Faculty 1

Study Program: Cyber Security

Tomi Jerenko

Study Project: Adversarial Machine Learning

Project Report

Cottbus, March 2019

Contents

List of Figures

Code Examples

List of Tables

List of Abbreviations and Acronyms

VM	Virtual Machine
ID	Identifier
AML	Adversarial Machine Learning
SVM	Support Vector Machine
kNN	k-Nearest Neighbors
AML	Adversarial Machine Learning
cmd	Command
ASCII	American Standard Code for Information Interrchange

1 Introduction

1.1 Motivation

In the world of computers encryption of data exchanged through the web is used to protect the sensitive information and provide its confidentiality. As the state of the art encryption algorithms are considered computationally infeasible to "break" and to recover plain text from cipher text, we try to circumvent the encryption and find another way to somehow "see" the encrypted content. By this we try to show, that encryption by itself does not provide sufficient confidentiality of data.

One method to achieve this is called "website fingerprinting". With this method we try to provide some level of confidence of what is believed to be hidden in the encrypted data of websites. The technique is to observe the patterns that appear in the encrypted packet transmission (i.e. loading time, number of packets exchanged, packet sizes...). Machine learning techniques are applied to compare the live encrypted data versus trained model and classify which encrypted data matches to its expected counterpart. We collect the raw data to train our model beforehand and thus we also know to which website it belongs to [16].

1.2 Study project

For the study project "Adversarial Machine Learning" we used the idea of website fingerprinting to obtain various information about the websites, extract features from obtained tcp/ip traces and try to identify the accessed website without breaking the encryption of exchanged data. To successfuly classify the accessed website within a given set of sites we used different machine learning techniques and approaches with different parameters. At the end we analyzed chosen machine learning techniques and compared their performance.

The project was done in 5 steps which are explained in detail in the upcoming chapters:

- Part I – Browser Automation (preparing browser for web crawling, implementing automated mechanism for web crawling, logging timestamps and connection details during crawling),
- Part II – Collecting the TCP/IP traces (extending the script to save TCP/IP traces with tcpdump, crawling 100 URLs - classes, collecting 35 traces for each website and making the error analysis of the collected data),
- Part III - Feature extraction (generating feature vectors by using the collected data and visualizing obtained features - 56 for each class),

- Part IV – Classification (informing ourselves about existing machine learning libraries, training classifier based on multiclass SVM, trying and applying other machine learning techniques - random fores and k-nearest neighbors),
- Part V – Evaluation of the classifier (evaluating the classifier by performing k-fold cross validation, documenting and visualizing the validation process, creating and analysing the confusion matrix).

1.3 Laboratory Setup

The laboratory was set up in the following way:

- The client virtual machine is running Firefox browser,
- We are observing traffic on client side with tcpdump,
- Proxy is given by another virtual machine (tinyproxy),
- Connection between client and proxy is encrypted via SSL (stunnel).

1.4 Programming languages, tools, libraries, drivers used

We ran the VMs on host system Windows 10 using VirtualBox 5.2 virtualization environment. The following operating systems, tools, programming languages, code editors, libraries etc. were used to satisfy the needs of our task:

- Windows 10 (host machine),
- Ubuntu Linux 18.04 LTS (guest virtual machine),
- Mozilla Firefox 62.0.3 (web browser) [2],
- Python 3.7.1 (programming language) [3],
- Tcpdump (tool for collecting and saving ip traces as .pcap files) [4],
- Wireshark (tool for looking into .pcap files) [5],
- Geckodriver (web browser engine that links Firefox browser with our code) [6],
- Splinter (python library for testing web applications) [8],
- Dpkt (library for python for extracting .pcap data) [9],
- Matplotlib (for visualizing data on graphs) [10],
- Libsvm (library for python for support vector machines algorithm) [11],

- Scikit-learn (python library for machine learning) [7].

The tools used by itself do not effect the final results. The only exception may be same classifiers used in different libraries due to the different implementation.

2 Browser Automation and Collecting TCP/IP Traces

Before writing and running the script we created new profile for Mozilla Firefox in which we had set up a proxy and disabled the following in the browser "about:config" settings to disable caching so that the every time a specific website is requested, all required data is transfered:

Parameter	Value
browser.cache.disk.enable	false
browser.cache.memory.enable	false
browser.cache.offline.enable	false
network.cookie.cookieBehavior	2

To automate url visits and collect their tcp/ip traces we had to write a script which performed the following steps in written order:

1. Open up a browser with specified user profile (the custom created profile with caching and cookies disabled),
2. Start and set tcpdump to listen on a specific interface "enp0s3" and save the exchanged packets (we configured our browser to use proxy and send data through this interface. This step was to avoid any other disturbances while collecting traces – e.g. OS updates etc.),
3. Automatically visit URL,
4. Wait for page to be fully loaded,
5. Log information such as visited url, time stamp before visiting, time stamp after visiting and any unexpected errors occurred during page load,
6. Save pcaps,
7. Terminate tcpdump instance,
8. Repeat from step 2 for each URL 35 times.

To hide the complexity of setting up and using geckodriver we used library Splinter [8]. Our final program shown in Code 1 was ran through terminal. The URLs visited are displayed in Figure 5. Each url was visited 35 times, that yielded total in 3500 pcap files (approximately 7 GB of data).

The naming structure "counter_formattedURLAddress_iterationNumber.pcap" was used, to distinguish between different pcap files. We also generated a log file, which contained errors in case something went wrong during page load (timeout, connection interrupted...). With its help we could later identify improper pcap files and remove them.

