Identifying a Mobile Device Using DNS Fingerprinting

In our research we examined the ability of DNS queries to uniquely identify mobile device. Network communication of mobile device is a valuable source of information about the device. From this communication it is possible to infer installed applications and patterns of user behavior. The goal of this study is to observe DNS communication and extract specific features from it that can be used to identify the given mobile device within unknown network data (e.g. in digital forensics).

In this case study, we show how the DNS mobile fingerprint can be created and the data mining method that can be used for the comparison of these fingerprints.

Contents

Problem Description

DNS system typically translates a domain name, e.g., www.vutbr.cz, to an IP address, e.g., 147.229.2.90. Domain name translation is required for any communication over IP. Before an IP datagram is sent, a DNS client sends a DNS query to the DNS server for domain name resolution. The client forwards the query to its primary DNS server that is either manually or dynamically configured. By observing DNS queries, their types, e.g., A, AAAA, PTR, or requested domains, we can learn much about the device. For instance, if the system supports IPv6, it requests AAAA records in DNS. Also, most of installed applications contact vendor servers for updates or synchronization, which yields in resolution of DNS names like spotify.com, github.com, facebook.net. Similarly to application, we can identify the operating system by observing queries to domains android.google.com, xioami.net, etc. Based on the frequency and uniqueness of requested domain names we can create a DNS fingerprint that identifies a unique device.

In our scenario, we will analyze captured DNS communication sent by a set of mobile devices. First, we process a training dataset where we extract selected fields from DNS communication. Based on frequencies of requested domain name look-ups we create a DNS fingerprint for each device detected in the dataset. In the second phase this DNS fingerprint will be compared with fingerprints obtained from other datasets. If similarity of fingerprints from two datasets is high, we can deduce that DNS communication comes from the same source device.

The hypothesis behind the research is that almost every software installed on the device leaves a unique trace in DNS communication. By collecting all these traces we can describe an individual device and distinguish the device from others.

Input Data

As source data, we will use captured DNS communication. From this communication we extract specific values using `tshark` (See https://www.wireshark.org/docs/man-pages/tshark.html 2019) command:

```
tshark -r dns.pcap -T fields -E separator=";" -e ip.src -e ip.dst -e dns.qry.type
-e dns.qry.name "dns.flags.response eq 0 and ip"
```

Using this command, we obtain the following values:

- Source IP address (DNS client's address)
- Destination IP address (DNS server's address)

- DNS query type, e.g., A, AAAA, PTR, etc.
- Requested domain name, e.g., www.googleapis.com

The source IP address is used to classify DNS requests that belong to the same device. Destination address is used to identify a primary DNS server. We get a list of tuples *(srcIP,serverIP,query-type,domain-name)* which will be later transformed into a table where each device identified by an IP address will be assigned a vector describing frequency of all DNS queries found in the dataset, see figure Txt to csv.pdf. Query type 1 means A record. The frequency vector forms a DNS fingerprint of the related device.

Method

The naïve solution uses raw frequency vector to build a DNS fingerprint. Then a DNS fingerprint of an unknown device can be compared to a set of DNS fingerprints of known devices. Based on similarities of two DNS fingerprints we can say whether these devices are same or not. However, raw frequency matching without normalization does not work well with real data.

Thus, we apply TF-IDF (Term Frequency-Inverse Term Frequency) method [1] that includes normalization and weighting. TF-IDF method is typically used to assign a document to the most similar document taken from a set of documents based on frequency of terms that are found in documents. In case of DNS fingerprinting, we will use domain names to represent TF-IDF terms. For each term in the individual document, Term Frequency $tf_{t,d}$ will be computed using frequency of term t in document d. Normalized Term Frequency is given by the following formula:

$$tf_{t,d} = 0.5 + \frac{0.5 * f_{t,d}}{MaxFreq(d)}$$

where $MaxFreq(d)$ is maximal frequency of any term in document d.

