SNAP: ROBUST TOOL FOR INTERNET-WIDE OPERATING SYSTEM FINGERPRINTING

A Thesis

by

ANKUR BHARATBHUSHAN NANDWANI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Science

SNAP: ROBUST TOOL FOR INTERNET-WIDE OPERATING SYSTEM FINGERPRINTING

A Thesis

by

ANKUR BHARATBHUSHAN NANDWANI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Dmitri Loguinov
Committee Members,	Donald Friesen
	Srinivas Shakkottai
Head of Department,	Valerie Taylor

December 2010

Major Subject: Computer Science

ABSTRACT

Snap: Robust Tool for Internet-wide Operating System Fingerprinting. (December 2010)

Ankur Bharatbhushan Nandwani, B.Tech., National Institute of Technology-Surat

Chair of Advisory Committee: Dr. Dmitri Loguinov

Different approaches have been developed for TCP/IP fingerprinting, but none of these approaches is suited for Internet-wide fingerprinting. In this work, we develop approaches that rigorously tackle the issue of noise and packet loss while carrying out Internet-wide fingerprinting. We then carry out an Internet-wide scan to determine the distribution of different operating systems on the Internet. The results of our scan indicate that there are approximately 8.9 million publicly accessible web-servers on the Internet running Linux, while there are nearly 9.6 million web-servers with different embedded operating systems.

To my parents and sister

ACKNOWLEDGMENTS

I would like to sincerely thank Dr. Loguinov for giving me the opportunity to work with him. I believe that working with him has prepared me to deal with any situation both professional and personal that I may encounter in the future. I would also like to thank Dr. Shakkotai and Dr. Friesen for being on my committee. This thesis would not have been possible without the direction given to me by Mr. Ajit Shelat, who has always been a source of inspiration to me.

I would also like to thank my colleagues at Internet Research lab, especially Derek Leonard and Sadhan Sood. Finally, all my achievements have been a direct result of the support and sacrifices of my parents and my sister.

TABLE OF CONTENTS

CHAPTER		Page
I	INTRODUCTION	1
	A. Motivation	1
	B. Our Contribution	2
	C. Ethical Implications	3
II	RELATED WORK	5
	A. Active Fingerprinting	5
	B. Passive Fingerprinting	7
	C. Defenses	8
III	OPERATING SYSTEM FINGERPRINTING	9
	A. General Problem Formulation	9
	B. Noise	11
	C. Packet Loss	12
	D. Snacktime	12
IV	SNAP	15
	A. Feature Vector	15
	B. Signature Database	19
	C. Signature Preprocessing	19
	1. Clustering	20
	2. Cumulative RTO Values	20
	D. Classification Algorithm	22
V	EVALUATION	25
VI	INTERNET SCAN	27
	A. Scan Statistics	27
	B. Results	27
	C. Verification	31
VII	DEFENSES	33
VIII	CONCLUSION	34

Page

REFERENCES 35

VITA 39

LIST OF TABLES

TABLE | Page

I Sample Snacktime Fingerprint and Signatures 14
II Signature Description 16
III Sample Signatures 17
IV 10-fold Cross-validation Accuracy 26
V SYN-ACK Scan Statistics 28
VI HTTP Scan Statistics 28
VII Top 5 Operating Systems 29
VIII Top 5 Clusters 30
IX Common Operating Systems 31
X Basic Verification 32

LIST OF FIGURES

FIGURE Page

1 RTO (*Not sent by all OS). 6

2 RTO Randomness 10

3 Effect of Noise 11

4 Snap 24

CHAPTER I

INTRODUCTION

With the exponential growth of computer networks, it has become a challenge to effectively manage and secure these networks. One of the integral components of managing and securing networks is to be aware of the operating system running on each machine connected to the network. Operating system fingerprinting as its name suggest aims to identify the OS of a remote machine, by either sending probes to the machine or by sniffing traffic originating from remote machines. While the former comes under the class of approaches known as active fingerprinting, the later belongs to passive fingerprinting, as no probes are sent to the target machine.

Operating System fingerprinting had its origins in banner grabbing, where one tried to connect to a remote machine on a particular service like FTP. Telnet and HTTP, and then use the displayed banner to identify the operating system. Since the banner can be easily be modified, banner grabbing gave way to techniques which base their identification on the TCP/IP stack. Identification, with TCP/IP stack is made possible because various RFC's like 1122 [1] and 2988 [2] that deal with TCP/IP, are ambiguous about certain features, while for some other features they provide no guidelines, thus resulting in developers having different interpretations of the TCP/IP protocol, leading to implementations unique to an OS.

A. Motivation

In this paper, we develop robust techniques that would allow us to fingerprint all the publicly accessible machines on the Internet. The main motivation behind this

---

The journal model is *IEEE Transactions on Automatic Control.*

work is that developing techniques to work at such a large scale would allow us to find the vulnerable population for a malware on the Internet in less than 40 minutes using a custom scanner. While fingerprinting, we make sure that we use the bare minimum packets and only use packets that conform to the TCP/IP protocol. Owing to limiting the number of probes, the techniques we develop can be used by the network administrators of large corporate networks to scan their networks in real time, without clogging the network, or resulting in denial of service attack. Also, since we use protocol conforming packets, the risk of bringing down machines which are not adequately equipped to handle malformed packets is greatly reduced.

B. Our Contribution

Over the past decade, many tools [3, 4, 5, 6, 7, 8, 9] have been developed to carry out operating system fingerprinting by analyzing the TCP/IP stack properties. These tools use different type and number of probes for fingerprinting purposes. While fingerprinting machines on the Internet one has to inherently deal with noise and packet loss. Whereas noise can result in unknown values for various fields of TCP/IP headers and random retransmission timeout value, packet loss which is to the extent of 3.8% [10] on the internet, can drastically change the observed retransmission timeout values (RTO). Hence, if we don't consider these factors while carrying out a scan, then the results we achieve are bound to be inaccurate. Current tools [6, 8] which use retransmission timeout values fail to address the issues of packet loss on one hand, while are also ineffective in dealing with noise. Thus the major contribution of this paper is to develop robust techniques that deal with noise and packet loss, and apply the developed techniques for Internet-wide OS fingerprinting.