3 Feature Extraction and Visualization

To extract and visualize data we had to first read the pcap files and parse their content. The features in this context represent the values or properties of pcaps by which we can generalize and compare them to each other. To achieve this, we used library "dpkt" [9] for python which helped us convert and extract the contents from binary to ASCII. The Code 2 extracts the following data (which will later be used to extract final features):

- File name,
- Packet number (sequential number),
- Time stamp,
- Packet length (bytes),
- Packet direction (outgoing as minus or incoming as plus),
- Number of outgoing packets,
- Length of outgoing packets,
- Number of incoming packets,
- Length of incoming packets,
- Outgoing and incoming packet sum difference (cumulative packet length).

The program reads all pcaps in specified folder, parses their content line by line and writes it to a file. Because of some files containing unreadable/corrupted data we check if any error is thrown while parsing the file. If so, we separate the corrupted file from non-corrupted ones. To check if packet is outgoing or incoming, we check the IP value of "Target IP". This means if IP is localhost's the packet is incoming and we mark it as "+" character otherwise its outgoing packet and we mark it as "-" character. We save parsed data in CSV format.

3.1 Interpolation

Interpolation is a method of constructing new data points within the range of known data points ("shrinking" the full set to get the subset of intermediate values) so we get the same amount of data points or samples for each entity, which we can later compare.

The next step as shown in Code 3 is to extract 50 samples of packet cumulative packet length for each website visit and write it to a file. To achieve this, we have to read the values of the packet length at every $1/50$th as shown on Figure 1. This means if we

have n number of packets we have to get the value at every $i/50 * n$ part. For example, if we have to take first sample and there are 372 packets ($1/50 * 372 = 7.44$) that means we have to get the value of 44% between packet 7 and 8. In short: we calculate the absolute distance between two packets, multiply it by the percentage at which we need to extract data and add it to the previous packet's cumulative length (cumulative sum of packets size over).

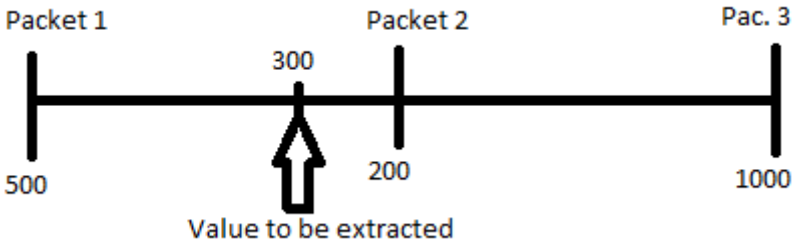

Figure 1: Packet Interpolation Example

Additionally we also write total number of packets exchanged, number of outgoing packets, number of incoming packets, total cumulative length of outgoing packets (bytes), total cumulative length of incoming packets (bytes) and their difference. This totally yields in $50 + 6 = 56$ features in total.

The program shown in Code 3 works as follows. At first we open the file with extracted packets data, we read it line by line, get values of the last record of the pcap (because last record holds total values for packet length, outgoing and incoming sum etc.), calculate 50 values of cumulative packet length for each pcap and save everything to a file.

3.2 Outlier Detection and Visualization

The next step was to visualize the extracted features so we could identify and remove corrupted data (outliers). Outliers represent the data that deviate from standard or average properties of its generalization. Removing the outliers is required to build a model for a classifier only with pure data set. To generate the graphs we used library matplotlib [10]. We filled the plots with extracted cumulative packet lengths and saved them as image. The program which fills plots with data and generates images is shown in Code 4 and example of generated graph is shown in Figure 2. There are 2 lines that stand out right away while looking at Figure 2 – the anomalies which we want to remove to get the clean data. With the help of the graphs we could identify which pcaps we need to exclude. As a result we have gotten clean data with clear and consistent patterns. Altogether we used 25 packets for each URL. The Figure 3 shows the graph after we removed anomalies and visualized 25 packets.

