Global Information Assurance Certification Paper

A beginner's practical in Intrusion Detection Analysis

GIAC Intrusion Detection In Depth
GCIA Practical Assignment v3.2

SANS Parliament Square 2002, London

Antonia Rana

Part 1: Describe the State of Intrusion Detection

Probing remote services for reconnaissance using amap

Abtract

Gathering information about a remote host is often the first step in launching an attack. In order to break into a system exploiting some kind of vulnerability it is important to find as much information as possible. Port scanning, OS fingerprinting , banner grabbing are only some of the techniques that can be used. This paper summarises briefly the most common intelligence gathering techniques in use today, describing some of the tools that employ such techniques. Finally, a tool (amap) is presented which can be used to probe remote systems in the attempt to recognise an application listening on a non standard port.

Introduction

Gathering information about a remote system is often considered the first step an "intelligent hacker"[1] takes in launching an attack against or gain privileged access to a target machine. Intelligence gathered in this research can provide useful information about vulnerabilities or misconfigurations that can be successfully exploited by the potentail intruder. The more a hacker knows about a particular system (e.g. the OS, the hardware architecture and services that are running), the greater are his or her chances of launching a successful attack. By knowing the operating system and system type, a hacker can do a little research and come up with a list of known vulnerabilities.
Ofir Arkin describes in [4] a series of steps that an "intelligent hacker" would take in this intelligence gathering attempt:

- **Footprinting**: this phase consists in gathering as much information as possible on the target from authorised source of information (IP address ranges, DNS servers, mail servers);
- **Scanning**: this phase consists in determining which hosts in the targeted network are alive and reachable (through ping sweeps), which services they offer (through port scanning) and which operating systems they run (OS fingerprinting);
- **Enumeration**: this phase consists in extracting valid accounts or exported resources, system banners, routing tables, SNMP information, etc.

The second phase has an impact particularly strong on all networks since the number of automated scanners is constantly increasing and so is this type of traffic on the borders of every network.
Arkin also classifies the scan types according to the protocol used, as follows:

[1] The term "intelligent hacker" is used here to designate individuals who have knowledge of the systems they are dealing with, a deep understanding of the way they work and can program their own exploit programs, in contrast to "lamers" who simply, mechanically, execute scripts written by others.

PING SWEEPS: consists in querying multiple hosts using ICMP packets. It is an old approach to mapping and the scan is fairly slow. Automated tools for this scan include fping and gping on Unix, Pinger on Windows

BROADCAST ICMP: consists in sending echo requests to the network and/or broadcast address. Some operating system (Unix machines in general) will send back an ECHO REPLY to the attacker source IP, others will ignore these packets.

NON-ECHO ICMP: consists in sending ICMP messages different from ECHO REQUEST. This is useful when ECHO REQUESTS (PING) are filtered. Messages used for this purpose are ICMP type 13 (Timestamp request) and type 17 (address mask request). Automated tools for this type of scan include icmpush and icmpquery[2].

TCP SWEEPS: consists in sending a TCP ACK or SYN. Receiving a RST response is an indication that there is a host. However, information provided by this type of scan is not completely reliable if the target is behind a firewall that can reply with an RST packet on behalf of the targeted host. Tools that can be used for this type of scan include nmap and hping[3].

UDP SWEEPS: consists in sending a UDP packet. This method relies on the ICMP Port unreachable message as a reply to a UDP packet sent to a closed UDP port. This type of scan too can be done using nmap and hping.

All the above are used to determine if a host is alive, i.e. those hosts on a targeted network that are alive.
Port scanning, on the other hand, is used to determine which services are running on a host.
Port scanning techniques include:

TCP connect() scan:

A SYN is sent to an "interesting" port;
If a SYN/ACK is received, a service is listening and the TCP handshake phase is concluded by sending an ACK.

TCP half-opening scan:

A SYN is sent to an "interesting" port;
If a SYN/ACK is received, a service is listening, a RST packet is sent to close the connection.

Stealth scan:

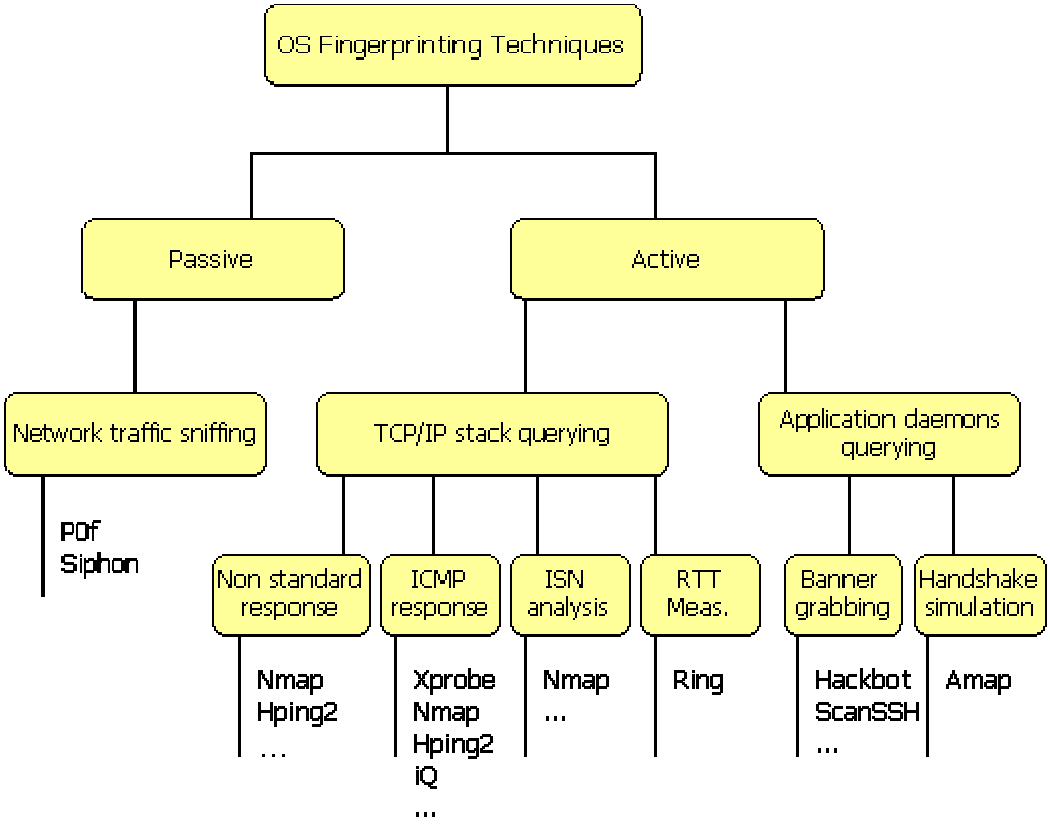
This is a technique that is meant to pass through filtering rules, not to be logged by system logging mechanisms. It consists in forging non-standard combination of TCP flags and relies on the fact that some filtering devices do not log a TCP connection if the three-way handshake is not completed.

[2] Available at http://packetstormsecurity.nl/UNIX/scanners/ (October 4th)
[3] Available at http://www.insecure.org/nmap and http://www.hping.org respectively (October, 4th)

SYN/ACK:

Packets are sent with SYN and ACK flags set. If a port is open, TCP replies with a RST because there is no SYN corresponding to the received SYN/ACK, otherwise the packet is discarded silently.

The techniques that are employed for port scanning are also successfully employed for identification of the remote operating systems (OS fingerprinting).
Basically, OS fingerprinting is a process for determining the operating system a remote host computer is running, based on characteristics of the data returned from the remote host. This can be as simple as connecting to the host and reading a service banner or as complex as statistical analysis of TCP initial sequence numbers and flags. OS fingerprinting is based on the fact that there are slight differences in the implementation of the TCP/IP stack from different vendors. In some cases, these differences can reveal information as detailed as the version number of the operating system and the processor architecture.
Tools are available today which that can tell with a high degree of precision which operating system is on the other side, by examining subtle details in the way TCP/IP was implemented in that particular system, they can be distinguished, according to the approach they follow, in passive and active fingerprinting.
The first approach consists in sending particular combinations of TCP flags or options (or ICMP messages) observing the responses obtained and comparing them to a database of known "fingerprints", while the second approach consists in monitoring (sniffing) incoming traffic and observing certain characteristics of the received packets.
Active port scanning and OS identification techniques are extensively described in [1], while [21] describes the basis of passive fingerprinting. More recently another approach has been described to remote fingerprinting based on the Round Trip Time (RTT) between a SYN and the SYN/ACK sent by the server. This approach is described in [16] which also presents a tool (ring) that has been implemented as a proof of concept for this approach.
An alternative method to TCP/IP stack fingerprinting is identification by using client application. These methods rely on the behaviour of certain daemons in error conditions or on the "greeting" information that some applications send as part of the application level handshaking process. Quite a number of network clients send revealing information about their host system, either directly or indirectly. Email clients, for example, often include a lot of information on their systems in the headers, [12] provides interesting information about the behaviour of the pine mail client in this respect. Web browsers also send this kind of information.
The different approaches to OS fingerprinting are summarised in the diagram in the following page (also described in [16]), some of the tools that employ the various techniques are also indicated.



Active TCP/IP Stack fingerprinting

Several publicly available tools exist that use active fingerprinting techniques. Of these tools nmap [1] seems to be the popular choice. Version 3.0 of nmap was released last August. Nmap uses several techniques for attempting to determine the host operating system from a network level, some of them primitive in their approach and others more complex, requiring a good understanding of the TCP/IP protocol. They include testing the response of the remote system to undefined combinations of TCP flags, TCP Initial Sequence Number (ISN) sampling, determining the default setting of the DF bit, TCP initial windows size, ToS setting, fragmentation handling, types and order of TCP options.

Nmap fingerprints a system in three steps: port scanning, which provides as a result a list of open and closed TCP and UDP ports; "ad-hoc forged" packets sending, analysis of the responses received and comparison against a database of known OS's behaviour (fingerprints).

