INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

FORENSICS CYBER-SECURITY

MEIC, METI

Lab Assignment III

Traffic Analysis

2018/2019

nuno.m.santos@tecnico.ulisboa.pt

Introduction

This assignment aims to improve your skills on traffic analysis. It is divided into two groups of exercises. In the first group, you have to analyze two network traces using a conventional packet analyzer, e.g., Wireshark, and answer a few questions about the collected traffic. In the second group, you will be exposed to advanced network forensic techniques, namely for website fingerprinting over the Tor network. Do not be intimidated by the lengthy description of this group as most of its text aims to help you solve this exercise without major hurdles by providing you with some necessary background about machine learning. Although it is not strictly necessary, we recommend you to use the Kali Linux distribution running on a forensically sound virtual machine in order to perform your analysis.

1 Network trace analysis

In this group, you have to analyze two independent packet traces which capture two scenarios of attacks that were performed over the network. It is your job to study each trace and understand what's going on. Justify all your answers based on evidence that can be retrieved from the respective traces.

Exercise 1: Consider the network trace lab3-pcap1.pcap, which can be downloaded from `http://turbina.gsd.inesc-id.pt/csf1819/lab3-pcap1.pcap`, and answer the following questions:

1. Indicate the complete URL of the original web request that led to the client being compromised.
2. What file type was requested in the final web request to the malicious server? Pick one:

   a. Windows executable
   b. JavaScript
   c. PDF
   d. Word document
   e. GIF

3. What is the number of the first frame that indicates that the client has been compromised?
4. At one point, the malicious server sends a malicious file to the client. What type of file it is? (Answer a, b, c, d, or e)

   a. Windows executable
   b. JavaScript
   c. PDF
   d. Word document
   e. GIF

5. What is the SHA1 hash of the malicious file?
6. What vulnerable software has been exploited (in the following format, Firefox 3.5, Firefox 3.6, Word 2010, IE7, Safari2, Chrome2, AdobeReader, IE9)?
7. When the capture ends, is the client still connected to the malicious attacker?
8. Can you indicate the corresponding CVE security bulletin that covers the vulnerability that was exploited here (answer in form of CVE-$year-$number)? Pick one of the following options:

   a. CVE-2011-3887

b. CVE-2010-0249

c. CVE-2010-3178

d. CVE-2010-1127

e. None of the above

9. From the capture, it is clear that the attacker gets a certain form of access (i.e. the interface), what (type of) access does the attacker "get" on the client?

Exercise 2: Download the network trace lab3-pcap2.pcap from `http://turbina.gsd.inesc-id.pt/csf1819/lab3-pcap2.pcap`, analyze it, and answer the following questions:

1. What is the frame that indicates that something strange might be going on?

2. What tool is generating this traffic?

3. What does this frame constitute the beginning of? (Answer a,b,c,d or e)

   a. TCP Scan

   b. Ack Scan

   c. UDP Scan

   d. Ping Scan

   e. Syn Flood

4. The attacker then runs two scans beginning just after six minutes and 24 minutes into the trace. However, these traces weren't to his/her liking as they were too slow. On the following scans, a switch was removed from the command to improve the speed, what was this switch (just the letters, case-sensitive)?

5. What switch was added to the final scan (case-sensitive)?

2 Website fingerprinting over Tor

In this section, you will be performing a website fingerprinting attack over Tor. Before you start, you must download the dataset and classifier available at `http://turbina.gsd.inesc-id.pt/csf1819/wf.tar.gz`. Your main task is to build a feature set which allows for a better identification of websites browsed over Tor's encrypted tunnel. Next, we introduce you to all the concepts and techniques that you need to master in order to solve this exercise.

2.1 Background on website fingerprinting

In networking systems, traditional encryption can obscure only the content of communications and does not hide metadata such as the size and direction of traffic over time. Low-latency anonymous communication tools, such as Tor, are able to obscure both the content and destination of communications by routing encrypted traffic through a number of network nodes. However, in order to support instant-messaging and an interactive web browsing experience, Tor does not significantly modify the shape of traffic patterns and closely preserves the packet timing and volume characteristics which are exclusive to a given web page. This fact renders Tor vulnerable to multiple traffic analysis attacks, commonly denominated *website fingerprinting*, aimed at unveiling the browsing habits of users.

