(optional, default see USING): Here the user defines the variable needed for the z-score and interquartile range. The standard values are 3 and 1.5 respectively. It is used as follows:

  ```
  VAR=float
  ```

- *SAVEAS* (optional, default None): This is used to specify the save location of the resulting table and graph. If the user does not specify it before, he will be asked to specify it later on in the algorithm. If the user wants to save it at '/home/user/anomalies/' and call it anomaly , he uses:

  ```
  SAVEAS='/home/user/anomalies/anomaly'
  ```

  Note that it produces to files in the folder anomalies, namely anomaly.txt containing the table and anomaly.png containing the graph.

Usage

If you want to find anomalies in the column 'Users' based on 'Processes' in the file 'Process' using z-score and save it '/home/user/anomalies/' in a script, use:

```
import cyberexplorer as ce

dataset, names = select('/home/user/saves/')
detect(x="Users", y="Processes", FROM="Process", USING="z-score",
SAVEAS="'/home/user/anomalies/anomaly', dataset=dataset, names=names)
```

Here dataset and names specify the dictionary containing the data set and the dictionary containing the names respectively.

To do the same thing in the wrapper use:

```
DETECT(x:Users, y:Processes, FROM:Process, USING:z-score,
SAVEAS:/home/user/anomalies/anomaly)
```

3.5.5 ADD

The ADD is used to add a new data batch to an already existing and processed data set. This can only be used if the data set you want to add a batch to is already processed using the READ operator (see 3.5.1). It uses 2 arguments:

- *new*: This is used to define the data batch you want to add to the already processed data set. The data batch will be processed according the stored dictionaries of the present data set. A new data set is created consisting out of both parts. The user needs to specify the path to the data batch. Make sure that the folder containing the data batch is clear of any noise, only the new data should be present. So, if the data batch is located at '/home/user/batch/', use the following:

  ```
  new='/home/user/batch/'
  ```

- *to*: This is used to specify the already present and processed data set and its dictionaries. You need to specify the path to the data set. So, if a data set is located in '/home/user/save/', use:

  ```
  to='/home/user/save/'
  ```

Usage

If the user wants to add a new data batch located in '/home/user/batch/', to the processed data set in '/home/user/save/' in a script, he uses the following:

```
import cyberexplorer as ce

add(new='/home/user/batch/', to='/home/user/save/')
```

To do the same thing in the wrapper use:

```
ADD(new:/home/user/save/, to:/home/user/save/
```

3.6 Technical information

This tool was created using python 3.6.0, numpy 1.13.1, pandas 0.20.1 and matplotlib 2.0.2. We used python because, together with its libraries, it allows for easy transformation, exploration and visualization of large data sets [2] [3] [4]. The library pandas was used because of its ability to load and save data rapidly into a data frame and because of its good integration with matplotlib [2]. Numpy was used because of its rapid calculation, which allowed us to rapidly analyze a large data set [3]. Matplotlib was used because Matplotlib makes publication worthy figures in a variety of formats. [4].

Chapter 4

Case study Los Alamos

To test our newly created system we will visualize the data set provided by the Los Alamos National security [1]. This data set contains the log files of their cyber security system, for more details see 7. To visualize this data set, several steps should be taken when using our algorithm.

1. Preprocessing of the data set.
2. Selection of the data set you want to use.
3. Visualize or explore the data set

4.1 Preprocessing of the data set

In order to visualize or explore any cyber security related data set by our algorithm, you first need to process the data. Our algorithm will transform all strings in the data set to integers. To do this our algorithm first converts all strings into unique integers. If there are multiple instances of a string, they get the same integer. Second two dictionaries are created, one for the references for string to integer and one for integers to string. Those dictionaries are used by our algorithm to look up the meaning of an integer or the other way around. Then the original data set is replaced by its integer values and stored it as an hdf file. To perform the case study, we first need to process the dataset. For this case study we use the Dns and process files of the Los Alamost data set (see 7. The data is located in '/data/user/datasets/' and we will save it in '/data/user/saves/'. To do this, we use the following code:

```
import cyberexplorer as ce

ce.read(new='/home/user/datasets/', to='/data/user/saves/new/')
```

When running this code, the user will be asked how much columns each file contains and the column names. The Dns file has 3 columns, called: Time, S_Comp (Source computer) and R_Comp (Resolved Computer). The

Process file has 5 columns, called: Time, User, Comp (Computer), Proc (Process) and Start_stop. The total time needed to process these two files combined was 1884,1046 seconds (53,1683 seconds for the Dns file and 1830,9363 seconds for the Process file) . The total RAM needed to perform this operation was the biggest for the Process file. The process file is about 15 gigabytes big and used about 34 gigabytes of RAM. To perform the same operation in the wrapper, use:

```
READ(new:/home/user/datasets/, to:/data/user/saves/new/)
```

4.2 Selection

In order to visualize or explore a data set, you first need to load the data. When the algorithm loads the data, it loads all preprocessed data sets into RAM for faster visualization and exploration. It also specifies the location of where to find all dictionaries need to explore and visualize the data set. After this the data is ready to be explored and visualized. To select the data we processed in the section before, we use the following script:

```
import cyberexplorer as ce

ce.select(Location='/data/user/saves/new/')
```

This script will load the Dns file and the Process file into RAM. This operation takes about 9.1 seconds to complete and requires about 18 gb of RAM. To perform the same operation in the wrapper, use:

```
SELECT(/data/user/saves/new/)
```

4.3 Exploration and Visualization

So in order to visualize and explore any data set, you need to preprocess and load the data. After that, you simply state which part of the data set you want to load. To this you use the functions given in chapter 3, or see this chapter for examples.