In version 3, nmap has introduced the following additional features:

- protocol scan, which determines which protocols (TCP, IGMP, GRE, UDP, ICMP, etc.) are supported by a given host;
- "idlescan" which performs a scan via a "zombie" machine;
- ICMP timestamp and netmask requests;
- detection of host uptime;
- option to specify payload length
- IP Identification Number and TCP Initial Sequence Number predictability report;
- "random IP" scanning mode is capable of skipping unallocated netblocks;

Another tool that is very popular for use in active scanning is xprobe based on the work described in [23]. Xprobe introduced the use of ICMP messages for OS fingerprinting. Its first version was not very flexible as it did not have a signatures database, and relied on a static decision tree hardcoded in the binary code to produce the results. Xprobe v2.0 [9] is an evolution of xprobe. It uses a "fuzzy" approach to analyse the results produced by its various tests on the remote system. In this approach each fingerprinting test is implemented as a separate module. Upon initialisation, xprobe2 builds its own vector of possible "test matches" (i.e. builds a matrix associating a starting value for the various operating system that the software recognises). When the test is executed, the received packet is examined, the result is scored and put in the matrix. The "score" can be one of:

- `YES(3)`
- `PROBABLY_YES(2)`
- `PROBABLY_NO(1)`
- `NO(0)`

Once all tests are run, the scores for each test are summed. The top-score OS is declared as the final result.
The system is modular, new tests can be implemented and added as additional modules.
Other tools that deploy similar techniques are hping [3] and iQ [13].

Passive fingerprinting

Passive host fingerprinting is the practice of determining a remote operating system by measuring the peculiarities of observed traffic without actively sending probes to the host.
Five parameters are particularly useful in this technique:

- The value of the "Time to Live" field (TTL) in the IP header
- The Initial Window Size in the TCP header
- The value of the "Don't Fragment" bit (DF) in the IP header
- The value of the "Type of Service" (TOS) field in the IP header
- The types of TCP options used (if any)

No single signature can reliably determine the remote operating system. However, by looking at several signatures and combining the information, the accuracy of identifying the remote host increases.
Passive fingerprinting was first described in [21]. Tools based on this technique include p0f [24] and siphon [12].
Passive fingerprinting has some limitations. If used to analyse incoming traffic, it will not help in gathering useful information about malicious users since applications that build their own packets (such as nmap, hping, xprobe, etc.) will not use the same signatures as the operating system. In addition, it is relatively simple for a remote host to modify the default values for the TTL, Window Size, DF or TOS settings and, indeed this is considered one the countermeasures system administrators could and should take against passive fingerprinting.

Using RTT for TCP/IP Stack fingerprinting

A new approach to remote OS fingerprinting at the TCP/IP stack level is described in [16]. The technique described here relies on the fact that timeouts and regeneration cycles between a SYN sent by the client and successive SYN/ACK sent by the server to complete the TCP handshake are loosely specified in the RFC, which means that

almost each OS uses its own method and set of values. Ring is a tool that has been implemented to prove how the Round Trip Time can be effectively used to recognise the remote OS.
A typical ring identification session has the following steps:

1. ring sends a SYN packet to an open port of the target
2. the target enters the state "SYN_RCVD" and sends back a SYN-ACK
3. Ring ignores the SYN-ACK
4. the target remains in the SYN_RCVD state while reinjecting SYN-ACK segments from time to time. ring measures times between these segments.

Ring is extensively described in Tod Beardsley's GIAC practical[4].

Banner grabbing

One of the oldest techniques used to identify a remote operating system is "banner grabbing", which consists in opening a connection to a remote application daemon and determining the operating system by examining the responses received from applications like telnet or ftp.
Tools that use this technique span from scanners like Hackbot [10] and ScanSSH[11] to ad-hoc scripts aimed at particular application services [18] [19]. Hackbot is a bannergrabber that can scan for ftp, mail, ssh banner and DNS version, can perform whois lookup and various types of web scanning including Nimda and "path revealing NT problems" [10]. ScanSSH is a scanner that probes SSH servers and classifies them according to their advertised version number.
Fingerprinting at the application level is also extensively described in [12].

Defeating Fingerprint

Various techniques have also been described to defeat fingerprinting. Among them, the simplest and most immediate is the modification of the default values of a TCP/IP stack implementation, such as the TTL, Window Size or TCP options.
Another interesting approach can be found in [8] which describes the design and implementation of a TCP/IP stack "fingerprint scrubber". A "fingerprint scrubber" is a tool aimed at restricting a remote user's ability to determine the operating system of another host on the network. It is a piece of software that is transparently interposed between the Internet and the network under protection (a typical position would be on the firewall) and performs a set of kernel modifications to avoid recognition of the operating system based on the characteristics of IP and TCP implementations. It works both at the network and transport layers by converting ambiguous traffic from a heterogeneous group of hosts into sanitized packets that do not reveal clues about the hosts' operating systems. For example for all the packets generated by all hosts in the protected network it normalizes the IP header flags, forces all ICMP error messages to contain data payloads of only 8 bytes, keeps track of the open TCP connections by following the three-way handshake, and blocking all TCP packets that do not belong to a valid three-way handshake sequence, reorders the TCP options within the TCP header. According to [8] the fingerprint scrubber was tested against nmap which was completely unable to determine the operating system with the scrubber interposed.

[4] GIAC practicals are available at http://www.giac.org/GCIA.php (October 4th)

Probing application level services: amap0.95

In the previous sections various approaches to remote information gathering were described that allow identification of the remote Operating System or of the version of a particular application running on a remote host. A further step ahead in gathering information about a remote host is provided by amap [25]. Amap is a scanning tool that probes services running on a remote server on a given port to identify the specific application that is listening on that specific port. Its purpose is to be used to identify services that are not running on the standard ports. This tool has been released on March 2002 under the GNU General Public License and can be downloaded from http://www.thehackerschoice.com/download.php?t=r&d=amap-0.95.tar.gz. . It is also available as a package in the Debian Linux distribution. Its authors describe it as "a next-generation scanning tool, it identifies applications and services even if they are not listening on the default port by creating a bogus-communication. amap has a growing database of know applications also including non-ASCII based applications and even enterprise services.".

The purpose of the following sections is to explain how amap works and to present the results of its use in a test environment.

Amap probes the target by sending a number of "trigger" packets at the rate of about one per millisecond. By default it sends 16 such packets, this value can be modified with the "-T" option, however I counted 11 such packets in my tests, probably because there are only 11 different triggers defined in the signature files for TCP based application protocols. These "trigger" packets are typically the initiating packet of an application protocol handshake (see SSL example in the following section). Amap has a list of "triggers" which include binary as well as text handshake messages.

Triggers are defined in the file: ***appdefs.trig***. The triggers currently defined are shown in the following table:

SUNRPC	T/U	0x80 00 00 28 18 72 db 5a 00 00 00 00 00 00 00 02 00 01 86 a0 00 00 00 02 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
SSL	T	0x80 80 01 03 01 00 57 00 00 00 20 00 00 16 00 00 13 00 00 0a 07 00 c0 00 00 66 00 00 07 00 00 05 00 00 04 05 00 80 03 00 80 01 00 80 08 00 80 00 00 65 00 00 64 00 00 63 00 00 62 00 00 61 00 00 60 00 00 15 00 00 12 00 00 09 06 00 40 00 00 14 00 00 11 00 00 08 00 00 06 00 00 03 04 00 80 02 00 80 63 b9 b9 19 c0 2b ae 90 74 4c 73 eb 8b cf d8 55 ea d0 69 82 1b ef 23 c3 39 9b 8e b2 49 3c 5a 79
DNS	U	0xb3 65 01 00 00 01 00 00 00 00 00 00 03 31 33 36 02 37 33 03 31 35 39 03 31 39 34 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 0c 00 01
DNS	U	0xdd d9 01 00 00 01 00 00 00 00 00 00 02 31 30 01 30 03 31 36 38 03 31 39 32 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 0c 00 01
DNS	T	0x00 1e 3b 6f 01 00 00 01 00 00 00 00 00 00 05 68 34 78 30 72 02 6e 6c 00 00 fc 00 01
NETBIOS	T/U	0x79 08 00 00 00 01 00 00 00 00 00 00 20 43 4b 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00 00 21 00 01
HTTP	T	"HEAD / HTTP/1.0\n\n"
LDAP	T	0x30 0c 02 01 01 60 07 02 01 02 04 00 80 00
SAP-R3	T	0x00 00 01 06 ff ff ff ff 0a 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3e 00 00 00 00 ff ff ff ff ff ff 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 70 65 6e 74

		65 73 74 00 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 2d 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00 ff ff ff ff 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00 10 04 02 00 0c 00 01 87 68 00 00 04 4c 00 00 03 e8 10 04 0b 00 20 ff 7f ca 0d c8 b3 66 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
RPCS	U	0x03 9b 65 42 00 00 00 00 00 00 00 02 00 0f 42 43 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
MOUNTD	T	0x80 00 00 60 77 b7 3b 30 00 00 00 00 00 00 00 02 00 01 86 a5 00 00 00 01 00 00 00 05 00 00 00 01 00 00 00 38 3c 71 4d 94 00 00 00 07 6b 70 6d 67 2d 70 74 00 00 00 00 00 00 00 00 00 00 00 00 07 00 00 00 00 00 00 00 01 00 00 00 0e 00 00 00 0f 00 00 00 10 00 00 00 11 00 00 ff fe 00 00 00 00 00 00 00 00
X_WINDOWS	T	0x6c 00 0b 00 00 00 12 00 10 00 00 00 4d 49 54 2d 4d 41 47 49 43 2d 43 4f 4f 4b 49 45 2d 31 00 00 c6 17 34 b7 89 ed 65 c0 93 fd d8 56 66 fa 52 40
SNMP_PUBLIC	U	0x30 82 00 2f 02 01 00 04 06 70 75 62 6c 69 63 a0 82 00 20 02 04 4c 33 a7 56 02 01 00 02 01 00 30 82 00 10 30 82 00 0c 06 08 2b 06 01 02 01 01 05 00 05 00
NTP	U	0xcb 00 04 fa 00 01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bf be 70 99 cd b3 40 00
LDAP	T	0x30 0c 02 01 01 60 07 02 01 02 04 00 80 00

The hex string in the table (indicated by a 0x before the first octet) is sent as the payload of the "trigger" packet in the first message sent after the completion of the TCP handshake or in the UDP datagram (depending on whether the service uses TCP or UDP as transport). This list can be expanded very easily, provided one knows the handshake message of the application that one wants to trigger.
Amap defines a format for describing the trigger:

```
PROTO_ID:<t|u>:<0|1>:<optional trigger data>
```

Where ":" is the separator and:

PROTO_ID: is the name of the application level protocol (service) for which a handshake trigger is provided (e.g. SSL, Telnet, etc.). This value is looked up when the "p" command line option is used.