In the first part of the paper we discuss the problem of fingerprinting using

RTO and the challenges associated with it. Also, we discuss how we augmented the signature vector and built the signature database. The fingerprinting methodology we use can be broken down into two steps, the first being data collection, where we send SYN probes similar to those sent by Windows Server 2008, to the target machines and log all the responses received from the target. The second step is classification, where by using Retransmission Timeout values in conjugation with various features derived from TCP/IP headers of the response packet, and applying a custom classification algorithm, we classify the target machines into different operating system.

In the next section we propose a custom classification scheme that robustly deals with both noise and packet loss. We then evaluate our classification scheme against the default scheme used by Snacktime for classification.

Finally, to get a picture of the distribution of different operating systems on the internet, we scan the internet for all the machines which have port 80 open, and at the same time fingerprint by running our classification algorithm on received response packets. Of the 37.8 million machines, which responded to our SYN probes with SYN-ACK packets, nearly 9.6 million machines are running different embedded OS, the major chunk of these embedded OS being VxWorks. VxWorks is a popular embedded OS mainly used in routers, modems and printers. This is followed by Linux which account for nearly 8.9 million machines. This is followed by Windows which collectively account for nearly 5.2 million machines.

C. Ethical Implications

The work presented in this paper can be used for both good and bad purposes. On one hand it can be used to secure a large network, while on the other hand, it can also be used by the hackers to surreptitiously scan the Internet for vulnerable machines. To

counteract this threat caused by the hackers, a simple solution would be to randomize the first RTO value, since in most cases successive RTO values are computed based on the initial RTO. Also, simply altering values of various TCP/IP header fields can drastically reduce the accuracy of fingerprinting. We hope that by publishing this work, during the future implementation of TCP/IP protocol in different OS, these techniques could be kept in mind, so that risk of detection could be reduced.

CHAPTER II

RELATED WORK

OS fingerprinting has been studied for a long time now, but most of the work has been done by independent hackers. Initial approaches used banner grabbing, but since the banner for most of the services can be easily changed, the focus shifted towards the use of TCP/IP stack for identification. Of the techniques that use TCP/IP stack, some send probes to the target machines [3, 5, 6, 7, 8, 9], while others involve sniffing traffic [7, 9]. At the same time some techniques have even made use of application level data for OS detection [5, 4]. Since OS fingerprinting poses a network security threat, work has also been to defend against fingerprinting.

A. Active Fingerprinting

Among the tools used for fingerprinting, Nmap [3] is the most actively developed and widely used tool. Developed by Fyodor, it has one of the largest user contributed signature database, and is fairly accurate. But it is of limited utility when scanning on the internet because it suffers from various issues. Firstly, it uses 16 packets per machine, which would result in lot of traffic when considering all the machines on the internet. Secondly, it uses malformed packet, thus allowing signatures to be written to detect its probes. In fact Snort [11] has signatures to detect nmap probes. Finally, it requires and open and closed port, for which it does a port scan, thereby again increasing the number of packets sent to a machine.

Since Nmap is the most popular tool, a lot of work has also been done to improve the same. Sarraute *et al.* [12] propose the use of neural networks to identify different versions of a same operating system. Medeiros *et al.* [13] also make use of neural networks, to add the capability of identification of unknown devices to Nmap. In

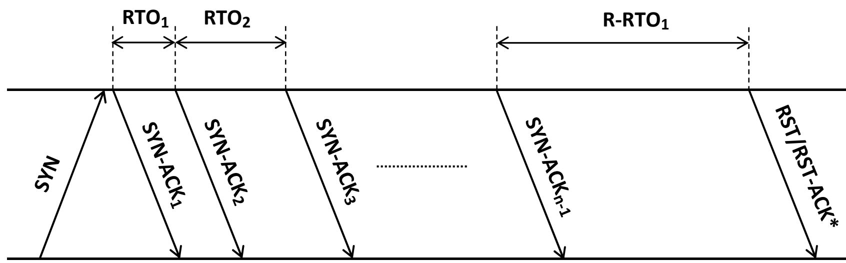


Fig. 1. RTO (*Not sent by all OS).

[14], Greenwald *et al.* discuss how information theoretic measures could be used to reduce the number of probes that are sent by Nmap, and at the same time get good results. Finally, Shu *et al.* [15], explore the use of Parameterized Extended Finite State Machines with Nmap for fingerprinting.

Xprobe2++ [5], is a tool which makes use of ICMP probes, SYN probes, and application layer requests for fingerprinting. It has a modular architecture,which allows the user to control which probes are sent to the target machine. By default the latest version uses 13 modules, which includes two application layer requests modules.

RING [8] and Snacktime [6], both base their detection on Retransmission timeout values. RING was developed as a proof of concept tool, and it uses Retransmission timeout values to detect the OS of a machine. The initial version of RING only made use of SYN-ACK RTO values and presence of RST or RST-ACK packet. Later version of RING gave an option of considering RTO values of FIN-ACK packets. Now, in order to get the RTO values for FIN-ACK packets, at least 3 packets need to be sent per machine, making the approach infeasible for the internet. Snacktime (Fig. 1) which is based on RING uses RTO values of SYN-ACK packets in combination with the observed Window Size and TTL values to determine the operating system

of the machine. In Snacktime, the client first sends a SYN packet to the remote machine and then listens for SYN-ACK packets. If it fails to get a response for 65 seconds(default value) it terminates the connection and proceeds with fingerprinting. Now, during fingerprinting only signatures which have the same length as that of the generated fingerprint are considered, and then using different weights for different features similarity between a signature and fingerprint is computed. The signature with the highest similarity to the fingerprint is returned as detected OS. Also, OS corresponding to signatures with second and third highest similarities are returned as probable matches.

Some work has also been done academically, in the domain of TCP/IP fingerprinting. In [16], Caballero *et al.* develop an approach similar to fuzz testing, to come up with new probes that could be used for fingerprinting. But, since non-standard probes are generated, IDS/IPS signatures could be written to detect the same, hence fingerprinting could be easily circumvented. Medeiros *et al.* in [17], present a technique to fingerprint OS based on only Initial Sequence Number, which is based on the work done in [18] . The problem with this approach is that it requires thousands of ISN samples per machine, making the approach infeasible for large scale scanning and easily prone to detection.

