OS fingerprint classification using a support vector machine

David Fifield

December 10, 2010

Abstract

An evaluation of using a support vector machine (SVM) to classify operating system fingerprints in the Nmap security scanner. In solving a simplified version of operating system classification, the SVM got marginally more accurate results than Nmap's built-in classifier.

1 Introduction

Remote operating system detection (OS detection) is any way of finding out the operating system system of a network host through analysis of network traffic. It has implications for network security, including network inventory, vulnerability detection, and exploit tailoring. OS detection is a feature of various network security programs including Nmap, p0f, and SinFP.

There are many ways of doing OS detection. This paper is concerned only with active, low-level detection. "Active" means that the scanner is free to send packets and not just sniff passively. "Low-level" means that it operates at the level of IP, TCP, and ICMP, and not, for example, by reading banners from application services.

OS detection is possible because of implementation differences in TCP/IP stacks. OS programmers are free to set some values within certain limits. Systems differ in how they handle unexpected or uncommon conditions. Implementation errors are sometimes visible in network traffic. OS detection seeks to elicit as many of these distinguishing characteristics as possible. Nmap sends over a dozen TCP, UDP, and ICMP probes, with various valid and invalid combinations of fields, and collects the responses into a data structure called an OS fingerprint. This is a fingerprint of Linux 2.6.34:

```
SEQ(SP=CC%GCD=1%ISR=CD%TI=Z%CI=Z%II=I%TS=8)
OPS(O1=M5B4ST11NW7%O2=M5B4ST11NW7%O3=M5B4NNT11NW7
   %O4=M5B4ST11NW7%O5=M5B4ST11NW7%O6=M5B4ST11)
WIN(W1=16A0%W2=16A0%W3=16A0%W4=16A0%W5=16A0%W6=16A0)
ECN(R=Y%DF=Y%T=40%W=16D0%O=M5B4NNSNW7%CC=Y%Q=)
T1(R=Y%DF=Y%T=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=Y%T=40%W=16A0%S=O%A=S+%F=AS%O=M5B4ST11NW7%RD=0%Q=)
T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
IE(R=Y%DFI=N%T=40%CD=S)
```

The format of the fingerprint and the meaning of its fields are described fully in [1]. A dotted notation is used to refer to tests within each line: in the above fingerprint it is the case that `ECN.W=16D0`, meaning that the TCP window size in the response to the Explicit Congestion Notification probe was 0x16D0. Compare with a fingerprint of Windows XP SP3:

```
SEQ(SP=105%GCD=1%ISR=10B%TI=I%II=I%SS=S%TS=0)
OPS(O1=M5B4NW0NNT00NNS%O2=M5B4NW0NNT00NNS%O3=M5B4NW0NNT00
   %O4=M5B4NW0NNT00NNS%O5=M5B4NW0NNT00NNS%O6=M5B4NNT00NNS)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=Y%TG=80%W=FFFF%O=M5B4NW0NNS%CC=N%Q=)
T1(R=Y%DF=Y%TG=80%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N%DF=N%TG=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=Y%TG=80%W=FFFF%S=O%A=S+%F=AS%O=M5B4NW0NNT00NNS%RD=0%Q=)
T4(R=Y%DF=N%TG=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=S%TG=80%CD=Z)
```

See that in this case `ECN.W=FFFF`. A few of other differences are in the TCP options (`OPS.O1` through `OPS.O6` and `T3.O`), don't fragment flags (`*.DF`), and explicit congestion notification status (`ECN.CC`). Windows responded to the T2 probe while Linux did not;

also the Windows fingerprint is missing lines T5, T6, and T7 because the host lacked a closed port.

This paper is not about the technical details of OS detection probes and the assembly of fingerprints. Rather, it is assumed that the fingerprints are given and the only remaining task is to classify them. The focus is on converting a fingerprint to a feature vector and then classifying it using a support vector machine.