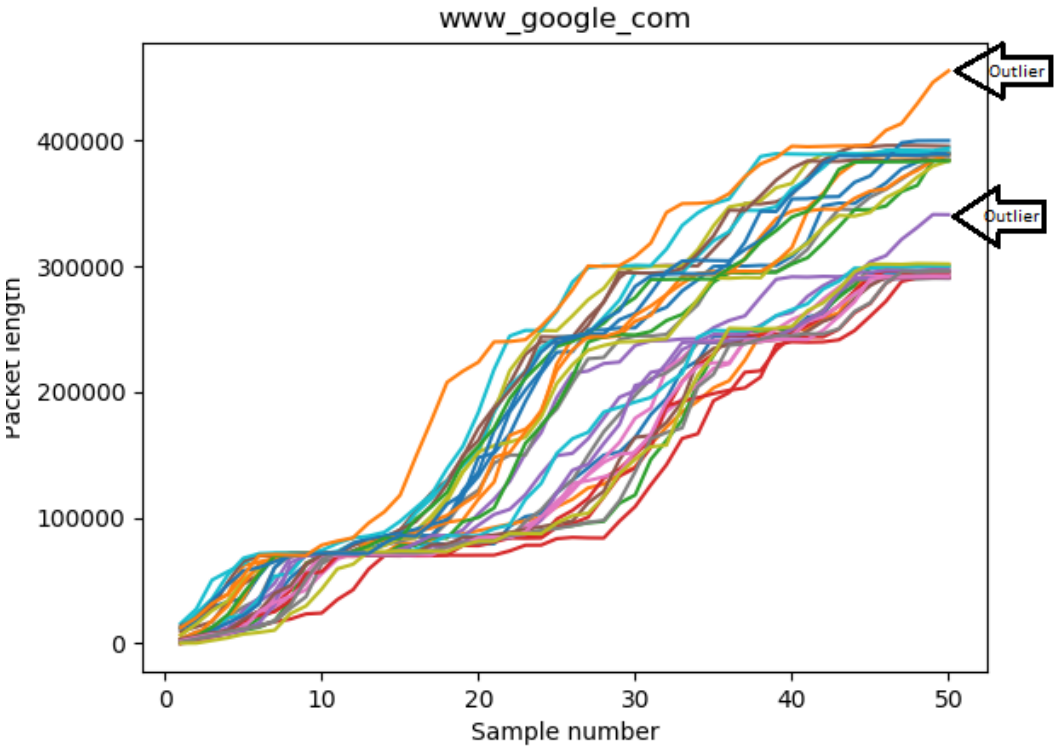

Figure 2: Visualized Features for Google Before Removing Anomalies

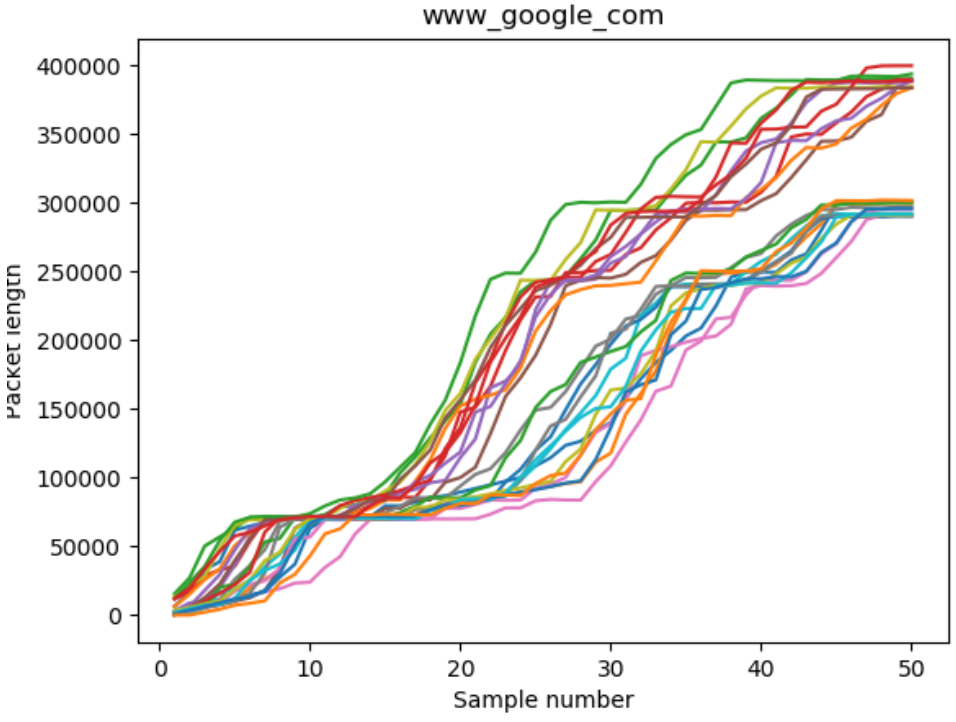

Figure 3: Visualized Features for Google After Removing Anomalies

It may happen that some websites have more outliers then others. This is due to the distance of the server, possible network outages or any other interference in between, server configuration (i.e. some websites have try to detect and block web crawlers so our requests may get blocked), service quality etc. Having this in mind we try to gather more packages so that we have enough of "clean" data left after removing outliers.

4 Development and Training of Different Machine Learning Approaches

To progress further towards our goal of our AML project the next step is to choose the classification algorithms. We will try to find the most accurate algorithm and its configuration.

For this task our chosen libraries with their classification algorithms were:

- Library "libsvm" with classifier "Multiclass SVM" [11],
- Library "scikit-learn" with classifier "kNN" [12],
- Library "scikit-learn" with classifier "Random Forest" [13].

Our first task was to put the clean data into such format which is compatible for training a model with corresponding libraries. Prerequisite for this step is to detect outliers and remove them as shown in previous chapter.

4.1 Libsvm and Multiclass SVM

Support vector machine - SVM is a classifier which discriminatively classifies data by its defined hyper plane in a two dimensional space. Multiclass SVM is therefore the same SVM classifier, but multiclass works with n dimensional spaces. The algorithm tries to find widest hyper plane between classes and then checks if test data falls inside or outside of that hyper plane. Different types of kernels can be used to generate a hyper plane. Those kernels are: Linear, Polynomial, Radial etc. [15]

To train a model which we use in libsvm's library and test the data we did the following:

- Put the data into such corresponding format which is used by the libsvm library: "<label> <index1>:<value> <index2>:<value> <indexN>:<value>". The label represents the number of the class of the same type (e.g. 1 for all Google URLs), index represents the serial number of corresponding instance's feature and value represents the actual value of some feature. In total there were 100 classes (1 for each different URL), 25 instances of each class and 56 features for each instance,
- When we have the data in the right format, we need to split it into two files - training and testing. Training file contains in total 20 instances of each class and training file contains 5 instances of each class,

- Scale the data (reason for this is normalization of data within a particular range, which can have performance benefits),
- Input a training file into algorithm to generate a model,
- Test the test data in test file against the generated model.