4.4 Case Study

In this section we will present some examples of how to explore and visualize cyber security system log files. To demonstrate it, we use the data set provided by the Los Alamos Labatory 7. Let's say we want a line graph presenting all Dns look ups over time in days. To visualize this, we use this script:

```
import cyberexplorer as ce

dataset, names = ce.select(Location='/data/user/saves/new/')
```

```
dataset, names = ce.plot(x='DAY(Time)', y='TOTAL(S_Comp)', FROM='Dns')
```

To generate this graph in the wrapper, use:

```
plot(x:DAY(Time), y:TOTAL(S_Comp), from:Dns)
```

The result of this query in the program can be seen in graph 4.1. We can see that the number of DNS lookups per day is low until day 29. From that day on, the number of DNS lookups per day is substantially higher than before. So, from day 29 on an event occurred that drastically increased the number of DNS lookups per day. This might be interesting to see for experts who might consider this to be odd behavior, or just a flaw in the data collection prior to day 29.

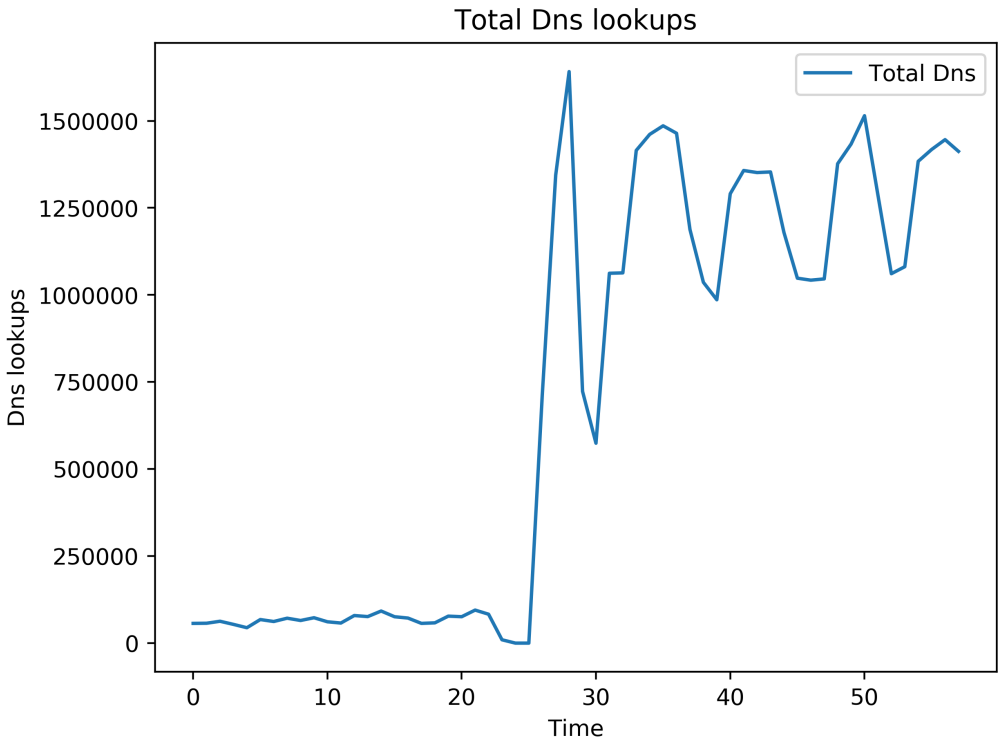

Figure 4.1: This graph shows the total Dns lookups over time in days

You can also visualize the Dns Look ups for a particular computer and compare it with the average number of Dns lookups for a particular Time. So, let's say we want to visualize the number of Dns lookups per day for Source computer *C4653* and compare that source computer with the average number of Dns lookups per day of all source computers. The script will be as follows:

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

dataset, names = ce.plot(x='DAY(Time)', y='TOTAL(S_Comp)', FROM='Dns',
```

```
WHERE='S_Comp == C4653', total:True, dataset=data, names=name)
```

To generate this graph in the wrapper, use:

```
plot(x:DAY(Time), y:TOTAL(S_Comp), from:Dns, where:S_Comp == C4653, total:true)
```

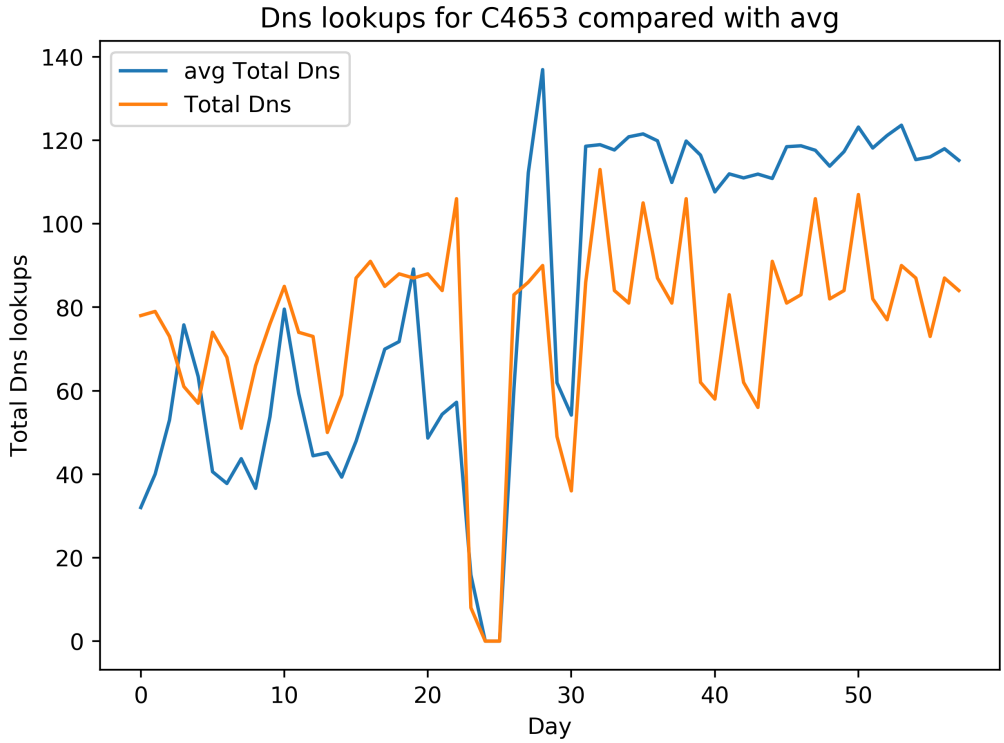

Figure 4.2: This graph shows the total Dns lookups over time in days for source computer C4653 compared with the average Dns lookups for all source computers over time in days. User C4653 is visualized in 'Total Dns'

The result of this query can be seen in graph 4.2. We can see that the total DNS lookups for source computer C4653 roughly matches that of the averages DNS lookups per day of all source computers. Meaning that source computer C4653 did not exhibit strange behavior during those 58 days. So C4653 will not be classified as an anomaly. To find a source computer that does not meet the average behavior, we need to apply anomaly detection. In order to detect those anomalies in the DNS lookup file we need to use the following script (this script uses the iqr method of anomaly detection):

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.detect(x='S_Comp', y='R_Comp', FROM='Dns', dataset=data, names=name)
```

To find the anomalies in the wrapper, use:

```
detect(x:S_Comp, y:R_Comp, from:Dns)
```

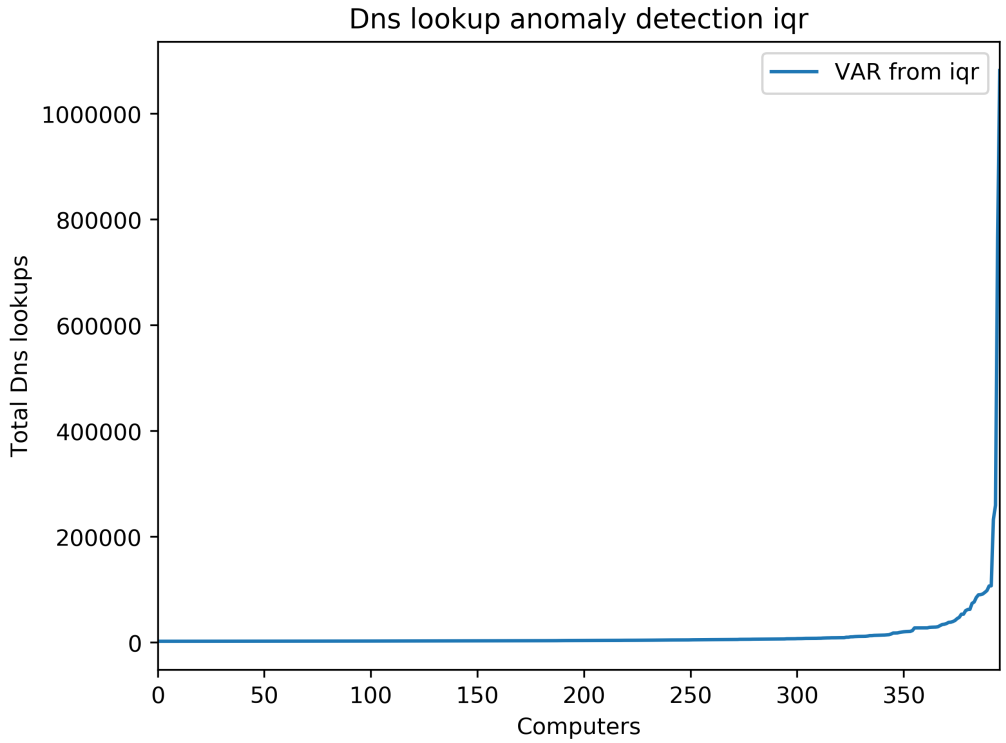

Figure 4.3: This graph shows the distribution of the anomalies in DNS lookups using the interquartile range method of anomaly detection.