"t" or "u" indicates whether TCP or UDP must be used as transport

"0|1" is a flag to mark "dangerous" protocols. These are applications that might crash if unexpected or long data is received". When the "H" command line option is specified, triggers with a value of 1 in this field will not be sent.

<optional trigger data> can be an hex string or a ascii string depending on the application. A hex string is identified by a leading "0x". All strings are terminated with a newline character ("\n"). A trigger string is not defined for application protocols that provide a banners string upon successful completion of the TCP handshake (e.g. mail servers, ftp servers, ssh daemons, etc.). These will be simply recognised with the

same mechanism used by any banner grabbing tool.
After the trigger has been sent, amap then looks up the response in a list, contained in the file ***appdefs.resp*** and prints out any match it finds.
The possible responses are contained in this file with the following format:

```
PROTO_ID:<response string>
```

Where ":" is the separator and:

PROTO_ID:	is the name of the application level protocol (service) containing the string <*response string*> in its response.
<*response string*>	can be an ASCII string or a binary string, like in the triggers and can be prepended with either a "^", meaning that the specified string must be found at the beginning of the response, or by a "/" meaning that the specified string must be found somewhere in the received string.

As for the "triggers", it is very easy to expand the list of "recognised" services by providing the appropriate description in this file.
Amap supports both tcp and udp protocols, ASCII and binary protocols and provides a number of options to tune the probe being sent. It can take an nmap machine-readable output file as its input file and probe the services that are listening on ports found open by nmap.
The options currently available are described below:

- i <*filename*>	Reads hosts and ports from the specified file. The format of this file is as obtained by nmap using the option "-m"
- sT	Scan only TCP ports
- sU	Scan only UDP ports
- d	Print the hex dump of the received response. The default is to print only the responses that are recognised
- b	Print ASCII banners if any are received from the probed service
- o <*filename*>	Log results to <*filename*>
- D <*filename*>	Reads triggers and responses definitions from <*filename*>, instead of the defaults appdefs.trig and appdefs.resp
-p <*protocol*>	Indicates that only the trigger associated to <*protocol*> must be used
-T n	Open "n" parallel connections. The default is indicated as 16 in the manual pages, however, I counted only 11 in all tests I made.
- t n	Wait "n" seconds for a response. Default is 5.
- H	Skip potentially harmful triggers. This swill skip triggers that are marked with the 1 flag in the triggers description file (appdefs.trig)

The syntax for running amap is:

```
amap [-sT|-sU] [options] [target port| -I <filename>]
```

Either –sT or –sU must be specified. "target" is the IP address or fully qualified name of the probed host and "port" is the probed port number. Target and port must not be specified if the "-i" option is used.

Testing amap

Amap was downloaded from http://www.thehackerschoice.com/ and compiled on a machine running RedHat Linux 7.2.
No changes were made to the default configurations.
The test environment included the RedHat 7.2 machine running amap at the address 10.0.0.2 and the "target" host running Debian 3.0 at the address 10.0.0.1, both hosts on the same subnet. A number of services were activated on the debian host, for most of them the default port was changed to verify that amap could correctly recognise the applications listening on the ports probed.
Tcpdump was activated on the RedHat host to record the traffic exchanged between the two hosts.
Amap was used to probe services listening on TCP ports.
Services were distributed as follows:

389/tcp	LDAP (not modified)
80/tcp	SSL (HTTPS)
31/tcp	FTP
21/tcp	SSH
22/tcp	TELNET

When amap was started, in each probe, 11 TCP connections were opened, SYN packets being sent at a few milliseconds one after the other. Amap forks as many child processes as the number of parallel connections specified with the –T option. Once the TCP handshake is completed, amap sends the one trigger packet per each trigger found in the appdefs.trig file for the chosen protocol (TCP in this case). In addition, it sends a trigger packet containing the string "\rnHELP\r\n".
Upon reception of the response from the server, amap checks in the appdefs.resp file for a match with the pre-defined responses. The response form the server can be either a banner or an error or a response to the handshake initiated by the amap trigger. Some application would also send error messages back to amap. As soon as a message is received from the server, the corresponding TCP connection is closed. Obviously, depending on the level of logging of the application listening on the probed port, an error will be recorded on the log file for each "wrong" trigger received. Finding eleven connections open from the same host all of which, except possibly one, generating errors on the application level protocol, could be a good indication of a probe from amap.
The next two sections describe the results of running amap against an application that responds with an ASCII banner (FTP) and an application that requires the successful completion of a binary handshake.

"Text banner" applications: ftp

The traces provided in this section show an extract of a probe on port 31 (running ftp). Amap was run on 10.0.0.2 with the following options:

```
Redhat#./amap -sT -d -b -o amap.result 10.0.0.1 31
```

For brevity, only some of the connections are shown and the payload is shown only for data transfer packets (PUSH and ACK bits set).

```
21:07:02.366476 10.0.0.2.1080 > 10.0.0.1.31: S [tcp sum ok] 1462036788:1462036788(0) win 5840 <mss 1460,sackOK,timestamp 118935 0,nop,wscale 0>
(DF) (ttl 64, id 14755, len 60)
21:07:02.366476 10.0.0.1.31 > 10.0.0.2.1080: S [tcp sum ok] 1366211808:1366211808(0) ack 1462036789 win 5792 <mss 1460,sackOK,timestamp 9844055
118935,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:07:02.366476 10.0.0.2.1080 > 10.0.0.1.31: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 118935 9844055> (DF) (ttl 64, id 14756, len 52)
21:07:02.366476 10.0.0.2.1080 > 10.0.0.1.31: P [tcp sum ok] 1:49(48) ack 1 win 5840 <nop,nop,timestamp 118935 9844055> (DF) (ttl 64, id 14757,
len 100)
0x0000   4500 0064 39a5 4000 4006 ecec 0a00 0002     E..d9.@.@.......
0x0010   0a00 0001 0438 001f 5724 e935 516e bce1     .....8..W$.5Qn..
0x0020   8018 16d0 c6df 0000 0101 080a 0001 d097     ................
0x0030   0096 3557 6c00 0b00 0000 1200 1000 0000     ..5Wl...........
0x0040   4d49 542d 4d41 4749 432d 434f 4f4b 4945     MIT-MAGIC-COOKIE
0x0050   2d31 0000 c617 34b7 89ed 65c0 93fd d856     -1....4...e....V
0x0060   66fa 5240                                   f.R@
21:07:02.366476 10.0.0.1.31 > 10.0.0.2.1080: . [tcp sum ok] ack 49 win 5792 <nop,nop,timestamp 9844055 118935> (DF) (ttl 64, id 35254, len 52)
21:07:02.366476 10.0.0.2.1081 > 10.0.0.1.31: S [tcp sum ok] 1456381211:1456381211(0) win 5840 <mss 1460,sackOK,timestamp 118935 0,nop,wscale 0>
(DF) (ttl 64, id 19852, len 60)
21:07:02.366476 10.0.0.1.31 > 10.0.0.2.1081: S [tcp sum ok] 1373954753:1373954753(0) ack 1456381212 win 5792 <mss 1460,sackOK,timestamp 9844056
118935,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:07:02.366476 10.0.0.2.1081 > 10.0.0.1.31: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 118935 9844056> (DF) (ttl 64, id 19853, len 52)
21:07:02.366476 10.0.0.2.1081 > 10.0.0.1.31: P [tcp sum ok] 1:15(14) ack 1 win 5840 <nop,nop,timestamp 118935 9844056> (DF) (ttl 64, id 19854,
len 66)
0x0000   4500 0042 4d8e 4000 4006 d925 0a00 0002     E..BM.@.@..%....
0x0010   0a00 0001 0439 001f 56ce 9d1c 51e4 e2c2     .....9..V...Q...
0x0020   8018 16d0 58f2 0000 0101 080a 0001 d097     ....X...........
0x0030   0096 3558 300c 0201 0160 0702 0102 0400     ..5X0....`......
0x0040   8000                                        ..
21:07:02.366476 10.0.0.1.31 > 10.0.0.2.1081: . [tcp sum ok] ack 15 win 5792 <nop,nop,timestamp 9844056 118935> (DF) (ttl 64, id 55194, len 52)
21:07:02.396476 10.0.0.1.31 > 10.0.0.2.1081: P [tcp sum ok] 1:69(68) ack 15 win 5792 <nop,nop,timestamp 9844080 118935> (DF) [tos 0x10]  (ttl 64,
id 55195, len 120)
0x0000   4510 0078 d79b 4000 4006 4ed2 0a00 0001     E..x..@.@.N.....
0x0010   0a00 0002 001f 0439 51e4 e2c2 56ce 9d2a     .......9Q...V..*
0x0020   8018 16a0 91a1 0000 0101 080a 0096 3570     ..............5p
0x0030   0001 d097 3232 3020 6465 6269 616e 2046     ....220.debian.F
0x0040   5450 2073 6572 7665 7220 2856 6572 7369     TP.server.(Versi
0x0050   6f6e 2036 2e34 2f4f 7065 6e42 5344 2f4c     on.6.4/OpenBSD/L
0x0060   696e 7578 2d66 7470 642d 302e 3137 2920     inux-ftpd-0.17).
0x0070   7265 6164 792e 0d0a                         ready...
21:07:02.396476 10.0.0.2.1081 > 10.0.0.1.31: . [tcp sum ok] ack 69 win 5840 <nop,nop,timestamp 118938 9844080> (DF) (ttl 64, id 19855, len 52)
21:07:02.396476 10.0.0.1.31 > 10.0.0.2.1081: P [tcp sum ok] 69:113(44) ack 16 win 5792 <nop,nop,timestamp 9844082 118938> (DF) [tos 0x10]  (ttl
64, id 55196, len 96)
0x0000   4510 0060 d79c 4000 4006 4ee9 0a00 0001     E..`..@.@.N.....
0x0010   0a00 0002 001f 0439 51e4 e306 56ce 9d2b     .......9Q...V..+
0x0020   8018 16a0 af5a 0000 0101 080a 0096 3572     .....Z........5r
0x0030   0001 d09a 3530 3020 2730 0c02 0101 6007     ....500.’0....`.
```

```
0x0040   0201 0204 273a 2063 6f6d 6d61 6e64 206e     ....':.command.n
0x0050   6f74 2075 6e64 6572 7374 6f6f 642e 0d0a     ot.understood...
21:07:02.396476 10.0.0.2.1081 > 10.0.0.1.31: R [tcp sum ok] 1456381227:1456381227(0) win 0 (DF) [tos 0x10]  (ttl 255, id 0, len 40)
21:07:02.406476 10.0.0.1.31 > 10.0.0.2.1080: P [tcp sum ok] 1:69(68) ack 49 win 5792 <nop,nop,timestamp 9844096 118935> (DF) [tos 0x10]  (ttl 64,
id 35255, len 120)
0x0000   4510 0078 89b7 4000 4006 9cb6 0a00 0001     E..x..@.@.......
0x0010   0a00 0002 001f 0438 516e bce1 5724 e965     .......8Qn..W$.e
0x0020   8018 16a0 6b58 0000 0101 080a 0096 3580     ....kX........5.
0x0030   0001 d097 3232 3020 6465 6269 616e 2046     ....220.debian.F
0x0040   5450 2073 6572 7665 7220 2856 6572 7369     TP.server.(Versi
0x0050   6f6e 2036 2e34 2f4f 7065 6e42 5344 2f4c     on.6.4/OpenBSD/L
0x0060   696e 7578 2d66 7470 642d 302e 3137 2920     inux-ftpd-0.17).
0x0070   7265 6164 792e 0d0a                         ready...
21:07:02.406476 10.0.0.2.1080 > 10.0.0.1.31: . [tcp sum ok] ack 69 win 5840 <nop,nop,timestamp 118939 9844096> (DF) (ttl 64, id 14758, len 52)
21:07:02.406476 10.0.0.2.1080 > 10.0.0.1.31: F [tcp sum ok] 49:49(0) ack 69 win 5840 <nop,nop,timestamp 118939 9844096> (DF) (ttl 64, id 14759,
len 52)
21:07:02.406476 10.0.0.1.31 > 10.0.0.2.1080: P [tcp sum ok] 69:103(34) ack 50 win 5792 <nop,nop,timestamp 9844097 118939> (DF) [tos 0x10]  (ttl
64, id 35256, len 86)
0x0000   4510 0056 89b8 4000 4006 9cd7 0a00 0001     E..V..@.@.......
0x0010   0a00 0002 001f 0438 516e bd25 5724 e966     .......8Qn.%W$.f
0x0020   8018 16a0 fa0e 0000 0101 080a 0096 3581     ..............5.
0x0030   0001 d09b 3530 3020 274c 273a 2063 6f6d     ....500.'L':.com
0x0040   6d61 6e64 206e 6f74 2075 6e64 6572 7374     mand.not.underst
0x0050   6f6f 642e 0d0a                             ood...
21:07:02.406476 10.0.0.2.1080 > 10.0.0.1.31: R [tcp sum ok] 1462036838:1462036838(0) win 0 (DF) [tos 0x10]  (ttl 255, id 0, len 40)
```