OS identification is harder than it sounds. In an ideal world, every operating system would have just one fingerprint, distinguishable from those of all others. In actuality, one OS can have dozens of fingerprints, even at the granularity of a Linux kernel micro revision, let also a Windows service pack. There are at least two reasons for this: some tests are sensitive to network conditions and (especially) packet-mangling routers, and fingerprints are affected by OS configuration. Nmap's current second-generation OS database has a simple pattern language that allows for some variation in fingerprints [1], but even so it contains over 2,600 fingerprints for 106 OS families from AIX to ZyNOS. Of these, 468 are Linux, 471 are Windows, and 1,110 are miscellaneous "embedded" operating systems of hardware devices. The size of this database and its maintenance costs are one reason for investigating whether machine learning can do the same job.

2 Previous work

Sarraute and Burroni [4] used a hierarchy of neural networks and Nmap's older first-generation OS database to do first coarse, then fine OS classification.

Zaid Aiman [5] outlined a plan to implement something similar for Nmap's second-generation database.

João Medeiros et al. [6] used a neural network and the second-generation Nmap database to do classification and build a map of operating system similarity. They used Euclidean distance of feature vectors as a measure of fingerprint similarity.

3 Data sources

The Nmap OS database is built from submissions by users. When Nmap fails to identify an operating system it prints a fingerprint and asks the user, if the OS is known, to submit it to an online form along with a classification. The submissions are integrated into the database a few times per year. The raw data for this experiment were 12,531 submissions that had already been integrated into the Nmap database between December 2006 and April 2010 (the training set), and 1,500 that were received between April 2010 and October 2010 but had not been integrated (the testing set).

Nmap's second-generation database was introduced in June 2006, so these are almost all the submissions available. It is necessary to go far into the past to counteract the bias that otherwise exists with the submission system: once an operating system is reliably matched, it becomes less likely to be submitted in the future.

Many submissions have to be discarded because they are of low quality or have uncertain identification. All submissions marked by Nmap as being suspect (using the pseudo-test `SCAN.G=N`) were removed. During manual database integration, submissions for common operating systems must include the output of `winver.exe` or `uname` or equivalent, to check that the submitter actually has access to the operating system ostensibly being identified. A robotic classification program did pattern matching against this `uname` output to put submissions into one of seven OS families; those that could not be positively identified were discarded. A human doing database integration will discard on the order of 40% of submissions, and this process is supposed to be even more conservative.

These seven OS families were chosen because (1) they are common, and (2) they are easily identified by their `uname` strings: Linux, Windows, FreeBSD, Darwin (Mac OS X), SunOS (Solaris), OpenBSD, and AIX.

As will be discussed in the section called "Procedure," errors (i.e., mistaken submissions) in the training set were only removed as they became apparent, but every submission in the testing set was carefully checked by hand (by cross-checking against the Nmap

database) to remove all errors.

After this winnowing process, the data broke down as follows:

OS	*# training*	*# testing*
Linux	3,246	570
Windows	1,100	71
FreeBSD	325	15
Darwin	293	9
SunOS	143	4
OpenBSD	79	9
AIX	74	9
total	5,260	687

4 Feature selection

An Nmap fingerprint consists of up to 119 tests [1]. They can be of five different classes:

1. Numeric, encoded in hexadecimal. These are mostly fixed-size positive integers taken directly from packet contents, like `T1.T` (T1 probe TTL). Members of this class:
   `T[1234567].T`, `ECN.T`, `U1.T`, `IE.T`,
   `T[1234567].TG`, `ECN.TG`, `U1.TG`, `IE.TG`,
   `T[234567].W`, `WIN.W[123456]`, `ECN.W`,
   `T[1234567].RD`, `SEQ.SP`, `SEQ.GCD`, `SEQ.ISR`,
   `U1.IPL`, `U1.UN`, `U1.RIPL`, `U1.RID`.