B. Passive Fingerprinting

p0f [9] is a tool primarily built for passive fingerprinting. It supports 4 modes of fingerprinting, they being the SYN mode, SYN-ACK mode, RST+ mode and the stray ACK mode. The first and the third mode support passive fingerprinting, while the second and the fourth mode are for active fingerprinting. The SYN mode has the largest signature database and seems to be the most developed, while the other

modes are not well supported. p0f makes use of options, various TCP/IP quirks and values of TCP/IP fields for OS detection.

C. Defenses

There exist many tools to defend against OS fingerprinting. Berrueta in his paper [19] provides an overview of tools to defend against Nmap. In [20], a proof of concept tool using Netfilter [21] kernel module for fingerprinting evasion is presented. Use and effectiveness of Honeyd against fingerprinting tools has been discussed in [22, 23]. Smart et. al. present a *fingerprinting scrubber* in [24] as a general solution to protection against TCP/IP fingerprinting to be implemented at the gateway. As stated in the paper, fingerprinting scrubber effectively deals with static approaches but is not well developed for temporal approaches like RING and Snacktime.

CHAPTER III

OPERATING SYSTEM FINGERPRINTING

In this section we first formulate the problem of OS fingerprinting, follow it by explaining the issue of noise and packet loss, and conclude by discussing Snacktime.

A. General Problem Formulation

The problem of Operating System fingerprinting is closely related to the problem of pattern matching, therefore we formally state our problem using the terminology of pattern matching. Assuming a set of c unique operating system types where $C = \{\omega_1, \ldots, \omega_c\}$, the signature of the i-th OS is a vector $S_i = \{S_{i,1}, \ldots, S_{i,n_i}\}$, where $S_{i,j}$ is the random variable represented the j-th feature of the OS i. Fig. 2 shows the randomness in RTO values. These values were measured for machines in our lab itself, hence they correspond to the randomness in the RTO values itself and not network delays.

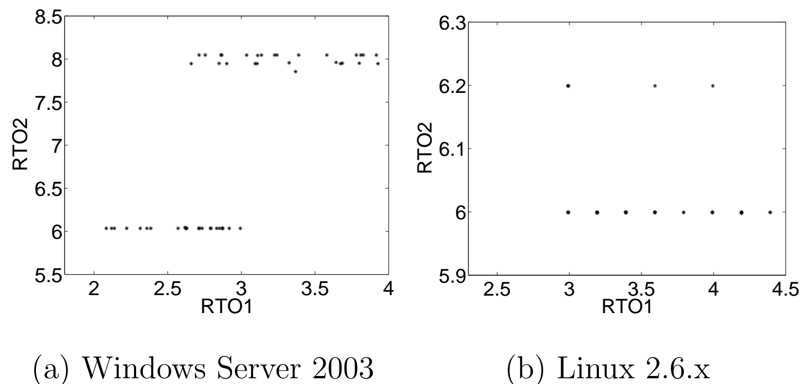
Now assuming that M is a random variable (vector of features) representing a response from a host on the Internet, the classification goal is to find the most likely class i_{opt} in C that could have generated M such that

$$i_{opt} = \arg\max_i P(M \in \omega_i | M = m). \quad (3.1)$$

Using Bayes rule (3.1) can be expressed as

$$i_{opt} = \arg\max_i \frac{P(M = m | M \in \omega_i) P(M \in \omega_i)}{P(M = m)}. \quad (3.2)$$

Now, the conditional probability $P(M = m | M \in \omega_i)$ is equivalent to the probability $P(S_i = m)$, and the probability of a measurement $M = m$ is constant for all



(a) Windows Server 2003 (b) Linux 2.6.x

Fig. 2. RTO Randomness

the signatures hence we can reduce (3.2) to

$$i_{opt} = \arg\max_{i}\{P(S_i = m)P(M \in \omega_i)\} \tag{3.3}$$

The work done in this paper deals with estimating the probability of the measured value m being equal to a signature S_i. It is necessary to note that it is impossible to compute $P(S_i = m)$, because of the human factor involved. For example, the users may arbitrarily decided to tweak OS settings. For an Internet-wide scan, *apriori* probability for a class can be computed iteratively but we leave it for future work.

As mentioned earlier many efforts have been made to solve the above problem using the TCP/IP stack, but few efforts have been made to reduce the number of probes sent to the target machine [7, 9, 6, 5], and to the best of our knowledge, no work has been done to robustly fingerprint a large number of machines. If we wish to accurately fingerprint all of the publicly addressable machines on the internet, we need to make sure that our approach is not only resilient to noise an packet loss, but is also polite. Now [5] and [6], which use RTO values for identification, are polite, but fail to address noise and packet loss. In the following subsections we address the problem of noise and packet loss associated with RTO by carrying out a rigorous

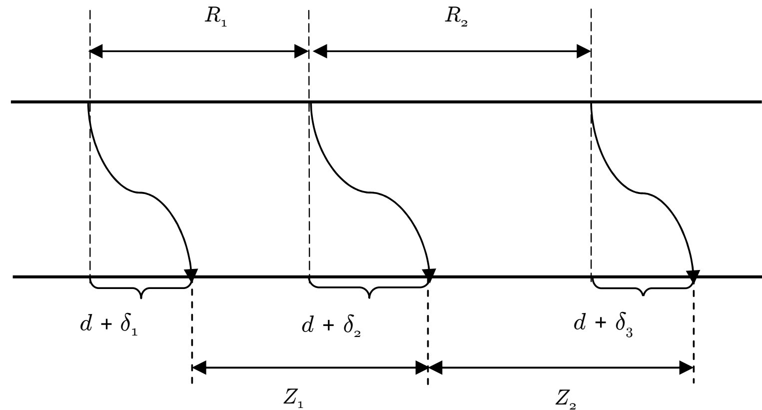


Fig. 3. Effect of Noise

analysis of the two problems.

B. Noise

The first issue we consider is that of noise on the Internet. In the ideal conditions assuming the delay between the sender and the receiver is d, and the retransmission timeout values at the sender and receiver are R and Z, respectively, the values of Z and R would be same. But, as shown in [25], normally there is jitter(δ) in the Internet packet delay, which is illustrated in Fig. 3. Therefore, due to the presence of jitter, the values Z_i and R_i are not the same and we have

$$Z_i = R_i + (\delta_{i+1} - \delta_i) \tag{3.4}$$

Thus it is required that the classification algorithm we use effectively deal with jitter, which we refer to as *noise* from now on.