Website fingerprinting attacks can be formulated as a classification problem where an adversary aims to unveil whether a user browsed a given web page (e.g., a site distributing child pornography). This classification problem is formulated in the literature according to two different settings of the attack which are known by the names of *closed world* and *open world*.

- **Closed-World:** The adversary monitors *n* web pages, and the user is assumed to be able to browse one of *n* web pages monitored by the adversary. To set up a website fingerprinting attack, the adversary first gathers a number of traffic traces from each of the *n* web pages by performing web requests. Then, the adversary extracts a feature set from the high-level description of each browsed web page and trains a model to identify the website using these features. When a user visits a web page, the adversary gathers the corresponding traffic trace and uses the trained model to make a prediction over which of the *n* web pages the user has visited. (In the following sections, we explain the concepts of feature set and training a model.)

- **Open-World:** A website fingerprinting attack in this setting follows a similar methodology but represents a more realistic attack. In this case, the adversary monitors *n* web pages, allows a user to visit a universe of non-monitored web pages, and wishes to assess whether (and which) a monitored page has been visited.

In this exercise, you will perform both closed- and open-world web fingerprinting attacks using a dataset that contains traces of Tor traffic. We provide you with sample Python code that demonstrates how to perform these attacks using a basic feature set. Your job will be to enhance this feature set in order to improve the effectiveness of the attacks and answer some questions based upon your findings.

2.2 Dataset

The provided dataset incorporates a list of 90 instances for 100 monitored pages, as well as 1 instance each of 9,000 non-monitored pages. These 100 monitored pages were compiled from a list of blocked web pages from China, the UK, and Saudi Arabia and are listed in `wf/data/knnlist.txt`. These include pages ranging from adult content, torrent trackers, and social media to sensitive religious and political topics. The list of 9,000 non-monitored pages was extracted from Alexa's top 10,000, in order, excluding those pages that are in the list of monitored pages by domain name.

Dataset structure: Rather than providing raw packet traces, the dataset comprises pre-processed encrypted Tor cell sequences corresponding to page loads. Keep in mind that Tor sends upstream and downstream data within 512-byte long encrypted cells so as to obfuscate traffic patterns. In this dataset, each cell sequence for a monitored site is stored in a file with name `W-L`, where `W` is the webpage's id, and `L` indicates a page load instance (e.g., `0-4` is the fourth page load of monitored webpage `0`). Cell sequences for non-monitored sites are stored in files with name `W`, corresponding to the webpage's id only. Each of these files contains, per row: t_i`<tab>`s_i with t_i and s_i indicating the time and size of the i-th cell. Since each Tor cell has a fixed sized of 512 bytes, s_i will only provide direction information: -1 or +1. The value +1 means an outgoing cell, i.e., transmitted from the client to the website.

2.3 Classifier

To perform website fingerprinting attacks using the provided dataset, you can use the classifier algorithm implemented by the Python code enclosed in file `wf.tar.gz`. The main program is `classify.py`, which essentially works in two stages: *training* and *testing*. In the training stage, the classifier uses 80% of existing traces (*training set*) to build a classification model of the traffic for each website. If the website is monitored, then the classifier creates a specific class just for that page, otherwise, it labels that cell sequence under a single class for all non-monitored websites. In the testing stage, the classifier uses 20% of the remaining flows (*testing set*) to launch the attack, i.e., guess which class of website traffic these flows belong to. By comparing the classifier output with the ground truth and checking how many flows of the testing set have been correctly identified, we can evaluate the quality of the classifier.