Code 5 shows how we read the file with containing extracted features and generated two files with mentioned libsvm format - "adm.train" which contains data for training a model and "adm.test" which contains data for testing. Figure 4 shows the structure of "adm.test" file containing test data.

```
1 1:165 2:77 3:88 4:296097 5:7355 6:303452 7:853.5 8:3497.5 9:6141.5
1 1:146 2:66 3:80 4:301874 5:6989 6:308863 7:-84.04 8:2.72 9:2095.32
1 1:216 2:98 3:118 4:394045 5:11898 6:405943 7:12147.12 8:23731.12 9
1 1:255 2:119 3:136 4:388900 5:12267 6:401167 7:11971.6 8:19790.8 9:
1 1:202 2:94 3:108 4:383715 5:10231 6:393946 7:3352.64 8:9144.64 9:1
2 1:483 2:220 3:263 4:1303285 5:23579 6:1326864 7:-644.2 8:5652.22 9
2 1:521 2:228 3:293 4:1305646 5:24193 6:1329839 7:-266.28 8:4953.72
2 1:519 2:239 3:280 4:1304082 5:24720 6:1328802 7:-448.74 8:4009.9 9
2 1:487 2:231 3:256 4:1301237 5:24342 6:1325579 7:-683.16 8:5761.84
2 1:501 2:232 3:269 4:1302720 5:24418 6:1327138 7:-661.4 8:4499.1 9:
```

Figure 4: Libsvm Ready Test Data

When we had the data in correct format the next step was to scale it and perform the validation. For scaling purpose we used libsvm's console tool for windows called "svm-scale.exe", for training "svm-train.exe" and for predicting "svm-predict.exe". To predict and get the results we've taken two different approaches:

- Validation using model. This means that we generate the model from train data and validate the test data against the model.
- K-fold cross validation. For this approach we have all data in single file and run k-fold cross validation using scaled file.

K-fold cross validation represents a method to estimate the skill of machine learning models. It works by shuffling the data set, splitting it into k groups, picking a group to use as a testing data and other groups as a training data. In the next round we take other group which hasn't been used as testing data yet, build a model with remaining groups and validate chosen group against the new model. This recursion is repeated until all groups have been used as test data. Results from each round are retained and summarized. [14]

Validation by creating a model was achieved by using the following commands:

- "svm-scale.exe -l -1 -u 1 -s range adm.train > adm.scaled.train" to scale train data. With parameters "-l" and "-u" we've set the lower and upper limit of scaled values, "-s" is for svm type, "range" represents the file with temporary values which need to be taken into account when scaling test file, "adm.train" is input file and "adm.scaled.train" is output file.
- "svm-scale.exe -l -1 -u 1 -s range adm.test > adm.scaled.test" to scale test data.
- "svm-train.exe -t 0 adm.scaled.train" to train and output a model. Parameter "-t" represents the kernel type (0 - linear) and "adm.scaled.train" is our scaled input. Command produces file "adm.scaled.train.model" which is our trained model.
- "svm-predict.exe adm.scaled.test adm.scaled.train.model adm.out". The input files are scaled test data and scaled model. The output is prediction summary.

For k-fold cross validation we ran the following commands in cmd prompt:

- "svm-scale.exe -l -1 -u 1 -s range adm.complete > adm.scaled.complete" to scale the training and testing data merged in one file.
- "svm-train.exe -t 0 -v 5 adm.scaled.complete" to preform k-fold cross validation. In our example we performed 5-fold cross validation. Parameter "-v" represents the number k. Number 5 was chosen because the training and test data previous validation type were also split into 20 for training and 5 for testing class instances.

4.2 Scikit-learn, k-NN and RandomForest

As the second and third machine learning technique of choice we've used library "scikit-learn" with its classifiers kNN and RandomForest.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the data set and uses averaging to improve the predictive accuracy and control over-fitting. [13]

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. [1]

First we've written a program, which transforms the data into required format:

- X = [[1],[2],[3],[4]] where X represents the two dimensional field of features for each class and numbers 1, 2, 3, 4 represent the feature values.
- Y = [1,1,2,3,4], where Y represents the class numbers.

Arrays X and Y were each split into two arrays with ratio 20(training data):5(test data). The training data was fit into the model and then we recursively checked if algorithm is capable of guessing right class number for each testing data. We have counted all the failed and right guesses, noted them into confusion matrix and calculated prediction percentage. Mentioned tasks were achieved with Code 6. The same code was used for RandomForest and kNN classifiers, modifying only the classifier instantiation.

5 Evaluation of the Developed Classifier

Our final step was to compare the collected results of predictions produced by classifiers, try different settings for those classifiers, visualize them in a confusion matrix and try to find which classifier produces most accurate results.

We used the following settings for the classifiers:

- Libsvm's multiclass SVM - kernel types 0, 1, 2, 3,
- Scikit-learn's random forest classifier - 2, 3, 5, 10, 30, 100 number of estimators,
- Scikit-learn's k-NN - 1, 2, 3, 4, 5 number of neighbors.

The results were visualized into 100x100 confusion matrix to determine classifier's bias towards certain classes. In a perfect 100% classification accuracy the confusion matrix would have number of tested data written across its diagonal. Confusion matrix was chosen, because it describes the performance of classification model efficiently. In the end the data set proved itself too small to actually determine bias. In our example the column header of the confusion matrix represents the class number that was predicted and the first cell of each row represents the actual class number.