C. Packet Loss

Next we consider the issue of packet loss, which can reach as high as 3.8% for SYN-ACK packets[10]. In ideal circumstances when there is no packet loss the RTO values measured at the receiver are approximately equal to the RTO values of the sender, but in the case of packet loss we have a different scenario. Therefore, considering Q as a vector representing the values measured at the receiver,and N as a vector indicating the packets which are actually received by the receiver we have

$$Q_j = \sum_{i=N_j}^{N_{j+1}-1} Z_i \tag{3.5}$$

where n is the number of packets received.

The above equation shows not only the length of the RTO vector gets reduced, but also the observed RTO value changes. As an example, for FreeBSD 7.x the RTO vector without any packet loss is $\{3, 6, 12\}$. However, should the third packet be lost, then the RTO vector in that case would be $\{3, 18\}$. Therefore, the classification scheme we use, should deal with the above issue effectively.

D. Snacktime

Here we explore the need for augmenting feature vector, by showing that features currently used by Snacktime are not enough in accurately identifying an Operating System. We motivate this by showing how Snacktime fails to address the issues discussed is the previous section. Therefore, let S_{ij} be the j-th feature of the i-th class of operating system, such that $i = 1, 2, 3, \ldots, n$ and $j = 1, 2, 3, \ldots, k_i$, where k_i is the number of features of the i-th class. Given, the above information, Snacktime only considers the operating systems which have the same number of features in the signature i.e. the value of k are same. Snacktime, uses the following formula

when using receiver window size and TTL, to determine the score corresponding to a particular class, and the class with maximum score is declared to be the predicted class.

$$\begin{aligned} Y_i &= \max(|X_j - S_{ij}|, 10^{-6}) \\ U_i &= \sum_{j=1}^{n} \lceil -\log_{10} Y_i \rceil + W_i + T_i \end{aligned} \tag{3.6}$$

where W_i and T_i are the weights associated with receiver window size and TTL. If the receiver window size of the fingerprint and the signature match, W_i is assigned a value of 3, other a value of zero is assigned. T_i, is a assigned a value of 2, if the TTL of fingerprint is .less than the TTL of the signature and is within 32 seconds, a value of 1 is assigned if TTL of fingerprint is only less than TTL of signature. In all other cases a value of 0 is assigned to T_i.

Now, if we consider the fingerprint and signatures in Table I, and assume that the TTL and Win values are equivalent, the total score for Signature 1 is 11, while that of Signature 2 is 6. Thus the fingerprint is determined to belong to OS being represented by class 1, when in fact it should have been assigned to Signature 2, which may be deviating from the fingerprint due to the presence of noise. This behavior of Snacktime can be attributed to the fact that the RTO values are compared individually, when, in fact, the whole vector should have been compared at once. In response to this problem, we first augment the feature vector and then present an approach in Part V, that effectively deals with noise and packet loss. Secondly, the weights assigned for the receiver window size and TTL are totally arbitrary without and rigorous analysis. Finally, discrete and random features are used together in the classification algorithm, when in fact they should be treated differently.

Table I. Sample Snacktime Fingerprint and Signatures

Type	RTO1	RTO2
Fingerprint	3.000000	24.000000
Signature 1	3.000000	12.000020
Signature 2	3.299999	25.500879

CHAPTER IV

SNAP

In this section, we first show the feature vector for the proposed solution. We follow it, by talking about our signature database, and how the signature are preprocessed before being used in the classification algorithm. We end this section by introducing the proposed solution.

A. Feature Vector

An important aspect of operating system fingerprinting is that a signature representing an operating system should clearly differentiate it from other OSes. Hence, in this sub-section we consider the features that aid us in achieving the above goal. The features that we use are shown in Table II.

We use eight additional features as compared to Snacktime, which only uses *Win*, *TTL* and *SA-RTO*. In addition to considering RTO values between the SYN-ACK packets we consider the RTO value between the last SYN-ACK packet and RST/RST-ACK packet *R-RTO*, as shown in Fig. 1, which helps us in resolving ambiguities. For example, by considering the *R-RTO* in Table III we can resolve the ambiguity between Mac OS 10.1.x and NetBSD 4.0.1 shown in Table III, which would otherwise have similar *SA-RTO* patterns. Moreover by default, we consider RTO values greater than 65 seconds, which again helps us in better differentiating OSes. For instance without considering the last *SA-RTO* of Linux 2.0.X (Table III), the *SA-RTO* vector of both Linux 2.0.x and Linux 2.6.x would have similar *SA-RTO* vectors.

Additionally, we have found a large variety of behavior pertaining to the RST/RST-ACK packets. Because, the behavior is not well defined in the RFC, the properties of there RST/RST-ACK response, or lack of a response are very OS dependent. Thus,

Table II. Signature Description

Feature	Description	Type
Win	Default Receiver Window Size	M
TTL	Initial Time to live	M
SA-RTO	RTO values between SYN-ACK packets	MNL
R-RTO	RTO value between the last SYN-ACK packet and RST/RST-ACK packet	MNL
DF	DF bit set or not	M
OPT	TCP options and their sequence	F
RST	RST packet sent by the target	F
RA	RST-ACK packet sent by the target	F
R-Broken	Non-zero ACK value in RST Packet	F
R-Win	Receiver window size in RST/RST-ACK is zeroed	F
R-Seq	Sequence number incremented in the RST/RST-ACK packet	F

Table III. Sample Signatures

Operating System	Win	TTL	DF	RST	RA	R-Win	R-Seq	R-Broken	OPT (Order Mainained)	SA-RTO	R-RTO
Widows 7	8192	128	1	1	0	0	0	1	MSS, NOP, Window Scale, SACK Permitted, Timestamp	3.00, 6.00	12.00
Linux Kernel 2.6.x	5792	64	1	1	0	0	0	0	MSS, SACK Permitted, Timestamp, NOP, Window Scale	3.79, 5.90, 12.10, 24.00, 48.20	N.A.
Linux Kernel 2.0	32736	64	0	0	0	0	0	0	MSS	3.00, 6.00, 12.00, 24.00, 48.02, 96.0	N.A.
Mac OS 10.1.x	33304	64	1	0	1	1	1	0	MSS, NOP, Window Scale, NOP, NOP, Timestamp	2.92, 6.00, 12.00, 24.00	30.00
NetBSD 4.0.1	32768	64	1	0	0	0	0	0	MSS, NOP, Window Sclae, NOP, NOP, Timestamp, SACK, NOP, NOP	2.92, 6.00, 12.00, 24.00	N.A

five of our features are based on RST/RST-ACK packet. Then, depending on whether a RST or RST/ACK is sent by the target, we set the appropriate value to 1. We then consider the various peculiarities of RST/RST-ACK packets to extract the features *R-Win*, *R-Seq*, and *R-Broken*, which are explained in Table II.