In general, there are many ways to implement a classifier. In this exercise we adopt the k-Nearest Neighbours (*k*-NN) classifier, which is a supervised machine learning algorithm. Intuitively, our *k*-NN classifier measures the distance between a cell sequence P of the training set and its k nearest neighbors of the training set (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor. If $k > 1$, P is classified by a majority vote of its neighbors. Figure 1 illustrates this idea for $k = 3$ and two different classes (blue squares and red triangles).

k-NN uses a distance metric d to describe how similar two cell sequences are. This distance is computed according to the value of different characteristics that describe a data point. Such characteristics are often called *features*. A feature can be something as simple as "the total number of cells within a cell sequence". It is possible to consider multiple features for computing this distance by using a *feature set*. In a feature set $F = f_1, f_2, ..., f_n$, each feature is a function f which takes in a cell sequence P and computes $f(P)$. Conceptually, we select each feature such that cell sequences from the same webpage are more likely to have similar features than cell sequences from different webpages. You will be using a modified version of k-NN which adjusts the weight of each feature according to its contribution in distinguishing members of different classes. In particular, more informative features will have a larger weight in distance calculations and have a larger impact in classification results. As you may have guessed by now, the quality of a classifier will greatly depend on what kind of features are used and what respective weights are assigned to them.

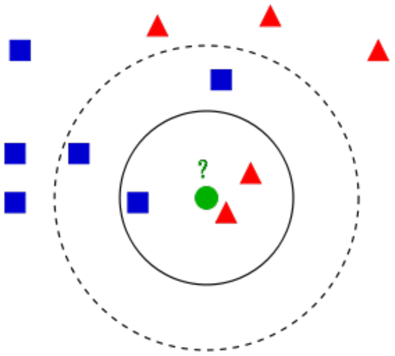

Figure 1: Example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Thus, more formally, our k-NN classifier works as follows. It starts with a set of training points $S_{train} = P_1, P_2, ..., P_n$, each point being a cell sequence. Let the class of P_i be $c(P_i)$; a class labels a monitored website or a group of non-monitored websites. Given a testing point $P_{test} \in S_{test}$, the classifier computes the distance d $(P_{test}$,$P_{train})$ for each $P_{train} \in S_{train}$. The algorithm then labels P_{test} based on the classes of the k closest training points. When the k neighbours of a testing point differ in class assignment, the k neighbours act as voters in a majority scheme with their votes weighted by their distance to the testing point. In the classifier provided in this lab, if any of the k neighbours disagree, it assigns the testing point to the non-monitored class, even if all of the k neighbours pointed to monitored pages (but different ones). That is, a testing point is assigned to a specific monitored page if and only if all k neighbours agree on that specific label.

Classifier setup: The classifier runs in holdout method with 80% of all monitored page samples used for training (7200 samples) and 20% used for testing (1800 samples), both in closed and open-world settings. For open-world, the percentage of non-monitored page samples used in training is configurable (up to a max. of 80% - 7200 samples), while 20% (1800 samples) of the non-monitored pages are always used for testing. The model is not trained with any of the non-monitored pages included in the test set.

Performance metrics: When analyzing the success of your website fingerprinting attacks, you should focus on different metrics depending on whether the attack is conducted in open or closed-world mode:

- **Closed-World:** In a closed-world attack, a user is assumed to be able to visit a particular website among n monitored websites. When measuring the effectiveness of this attack, we are mostly interested in its Accuracy (ACC), that is, the fraction between the number of correct predictions over the total number of predictions. An effective attack should exhibit a large accuracy, meaning that the majority of accesses to monitored pages are successfully identified.

- **Open-World:** In an open-world attack, a user is assumed to be able to browse either a monitored or non-monitored page. Here, we are interested in analyzing the True Positive Rate (TPR) and False Positive Rate (FPR) trade-off of the classifier. You may interpret TPR as the probability that a monitored page is classified as the correct monitored page, and FPR as the probability that an non-monitored page is incorrectly classified as a monitored page. Ideally, a successful attack should strive for: a large TPR, in order to ensure that most accesses to monitored sites are actually identified; a low FPR, in order to minimize the risk of labeling accesses to non-monitored pages as monitored pages (for instance, incorrectly blaming an innocent user for allegedly accessing a webpage distributing child-pornography).

2.4 Performing a website fingerprinting attack

Before performing your first website fingerpriting attack, download the code and dataset from `http://turbina.gsd.inesc-id.pt/csf1819/wf.tar.gz`. Then, install the required Python packages, and build the classifer code:

```
$ cd wf
$ pip install -r requirements.txt
$ cd code/attacks && make && cd -
```

Extracting traffic features: You may use `wf/code/fextractor.py` to extract features from the dataset and save the output to a folder. You may find an example below:

```
$ cd code
$ python fextractor.py --traces ../data/cells/ --out ../data/features
```

where `--traces` should point to the folder containing the Tor cells dataset, and `--out` to the folder where to save the extracted features.