Amap successfully recognised ftp listening on port 31:

```
Amap v0.95 started at Fri Sep 27 21:07:02 2002
Ports: 0, triggers 0. Total amount of tasks to perform: 11
Protocol on IP 10.0.0.1 port 31 tcp matches FTP - banner: 220 debian FTP server
(Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.\r\n
Unidentified ports: None.
Amap v0.95 ended at Fri Sep 27 21:07:02 2002
```

Recognition of the ftp service is based on the banner received from the server. In particular, the match of the response received from the server with the string:

On the server side, the following error messages are logged in the syslog file. Error messages are also sent back to the client.

```
Sep 27 21:05:26 debian ftpd[2905]: <--- 220
Sep 27 21:05:26 debian ftpd[2905]: debian FTP server (Version 6.4/OpenBSD/Linux-ftpd-
0.17) ready.
Sep 27 21:05:26 debian ftpd[2905]: command:
Sep 27 21:05:26 debian ftpd[2905]: <--- 500
Sep 27 21:05:26 debian ftpd[2905]: '': command not understood.
Sep 27 21:05:26 debian ftpd[2905]: command: HELP
Sep 27 21:05:26 debian ftpd[2905]: lost connection
Sep 27 21:05:26 debian in.ftpd[2906]: connect from 10.0.0.2
Sep 27 21:05:26 debian ftpd[2906]: <--- 220
Sep 27 21:05:26 debian ftpd[2906]: debian FTP server (Version 6.4/OpenBSD/Linux-ftpd-
0.17) ready.

[snip - same log for each attempted connection]
```

"Binary handshake" application: SSL

The traces provided here show how amap can simulate an SSL connection and recognise an SSL application running on port 80.
The steps involved in the SSL handshake are as follows:

1. The client sends the CLIENT_HELLO message containing:
    - Client's SSL version number
    - Supported ciphering schemes
    - Challenge
2. The server sends the SERVER_HELLO message containing:
    - Handshake type (server hello)
    - Server's SSL version
    - Cipher settings
    - Cipher suite
    - Session_ID
    - Random number
    - Timestamp
    - Compression method
3. The server then sends its certificate
    - Handshake type (certificate)
    - Server certificate

Messages 2 and 3 can be combined into a single message like in the trace below.
The trigger that is used for SSL probing is the starting message of the SSL handshake, i.e. the CLIENT_HELLO message. The binary string contained in the appdefs.trig file and actually sent by amap is:

```
0x80 80 01 03 01 00 57 00 00 00 20 00 00 16 00 00 13 00 00 0a 07 00 c0
00 00 66 00 00 07 00 00 05 00 00 04 05 00 80 03 00 80 01 00 80 08 00
80 00 00 65 00 00 64 00 00 63 00 00 62 00 00 61 00 00 60 00 00 15 00
00 12 00 00 09 06 00 40 00 00 14 00 00 11 00 00 08 00 00 06 00 00 03
04 00 80 02 00 80 63 b9 b9 19 c0 2b ae 90 74 4c 73 eb 8b cf d8 55 ea
d0 69 82 1b ef 23 c3 39 9b 8e b2 49 3c 5a 79
```

The decoded equivalent of this string is (decoding has been obtained using ethereal [22]:

```
SSLv2 Record Layer: Client Hello
    Length: 128
    Handshake Message Type: Client Hello (1)
    Version: TLS 1.0 (0x0301)
    Cipher Spec Length: 87
    Session ID Length: 0
    Challenge Length: 32
    Cipher Specs (29 specs)
        Cipher Spec: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x000016)
        Cipher Spec: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x000013)
        Cipher Spec: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x00000a)
        Cipher Spec: SSL2_DES_192_EDE3_CBC_WITH_MD5 (0x0700c0)
        Cipher Spec: TLS_DHE_DSS_WITH_RC4_128_SHA (0x000066)
        Cipher Spec: TLS_RSA_WITH_IDEA_CBC_SHA (0x000007)
        Cipher Spec: TLS_RSA_WITH_RC4_128_SHA (0x000005)
        Cipher Spec: TLS_RSA_WITH_RC4_128_MD5 (0x000004)
        Cipher Spec: SSL2_IDEA_128_CBC_WITH_MD5 (0x050080)
        Cipher Spec: SSL2_RC2_CBC_128_CBC_WITH_MD5 (0x030080)
        Cipher Spec: SSL2_RC4_128_WITH_MD5 (0x010080)
        Cipher Spec: SSL2_RC4_64_WITH_MD5 (0x080080)
        Cipher Spec: TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA (0x000065)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x000064)
        Cipher Spec: TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x000063)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x000062)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC2_CBC_56_MD5 (0x000061)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC4_56_MD5 (0x000060)
        Cipher Spec: TLS_DHE_RSA_WITH_DES_CBC_SHA (0x000015)
        Cipher Spec: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x000012)
        Cipher Spec: TLS_RSA_WITH_DES_CBC_SHA (0x000009)
        Cipher Spec: SSL2_DES_64_CBC_WITH_MD5 (0x060040)
        Cipher Spec: TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x000014)
        Cipher Spec: TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA (0x000011)
        Cipher Spec: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x000008)
        Cipher Spec: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x000006)
        Cipher Spec: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x000003)
        Cipher Spec: SSL2_RC2_CBC_128_CBC_WITH_MD5 (0x040080)
        Cipher Spec: SSL2_RC4_128_EXPORT40_WITH_MD5 (0x020080)
    Challenge (63 b9 b9 19 c0 2b ae 90 74 4c 73 eb 8b cf d8 55 ea d0 69 82 1b ef 23 c3
39 9b 8e b2 49 3c 5a 79)
```

The response received from the server that allows amap to recognize SSL is (the decoded format has been obtained using ethereal [22], the content of the certificate is not shown for brevity, but it can be seen in the trace in the following section)

```
    TLS Record Layer: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.0 (0x0301)
        Length: 74
        Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 70
            Version: TLS 1.0 (0x0301)
            Random.gmt_unix_time: Sep 27, 2002 20:05:00.000000000
            Random.bytes
            Session ID Length: 32
            Session ID (32 bytes)
            Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
            Compression Method: null (0)
    TLS Record Layer: Certificate
        Content Type: Handshake (22)
```

```
        Version: TLS 1.0 (0x0301)
        Length: 590
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 586
            Certificates Length: 583
            Certificates (583 bytes)
                Certificate Length: 580
                Certificate (580 bytes)
    TLS Record Layer: Server Hello Done
        Content Type: Handshake (22)
        Version: TLS 1.0 (0x0301)
        Length: 4
        Handshake Protocol: Server Hello Done
            Handshake Type: Server Hello Done (14)
            Length: 0
```

Amap was run on 10.0.0.2 with the following options:

```
Redhat#./amap -sT -d -b -o amap.result 10.0.0.1 80
```

The following page shows the tcpdump log recorded during the probing on port 80. Hex dump is shown only for data transfers for brevity. All the connections opened by amap are shown as well as the all the triggers sent in one run of amap. Payload in red is the triggers sent by amap (Application protocol probed is indicated beside). Payload in blue is the response sent by the server. In this case, the probed service replies only to the correct trigger (i.e. the SSL CLIENT_HELLO handshake message).