Support for various TCP options differs between operating systems, and is OS dependent, as RFC 1232 is not specific about various implementation details. As a result, some operating systems support all of the standard TCP options (MSS, Window Scaling, SACK Permitted, SACK, NOP, and EOL), while others only support a subset of them. Moreover, the ordering of options in the TCP header of SYN-ACK packets varies among operating systems. So, similar to most other fingerprinting tools we also make use of TCP options (*Opt*) as one of the features of our signature.

The last feature we consider *DF* pertains to the Don't Fragment bit in the IP header. Some operating systems set this bit by default, while others clear this bit, so we identify this behavior and accordingly set the value of *DF* to 1 or 0. Most of the features discussed above have been used in other tools, with the exceptions of *R-Win* and *R-Seq*, which to the best of our knowledge haven't been used before.

It is important to understand that different features in the signature have different properties. Certain features like *Win*, *DF*, and *TTL* can be changed easily by either altering registry values or by editing specific configuration files, while features like *Opt* and those related to RST/RST-ACK packets are hard to change without hacking the kernel. Hence, based on the above nature of the features, features can be either Modifiable represented by *M* or Fixed represented by *F*. The remaining random features, *SA-RTO* and *R-RTO*, are not only random but are altered in noisy and lossy condition. Therefore, we identify the lossy nature of this features using *L*, random nature using *R* and noisy nature using N. Finally, note that, for embedded devices, unless they are behind normalizers [26] or scrubbers [27], it is difficult to

change any of the above mentioned features.

B. Signature Database

We now discuss our signature database, which currently contains 115 signatures. We prepared these signatures by either installing the operating system, or by using already-installed systems. Using our signature we can not only distinguish different operating systems like Windows, Linux etc, but we can also identify different versions of Windows, Linux, FreeBSD etc. We collected 50 samples for each operating system, so that we could use them as training data in our classification algorithm. These samples were collected over a period of 24 hours to account for different load conditions.

While preparing our signature database, we noticed that, in general, the first RTO value had much larger variations as compared to other RTO values. This could explained by noting that the first RTO timer might be starting between two clock ticks, hence resulting on an average error of half the clock tick period [28]. Also, we noticed that Windows Server 2003 was the only OS which seemed to have multiple RTO patterns. The first RTO ranges from anywhere between 2.5 and 4.5 seconds, while the second RTO was between 5.8 and 9.5 seconds.

C. Signature Preprocessing

In this section we provide details about the preprocessing we do with the signatures to make Snap robust against noise and packet loss.

1. Clustering

As a first step we group signatures together into different Clusters. This clustering is done based on the length of the signatures, type of the packet sent and overlapping of RTO values. As, an example all the operating systems which have a signature of length 4, which send only SYN-ACK packets and have overlapping RTO values are grouped together into one cluster. By overlapping RTO values we mean that the interval formed by the min and max value of the j-*th* RTO of S_i and by considering a threshold of 0.1 sec, should overlap with the corresponding interval for S_k. If all the RTO's of two signatures overlap and they satisfy the other two criteria of same length and type of packets, then they are grouped together. Now, in order for a new signature to be a part of this cluster, it should satisfy the above three requirements with all the signatures already present in the cluster. By applying clustering on our signature database, we end up with 58 cluster, and the average size of a cluster being 2.

Now, the main motivation behind clustering is to eliminate the affect of noise. By using clustering, instead of find an exact match for an unknown sample, we look for a cluster with similar RTO vector, thereby ignoring the noise that may have been introduced as a result of delay jitter. After finding the appropriate cluster we use other features like TCP options, receiver window size, TTL and DF to select an operating system which best matches the given unknown sample.

2. Cumulative RTO Values

We make use of cumulative RTO values in conjugation with binary search to deal with the packet loss. For cumulative RTO values, instead of considering the time difference between two SYN-ACK packets, we consider the time difference relative to

the first packet. The i-*th* Ccumulative RTO (C_i) is given using the following equation

$$C_i = \sum_{j=0}^{i-1} Z_j. \tag{4.1}$$

As we saw earlier, in case of packet loss the measured RTO values Z_i are drastically affected and also the number of RTO values are reduced. In the case of C_i only the number of RTO values get affected but the values remain the same, as the C_i values are measured relative to the first received packet. The only exception to this is when the first SYN-ACK packet gets lost, in which case all the C_i values get affected.

Let C_j represent the j-*th* cumulative RTO value of the fingerprint, and let C'_{ij} represent the j-*th* RTO value of the i-*th* signature. Now, to detect packet loss when considering a signature S_i, we do binary search for each of the C_j value on C'_i. Now, if Z and S_i are of the same length, we skip the binary search and assign a correspondence between them such that Z_j corresponds to the S_{ij}. If the signature is longer than fingerprint, then some of the values in the signature have no correspondence in the fingerprint and this help us in detecting the packet loss.

For e.g., the RTO vector for FreeBSD is $\{3, 6, 12\}$, which when represented in cumulative RTO form would be $\{3, 9, 21\}$ If the third packet is lost, the original RTO vector for the fingerprint would be $\{3, 18\}$, while the same in cumulative format would be $\{3, 21\}$. Now, on doing a binary search for 3 on the FreeBSD signature vector would result in it corresponding with the first RTO of the signature, while doing it for 21 would result it in corresponding to the third RTO of FreeBSD vector. Thus, the second RTO of the FreeBSD vector is unassigned thereby indicating the third packet was not received at the receiver end.