This query will provide the user with a table containing all the anomalies, their total number of DNS lookups, and there difference from the interquartile range as well as the distribution of the anomalies, which can be seen in graph 4.3. A part of the anomalies can be seen in table 8.1. When we look at this part we can see that source computer *C585* has the most DNS lookups and therefore scores the highest anomaly score. To see if this computer really shows unusual behavior, we need to visualize his total DNS lookups per day against that the average. To do this, we need the following script:

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.plot(x="DAY(Time)", y="TOTAL(R_Comp)", FROM='Dns', WHERE='S_Comp == C585',
average='true', dataset=data, names=name)
```

To generate the same graph in the wrapper, use:

```
PLOT(x:DAY(Time), y:TOTAL(R_Comp), from:Dns, where:S_Comp ==
C585, average:true)
```

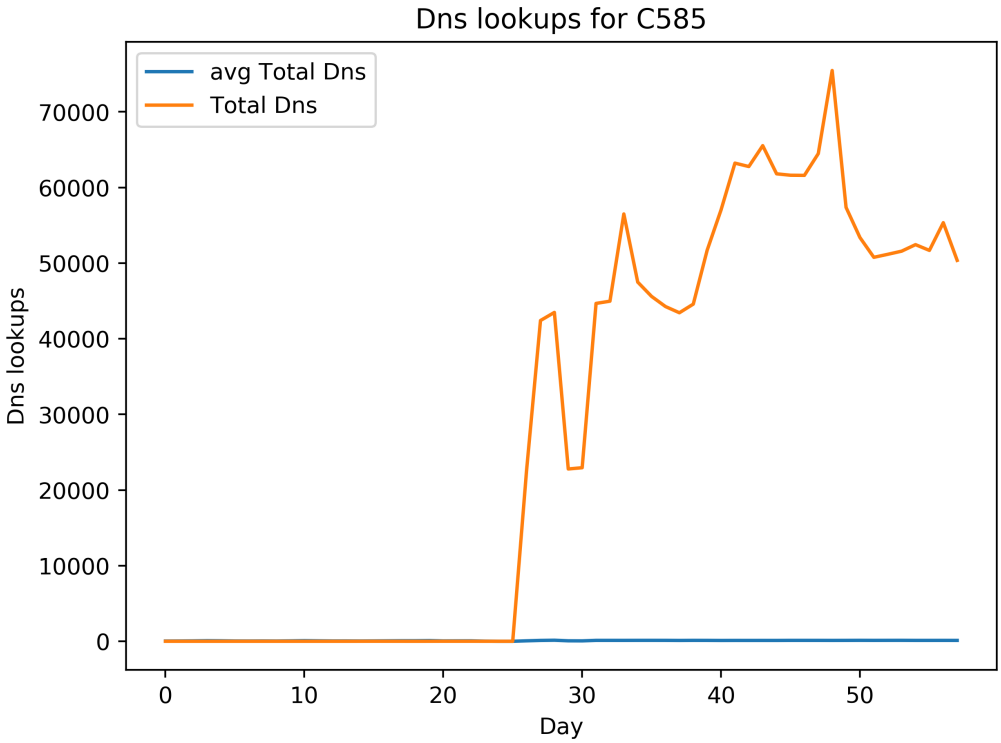

Figure 4.4: This graph shows the total Dns lookups over time in days for source computer C585 compared with the average Dns lookups for all source computers over time in days.

The resulting graph of this query can be seen in graph 4.4. We can see that the number of DNS lookups are practically 0 before day 25. After day 25 the number of daily DNS lookups is extremely high compared with the average. This might indicate that this computer was new and started to be used intensively after day 25, this does not mean that this computer is an anomaly. It was detected as such because of its unusually high total number of Dns lookups. But after visualizing his behavior we can conclude that it is not completely strange behavior.

We can do the same for the Process start and stop events. If a cyber security expert wants to know how many unique processes there are over time for each day, he needs to run this script:

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.plot(x="DAY(Time)", y="TOTAL(Process)", FROM="Proc", dataset=data, name=name)
```

To generate the same graph in the wrapper, use:

```
plot(x:DAY(Time), y:TOTAL(Process), from:Proc)
```

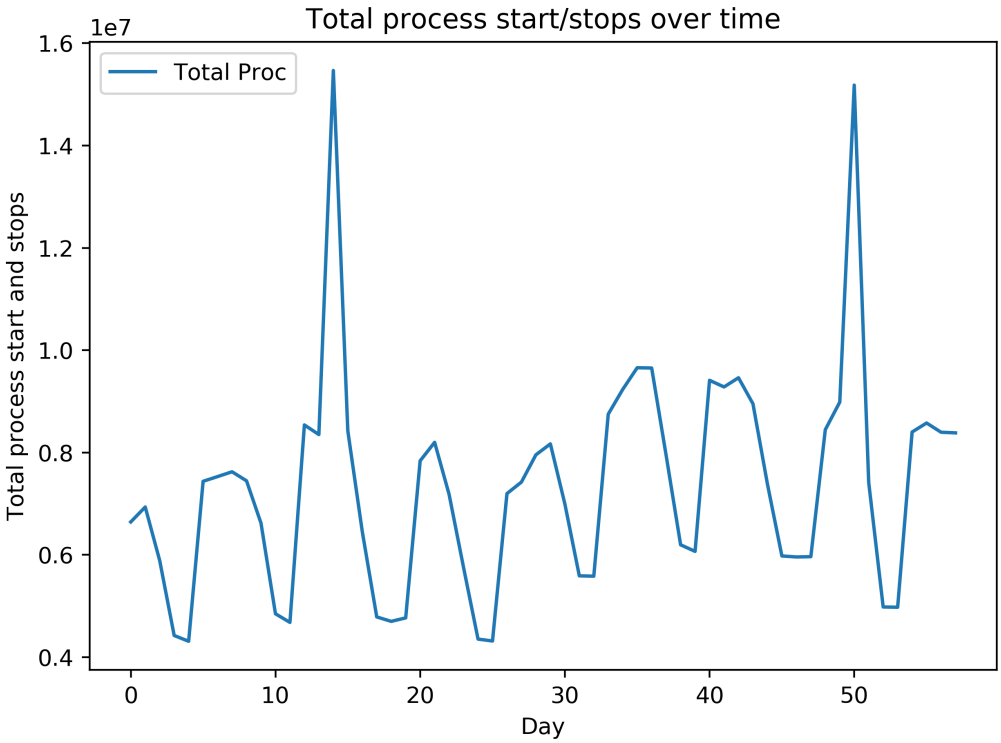

Figure 4.5: This graph shows the total number of process start and stops per day over time