The tcpdump record of the amap probes is (hex dump of the packet is shown only for data exchange packets and not for SYN, SYN/ACK, ACK, FIN, RST packets for brevity)

```
21:06:36.236476 10.0.0.2.1060 > 10.0.0.1.80: S [tcp sum ok] 1428200370:1428200370(0) win 5840 <mss 1460,sackOK,timestamp 116322 0,nop,wscale 0>
(DF) (ttl 64, id 24016, len 60)
21:06:36.236476 10.0.0.1.80 > 10.0.0.2.1060: S [tcp sum ok] 1345344243:1345344243(0) ack 1428200371 win 5792 <mss 1460,sackOK,timestamp 9817924
21:06:36.236476 10.0.0.2.1060 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116322 9817924> (DF) (ttl 64, id 24017, len 52)
21:06:36.256476 10.0.0.2.1060 > 10.0.0.1.80: P [tcp sum ok] 1:9(8) ack 1 win 5840 <nop,nop,timestamp 116324 9817924> (DF) (ttl 64, id 24018, len
60)
0x0000   4500 003c 5dd2 4000 4006 c8e7 0a00 0002     E..<].@.@.......
0x0010   0a00 0001 0424 0050 5520 9bb3 5030 52f4     .....$.PU...P0R.
0x0020   8018 16d0 6e85 0000 0101 080a 0001 c664     ....n..........d
0x0030   0095 cf44 0d0a 4845 4c50 0d0a               ...D..HELP..                  GENERIC "HELP"
21:06:36.256476 10.0.0.1.80 > 10.0.0.2.1060: . [tcp sum ok] ack 9 win 5792 <nop,nop,timestamp 9817937 116324> (DF) (ttl 64, id 40369, len 52)
21:06:36.276476 10.0.0.2.1061 > 10.0.0.1.80: S [tcp sum ok] 1428096922:1428096922(0) win 5840 <mss 1460,sackOK,timestamp 116326 0,nop,wscale 0>
(DF) (ttl 64, id 49660, len 60)
21:06:36.276476 10.0.0.1.80 > 10.0.0.2.1061: S [tcp sum ok] 1332924456:1332924456(0) ack 1428096923 win 5792 <mss 1460,sackOK,timestamp 9817966
116326,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.276476 10.0.0.2.1061 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116326 9817966> (DF) (ttl 64, id 49661, len 52)
21:06:36.286476 10.0.0.2.1061 > 10.0.0.1.80: P [tcp sum ok] 1:45(44) ack 1 win 5840 <nop,nop,timestamp 116327 9817966> (DF) (ttl 64, id 49662,
len 96)
0x0000   4500 0060 c1fe 4000 4006 6497 0a00 0002     E..`..@.@.d.....
0x0010   0a00 0001 0425 0050 551f 079b 4f72 d029     .....%.PU...Or.)
0x0020   8018 16d0 39e0 0000 0101 080a 0001 c667     ....9.........g
0x0030   0095 cf6e 8000 0028 1872 db5a 0000 0000     ...n...(.r.Z....
0x0040   0000 0002 0001 86a0 0000 0002 0000 0004     ................              SUNRPC
0x0050   0000 0000 0000 0000 0000 0000 0000 0000     ................
21:06:36.286476 10.0.0.1.80 > 10.0.0.2.1061: . [tcp sum ok] ack 45 win 5792 <nop,nop,timestamp 9817968 116327> (DF) (ttl 64, id 8007, len 52)
21:06:36.286476 10.0.0.1.80 > 10.0.0.2.1061: R [tcp sum ok] 1:1(0) ack 45 win 5792 <nop,nop,timestamp 9817968 116327> (DF) (ttl 64, id 8008, len
52)
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: S [tcp sum ok] 1428512848:1428512848(0) win 5840 <mss 1460,sackOK,timestamp 116330 0,nop,wscale 0>
(DF) (ttl 64, id 42554, len 60)
21:06:36.316476 10.0.0.1.80 > 10.0.0.2.1062: S [tcp sum ok] 1345455902:1345455902(0) ack 1428512849 win 5792 <mss 1460,sackOK,timestamp 9818002
116330,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116330 9818002> (DF) (ttl 64, id 42555, len 52)
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: P [tcp sum ok] 1:131(130) ack 1 win 5840 <nop,nop,timestamp 116330 9818002> (DF) (ttl 64, id 42556,
len 182)
0x0000   4500 00b6 a63c 4000 4006 8003 0a00 0002     E....<@.@.......
0x0010   0a00 0001 0426 0050 5525 6051 5032 071f     .....&.PU%`QP2..
0x0020   8018 16d0 c598 0000 0101 080a 0001 c66a     ...............j
0x0030   0095 cf92 8080 0103 0100 5700 0000 2000     ..........W.....
0x0040   0016 0000 1300 000a 0700 c000 0066 0000     .............f..
0x0050   0700 0005 0000 0405 0080 0300 8001 0080     ................              SSL     Client Hello
0x0060   0800 8000 0065 0000 6400 0063 0000 6200     .....e..d..c..b.
0x0070   0061 0000 6000 0015 0000 1200 0009 0600     .a..`...........
```

```
0x0080   4000 0014 0000 1100 0008 0000 0600 0003     @...............
0x0090   0400 8002 0080 63b9 b919 c02b ae90 744c     ......c....+..tL
0x00a0   73eb 8bcf d855 ead0 6982 1bef 23c3 399b     s....U..i...#.9.
0x00b0   8eb2 493c 5a79                              ..I<Zy
21:06:36.316476 10.0.0.1.80 > 10.0.0.2.1062: . [tcp sum ok] ack 131 win 6432 <nop,nop,timestamp 9818004 116330> (DF) (ttl 64, id 51341, len 52)
21:06:36.316476 10.0.0.1.80 > 10.0.0.2.1062: P [tcp sum ok] 1:684(683) ack 131 win 6432 <nop,nop,timestamp 9818005 116330> (DF) (ttl 64, id
51342, len 735)
0x0000   4500 02df c88e 4000 4006 5b88 0a00 0001     E.....@.@.[.....
0x0010   0a00 0002 0050 0426 5032 071f 5525 60d3     .....P.&P2..U%`.
0x0020   8018 1920 27e2 0000 0101 080a 0095 cf95     ....'...........
0x0030   0001 c66a 1603 0100 4a02 0000 4603 013d     ...j....J...F..=
0x0040   94ab dc96 1264 6a9a bf84 18e8 f9e9 9205     .....dj.........
0x0050   9c33 c2eb 042a 42f3 bff6 bae5 02b9 e920     .3...*B.........
0x0060   ada7 e88e 0a65 e619 c0fb b421 4fb0 3631     .....e.....!O.61
0x0070   ac69 b1e8 a51a 0e49 f419 1ee4 4ff1 77f2     .i.....I....O.w.
0x0080   000a 0016 0301 024e 0b00 024a 0002 4700     .......N...J..G.
0x0090   0244 3082 0240 3082 01a9 a003 0201 0202     .D0..@0.........
0x00a0   0100 300d 0609 2a86 4886 f70d 0101 0405     ..0...*.H.......
0x00b0   0030 6631 0b30 0906 0355 0406 1302 4742     .0f1.0...U....GB
0x00c0   310f 300d 0603 5504 0813 064c 6f6e 646f     1.0...U....Londo
0x00d0   6e31 0f30 0d06 0355 0407 1306 4c6f 6e64     n1.0...U....Lond
0x00e0   6f6e 310c 300a 0603 5504 0a13 0341 4141     on1.0...U....AAA
0x00f0   310c 300a 0603 5504 0b13 0341 4141 3119     1.0...U....AAA1.
0x0100   3017 0603 5504 0313 1061 6161 6161 612e     0...U....aaaaaa.
0x0110   6262 6262 622e 6363 6330 1e17 0d30 3230     bbbbb.ccc0...020
0x0120   3931 3332 3235 3932 335a 170d 3032 3130     913225923Z..0210
0x0130   3133 3232 3539 3233 5a30 6631 0b30 0906     13225923Z0f1.0..
0x0140   0355 0406 1302 4742 310f 300d 0603 5504     .U....GB1.0...U.
0x0150   0813 064c 6f6e 646f 6e31 0f30 0d06 0355     ...London1.0...U
0x0160   0407 1306 4c6f 6e64 6f6e 310c 300a 0603     ....London1.0...
0x0170   5504 0a13 0341 4141 310c 300a 0603 5504     U....AAA1.0...U.
0x0180   0b13 0341 4141 3119 3017 0603 5504 0313     ...AAA1.0...U...
0x0190   1061 6161 6161 612e 6262 6262 622e 6363     .aaaaaa.bbbbb.cc
0x01a0   6330 819f 300d 0609 2a86 4886 f70d 0101     c0..0...*.H.....
0x01b0   0105 0003 818d 0030 8189 0281 8100 b063     .......0.......c
0x01c0   ad97 cf77 492e 4b9a 4ab9 7b98 5523 376a     ...wI.K.J.{.U#7j
0x01d0   2a2d a5c7 e40e 44b3 181d d289 597b 344a     *-....D.....Y{4J
0x01e0   3933 df30 56dd 2760 a493 91f0 e658 4846     93.0V.'`.....XHF
0x01f0   5f02 bab2 6c4a d0ce a211 5223 075e 6f2f     _...lJ....R#.^o/
0x0200   2782 b01b a5b9 c407 7017 0cd9 d610 9ae5     '.......p.......
0x0210   f331 ac8f 011b 9045 7b52 f8ff 4f19 6643     .1.....E{R..O.fC
0x0220   924e f7f1 fce0 065e 5042 e4bc a766 3872     .N.....^PB...f8r
0x0230   178f e414 7d5c 1f34 1fc1 c3c4 ebe3 0203     ....}\.4........
0x0240   0100 0130 0d06 092a 8648 86f7 0d01 0104     ...0...*.H......
0x0250   0500 0381 8100 6a09 56d2 65f3 1930 60de     ......j.V.e..0`.
```