D. Classification Algorithm

In this subsection we discuss the algorithm used in Snap to detect the operating system of the target machine. As shown in Fig. 4, we begin by grouping the signatures into different clusters. This is followed by filtering the signatures based on reset parameters. If the RST or RST-ACK packets are not observed in the fingerprint, we skip this filtering process, as they might have been lost during transit. We do this filtering because the reset properties are highly implementation specific as RFC's make no mention about their implementation details. Next, we determine the best cluster corresponding the signature. We follow this by filtering based on the TCP options. Now, if no signature in the cluster has matching TCP options, we return with a no match. But, if there are signatures with the same ordering and support for TCP option, we determine which signature in the cluster has the most similar Win, DF and TTL values. If there is more than one signature having the maximum match for Win, DF and TTL values then we call it as a tie and return the cluster, otherwise the OS corresponding to the signature with the highest match for Win, DF and TTL values is returned as the predicted OS. Now the most important part of the algorithm discussed above is the identification of the correct cluster. The algorithm we use for this purpose is shown in Algorithm 1. Here we consider only those signatures which have a length greater than equal to the length of the fingerprint. Next, we perform binary search for all the remaining signatures. During binary search if more than one C_j value corresponds to any C'_{ij} value, we discard that signature. For each of the remaining signature we calculate the probability of it matching the fingerprint using a modified K-NN algorithm, which is explained next. Finally, we return the cluster corresponding to the signature with the highest probability.

In the modified K-NN algorithm, to compute the probability for each candidate

Algorithm 1 Snap RTO Matching Algorithm

$c_i = 1$ for all signatures {Candidates}

for all S_i where $Len_Z >= Len_{S_i}$ **do**

  for all C_j **do**

    $k = binarysearch(C_j, C'_i)$ {Returns corresponding signature RTO, -1 returned if the determined RTO is already assigned}

    if $k = -1$ **then**

      $c_i = 0$

      break

    else

      $pos_{ij} = k$ {Store corresponding Signature RTO position}

    end if

  end for

end for

$P = modified_k\text{-}NN(Z, S_i, c, pos)$

$P_i = P_i \times 0.038^{(Len_Z - Len_{S_i})}$ for all signatures

return $\max P$

signature i, we consider all the samples of a signature. Using the following equations we compute the probability for the fingerprint being generated by a particular operating system under the assumption of delay jitter having laplace distribution with a mean of 0.1 seconds.

$$P_i = P(S_i + \delta = Z) = \sum_{j=1}^{50} P(|\delta| = |S_i - Z|)$$
$$P(|\delta| = |S_i - Z|) = \prod_{k=1}^{n_k} P(\delta_{ik} = S_{ik} - Z_k) \tag{4.2}$$

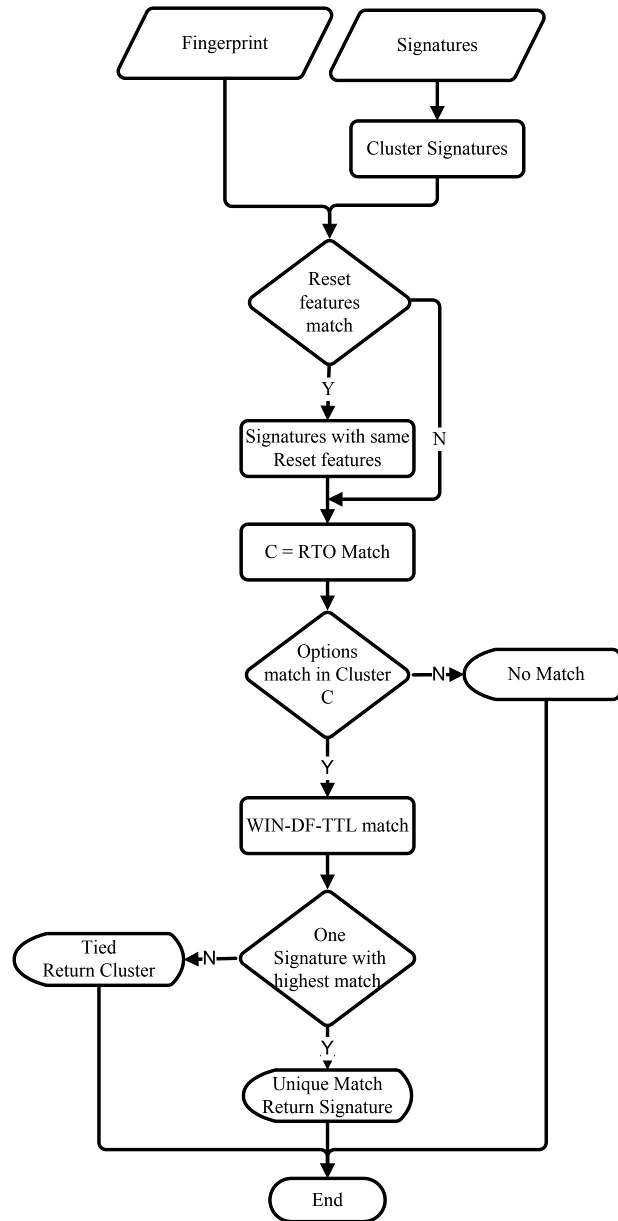


Fig. 4. Snap

CHAPTER V

EVALUATION

We evaluate Snap against Snacktime under four different conditions by using 10-fold cross-validation. In order to carry out 10-fold cross-validation, during each iteration we divide the data into two parts, test and training. The training data is used to train the classifier, while the test data is used to check the accuracy of the trained model. This process is repeated 10 times, each time selecting different data as test and training data.

In order to simulate the lossy and noisy nature of traffic on the Internet, we carry out Cross-validation under four different conditions. In the first case we assume that there is no noise and packet loss, while in the second case we assume noise affecting the RTO values. We assume that the jitter values are distributed as per Laplace distribution with a mean of 0.1 secs. The generated delay is then added or subtracted from the RTO values to simulate noise. As show earlier, packet loss on the internet is to the extent of 3.8% for SYN-ACK packets. In the third case, we randomly drop 3.8% of the packets we are considering to simulate packet loss. It is interesting to note that though the packet loss is to the extent of 3.8% the number of samples affected by packet loss are much larger. Finally, in the fourth case we introduce both noise and packet loss using the above mentioned models. Here, it is important to note that in all the cases noise and packet loss is only applied to the test data, while for training purposed we consider no noise and packet loss.

Table IV shows accuracy of Snap and Snacktime when considering the complete algorithm. As the table shows Snap performs significantly better than Snacktime under all conditions. Since Snacktime only considers signatures of same length as candidates, in presence of packet loss the accuracy falls by nearly 12%, when compared

Table IV. 10-fold Cross-validation Accuracy

Conditon	Snap Accuracy	Snacktime Accuracy
Normal	99.92%	84.62%
Noise	98.39%	81.43%
Packet Loss	98.86%	74.35%
Noise + Packet Loss	96.32%	72.45%

to normal conditions. This conforms to the above result which shows that a large number of samples are affected by just 3.8% loss of SYN-ACK packets.