The result of this query can be seen in graph 4.5. When we look at the graph we can clearly see that there is a trend with two spikes around day 15 and day 50. This might be an indication to a cyber security expert that on those days something happened which needs to be looked into. We can do that by checking of there were more processes being started and stopped used on those days. The script will be as follows:

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.plot(x="DAY(Time)", y="UNIQUE(Process)", FROM="Proc",
    dataset=data, name=name)
```

To generate the same graph in the wrapper, use:

```
plot(x:DAY(Time), y:UNIQUE(Process), from:Proc)
```

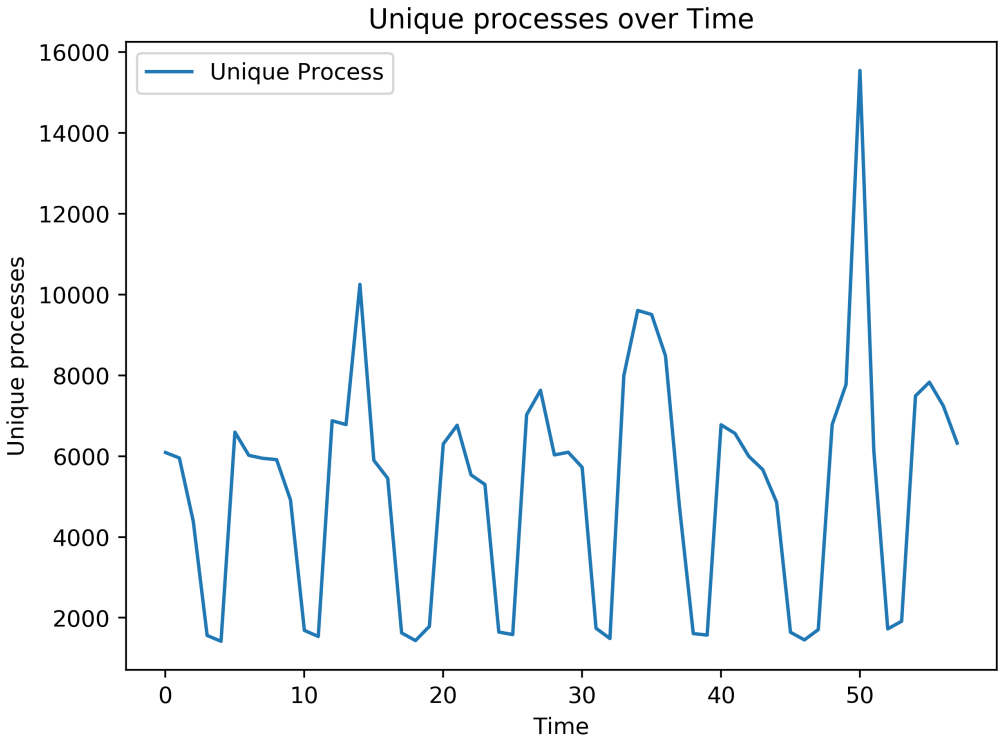

Figure 4.6: This graph shows the total number of unique process per day over time

The resulting graph can be seen in 4.6. When we look at this graph we can only see 1 clear spike around day 50. The spike around day 15 is not as clear. This is an indication that on day 15 one or more processes were used more frequently than the days before. On day 50 more different processes were being used which indicates that either new processes where started, some of them might be attempts to break the security and might alert a cyber security expert.

We can also detect users with an unusually high or low number of process start stop events. To do this we use the z-score rule which results in the following script (this query uses the z-score anomaly detection):

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.detect(x='User', y='Start_stop', FROM='Proc', USING='z-score',
    dataset=data, names=name)
```

To generate the same table and graph in the wrapper, use:

```
detect(x:User, y:Start_stop, from:Proc, using:z-score)
```