2. Enumeration, one of a finite set of symbolic values. Examples are `T1.R` (responsiveness to T1 probe), which may be either `Y` or `N`, and `U1.RIPCK` (integrity of IP checksum), which may be `G`, `Z`, or `I`. Members of this class:
   `T[1234567].R`, `ECN.R`, `U1.R`, `IE.R`,
   `T[1234567].DF`, `ECN.DF`, `U1.DF`,
   `T[1234567].A`, `T[1234567].S`,
   `SEQ.SS`, `ECN.CC`, `IE.DFI`, `U1.RIPCK`, `U1.RUD`.

3. Enumeration or numeric. These may be like either class 1 or class 2. An example is `IE.CD` (ICMP response code), which may be any of `Z`, `S`, or `O`, or a number like `38` if none of the enumeration values fit. `U1.RIPL` and `U1.RID` would appear to be in this class because they may be numeric or have the value `G`, but `G` only stands for a specific number so it is more convenient to handle these as class 1. Members of this class:
   `SEQ.TI`, `SEQ.CI`, `SEQ.II`,
   `SEQ.TS`, `U1.RUCK`, `IE.CD`.

4. Set. These are marked by the presence or absence of letters from a finite set. `T1.F` through `T7.F` (TCP flags) are examples using the set `EUAPRSF`. Members of this class:
   `T[1234567].F`, `T[1234567].Q`, `ECN.Q`.

5. TCP options. `OPS.O1` through `OPS.O6` as well as `T2.O` through `T7.O` encode TCP options as strings like `M5B4ST11NW7`. Members of this class:
   `OPS.O[123456]`, `T[234567].O`, `ECN.O`.

Each test is converted to a fixed-size numeric vector as follows. The concatenation of all these becomes the overall feature vector. Numeric values map directly to integers.

`T1.T=40` → $[64]$

Enumerations are converted to a sequence of 0/1 indicators, one for each possible value, according to the advice in [3]. The only exceptions are the `Y/N` enumerations, which become a single indicator.

`U1.RIPCK=G` → $[1, 0, 0]$
`U1.RIPCK=Z` → $[0, 1, 0]$
`U1.RIPCK=I` → $[0, 0, 1]$
`T1.R=N` → $[0]$
`T1.R=Y` → $[1]$

Enumeration/numeric tests get categorical indicators followed by one element for the numeric value. When an enumerated value is used, the spot for the numeric data is -1. When a numeric value is used, all the categorical indicators are -1.

`IE.CD=G` → $[1, 0, 0, -1]$
`IE.CD=S` → $[0, 1, 0, -1]$
`IE.CD=O` → $[0, 0, 1, -1]$
`IE.CD=38` → $[-1, -1, -1, 56]$

TCP flags are represented by an indicator for each flag.

`T1.F=AS` → $[0, 0, 1, 0, 0, 1, 0]$
`T1.F=R` → $[0, 0, 0, 0, 1, 0, 0]$

TCP options, because of their more complex structure, were ignored. (See "Continuing work.")

Whenever a test was missing, all its elements were assigned -1. The concatenation of all these subvectors leads to a feature vector of 240 elements.

5 Procedure

LIBSVM 3.0 [2] is the support vector machine implementation used. The training of the SVM went according to the recommendations in [3]. Specifically, the training set was first converted to a set of feature vectors. The elements of each vector were scaled to lie in the range $[-1, 1]$. Using the RBF kernel $K(x_1, x_2) = \exp(-\gamma||x_1 - x_2||^2)$, grid search found good values of C and γ using five-fold cross validation. All this is encapsulated in the `easy.py` and `grid.py` scripts that come with LIBSVM.

The feature vector and training set were refined over eight iterations. After each of the first few iterations, each incorrectly classified submission from the training set was checked for plausibility by matching it against the Nmap database. The matching program shows the ten closest matches and for common operating systems all ten are usually variations of the same OS family. It would be surprising, for example, for a Linux result to be in the top ten for a Windows fingerprint. Submissions that appeared to be wrong were excluded from later iterations. 43 submissions were excluded in this way.

All 1,500 testing submissions were individually checked for correctness. 11 of these were excluded.