SSL Server Hello + Certificate

```
0x0260   f78c e403 f95e 0dc4 d12d f3fb eec9 e693     .....^...-......
0x0270   e984 1a29 15be 099d 15f6 c88d ca52 2a2f     ...).........R*/
0x0280   8b25 9a1e 0dbf aa49 4925 943a effd 2dba     .%.....II%.:..-.
0x0290   454b 47fb 7fa0 8946 31d7 e14b ebf8 4b00     EKG....F1..K..K.
0x02a0   72d8 01cc 63ff da29 659f 335a 88ff bcbd     r...c..)e.3Z....
0x02b0   d970 3694 4c58 483e ce18 7e60 b261 fdd0     .p6.LXH>..~`.a..
0x02c0   4722 2792 cbe5 a17b 2001 42e3 4d64 e842     G"'....{..B.Md.B
0x02d0   322d 352f 9a42 1603 0100 040e 0000 00       2-5/.B.........
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: . [tcp sum ok] ack 684 win 6830 <nop,nop,timestamp 116330 9818005> (DF) (ttl 64, id 42557, len 52)
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: F [tcp sum ok] 131:131(0) ack 684 win 6830 <nop,nop,timestamp 116330 9818005> (DF) (ttl 64, id
42558, len 52)
21:06:36.316476 10.0.0.1.80 > 10.0.0.2.1062: F [tcp sum ok] 684:684(0) ack 132 win 6432 <nop,nop,timestamp 9818006 116330> (DF) (ttl 64, id
51343, len 52)
21:06:36.316476 10.0.0.2.1062 > 10.0.0.1.80: . [tcp sum ok] ack 685 win 6830 <nop,nop,timestamp 116330 9818006> (DF) (ttl 64, id 42559, len 52)
21:06:36.356476 10.0.0.2.1063 > 10.0.0.1.80: S [tcp sum ok] 1429687260:1429687260(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 36274, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1063: S [tcp sum ok] 1332232310:1332232310(0) ack 1429687261 win 5792 <mss 1460,sackOK,timestamp 9818037
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1063 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818037> (DF) (ttl 64, id 36275, len 52)
21:06:36.356476 10.0.0.2.1063 > 10.0.0.1.80: P [tcp sum ok] 1:29(28) ack 1 win 5840 <nop,nop,timestamp 116334 9818037> (DF) (ttl 64, id 36276,
len 80)
0x0000   4500 0050 8db4 4000 4006 98f1 0a00 0002     E..P..@.@.......
0x0010   0a00 0001 0427 0050 5537 4bdd 4f68 4077     .....'.PU7K.Oh@w
0x0020   8018 16d0 6955 0000 0101 080a 0001 c66e     ....iU.........n
0x0030   0095 cfb5 001e 3b6f 0100 0001 0000 0000     ......;o........              DNS (TCP)
0x0040   0000 0568 3478 3072 026e 6c00 00fc 0001     ...h4x0r.nl.....
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1063: . [tcp sum ok] ack 29 win 5792 <nop,nop,timestamp 9818037 116334> (DF) (ttl 64, id 36802, len 52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1063: R [tcp sum ok] 1:1(0) ack 29 win 5792 <nop,nop,timestamp 9818038 116334> (DF) (ttl 64, id 36803, len
52)
21:06:36.356476 10.0.0.2.1064 > 10.0.0.1.80: S [tcp sum ok] 1429508463:1429508463(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 22273, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1064: S [tcp sum ok] 1334138529:1334138529(0) ack 1429508464 win 5792 <mss 1460,sackOK,timestamp 9818040
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1064 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818040> (DF) (ttl 64, id 22274, len 52)
21:06:36.356476 10.0.0.2.1064 > 10.0.0.1.80: P [tcp sum ok] 1:51(50) ack 1 win 5840 <nop,nop,timestamp 116334 9818040> (DF) (ttl 64, id 22275,
len 102)
0x0000   4500 0066 5703 4000 4006 cf8c 0a00 0002     E..fW.@.@.......
0x0010   0a00 0001 0428 0050 5534 9170 4f85 56a2     .....(.PU4.pO.V.
0x0020   8018 16d0 6c6d 0000 0101 080a 0001 c66e     ....lm.........n
0x0030   0095 cfb8 7908 0000 0001 0000 0000 0000     ....y...........
0x0040   2043 4b41 4141 4141 4141 4141 4141 4141     .CKAAAAAAAAAAAAA              NETBIOS
0x0050   4141 4141 4141 4141 4141 4141 4141 4141     AAAAAAAAAAAAAAAA
0x0060   4100 0021 0001                             A..!..
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1064: . [tcp sum ok] ack 51 win 5792 <nop,nop,timestamp 9818040 116334> (DF) (ttl 64, id 39194, len 52)
```

```
21:06:36.356476 10.0.0.2.1065 > 10.0.0.1.80: S [tcp sum ok] 1432105165:1432105165(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 63139, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1065: S [tcp sum ok] 1343970302:1343970302(0) ack 1432105166 win 5792 <mss 1460,sackOK,timestamp 9818041
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1065 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818041> (DF) (ttl 64, id 63140, len 52)
21:06:36.356476 10.0.0.2.1065 > 10.0.0.1.80: P [tcp sum ok] 1:18(17) ack 1 win 5840 <nop,nop,timestamp 116334 9818041> (DF) (ttl 64, id 63141,
len 69)
0x0000   4500 0045 f6a5 4000 4006 300b 0a00 0002     E..E..@.@.0.....
0x0010   0a00 0001 0429 0050 555c 30ce 501b 5bff     .....).PU\0.P.[.
0x0020   8018 16d0 a498 0000 0101 080a 0001 c66e     ...............n
0x0030   0095 cfb9 4845 4144 202f 2048 5454 502f     ....HEAD./.HTTP/              HTTP
0x0040   312e 300a 0a                                1.0..
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1065: . [tcp sum ok] ack 18 win 5792 <nop,nop,timestamp 9818041 116334> (DF) (ttl 64, id 37958, len 52)
21:06:36.356476 10.0.0.2.1066 > 10.0.0.1.80: S [tcp sum ok] 1428364229:1428364229(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 56747, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1066: S [tcp sum ok] 1341305709:1341305709(0) ack 1428364230 win 5792 <mss 1460,sackOK,timestamp 9818041
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1066 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818041> (DF) (ttl 64, id 56748, len 52)
21:06:36.356476 10.0.0.2.1066 > 10.0.0.1.80: P [tcp sum ok] 1:15(14) ack 1 win 5840 <nop,nop,timestamp 116334 9818041> (DF) (ttl 64, id 56749,
len 66)
0x0000   4500 0042 ddad 4000 4006 4906 0a00 0002     E..B..@.@.I.....
0x0010   0a00 0001 042a 0050 5523 1bc6 4ff2 b36e     .....*.PU#..O..n
0x0020   8018 16d0 7ce0 0000 0101 080a 0001 c66e     ....|..........n
0x0030   0095 cfb9 300c 0201 0160 0702 0102 0400     ....0....`......              LDAP
0x0040   8000                                        ..
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1066: . [tcp sum ok] ack 15 win 5792 <nop,nop,timestamp 9818041 116334> (DF) (ttl 64, id 31959, len 52)
21:06:36.356476 10.0.0.2.1067 > 10.0.0.1.80: S [tcp sum ok] 1444452867:1444452867(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 23661, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1067: S [tcp sum ok] 1338699452:1338699452(0) ack 1444452868 win 5792 <mss 1460,sackOK,timestamp 9818042
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1067 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818042> (DF) (ttl 64, id 23662, len 52)
21:06:36.356476 10.0.0.2.1067 > 10.0.0.1.80: P [tcp sum ok] 1:267(266) ack 1 win 5840 <nop,nop,timestamp 116334 9818042> (DF) (ttl 64, id 23663,
len 318)
0x0000   4500 013e 5c6f 4000 4006 c948 0a00 0002     E..>\o@.@..H....
0x0010   0a00 0001 042b 0050 5618 9a04 4fca eebd     .....+.PV...O...
0x0020   8018 16d0 9a96 0000 0101 080a 0001 c66e     ...............n
0x0030   0095 cfba 0000 0106 ffff ffff 0a00 0000     ................
0x0040   0000 00ff ffff ffff ffff ffff ffff ffff     ................
0x0050   ffff ffff ffff 3e00 0000 00ff ffff ffff     ......>.........
0x0060   ff20 2020 2020 2020 2020 2020 2020 2020     ................
0x0070   2020 2020 2020 2020 2020 2020 2020 2020     ................              SAP-R3
0x0080   2020 2020 2020 2020 2070 656e 7465 7374     .........pentest
0x0090   0020 2020 2020 2020 2020 2020 2000 0000     ................
0x00a0   0000 2d20 2020 2020 2020 2020 2020 2020     ..-.............
0x00b0   2020 2020 2020 0000 0000 0000 0000 ffff     ................
```

```
0x00c0   ffff 0000 0000 0100 0000 0000 0000 0000     ................
0x00d0   0000 0000 0000 0000 0000 0000 0000 0000     ................
0x00e0   0000 0000 0000 0000 0000 0000 0000 0000     ................
0x00f0   0000 0000 0000 0000 0000 0000 0000 0000     ................
0x0100   0010 0000 0000 0000 1004 0200 0c00 0187     ................
0x0110   6800 0004 4c00 0003 e810 040b 0020 ff7f     h...L........... 
0x0120   ca0d c8b3 6600 0400 0000 0000 0000 0000     ....f...........
0x0130   0000 0000 0000 0000 0000 0000 0000          ..............
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1065: R [tcp sum ok] 1:1(0) ack 18 win 5792 <nop,nop,timestamp 9818042 116334> (DF) (ttl 64, id 37959, len
52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1067: . [tcp sum ok] ack 267 win 6432 <nop,nop,timestamp 9818042 116334> (DF) (ttl 64, id 8455, len 52)
21:06:36.356476 10.0.0.2.1068 > 10.0.0.1.80: S [tcp sum ok] 1428253576:1428253576(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 44518, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1068: S [tcp sum ok] 1331826576:1331826576(0) ack 1428253577 win 5792 <mss 1460,sackOK,timestamp 9818042
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1068 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818042> (DF) (ttl 64, id 44519, len 52)
21:06:36.356476 10.0.0.2.1068 > 10.0.0.1.80: P [tcp sum ok] 1:101(100) ack 1 win 5840 <nop,nop,timestamp 116334 9818042> (DF) (ttl 64, id 44520,
len 152)
0x0000   4500 0098 ade8 4000 4006 7875 0a00 0002     E.....@.@.xu....
0x0010   0a00 0001 042c 0050 5521 6b89 4f62 0f91     .....,.PU!k.Ob..
0x0020   8018 16d0 d1db 0000 0101 080a 0001 c66e     ...............n
0x0030   0095 cfba 8000 0060 77b7 3b30 0000 0000     .......`w.;0....
0x0040   0000 0002 0001 86a5 0000 0001 0000 0005     ................              MOUNTD
0x0050   0000 0001 0000 0038 3c71 4d94 0000 0007     .......8<qM.....
0x0060   6b70 6d67 2d70 7400 0000 0000 0000 0000     kpmg-pt.........
0x0070   0000 0007 0000 0000 0000 0001 0000 000e     ................
0x0080   0000 000f 0000 0010 0000 0011 0000 fffe     ................
0x0090   0000 0000 0000 0000                         ........
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1064: R [tcp sum ok] 1:1(0) ack 51 win 5792 <nop,nop,timestamp 9818042 116334> (DF) (ttl 64, id 39195, len
52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1068: . [tcp sum ok] ack 101 win 5792 <nop,nop,timestamp 9818042 116334> (DF) (ttl 64, id 49195, len 52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1066: R [tcp sum ok] 1:1(0) ack 15 win 5792 <nop,nop,timestamp 9818042 116334> (DF) (ttl 64, id 31960, len
52)
21:06:36.356476 10.0.0.2.1069 > 10.0.0.1.80: S [tcp sum ok] 1430921132:1430921132(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 55746, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1069: S [tcp sum ok] 1336768757:1336768757(0) ack 1430921133 win 5792 <mss 1460,sackOK,timestamp 9818043
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1069 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818043> (DF) (ttl 64, id 55747, len 52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1067: R [tcp sum ok] 1:1(0) ack 267 win 6432 <nop,nop,timestamp 9818043 116334> (DF) (ttl 64, id 8456, len
52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1068: R [tcp sum ok] 1:1(0) ack 101 win 5792 <nop,nop,timestamp 9818043 116334> (DF) (ttl 64, id 49196,
len 52)
21:06:36.356476 10.0.0.2.1069 > 10.0.0.1.80: P [tcp sum ok] 1:49(48) ack 1 win 5840 <nop,nop,timestamp 116334 9818043> (DF) (ttl 64, id 55748,
len 100)
0x0000   4500 0064 d9c4 4000 4006 4ccd 0a00 0002     E..d..@.@.L.....
```

```
0x0010   0a00 0001 042d 0050 554a 1fad 4fad 78f6     .....-.PUJ..O.x.
0x0020   8018 16d0 478e 0000 0101 080a 0001 c66e     ....G.........n
0x0030   0095 cfbb 6c00 0b00 0000 1200 1000 0000     ....l...........
0x0040   4d49 542d 4d41 4749 432d 434f 4f4b 4945     MIT-MAGIC-COOKIE              X-windows
0x0050   2d31 0000 c617 34b7 89ed 65c0 93fd d856     -1....4...e....V
0x0060   66fa 5240                                   f.R@
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1069: . [tcp sum ok] ack 49 win 5792 <nop,nop,timestamp 9818043 116334> (DF) (ttl 64, id 46925, len 52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1069: R [tcp sum ok] 1:1(0) ack 49 win 5792 <nop,nop,timestamp 9818043 116334> (DF) (ttl 64, id 46926, len
52)
21:06:36.356476 10.0.0.2.1070 > 10.0.0.1.80: S [tcp sum ok] 1431629210:1431629210(0) win 5840 <mss 1460,sackOK,timestamp 116334 0,nop,wscale 0>
(DF) (ttl 64, id 11252, len 60)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1070: S [tcp sum ok] 1336236285:1336236285(0) ack 1431629211 win 5792 <mss 1460,sackOK,timestamp 9818044
116334,nop,wscale 0> (DF) (ttl 64, id 0, len 60)
21:06:36.356476 10.0.0.2.1070 > 10.0.0.1.80: . [tcp sum ok] ack 1 win 5840 <nop,nop,timestamp 116334 9818044> (DF) (ttl 64, id 11253, len 52)
21:06:36.356476 10.0.0.2.1070 > 10.0.0.1.80: P [tcp sum ok] 1:15(14) ack 1 win 5840 <nop,nop,timestamp 116334 9818044> (DF) (ttl 64, id 11254,
len 66)
0x0000   4500 0042 2bf6 4000 4006 fabd 0a00 0002     E..B+.@.@.......
0x0010   0a00 0001 042e 0050 5554 ed9b 4fa5 58fe     .......PUT..O.X.
0x0020   8018 16d0 0590 0000 0101 080a 0001 c66e     ...............n
0x0030   0095 cfbc 300c 0201 0160 0702 0102 0400     ....0....`......              LDAP
0x0040   8000                                        ..
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1070: . [tcp sum ok] ack 15 win 5792 <nop,nop,timestamp 9818044 116334> (DF) (ttl 64, id 38722, len 52)
21:06:36.356476 10.0.0.1.80 > 10.0.0.2.1070: R [tcp sum ok] 1:1(0) ack 15 win 5792 <nop,nop,timestamp 9818044 116334> (DF) (ttl 64, id 38723, len
52)
21:06:41.256476 10.0.0.2.1060 > 10.0.0.1.80: F [tcp sum ok] 9:9(0) ack 1 win 5840 <nop,nop,timestamp 116824 9817937> (DF) (ttl 64, id 24019, len
52)
21:06:41.256476 10.0.0.1.80 > 10.0.0.2.1060: F [tcp sum ok] 1:1(0) ack 10 win 5792 <nop,nop,timestamp 9822937 116824> (DF) (ttl 64, id 40370, len
52)
21:06:41.256476 10.0.0.2.1060 > 10.0.0.1.80: . [tcp sum ok] ack 2 win 5840 <nop,nop,timestamp 116824 9822937> (DF) (ttl 64, id 24020, len 52)
```