CHAPTER VI

INTERNET SCAN

A. Scan Statistics

In order to determine the distribution of different operating systems on the Internet, we scan for all the publicly accessible machines running a web server on the Internet, using a custom scanner. We conducted the scan from a Windows Server 2008 machine with 16GB of RAM and 2 Dual core 1.70 GHz AMD Opteron processors. The SYN probe that was sent to each machine reassembles the SYN packet sent by a Windows Server 2008 machine with ECN and TCP timestamp option enabled. While, fingerprinting live hosts running a web server, we also did banner grabbing, so as to use the information for verification purposes.

As you can see in Table V nearly 37.8 million machines replied to our SYN probes. Of this majority of them supported MSS option, while nearly 2.32% machines support ECN. As Table VI shows nearly 75% of the hosts that responded to out SYN probe also replied to our GET request. The average size of the downloaded page that we could parse was 3.351 KB. This is due to the fact that nearly 10 million hosts replied with 4xx messages, which typically are smaller compared to other responses, as they only contain the error code and their description.

B. Results

In this subsection we analyze of the results of SYN scan used for OS detection. Of the nearly 37.8 million machines on the internet, we were able to uniquely identify 66.25% of these machines. By unique identification we mean that we were able to narrow down the OS running on that machine to one unique OS. As Table VII shows

Table V. SYN-ACK Scan Statistics

Property	# of hosts
Live Hosts	37,808,896
ECN Support	878,579 (2.32%)
MSS	37,610,829
Window Scaling	31,412,934
Timestamps	28,270,643
SACK Permitted	23,041,898
End Of Options List	2,349,369

Table VI. HTTP Scan Statistics

Property	# of hosts
Responses	28,592,538
Connection reset by peer	3,294,344
No data	2,942,658
Select Timeout	2,734,952
Connection timed out	371,262
Connection Refused	30,770
Software caused connection abort	1,341

Table VII. Top 5 Operating Systems

Operating System	# of hosts
Linux 2.6.x/2.4.x	7,315,946
VxWorks A	3,698,707
VxWorks B	2,044,608
Windows Server 2003 SP1 SP2	1,657,696
Windows XP 2002 SP3/Microsoft Server 2003	1,565,008

Linux 2.4.x/2.6.x is the most popular OS, with nearly 1/4th of the uniquely identified machines running it. The next most popular operating system is VxWorks, an embedded Real-time Operating System, used extensively in routers, modems, and printers etc. Here the second and the third most popular OS correspond to two different signatures for VxWorks. Ideally the port 80 should be blocked on these devices, but as the results show, it is not done, thus showing how insecure devices on the Internet are. At the fourth and fifth spot we have two different versions of Windows, It is interesting to see Windows XP at the 5th spot, but as TCP/IP stack of XP/2000 and 2003 are nearly the same, in many cases it is difficult to differentiae Windows Server 2003 from XP and 2000.

As shown earlier, we try to predict the cluster the machine may belong to, if we are not able to find an unique match for the machine. We end up identifying clusters for 10.35% of the machines (Table VIII), we found on the Internet. Linux again occupies the top spot for having the maximum number of hosts in a cluster. We have 4 different signatures for Linux, and these signatures are distinguished from each other by use of different receiver window size values. So, if we get a new receiver window size value with all the other features same, we end up assigning the machine to the

Table VIII. Top 5 Clusters

Cluster	# of hosts
Linux	2,601,436
ADSL Routers Cluster	419,952
Windows A	264,133
FreeBSD+VxWorks	237,680
Windows B	149,539

cluster. Hence, such a large number of values in the Linux cluster show that in many instance receiver window size values are changed by the user. Besides VxWorks being used extensively for routers and modems, we also found a large number of routers and modems running a particular OS which we could not identify, hence we label it as ADSL router, which is at the second position among the top 5 clusters. This is followed by a cluster of windows, composed of different versions of XP/2000/2003 in the third and fifth position. Finally, as the signature of a version of FreeBSD and one of the signatures of VxWorks i differ only in receiver window size, they are clustered together and they occupy the fourth position

Finally, in Table IX we show the number of hosts running various popular operating systems. It is interesting to see embedded systems occupying the top spot, thus indicating a large number of routers and modems run web servers. As expected embedded systems are followed by Linux, which is followed by Windows, FreeBSD and Mac. It should be noted that, the numbers presented in IX correspond to only uniquely identified OS.

Table IX. Common Operating Systems

Operating System	# of Hosts
Embedded	9,661,843
Linux	8,900,335
Windows	5,218,990
BSD	826,812
MacOS	80,918

C. Verification

Using the http-scan we did along with fingerprinting scan, we carry out basic verification. In order to verify the operating system of the machine we either used the HTTP server response header or the web-page itself. Also we manually looked at the fingerprint to identify the OS running on the target machine. The results of our verification are shown in Table X. Based on the above methodology and information, these results only give a basic idea about the accuracy of our scan and the results may vary as per the interpretation. In future we plan to carry out a much thorough verification and also compare our results with Nmap. As the results in Table X show, we had nearly perfect accuracy in almost all the cases except for the 5th OS, which had a lower accuracy because of a previously unknown OS is assigned to that class. If we include the signature for that unknown OS in our signature database, we again get 100% accuracy.

Table X. Basic Verification

OS	Correct	Incorrect	New signaturs
Linux 2.6.x/2.4.x	49	1	0
VxWorks A	50	0	0
VxWorks B	50	0	0
Windows Server 2003 SP1 SP2	50	0	0
Windows XP 2002 SP3/Microsoft Server 2003	45	0	5

CHAPTER VII

DEFENSES

Operating system fingerprinting poses a security threat, because once the operating system of a machine is known, one can tailor the attack based on it. Hence, in order to deal with this threat effectively it is required that we obfuscate the machine's operating system. One simple way to defeat IRLSnack is to randomize the number of retransmission tries [6] and also randomize the initial retransmission timeout value. Randomizing the initial timeout would suffice because in most cases, as successive timeout values in most as are computed using initial timeout values. To further add a layer of obfuscation one can make changes to the values of TCP/IP header fields, so as to resemble that of some other operating systems. The defenses suggested till now require that changes be made to each individual host, a network administrator can avoid doing that by ensuring that for all outgoing packets, the TCP/IP header values are similar to an arbitrary OS, and also by ensuring that for all machines in the network the number and value of RTO is again same as some random OS, preferably not present on the network.