5.1 Results

In Table 1 we've gathered the results for libsvm's Multiclass SVM classifier. We've tried 4 different kernel types and results showed that linear was most accurate in both 20/5 with generating a model variant and 5-fold cross validation variant. The accuracy of cross validation was 11.01% higher in total average. Linear kernel type proved itself as the most accurate and appropriate to approach problem. This is due to the type of data we are comparing and its distribution. The k-fold variant is more accurate, because it uses more testing/training groups to calculate average overall accuracy. It might have happened that our train model wasn't built from average representatives or test data contained some bias.

libsvm - Multiclass SVM		
Kernel type	20 train/5 test with model	5-fold no model
0 - linear	67.40%	80.44%
1 - polynomial	56.40%	68.56%
2 - radial basis	63%	76%
3 - sigmoid	46.80%	44.64%

Table 1: Results for Multiclass SVM (libsvm)

Table 2 shows the results of scikit-learn's random forest classifier. The higher the number of estimators was, the higher was the accuracy percentage. Time required to classify data increased as the number of estimators increased.

Scikit-learn - Random Forest	
Estimators no.	Prediction
2	82.39%
3	90.6%
5	94.2%
10	97.6%
30	98.4%
100	99.4%

Table 2: Results for Random Forest (scikit-learn)

Table 3 shows the results of scikit-learn's k-NN classifier. Number of nearest neighbors 1 has proven itself to be the most accurate. Number of neighbors 3 has proved itself to be the second most accurate. As the number of neighbors get increased, we can observe that accuracy starts dropping.

Scikit-learn - k-NN	
Neighbors no.	Prediction
1	96%
2	94.2%
3	94.6%
4	93.6%
5	93%

Table 3: Results for k-NN (scikit-learn)

As observed, we can conclude that in this specific settings, with this specific type of data the classifier "Random Forest" was the most accurate, but got slower as the numbers of estimators increased. "K-NN" was the second and the "Multiclass SVM" with the lowest accuracy was the last. For the practical scenario we therefore have to consider, if we want faster classification at the cost of accuracy or the other way around.

6 Conclusion

In the course of the study project we've got to know about the idea of website fingerprinting and using it in practice. The laboratory was set up in perfect conditions - observing isolated traffic being sent over the wire for specific website. Although we couldn't see what was the actual data being sent over the wire (because it was encrypted), we successfully guessed which website was visited by observing the patterns of sent packets. This experiment therefore showed us, that it is possible to guess with high confidence which websites are users browsing in encrypted networks such as Tor, IPSec or other encryption methods such as using SSL/TLS etc. in the given known set of websites.

In real life scenario the results would have been harder to obtain because we cannot control our targets data that is sent over the wire and we cannot isolate encrypted target packets from other sent packets. In our experiment we used only selected(100) amount of URLs, in the real world scenario this is not practical because websites are changing constantly, new ones emerge, some get deleted etc. We would need to have a lot of computing power available and constantly visit websites to check for new content, update our graphs and train new models.

To protect against website fingerprinting from a target's perspective, the target may either pad the packets sent with dummy values or visit two sites with different patterns at the same time. The first mentioned possibility of padding the packets with dummy data is to change the pattern of exchanged packets and discard the dummy data afterwards. The second option, to visit two different URLs at the same time causes attacker to observe two separate loads merged in one single pattern (because he is only able to observe encrypted traffic) thus he is unable to distinguish what the separate patterns of requested websites looks like.

A Appendix

```
from splinter import Browser
import urllib
import socket
import datetime
import subprocess
import os
import signal

# about:config
# browser.cache.disk.enable
# browser.cache.memory.enable
# browser.cache.offline.enable
# network.cookie.cookieBehavior

globalcounter = 0

def visitPage(x, counter):
    global globalcounter
    print(f"Iteration: {globalcounter}, URL: {x}, sequence: {counter}")
    localIp = socket.gethostbyname(socket.gethostname())
    temp = urllib.parse.urlparse(x)
    idstr = temp[1].replace(".", "_")+f"""_{counter}"""
    subprocess.Popen(["sudo", "-S", "tcpdump", "-i", "enp0s3", "-U", "-w",
                      f"""tcpdump/{globalcounter}_{idstr}.pcap"""])
    globalcounter += 1
    returnstr = ""
    try:
        timestampBefore = datetime.datetime.now()
        browser.visit(x)
        timestampAfter = datetime.datetime.now()
        returnstr = f"""{{
            id: "{idstr}",
            url: "{x}",
            startpoint: "{localIp}",
            timestampBefore: "{timestampBefore}",
            timestampAfter: "{timestampAfter}",
            totalRequestTime: "{timestampAfter - timestampBefore}",
            counter: "{counter}"
            }}"""
    except Exception as err:
        returnstr = f"""{{
            url: "{x}",
            startpoint: "{localIp}",
            exception: "{str(err)}",
            counter: {counter}
            }}"""
    tcpdump_pids = subprocess.check_output(["pidof", "tcpdump"]).split()
    for pid in tcpdump_pids:
        print(f"Killing tcpdump with pid: {pid}")
        os.system(f"""sudo kill {int(pid)}""")
    return returnstr

if __name__ == '__main__':
    with Browser('firefox', profile=f"""/home/mallory/.mozilla/firefox
                                        /avvu11al.admnocache""") as browser:
        subprocess.call(["sudo", "ls"])
        browser.driver.set_page_load_timeout(60)
```

```
        with open('urls.txt') as urls:
            url = [x.strip() for x in urls.readlines()]
            f = open("log.txt", "a+")
            for x in url:
                for repeat in range(35):
                    f.write(visitPage(x, repeat))
            f.close()
```