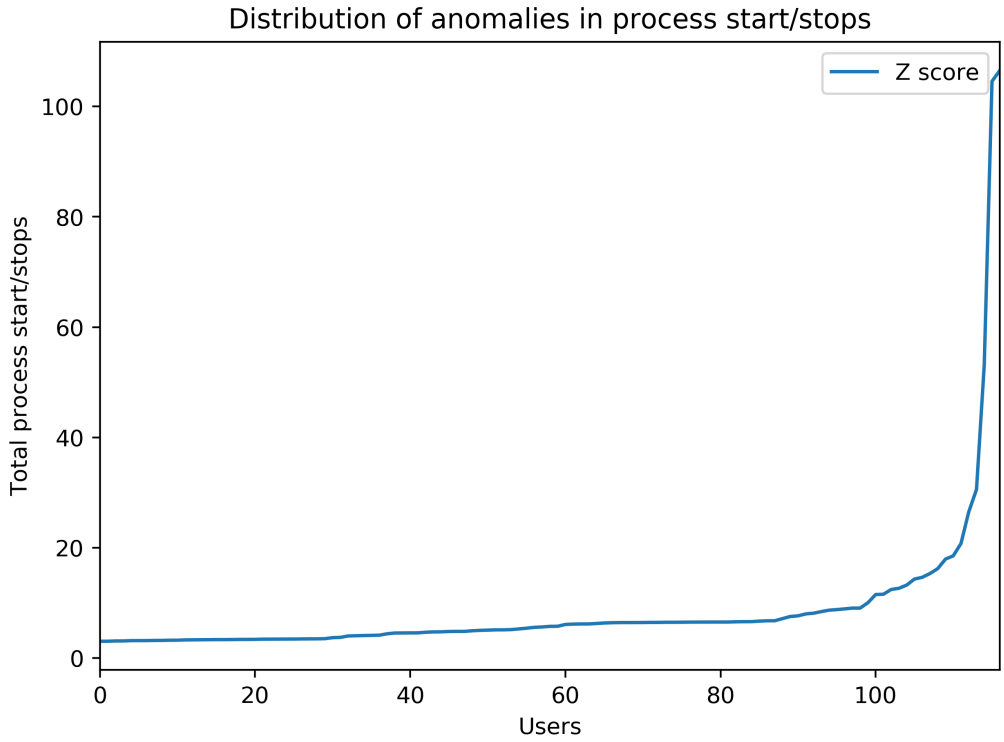

Figure 4.7: This graph shows the distribution of the anomalies for users in process start stop using the three sigma method of anomaly detection.

The distribution of the anomalies can be seen in graph 4.7 and a part of the resulting table can be seen in table 8.2. When we look at the distribution we can see that there are a few users with an extremely high number of process start and stop events. one of those users is user *C5170$@DOM1*. To check if this user can be classified as an anomaly we need to visualize his behavior. To visualize his behavior we need to following script:

```
import cyberexplorer as ce

data, name = ce.select(Location='/data/user/saves/new/')

ce.plot(x="DAY(Time)", y="Start_stop", FROM='Proc', WHERE='User == C5170$@DOM1',
average='true', dataset=data, names=name)
```

To generate the same graph in the wrapper, use:

```
plot(x:DAY(Time), y:Start_stop, from:Proc, where:User == C5170$@DOM1, average:true)
```

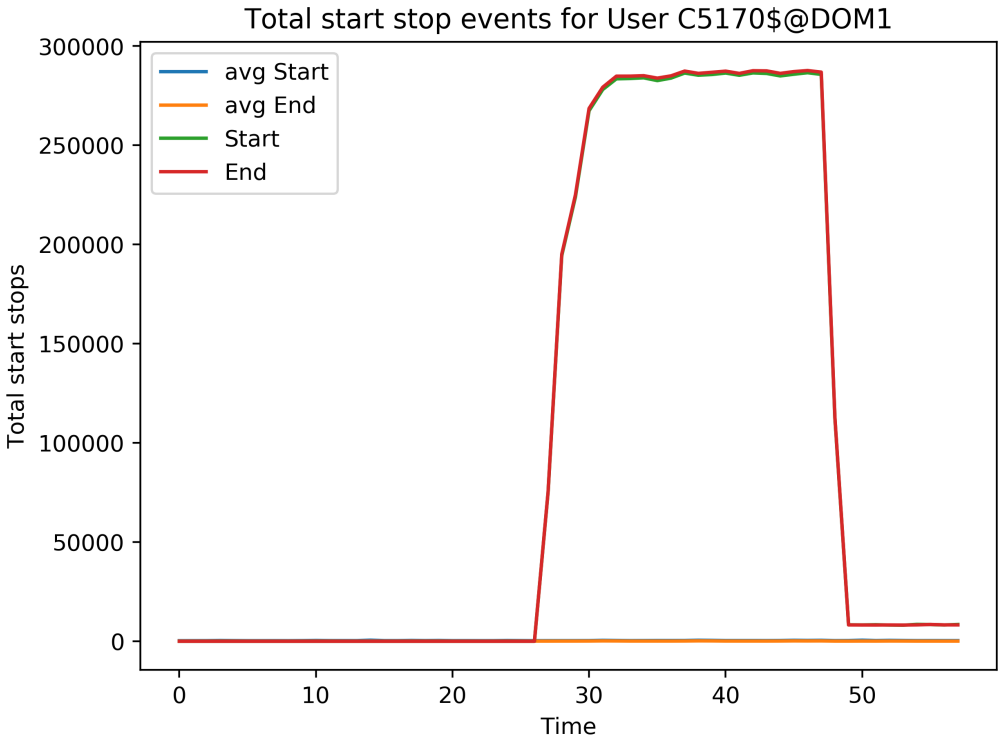

Figure 4.8: This graph shows the number of start stop events of user C5170$@DOM! compared with the average.

The resulting graph can be seen in 4.8, where User C5170$@DOM1 is represented in the lines 'Start' and 'End'. We can see that this user both starts and stops a lot of processes in the period ranging from day 28 to day 50. In this period is total start and stops is way higher than the average start and stops for each days. But apart from the extremely high number of total start and stops, this is not abnormal behavior. This user may have started and stopped a lot programs because he was running a system wide update and therefore had to start a lot of processes.