Classify the dataset: You may use `wf/code/classify.py` to issue a website fingerprinting attack over the provided Tor cell sequences. Below, you may find an example for conducting both a closed and open-world attack:

```
#Closed-world attack
$ python classify.py --features ../data/features --world closed --k 5 --out
    res_closed_k5
```

where `--features` points to the folder containing the features obtained in the previous step, `--world` states the attack mode we are interested in, `--k` selects the number of neighbours used when predicting a label, and `--out` points to the location where to save the file containing the classification results.

```
#Open-world attack
$ python classify.py --features ../data/features --train 0.8 --world open
   --k 5 --out res_open_k5_1800
```

where `--features` points to the folder containing the features obtained in the previous step, `--train` gives the percentage of non-monitored samples to include in the training set, `--world` states the attack mode we are interested in, `--k` selects the number of neighbours used when predicting a label, and `--out` points to the location where to save the file containing the classification results.

You will notice that you will get poor results after running this example. This is due to the fact that the feature set being used by the classifier is not sufficiently good to correctly distinguish amongst different page loads.

2.5 Questions

Based on the provided dataset and classifier code, perform the following exercises. Note that you **must not** alter any of the hardcoded parameters of the classifier in your experiments, such as the proportion 80% / 20% between training and testing set sizes.

Exercise 1: Improve the accuracy result of the closed-world attack, when using *k*=5 neighbours, by augmenting your feature set. You can do this by generating multiple descriptive characteristics of Tor cell sequences, for instance based on their time and direction. You may increase the feature set of the classifier by adding new features in `wf/code/fextractor.py`'s `extract` function. Read the comments in the code for more details on how to add new features. Aim for at least 70% accuracy. Indicate: (a) the selected feature set, and (b) the accuracy of your classifier using this feature set. Give an account of other features you have tried before converging into this feature set.

Exercise 2: Using your best feature set, answer the following questions:

k	**1**	**2**	**5**	**10**	**15**
ACC					

Table 1: Accuracy for different values of *k*.

k **nm%**	**1**	**2**	**5**	**10**	**15**
0.1					
0.2					
0.4					
0.6					
0.8					

Table 2: Trade-off between TPR and FPR for different values of *k* and percentage of non-monitored pages in the training set.

1. Analyze the influence of the choice of *k* neighbours in the accuracy of the closed-world attack. First, build a table similar to Table 1 where each cell represents the accuracy obtained by the classifier when running the closed-world attack with $k = 1, 2, 5, 10, 15$. Can you spot a trend in the accuracy of the classifier as *k* increases? Explain the variations observed.
2. Analyze the influence of the number of *k* neighbours and the percentage of non-monitored sites included in the training set (*nm*) in the TPR/FPR trade-off of the open-world attack. First, build a table similar to Table 2 where each cell shall be filled with the TPR and FPR obtained by the classifier when running the open-world attack with $k = 1, 2, 5, 10, 15$ and $nm = 0.1, 0.2, 0.4, 0.6, 0.8$. Can you spot any trend in the TPR/FPR trade-off alongside the variation of *k* and *nm*? Justify.
3. According to your results in Table 2, which configuration leads to a more successful attack? Do you think an attack with such a TPR / FPR trade-off may be effective in practice? Justify.

Deliverables

Write a forensic report containing your answers to these questions. The deadline for this work is December 14^{th}. Until then, you must upload to Fenix a compressed zip file containing three deliverables:

- **Report**: A document in which you answer the aforementioned questions. We recommend that you use the template that can be downloaded from the course website. For the first group, you must identify all relevant evidentiary items obtained from the network traces which support your claims.
- **Evidence Artifacts:** All relevant evidence artifacts recovered during the forensic analysis of the exercises of Section 1. Please make sure that the respective file names and MD5 values are indicated in the report. For Section 2 exercises it is not necessary to provide evidence artifacts.
- **Auxiliary Items:** Programs, scripts, and additional documents that you have produced during the investigation which are important to justify your results must also be included. In particular, for Section 2, include the code of your feature extractor script (`wf/code/fextractor.py`).

Good luck!