Code 1: TCP/IP Traces Collecting Program

- https://www.google.com/
- https://www.facebook.com/
- https://www.heise.de/
- https://www.b-tu.de/
- https://www.amazon.com/
- https://www.dotdash.com/
- https://www.ebay.com/
- http://www.spiegel.de/
- https://www.bahn.com/i/view/index.shtml
- https://en.wikipedia.org/wiki/Main_Page
- https://twitter.com/
- https://www.instagram.com/
- https://www.linkedin.com/
- https://www.flickr.com/
- https://www.microsoft.com/de-de/
- https://www.reddit.com/
- https://www.nytimes.com/
- https://www.jimdo.com/
- https://www.blogger.com/
- https://github.com/
- https://www.mozilla.org/
- https://myspace.com/
- https://www.theguardian.com/
- https://edition.cnn.com/us
- https://www.paypal.com/
- https://www.huffingtonpost.de/
- https://www.dropbox.com/
- https://www.xing.com/
- https://www.owasp.org/index.php/Main_Page
- https://www.skype.com/
- https://www.springer.com/
- https://www.oracle.com/index.html
- https://www.debian.org/
- https://www.android.com/
- https://www.office.com/
- https://www.alexa.com/
- https://www.netflix.com/
- https://opensource.org/
- https://www.icann.org/
- https://www.oracle.com/sun/index.html
- https://www.snapchat.com/
- https://www.symantec.com/
- https://www.avira.com/
- https://us.norton.com/
- http://www.aboutads.info/
- https://www.microsoft.com/
- https://www.python.org/
- https://siteorigin.com/
- http://ewebdevelopment.com/
- https://www.justice.gov/
- https://www.warnerbros.com/
- https://java.com/
- https://www.playstation.com/
- https://www.iso.org/home.html
- https://moodle.org/
- https://web.de/
- https://www.gmx.net/
- https://www.msn.com/
- https://www.unicef.org/
- https://www.nist.gov/
- https://www.dmca.com/
- http://taz.de/
- http://www.faz.net/aktuell/
- https://www.whatsapp.com/
- https://www.samsung.com/us/
- http://www.kicker.de/
- https://translate.google.de/
- https://www.transfermarkt.de/
- https://www.sparkasse.de/
- https://www.deutsche-bank.de/
- https://www.ing-diba.de/
- https://www.dkb.de/
- https://www.commerzbank.de/
- https://www.dhl.de/
- https://www.wolframalpha.com/
- https://www.history.de/
- https://www.sueddeutsche.de/
- https://www.bild.de/
- https://www.chefkoch.de/
- http://derbiokoch.de/
- https://www.tagesschau.de/
- https://www.wetteronline.de/
- https://www.wetter.com/
- https://prezi.com/
- https://www.mindmeister.com/
- https://www.alternate.de/
- https://www.notebooksbilliger.de/
- https://de.napster.com/
- https://www.deezer.com/de/
- https://www.spotify.com/
- https://soundcloud.com/
- https://www.deutschlandradio.de/
- http://www.surfmusik.de/
- https://www.kino.de/
- https://www.cinema.de/
- https://www.imdb.com/
- http://www.netzkino.de/
- https://www.watchbox.de/
- https://www.expedia.de/
- https://www.opodo.de/

Figure 5: Visited URLs

```
import dpkt
import socket
import os
import shutil

f = open("pcapData.txt", "a+")
if os.path.isfile("pcapData.csv") == False:
    f.write("Filename,Packet number,Timestamp,Length,Direction,
             Outgoing packets count,Outgoing packets sum,
             Incoming packets count,Incoming packets sum,
             Packet difference\n")

removedFilesCount = 0
currentFile = 1
```

```
filesCount = len(os.listdir("."))
for filename in os.listdir("."):
    print(f"Progress: {filesCount}/{currentFile} => {filename}")
    currentFile += 1
    if filename[len(filename)-5:] != ".pcap":
        continue
    pcfile = open(filename, "rb")
    pcap = dpkt.pcap.Reader(pcfile)
    counter = 0
    ioDataSum = 0
    outgoingPacketsCount = 0
    incomingPacketsCount = 0
    outgoingPacketsSum = 0
    incomingPacketsSum = 0
    direction = ""
    buflen = 0
    contentToWrite = ""
    try:
        if filename[-7:] == "_1.pcap":
            raise Exception()
        for ts, buf in pcap:
            counter += 1
            eth = dpkt.ethernet.Ethernet(buf)
            ip_hdr = eth.data
            buflen = int(len(buf))
            if socket.inet_ntoa(ip_hdr.dst) == "192.168.0.2":
                incomingPacketsCount += 1
                incomingPacketsSum += buflen
                ioDataSum += buflen
                direction = "+"
            else:
                outgoingPacketsCount += 1
                outgoingPacketsSum += buflen
                ioDataSum -= buflen
                direction = "-"
            contentToWrite += f"{filename},{counter},{ts},{buflen},
                        {direction},{outgoingPacketsCount},
                        {outgoingPacketsSum},{incomingPacketsCount},
                        {incomingPacketsSum},{ioDataSum}\n"
        if contentToWrite == '':
            raise Exception()
        f.write(contentToWrite)
        pcfile.close()
    except:
        pcfile.close()
        shutil.move(
            f"""{filename}""", f"""../removedfiles/{filename}""")
        removedFilesCount += 1
print(f"----------Total {removedFilesCount} files removed----------")
f.close()
```