Amap successfully recognized SSL listening on port 80. Here is the content of the results file:

```
Amap v0.95 started at Fri Sep 27 21:06:36 2002
Ports: 0, triggers 0. Total amount of tasks to perform: 11
Protocol on IP 10.0.0.1 port 80 tcp matches SSL - banner: JF=dj3*B
\ne!O61iIOw\nNJGD0@00\r\t*H\r0f10\tUGB10\rULondon10\rULondon10\nU\nAAA10\nUAAA10Uaaaaa
a.bbbbb.ccc0\r020913225923Z\r021013225923Z0f10\tUGB10\rULondon10\rULondon10\nU\nAAA10\
nUAAA10Uaaaaaa.bbbbb.ccc00\r\t*H\r0cwI.KJU#7j*-DY4J930V'`XH
Unidentified ports: None.
Amap v0.95 ended at Fri Sep 27 21:06:41 2002
```

The “banner” is, in fact, the message sent by the server in response to the handshake initiated by amap.
The apache-ssl daemon recorded the following error messages in its error log file, showing that the server received also messages that were not recognized as valid “client_hello” message. There was one such entry per each of the triggers sent by amap.

```
[snip]
[Fri Sep 27 21:05:00 2002] [error] SSL_accept failed
[Fri Sep 27 21:05:00 2002] [error] error:140760FC:SSL
routines:SSL23_GET_CLIENT_HELLO:unknown protocol
[Fri Sep 27 21:05:00 2002] [debug] apache_ssl.c(287): SSL_accept returned 0
[Fri Sep 27 21:05:00 2002] [error] SSL_accept failed
[Fri Sep 27 21:05:00 2002] [error] error:140760FC:SSL
routines:SSL23_GET_CLIENT_HELLO:unknown protocol
[Fri Sep 27 21:05:00 2002] [error] SSL_accept failed
[Fri Sep 27 21:05:00 2002] [error] error:1407609C:SSL
routines:SSL23_GET_CLIENT_HELLO:http request
[Fri Sep 27 21:05:00 2002] [error] SSL_accept failed
[Fri Sep 27 21:05:00 2002] [error] error:140760FC:SSL
routines:SSL23_GET_CLIENT_HELLO:unknown protocol
[snip]
```

Detecting amap probes

Amap is not very stealthy is run in its default mode. 11 parallel connections each one, with the exception of one possibly, sending an unexpected message at the application protocol level are surely recorded in the application log file, provided that the application maintains a good logging level. In the test that I made, the probes on the ssh port did not leave any trace at the application level, since no logging was enabled for this application. On the other hand extensive logging was available for the ftp, http and http-ssl applications. Therefore the most effective means to detect this probe is to maintain and check logs at the application level. After all if your mail server receives a NETBIOS request, something strange must be happening.
Apart from logging at the application level, it is difficult to detect an amap probe since it uses the OS system calls to the TCP/IP stack and therefore no signature can be found at the level of the TCP, UDP or IP packet. Nevertheless, it is still possible to write a snort rule that is able to detect probes from amap when it is run in its default mode. In fact, we can observe that in all attempts, amap always sends the trigger for the mount service, specifying a machine name that is hard-wired in the binary string it sends for this type of trigger. The machine name is “kpmg-pt” and it can be found in any default probe from this tool.
It is possible to write a rule that looks for this string in the payload for each service that we possibly want to monitor against this type of probes.
For instance I wrote the following rules for my test environment:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 389 (msg:"AMAP probe attempt on the
LDAP server"; flags:A+; content:"kpmg-pt"; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"AMAP probe attempt on the
HTTPS Server"; flags:A+; content:"kpmg-pt"; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 31 (msg:"AMAP probe attempt on the
FTP server"; flags:A+; content:"kpmg-pt"; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"AMAP probe attempt on the
HTTP server"; flags:A+; content:"kpmg-pt"; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 23 (msg:"AMAP probe attempt on the
ssh server"; flags:A+; content:"kpmg-pt"; classtype:attempted-recon;)
```

Obviously, $HOME_NET could be substituted with the IP address of the host on which the specific service is running.

The following alerts were produced by snort, running in NIDS mode with the –dv and –c <snort-conf> options:

```
[**] [1:0:0] AMAP probe attempt on the LDAP server [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:05:01.746476 10.0.0.2:1057 -> 10.0.0.1:389
TCP TTL:64 TOS:0x0 ID:49249 IpLen:20 DgmLen:152 DF
***AP*** Seq: 0x4F7E6127  Ack: 0x4A5394D0  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 106873 9723423

[**] [1:0:0] AMAP probe attempt on the HTTPS Server [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:06:36.356476 10.0.0.2:1068 -> 10.0.0.1:80
TCP TTL:64 TOS:0x0 ID:44520 IpLen:20 DgmLen:152 DF
***AP*** Seq: 0x55216B89  Ack: 0x4F620F91  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 116334 9818042

[**] [1:0:0] AMAP probe attempt on the FTP server [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:07:02.366476 10.0.0.2:1079 -> 10.0.0.1:31
TCP TTL:64 TOS:0x0 ID:20302 IpLen:20 DgmLen:152 DF
***AP*** Seq: 0x56F09FDC  Ack: 0x5118D111  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 118935 9844055

[**] [1:0:0] AMAP probe attempt on the HTTP server [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:07:26.436476 10.0.0.2:1090 -> 10.0.0.1:21
TCP TTL:64 TOS:0x0 ID:39814 IpLen:20 DgmLen:152 DF
***AP*** Seq: 0x58F7427B  Ack: 0x5293A102  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 121342 9868125