CHAPTER VIII

CONCLUSION

In this paper we developed robust techniques so as to fingerprint all the publicly available machines on the internet. We were successful in identifying nearly 78% percent of the machines we discovered during our scan. In future to further increase the detection, we plan to start an open source project so as to increase the number of signature in the signature database. Also, we intend to do scans on different ports thereby obtaining a much larger picture of OS distribution on the internet.

REFERENCES

[1] IETF, "RFC 1122," http://www.faqs.org/rfcs/rfc1122.html; accessed October 2008.

[2] V. Paxson and M. Allman, "RFC 2988," http://www.faqs.org/rfcs/rfc2988.html; accessed November 2008.

[3] G. F. Leon, "Nmap," http://insecure.org/; accessed October 2008.

[4] E. Kollmann, "Chatter on the wire: A look at DHCP traffic," http://myweb.cableone.net/xnih/download/chatter-dhcp.pdf; accessed April 2010.

[5] F. V. Yarochkin, O. Arkin, M. Kydyraliev, S. Dai, Y. Huang, and S. Kuo, "Xprobe2++: Low volume remote network information gathering tool," in *Dependable Systems Networks*, Lisbon, Portugal, June 2009, pp. 205–210.

[6] T. Beardsley, "Snacktime," http://www.planb-security.net/wp/snacktime.html; accessed November 2008.

[7] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *Journal in Computer Virology*, vol. 6, pp. 197–205, November 2010.

[8] F. Veysset, O. Courtay, O. Heen, and Intranode Research Team, "New tool and technique for remote operating system fingerprinting," http://www.ouah.org/ring-full-paper.pdf; accessed November 2008.

[9] M. Zalewski, "p0f," http://lcamtuf.coredump.cx/p0f.shtml; accessed Novemeber 2008.

[10] H. K. J. Chu, "Tuning TCP parameters for the 21st Century," July 2009, http://www.ietf.org/proceedings/75/slides/tcpm-1.pdf; accessed April 2010.

[11] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proc. 13th USENIX Conference on System Administration*, Berkeley, CA, August 1999, pp. 229–238.

[12] C. Sarraute and J. Burroni, " Using neural networks to improve classical operating system fingerprinting techniques," *Electronic Journal of SADIO*, vol. 8, no. 1, pp. 35–57, January 2008.

[13] J. P. S. Medeiros, A. M. Brito, and P. S. M. Pires, "A data mining based analysis of Nmap operating system fingerprint database," in *Computational Intelligence in Security for Information Systems*, A. Herrero, P. Gastaldo, R. Zunino, and E. Corchado, Eds., vol. 63, pp. 208–221. Berlin: Springer, September 2009.

[14] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proc. WOOT '07: First USENIX Workshop on Offensive Technologies*, Berkeley, CA, August 2007, pp. 1–10.

[15] S. Guoqiang and D. Lee, "Network protocol system fingerprinting a formal approach," in *INFOCOM*, Barcelona, Spain, April 2006, pp. 1–12.

[16] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. S., and A. Blum, "FiG: Automatic fingerprint generation," in *Proc. Network and Distributed System Security Symposium*, San Diego, CA, Febuary 2007, http://www.cs.cmu.edu/s̃hobha/research/fig_ndss07_cr.pdf; accessed April 2010.

[17] J. P. S. Medeiros, A. M. Brito, and P. S. M. Pires, "An effective TCP/IP fingerprinting technique based on strange attractors classification," in *Data*

privacy management and autonomous spontaneous security, J. Garcia-Alfaro, G. Navarro-Arribas, N. Cuppens-Boulahia, and Y. Roudier, Eds., vol. 5939, pp. 208–221. Berlin: Springer, 2009.

[18] M. Zalewski, "Strange attractors and TCP/IP sequence number analysis," http://lcamtuf.coredump.cx/oldtcp/tcpseq.html; accessed April 2010.

[19] D. B. Berrueta, "A practical approach for defeating Nmap OS-fingerprinting," http://www.zog.net/Docs/nmap.html; accessed May 2010.

[20] R. Beck, "Passive-aggressive resistance: OS fingerprint evasion," *Linux J.*, vol. 2001, no. 89, pp. 1, 2001.

[21] D. Napier, "IPTables/NetFilter Linuxs nextgeneration stateful packet filter," *SysAdmin Magazine*, vol. 10, pp. 8, 10, 12, 14, 16, November 2001.

[22] C. Valli, "Honeyd - a OS fingerprinting artifice," in *Proc. 1st Australian Computer, Network and Information Forensics Conference*, Scarborough, Western Austraila, Novemeber 2003, scissec.scis.ecu.edu.au/conference_proceedings/2003/forensics/pdf/24_final.pdf; accessed May 2010.

[23] N. Provos, "A virtual honeypot framework," in *Proc. 13th USENIX Security Symposium*, San Diego, CA, August 2004, pp. 1–14.

[24] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP stack fingerprinting," in *Proc. 9th USENIX Security Symposium*, Berkeley, CA, 2000, pp. 17–17.

[25] R. Sojka, "IP packet delay variation," M.S. thesis, University of Applied Sciences Wiener Neustadt, Wiener Neustadt, Austria, March 2002.

[26] H. Mark, Ley, and V. Paxson, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *10th USENIX Security Symposium*, Berkeley, CA, August 2001, p. 9.

[27] G. R. Malan, D. Watson, F. Jahanian, and P. Howell, "Transport and application protocol scrubbing," in *INFOCOM*, Tel Aviv, Israel, March 2000, IEEE Computer and Communication Societies, pp. 1381–1390.

[28] R. Stevens, *TCP/IP illustrated, volume 1: The protocols*, Addison-Wesley, Reading, MA, 1994.

VITA

Ankur Bharatbhushan Nandwani received his Bachelors of Technology degree in computer engineering from National Institute of Technology-Surat, India. He received his Masters of Science degree in computer science in December 2010 at Texas A&M University, College Station.

His research interest include Internet measurements and Network security. He can be reached at:

c/o Department of Computer Science and Engineering

Texas A&M University

College Station

Texas - 77843

The typist for this thesis was Ankur Nandwani.