Code 2: Pcap Parser Program

```
import csv
from itertools import islice

f = open("extractedFeatures.txt", "a+")
fileCount = 0
```

```
def writeHead(pcapList):
    global fileCount
    fileCount += 1
    f.write(f"""#name: {pcapList[0]}\n""")
    f.write(f"""{pcapList[1]}\n""")
    f.write(f"""{pcapList[5]}\n""")
    f.write(f"""{pcapList[7]}\n""")
    f.write(f"""{pcapList[9]}\n""")
    f.write(f"""{pcapList[6]}\n""")
    f.write(f"""{pcapList[8]}\n""")
    f.write(f"""#values: \n""")

def writeValue(value):
    f.write(f"""{value}\n""")

def sampler(sampleSize, pcapList):
    for x in range(1, sampleSize+1):
        listIndex = int(len(templist)*x/sampleSize)

        if x == sampleSize:
            writeValue(templist[-1][9])
        elif len(templist) \% sampleSize == 0:
            writeValue(templist[listIndex-1][9])
        else:
            result = 0
            biggerNum = 0
            smallerNum = 0

            if int(templist[listIndex-1][9]) > int(templist[listIndex][9]):
                biggerNum = int(templist[listIndex-1][9])
                smallerNum = int(templist[listIndex][9])
            else:
                biggerNum = int(templist[listIndex][9])
                smallerNum = int(templist[listIndex-1][9])

            distance = abs(biggerNum-smallerNum)
            modInPercent = len(templist) / sampleSize - \
                len(templist) // sampleSize
            result = round(smallerNum + distance * modInPercent, 2)

            writeValue(result)

with open('./tcpdump/pcapData.txt') as csvfile:
    i = 0
    reader = csv.reader(csvfile)
    templist = []
    sampleSize = 50
    itemsCounter = 0

    for row in reader:
        if i == 0 or i == 1:
            if i == 1:
                templist.append(row)
                itemsCounter += 1
            i += 1
            continue

        if templist[-1][0] != row[0]:
            itemsCounter += 1
```

```
            print(templist[-1][0])
            writeHead(templist[-1])
            sampler(sampleSize, templist)
            templist.clear()
            templist.append(row)
        else:
            templist.append(row)

    print(templist[-1][0])
    writeHead(templist[-1])
    sampler(sampleSize, templist)
    print(f"Different pcaps file count: {itemsCounter}")
    print(f"Pcap file count: {fileCount}")
    f.close()
```

Code 3: Interpolation Program

```
import matplotlib.pyplot as plt
import numpy as np

def getCounter(counter):
    pcapId = str(counter)
    if len(pcapId) == 1:
        pcapId = "00"+pcapId
    elif len(pcapId) == 2:
        pcapId = "0"+pcapId
    return pcapId

def extractName(pcapName):
    name = pcapName[13:]
    counter = 0
    for s in reversed(name):
        if s == "_":
            break
        counter += 1
    counter += 1
    name = name[0:-counter]
    return name

with open('./extractedFeatures.txt') as featuresFile:
    pcapCounter = 0
    takeValue = False
    pcapInProgress = ""
    xAxis = [i for i in range(1, 51)]
    yAxis = []
    plt.xlabel('Sample number')
    plt.ylabel('Packet length')

    for line in featuresFile.readlines():
        if line[:5] == "#name":
            if pcapInProgress == "":
                pcapInProgress = extractName(line)
                plt.title(pcapInProgress)
                continue
            elif extractName(line) != pcapInProgress:
                print(f"Working on: {pcapInProgress}")
                plt.savefig(
                    f"Visualisation/{getCounter(pcapCounter)}_
```

```
                {pcapInProgress}")
                pcapCounter += 1
                pcapInProgress = extractName(line)
                plt.cla()
                plt.title(pcapInProgress)
                plt.xlabel('Sample number')
                plt.ylabel('Packet length')
                yAxis = []

            if len(yAxis) != 0:
                plt.plot(np.array(xAxis), np.array(yAxis))
            yAxis = []
            takeValue = False
        elif line[:7] == "#values":
            takeValue = True
        elif takeValue == True:
            yAxis.append(float(line))
    plt.savefig(f"Visualisation/{getCounter(pcapCounter)}_{pcapInProgress}")
```

Code 4: Visualization Program

```
train = open("adm.train", "a+")
test = open("adm.test", "a+")

def extractName(pcapName):
    name = pcapName[13:]
    counter = 0
    for s in reversed(name):
        if s == "_":
            break
        counter += 1
    counter += 1
    name = name[0:-counter]
    return name

def writeToFile(counter, text):
    text += "\n"
    train.write(text)
    if counter < 20:
        train.write(text)
    else:
        test.write(text)

with open('./extractedFeatures.txt') as extractedFeatures:
    pcapCounter = 0
    pcapInProgress = ""
    lineToWrite = ""
    write = False
    featureCounter = 1
    urlCounter = 1
    for line in extractedFeatures:
        if line[:5] == "#name":
            if pcapInProgress == "":
                pcapInProgress = extractName(line)
                lineToWrite = f"{urlCounter}"
                continue
            elif extractName(line) != pcapInProgress:
                writeToFile(pcapCounter, lineToWrite)
```

```
                pcapCounter = 0
                pcapInProgress = extractName(line)
                urlCounter += 1
                featureCounter = 1
                lineToWrite = f"{urlCounter}"
                continue
            writeToFile(pcapCounter, lineToWrite)
            pcapCounter += 1
            featureCounter = 1
            lineToWrite = f"{urlCounter}"

        if "#name" in line or "#values" in line:
            continue

        lineToWrite += f" {featureCounter}:{line[:-1]}"
        featureCounter += 1
    writeToFile(pcapCounter, lineToWrite)
    train.close()
    test.close()
```