In this section we described how to perform an exploratory analysis on a data set containing cyber security log files. We searched for anomalies and visualized there behavior.

4.5 Usability

In table 8.3 the total time needed to compute the graphs is stated. Note that the distribution has not been included since those times are similar to the total time needed to generate the anomalies. We can see that al graphs generated for Dns are much faster than all the graphs generate for process start stop events. Which makes sense with how numpy bincounts works. Bincount is a linear computation meaning that the bigger the array, the longer it takes to compute. But the time it takes longer is linear. The total size of Dns lookups is approximately 0.5 gb and the total size of Process start stop events is approximately 8.5 gb. Meaning that process start stop events is about 17 times bigger than Dns lookups. When we look at the total time needed

to generate the anomalies 8.4, the times are only 5 times as big. When we look at the total time needed to generate anomalies, we can see that the time needed is about 10 times as big.

In terms of usage, the user needs to have programming experience. Both the scripting part and the wrapper requires some basic knowledge of programming. This tool is therefore restricted to users who have the required programming skills.

Chapter 5

Conclusions

The goal of this project was to create a system/tool that helped cyber security experts in gaining insight in the log files created by cyber security systems. The tool should allow an user to easily visualize and explore a data set containing the cyber security system log files. To test if our newly created tool meets the requirements we performed a case study on a data set provided by Los Alamos 7. We analyzed the data set by detecting anomalies and visualizing the anomalies. For each graph and table we timed the amount of seconds it took to generate the graph or anomalies.
In terms of speed, our tool performs quite well. It takes under 10 seconds to load a 16 gb data set into RAM. When we look at the generation of the graphs, we can see that the total time needed to generate the graphs between the two files (”Dns lookups” and ”Processes”) on average only increases by factor 4 to 5 while the process file is about 17 times as big. The same applies to the anomaly detection part, there the total time needed increases by factor 10. Only the READ operator performs significantly worse when handling bigger files. The total time needed to process the data increases by factor 30. So we can state that our tool is performing good when handling processed data, but slow when processing new data.

When we look at RAM usage, there is room for improvement. The READ operator uses a maximum of about 30 gigabytes of RAM to process the process file of the data set. The PLOT operator uses a maximum of about 62 gigabytes to generate a graph of the process file. The DETECT operator uses a maximum of about 42 gigabytes to find anomalies. The SELECT operator uses 19 gigabytes of RAM to load the Dns lookups file and process file into RAM. From this numbers we can conclude that our tool is not very efficient when it comes to RAM usage so it is best use to have about 4 times as much RAM as the dataset is big.

When we look at overall usage, this tool can be used to full extend by users which some basic programming skill. Making use of only 5 easy functions, the user can specify which part of the data in which file he wants to explore. Users who do not possess those skills can use the wrapper, although it requires some basic command line skills. The wrapper can be used to visualize and explore the data set using only 5 simple commands.

So overall we can say that our tool functions quite well, it is fast and relatively easy to use for experienced user. The only downside this tool has is that its RAM usage is far from efficient.

Chapter 6

Future extensions

Currently the wrapper works as a command line tool, this restricts to usage of the tool the user with command line experience. Some target users will not use this tool simply because they lack command line or/and programming knowledge. Making the wrapper graphical will eliminate this restriction and make it available for more users. A python package that could be used to make it graphical is Tkinter [17].
Another thing that could be improved to the program is the inter activeness. A user might generate an overview of all days and on seeing the graph wants to take a look at a period inside the graph. Currently the user has to regenerate the graph and specify the period he wants to take a look at. Optimally the user can just click on the part he wants to take a closer look at and immediately see the result.

Another part that should be improved is the RAM usage. Currently the program uses about 4 times as much RAM than the data set is big. That is not very efficient and is something that should definitely be looked into.

Currently our tool only visualizes in a 1-D color and dimensionality format (see 2.2). This limits the amount of information that can be shown in one graph. To be able to show more information in one graph we should extend to dimensionality to 2-D or even 3-D.

Also the number of anomaly detection methods in our program should be increased. Currently only 2 methods are installed as an example but that number should increase to find more anomalies. Also the complexity of those anomaly detection methods should increase to find better anomalies.

Bibliography

[1] A. D. Kent, *Cybersecurity Data Sources for Dynamic Network Research*, in Dynamic Networks in Cybersecurity, 2015.

[2] *Python Data Analysis Library*. Retrieved on 6 August 2017, from `http://pandas.pydata.org/`

[3] *NumPy*, NUMFOCUS. retrieved on 6 August 2017, from `http://www.numpy.org/`

[4] *Matplotlib*, NUMFOCUS. retrieved on 6 August 2017, from `https://matplotlib.org/`

[5] Alexander D. Kent *Comprehensive, Multi-Source Cyber-Security Events*, Dynamic Networks in Cybersecurity, Los Alamos National Laboratory (2015). 10.17021/1179829

[6] Choo, Kim-Kwang Raymond. *The cyber threat landscape: Challenges and future research directions.* Computers & Security 30.8 (2011): 719-731.

[7] Lavigne, Valrie, and Denis Gouin. *Visual Analytics for cyber security and intelligence*, The Journal of Defense Modeling and Simulation 11.2 (2014): 175-199.

[8] D'Amico, Anita, and Michael Kocka. *Information assurance visualizations for specific stages of situational awareness and intended uses: lessons learned.* Visualization for Computer Security, 2005.(VizSEC 05). IEEE Workshop on. IEEE, 2005.