Term frequency suffers from a serious problem: all terms are considered equally important when it comes to assessing relevancy on a query. This means, when a term is present in almost every document, it distorts the ability to distinguish documents based on term frequency. Using *Inverse document frequency* (IDF) we can weight frequency by the number of documents in the collection that contain a term t. Thus, terms that are present in the small number of documents will be given the higher weight. IDF can be computed using the following formula:

$$idf_t = \log \frac{N}{df_t}$$

where N is the number of all documents and df_t is a number of documents that contain term t. Combination of TF and IDF creates the TF-IDF weighting scheme that assigns a weight to term t in document d by the following equation:

$$tfidf_{t,d} = tf_{t,d} \times idf_t$$

Evaluation

In our case, the DNS fingerprint is expressed as a TF-IDF vector that contains weighted and normalized frequencies of domain names found in the dataset. Following table displays frequencies for selected domain names of devices F0-F3. Column F0 denotes an unknown device that we will be matched against fingerprints of devices F1 to F3.

	F0	F1	F2	F3
clients4.google.com	1	1	2	16
graph.facebook.com	2	0	9	0
mtalk.google.com	4	0	2	1
pxl.jivox.com	1	1	0	0
www.google.com	6	3	5	8
www.googleapis.com	4	6	8	6

For example, the first domain name *clients4.google.com* has been found in communication of all devices with frequencies 1, 1, 2, or 16 respectively. Domain *px1.jivox.com* appeared only one time in communication of device F0 and F1.

Based on raw frequencies, we can compute $tf_{t,d}$ and idf_t values for each domain name using equations from the previous section, see the following table. In this table, each column TF_{xx} includes a list of TF frequencies for all domain names requested by a given device. By applying IDF frequency we get an TF-IDF value that represents a DNS fingerprint of the device.

	TF_{F0}	TF_{F1}	TF_{F2}	TF_{F3}	IDF
clients4.google.com	0,583	0,583	0,611	1,0	0
graph.facebook.com	0,667	0	1,0	0	0,477
mtalk.google.com	0,833	0	0,611	0,531	0,176
pxl.jivox.com	0,583	0,583	0	0	0,477
www.google.com	1,0	0,75	0,778	0,75	0
www.googleapis.com	0,833	1,0	0,944	0,688	0

Comparison of DNS fingerprints can be evaluated using cosine similarity [2]. The resulting score describes a level of similarity between documents q and d, in our case between two DNS fingerprints:

$$score(q,d) = \frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}$$

$\vec{V}(q)\cdot\vec{V}(d)$ is a dot product of weighted DNS fingerprints $tfidf_{t,d}$. For our input data, similarity score of DNS fingerprint *F0* towards DNS fingerprints *F1*, *F2* and *F3* is in the following table. Based on the score, the most similar fingerprint to F0 seems to be F2.

	F1	F2	F3
F0	0.621	0.767	0.333

Source codes

As a part of this case study we provide the following source codes:

- SQL queries that allow to create the database containing real captured DNS data (DNS database.sql).
- python script for IDF values computation (CountIDF.py). This script produces the SQL queries, that can be used to store the IDF values in the provided database. It is not necessary to run this script, because the provided database already contains precomputed IDF values.
- python script that queries SQL database and compute TF and score values for database

values using IDF value stored in the database (TF IDF db.py).

We provide the SQL queries instead of csv files, since the database provide better data control and manipulation. However, the DNS fingeprints can be easily extracted from the provided database.

For the students that are interested in data processing we also provide:

- raw DNS data extracted from tshark output (Dataset1.txt, Dataset2.txt).
- python script for selection of unicast DNS data (SelectUC.py).
- python script that allows to sort selected unicast data (SortUC.py).
- SQL queries that insert basic values (mac and IP addresses) into table mobiles (Address insert.sql). The database schema can be found in the file DNS database.sql.
- python script that creates SQL queries that enables to store sorted unicast data in the database (Insert to db.py).

Notes

[1] Christopher D. Manning, H. S., Prabhakar Raghavan: Introduction to Information Retrieval. Cambridge University Press, 2008, ISBN 9780521865715.

[2] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.