Code 5: Program for Conversion of Features to Libsvm Format

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

# first value in array is always class number
train = []
test = []


def extractName(pcapName):
    name = pcapName[13:]
    counter = 0
    for s in reversed(name):
        if s == "_":
            break
        counter += 1
    counter += 1
    name = name[0: -counter]
    return name


def buildArrays(counter, dataArray):
    global test
    global train
    if counter < 20:
        train.append(dataArray)
    else:
        test.append(dataArray)


with open('./extractedFeatures.txt') as extractedFeatures:
    tempArr = []
    pcapCounter = 0
    pcapInProgress = ""
    urlCounter = 1
```

```
    tempArr.append(f"{urlCounter}")
    for line in extractedFeatures:
        if line[:5] == "#name":
            if pcapInProgress == "":
                pcapInProgress = extractName(line)
                continue
            elif extractName(line) != pcapInProgress:
                buildArrays(pcapCounter, tempArr)
                tempArr = []
                pcapCounter = 0
                pcapInProgress = extractName(line)
                urlCounter += 1
                tempArr.append(f"{urlCounter}")
                continue
            buildArrays(pcapCounter, tempArr)
            tempArr = []
            tempArr.append(f"{urlCounter}")
            pcapCounter += 1

        if "#name" in line or "#values" in line:
            continue
        tempArr.append(float(line))
    buildArrays(pcapCounter, tempArr)
    tempArr = []

trainDataArrData = []
trainDataArrClassNum = []
for x in train:
    trainDataArrData.append(x[1:len(x)])
    trainDataArrClassNum.append(int(x[0]))

testDataArrData = []
testDataArrClassNum = []
for x in test:
    testDataArrData.append(x[1:len(x)])
    testDataArrClassNum.append(int(x[0]))

# prediction part
#neigh = KNeighborsClassifier(n_neighbors=2)
neigh = RandomForestClassifier(n_estimators=5)
neigh.fit(trainDataArrData, trainDataArrClassNum)
i = 0
successful = 0
diffusionMatrix = [[0 for x in range(100)] for y in range(100)]
while i < len(testDataArrData):
    predicted = neigh.predict([testDataArrData[i]])
    if testDataArrClassNum[i] != predicted:
        diffusionMatrix[testDataArrClassNum[i]-1][int(predicted[0])] += 1
        print(
            f"Iteration: {i} classified as {predicted};
                should be {testDataArrClassNum[i]}")
    else:
        successful += 1
        diffusionMatrix[testDataArrClassNum[i] -
                        1][testDataArrClassNum[i]-1] += 1
    i += 1

print(f"Prediction: {successful/len(testDataArrData)*100}")
i = 0
for x in diffusionMatrix:
```

```
    if i == 100:
        i = 1
        print(f"\n")
    print(f"{x},")
```

Code 6: Program for Conversion to Scikit-learn Format and Confusion Matrix

References

[1] 1.6. nearest neighbors — scikit-learn 0.20.3 documentation. https://scikit-learn.org/stable/modules/neighbors.html#neighbors. Accessed: 7.3.2019.

[2] Download firefox — free web browser — mozilla. https://www.mozilla.org/en-US/firefox/new/. Accessed: 2.5.2019.

[3] Download python | python.org. https://www.python.org/downloads/. Accessed: 2.5.2019.

[4] Tcpdump/libpcap public repository. https://www.tcpdump.org/. Accessed: 2.5.2019.

[5] Wireshark · go deep. https://www.wireshark.org/. Accessed: 2.5.2019.

[6] Releases · mozilla/geckodriver · github. https://github.com/mozilla/geckodriver/releases. Accessed: 2.5.2019.

[7] scikit-learn: machine learning in python — scikit-learn 0.20.3 documentation. https://scikit-learn.org/stable/. Accessed: 2.5.2019.

[8] Splinter — splinter 0.10.0 documentation. https://splinter.readthedocs.io/en/latest/. Accessed: 19.1.2019.

[9] dpkt — dpkt 1.9.2 documentation. https://dpkt.readthedocs.io/en/latest/. Accessed: 19.1.2019.

[10] Matplotlib: Python plotting — matplotlib 3.0.2 documentation. https://matplotlib.org/. Accessed: 19.1.2019.

[11] Libsvm – a library for support vector machines. https://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed: 6.3.2019.

[12] sklearn.neighbors.kneighborsclassifier — scikit-learn 0.20.3 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed: 6.3.2019.

[13] 3.2.4.3.1. sklearn.ensemble.randomforestclassifier — scikit-learn 0.20.3 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed: 6.3.2019.

[14] A gentle introduction to k-fold cross-validation. https://machinelearningmastery.com/k-fold-cross-validation/. Accessed: 7.3.2019.

[15] Chapter 2 : Svm (support vector machine) — theory – machine learning 101 – medium. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. Accessed: 7.3.2019.

[16] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website Fingerprinting at Internet Scale. *NDSS*, 2016.