[9] Fink, Glenn A., et al. *Visualizing cyber security: Usable workspaces.* Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on. IEEE, 2009.

[10] Ma, Kwan-Liu. *Cyber security through visualization.* Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60. Australian Computer Society, Inc., 2006.

[11] Goodall, John R. *Introduction to visualization for computer security.* VizSEC 2007. Springer Berlin Heidelberg, 2008. 1-17.

[12] Card, Stuart K., Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[13] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. *Anomaly detection: A survey.* ACM computing surveys (CSUR) 41.3 2009: 15.

[14] Patcha, Animesh, and Jung-Min Park. *An overview of anomaly detection techniques: Existing solutions and latest technological trends.* Computer networks 51.12 (2007): 3448-3470.

[15] Bhuyan, Monowar H., Dhruba K. Bhattacharyya, and Jugal K. Kalita. *Towards Generating Real-life Datasets for Network Intrusion Detection.* IJ Network Security 17.6 (2015): 683-701.

[16] Guzdial, Mark, et al. *Analyzing and visualizing log files: A computational science of usability*. Georgia Institute of Technology, 1994.

[17] *Tkinter*, retrieved on 6 August 2017, from `https://wiki.python.org/moin/TkInter`

Chapter 7

Appendix A

7.1 Los Alamos data set

The data set is obtained from the Los Alamost Laboratory [1]. The data set consists out of 5 files [5]:

- *User and computer authentication events*: The biggest and most significant part of the data set. This file shows all authentication events on the network. It contains the following information [5]:
  - *Time*: Time of authentication event.
  - *Source User@Domain*: The user initiating the the authentication event.
  - *Destination User@Domain*: The user the authentication event is mapping to.
  - *Source Computer*: The computer originating the authentication event.
  - *Destination Computer*: The computer the authentication event is terminating at.
  - *Authentication type*: This indicates the type of authentication occurring.
  - *Logon Type*: The type of authentication.
  - *Authentication Orientation*: How the authentication event is being used.
  - *Success or Failure*: Indicating whether the authentication type was successful or failed.
- *Process start en stop events*: This tells whether a program has started, or stopped and on which computer. It contains the following information [5]:
  - *Time*: Time of process start/stop.
  - *User*: User initiating the start/stop event.
  - *Computer*: The computer the process is initiating.

  - *Process*: The process that is starting or stopping.
  - *Start or Stop*: Indicating whether the process is starting or stopping.

- *DNS lookup events*: DNS lookup events provide an inferred connection event between two computers within the network [5]. It contains the following information.
  - *Time*. Time the DNS lookup event occurred.
  - *Source Computer*. The computer the DNS lookup event is terminating at.
  - *Resolved Computer* Computer that is resolved

- *Network flow events* This file shows the network flows from the computers, and how it is being send. [5]
  - *Time*: The time of the network flow event.
  - *Connection Duration*: Time the connection lasted.
  - *Source Computer*: The computer originating the network flow event.
  - *Source Port*: The port originating the network flow event.
  - *Destination Computer*: The computer the network flow event is terminating at.
  - *Destination Port*: The port the network flow is terminating at.
  - *Protocol*: Shows how the information is begin transferred.
  - *Packet Count*: The amount of packets send by the network flow event.
  - *Byte Count*: The size of the packet send by the network flow event.

- *Red Team compromise events*: Intentionally created events to test the the security of the computers and network within the enterprise network by a group of authorized attackers [5]. It contains the following information:
  - *Time*: The time the attack took place.
  - *User@Domain*: The “hacked“ user.
  - *Source Computer*: The computer used for the attacks.
  - *Destination Computer*: The “target“ computer.

Chapter 8

Appendix B

8.1 Anomalies in Dns lookups

Outliers	Counts	Difference from iqr
C585	1624999	1623060.0
C743	1149807	1147868.0
C1823	391719	389780.0
C5741	350885	348946.0
C3380	163298	161359.0
C17490	162708	160769.0
C1193	149849	147910.0
C22235	144637	142698.0
C2091	140060	138121.0

Table 8.1: The top source computers with the highest total of DNS lookups

8.2 Anomalies in process start/stop events

Outliers	Counts	Z score
C5170$@DOM1	11599040	106.51807313794612
C3552$@DOM1	11386283	104.56211091112432
C1685$@DOM1	5790973	53.122134753544394
C1624$@DOM1	3335063	30.543948407271866
C236$@DOM1	2893576	26.485177623547372
C492$@DOM1	2270393	20.7560011057478
C1707$@DOM1	2027488	18.522875969060653
C20481$@DOM1	1962912	17.929202356633123
C972$@DOM1	1774822	16.200013962160035

Table 8.2: The users with the most process start/stops

8.3 Total time needed to generate graphs

Graph	Time to compute
Total Dns lookups 4.1	14,8359
Dns lookups for C4653 4.2	21,3762
Dns lookups for C585 4.4	17,6269
Total process start/stops over time 4.5	80,0198
Unique processes over Time 4.6	83,9447
proc_process_C5170$@DOM1_startstop 4.8	126,7573

Table 8.3: Total time needed to generate the graphs

8.4 Total time needed to generate anomalies

Anomaly	Time to compute
Anomalies in Dns lookups 8.1	36,4436
Anomalies in process start/stop events 8.2	379,8645

Table 8.4: Total time needed to generate the graphs