In a typical example of a submission that was manually excluded, one submitter claimed that the operating system was "Linux 2.6.34." However, when cross-checked using Nmap, the nearest matches were

```
96% Microsoft Windows XP Professional SP2
96% Microsoft Windows XP SP2 - SP3
96% Microsoft Windows XP SP3
95% Microsoft Windows XP SP2
94% Microsoft Windows Server 2003 SP2
```

The submitter likely scanned the wrong machine by mistake, or sent the classification of the scanning host instead of the one being scanned.

After the final iteration, the SVM correctly classified 5,237 of 5,260 training examples (99.5%). Training and cross-validation of the final iteration took about 90 minutes with `easy.py`. Experiments during the earlier iterations included using integers for enumeration tests instead of categorical indicators, using two-dimensional vectors for binary features instead of one-dimensional, (accidentally) leaving out a couple of tests, and artificially adjusting fingerprints to correct a byte ordering bug present in older versions of Nmap. The training and testing accuracy never dropped below 97% so these changes are likely not of much consequence. (The byte order fix was kept.)

6 Results

The SVM classified 685 of 687 testing examples correctly (99.7%). Both errors were Darwin misclassified as FreeBSD.

Nmap's detection engine, taking the best match and accepting any match at 80% confidence or more, classified 683 of 687 correctly (99.4%). The errors were AIX misclassified as an embedded messaging server, AIX as Darwin, Darwin as FreeBSD, and Windows as a wireless access point. (There were also 10 instances of Linux classified as "embedded," but on inspection all of these embedded devices do appear to run Linux.) If 90% or greater confidence is required of Nmap, there is only one mistaken classification, AIX as Darwin, but 26 submissions have no result at all, for a total accuracy of 660 in 687 (96.1%).

7 Continuing work

These results are promising, but it must be kept in mind that this particular SVM will do well only when the operating system is known a priori to be in one of the seven selected families. Unlike Nmap's matching algorithm, it will always return a result in one of these families, even if the fingerprint is unlike anything seen before. A practical matching algorithm will have to be aware of novel fingerprints and at least avoid returning a very uncertain answer.

It is limiting to have to declare possible OS matches in advance as is done here. "Windows" is unsatisfying when the technology can reliably distinguish between "Windows 2000 SP4" and "Windows XP SP1." The authors of [4] used a hierarchy of classifiers, so once

"Windows" was known they could get a more specific answer by turning to a specialized Windows classifier. Even this doesn't answer for the many undocumented "embedded" operating systems known only by the hardware they run on, such as

```
Polycom SoundPoint 500 or 601 IP phone
W&T Web-IO Thermometer model 57101
Xerox WorkCentre Pro C2128 printer
```

One possibility, as is done in [4], is to put all such miscellaneous fingerprints into a single category.

TCP options are a great source of variation and should be part of the feature vector. The references show some ways of doing this.

The source code of the tools developed for this project is at `http://www.bamsoftware.com/stanford/cs229/`.

References

[1] Gordon "Fyodor" Lyon. *Nmap Network Scanning*. Chapter 8, "Remote OS Detection" (2009). `http://nmap.org/book/osdetect.html`.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. "A Practical Guide to Support Vector Classification." `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

[4] Carlos Sarraute and Javier Burroni. "Using Neural Networks to improve classical Operating System Fingerprinting techniques." Electronic Journal of SADIO vol. 8, no. 1, pp. 35–47 (2008). `http://www.coresecurity.com/files/attachments/Sarraute_EJS.pdf`

[5] Zaid Aiman. "Ideas: Nmap Fingerprint Analyzer." nmap-dev mailing list (2008). `http://seclists.org/nmap-dev/2008/q2/2`

[6] João Medeiros et al. "Automating Security Tests for Industrial Automation Devices Using Neural Networks." IEEE Conference on Emerging Technologies and Factory Automation (2007). "Nmap OS Database and Artificial Neural Networks." nmap-dev mailing list (2008). `http://seclists.org/nmap-dev/2008/q1/325`