[**] [1:0:0] AMAP probe attempt on the ssh server [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:07:51.176476 10.0.0.2:1101 -> 10.0.0.1:23
TCP TTL:64 TOS:0x0 ID:36800 IpLen:20 DgmLen:152 DF
***AP*** Seq: 0x59D46C17  Ack: 0x548C9F48  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 123816 9892868
```

Obviously these rules fail, if one runs amap with the –p <protocol> options specifying the triggers that should be used and not using the "mount" trigger. But then again, being a tool that targets the application level, detection is done most appropriately at the application level by careful monitoring of "strange" messages sent to the server.

Conclusions

Tools like amap are an additional proof that "security through obscurity" is not the right approach to secure a network: simply running a service on a different port is not sufficient to go unnoticed. However, amap can be very useful for system administrators in finding "hidden" services, in those cases where users run unauthorised services and try to disguise them using a non-standard port. In this function it can be usefully used in collaboration with tools like nmap. The list of signatures (triggers and responses) is customisable and can be easily expanded with the addition of signatures of proprietary protocols. Like its authors say: "With amap, you will be able to identify that SSL server running on port 3445 and some oracle listener on port 23!".

References

[1] Fyodor, *Remote OS detection via TCP/IP stack fingerprinting* URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html (October, 4th)

[2] *List of fingerprints for passive fingerprint monitoring* URL: http://project.honeynet.org/papers/finger/traces.txt (October, 4th)

[3] *Hping man page*, URL: http://www.hping.org/manpage.html (October, 4th)

[4] Arkin, Orfin, *Network scanning techniques: Understanding how it is done* URL: http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf (October, 4th)

[5] *Identifying ICMP Hackery Tools Used In The Wild Today* URL: http://www.sys-security.com/archive/securityfocus/icmptools.html (October, 4th)

[6] *Preventing remote OS Detection* URL: http://www.pgci.ca/common/p_fingerprint.htm (October, 4th)

[7] Beck, Rob, *Passive-Aggressive Resistance: OS Fingerprint Evasion,* Linux Journal URL: http://www.linuxjournal.com/article.php?sid=4750 (October, 4th)

[8] Malan, G. Robert, Jahanian Farnam, Smart, Matthew, *Defeating TCP/IP Stack Fingerprint*, Proc. 9th Usenix Symposium, Denver, CO, August 14-17, 2000

[9] Arkin, Orfin, Yarochkin, Fyodor *Xprobe v2.0: A fuzzy approach to remote active operating system fingerprinting,* August 2002 URL: http://www.sys-security.com/archive/papers/Xprobe2.pdf (October, 4th)

[10] *HACKBOT 2.11* URL: http://ws.obit.nl/hackbot/documentation.txt (October, 4th)

[11] Provos, Niels, Honeyman, Peter, *ScanSSH - Scanning the Internet for SSH servers*, URL: www.citi.umich.edu/techreports/reports/citi-tr-01-13.pdf (October, 4th)

[12] Nazario, Jose *Passive Fingerprinting using Network Client Applications*, Nov. 2000 URL: http://www.crimelabs.net/docs/passive.html (October, 4th)

[13] Bursztein, Lupin *iQ Overview*, URL: www.bursztein.net/secu/iQ/iQ-fr.pdf (October, 4th)

[14] *OS Identification*, Unix Insider 12/8/00 URL: http://www.itworld.com/Comp/2124/swol-1208-buildingblocks/ (October, 4th)

[15] Glaser, Thomas *TCP/IP Stack Fingerprinting Principles*, , SANS Intrusion Detection FAQ, October 2000
URL: http://www.sans.org/newlook/resources/IDFAQ/TCP_fingerprinting.htm (October, 4th)
[16] Veysset, Franck, Courtay, Olivier, and Heen, Olivier, *New tool and technique for remote operating system fingerprinting*
URL: http://www.intranode.com/pdf/techno/ring-full-paper.pdf (October, 4th)
[17] Vision, Max, *Passive Host Fingerprinting*,
URL: http://www.whitehats.com/library/passive (October, 4th)
[18] *Advanced Remote OS Detection Methods/Concepts using Perl*
URL: http://cert.uni-stuttgart.de/archive/bugtraq/2001/02/msg00195.html (October, 4th)
[19] *Examining Remote OS Detection using LPD querying*, Feb 2001
URL: http://old.lwn.net/2001/0222/a/sec-lpddetect.php3 (October, 4th)
[20] Fernando Martins, *IDS: passive mapping: an offensive use of IDS*,
URL: http://www.shmoo.com/mail/ids/apr00/msg00014.shtml (October, 4th)
[21] *Know Your Enemy:Passive Fingerprinting, Honeynet project*
URL: http://project.honeynet.org/papers/finger/ (October, 4th)
[22] Ethereal, URL: http://www.ethereal.com (October, 4th)
[23] Arkin, Orfin, *ICMP usage in scanning*
URL:http://www.sys-security.com/html/projects/icmp.html (October, 4th)
[24] P0f README, URL: http://www.stearns.org/p0f/README (October, 4th)
[25] Amap README URL:
http://www.thehackerschoice.com/download.php?t=r&d=amap-0.95.tar.gz (October, 4th)

Part 2 - Network Detects

Detect #1: Looking for ssh servers

Trace Log

```
[snip - first address is x.y.z.67]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/05-17:23:40.059051 217.195.194.105:20 -> x.y.z.128:22
 TCP TTL:236 TOS:0x0 ID:12859 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x905D891C Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/05-17:25:03.903557 217.195.194.105:20 -> x.y.z.129:22
 TCP TTL:236 TOS:0x0 ID:31083 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x489D34B0 Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/05-17:26:39.277130 217.195.194.105:20 -> x.y.z.130:22
 TCP TTL:236 TOS:0x0 ID:60917 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x76017BFE Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/05-17:27:09.031427 217.195.194.105:20 -> x.y.z.131:22
 TCP TTL:236 TOS:0x0 ID:25151 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0xBB41833B Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

[snip - destination IP increment by one]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/05-23:59:41.655592 217.195.194.105:20 -> x.y.w.250:22
 TCP TTL:236 TOS:0x0 ID:49447 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x9B009A3D Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/06-00:00:25.830168 217.195.194.105:20 -> x.y.w.251:22
 TCP TTL:236 TOS:0x0 ID:28091 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x617761F6 Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/06-00:01:23.642390 217.195.194.105:20 -> x.y.w.252:22
 TCP TTL:236 TOS:0x0 ID:20365 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0xAC8CB38E Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/06-00:02:59.168129 217.195.194.105:20 -> x.y.w.253:22
 TCP TTL:236 TOS:0x0 ID:50349 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x9E845F7E Ack: 0x0 Win: 0x3FFF TcpLen: 20
```

```
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/06-00:03:28.787919 217.195.194.105:20 -> x.y.w.254:22
 TCP TTL:236 TOS:0x0 ID:14393 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0xA958E0FF Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

[scan starts again the following day to complete the first 255-addresses
netblock]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/07-17:12:03.138461 217.195.194.105:20 -> x.y.z.1:22
 TCP TTL:236 TOS:0x0 ID:9173 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0xAF643E15 Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/07-17:13:29.351675 217.195.194.105:20 -> x.y.z.2:22
 TCP TTL:236 TOS:0x0 ID:29847 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x5BB0B713 Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/07-17:15:14.552546 217.195.194.105:20 -> x.y.z.3:22
 TCP TTL:236 TOS:0x0 ID:3825 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0xB24A8B8A Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

[snip]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/07-18:14:17.310046 217.195.194.105:20 -> x.y.z.65:22
 TCP TTL:236 TOS:0x0 ID:7631 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x637DF372 Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]

 [**] [1:503:2] MISC Source Port 20 to <1024 [**]
 [Classification: Potentially Bad Traffic] [Priority: 2]
 07/07-18:15:54.563805 217.195.194.105:20 -> x.y.z.66:22
 TCP TTL:236 TOS:0x0 ID:39345 IpLen:20 DgmLen:40 DF
 ******S* Seq: 0x553FF4FF Ack: 0x0 Win: 0x3FFF TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS06]
```

1. Source of Trace

NIDS sensor located on a network that I manage.

2. Detect was generated by

This detect was generated by snort 1.8.7 build 128 in full alert mode with the standard ruleset.

Each alert contains the following information:
1st line:
[**] ... [**] = snort-signature,
2nd line:
[Classification: ....] = classification of the alert
[Priority: ...] = priority assigned to the alert (indicates the severity of the alert),

3rd line:

MM/DD-hh:mm:ss.cccccc = timestamp,
A.B.C.D:X = source address and source port
-> = traffic direction
E.F.G.H:Y = destination address and destination port
Protocol (TCP, UDP, etc.)

4th line:

Time to live (TTL) - as specified in the IP header
Type of Service (TOS) - as specified in the IP header
IP Identification number - as specified in the IP header
Length of the IP header
Length of the IP datagram
Don't fragment bit (DF)

5th line:

List of TCP flags in the TCP header ("*" indicates that the flag corresponding to the option in that position is not set)
TCP Sequence number
TCP Acknowledge number
TCP Window size
Length of the TCP header

6th line:

[...] references

The destination addresses have been sanitized, and represented as x.y.z.N, and x.y.w.N (N is a number that ranges in the trace from 1 to 255) to indicate two different 255-addresses net blocks.

The rule that triggered the alert is:

```
alert tcp $EXTERNAL_NET 20 -> $HOME_NET 0:1023 (msg: "MISC Source
Port 20 to<1024"; flags: S; classtype: bad-unknown; reference:
arachnids,6;)
```

Basically these alerts were generated because TCP connections were attempted using a source port outside the range of the ephemeral source ports (usually above 1024). The scan was not detected by the portscan preprocessor as this is activated with the default threshold which detects a portscan when there are UDP or TCP SYN packets from the same source to four different destinations in less than three seconds. In this scan we have a packet about every minute which is far above the threshold of the portscan preprocessor.

3. Probability the source address was spoofed

The probability that the source address is spoofed is low. The "attacking" host is trying to determine if the ssh service is running on all hosts of two 255-addresses net blocks, probably in an information gathering attempt for a possible future exploit. For this attempt to succeed, the remote system needs to receive replies to its probes, for this reason, unless it is positioned between the possibly spoofed address and the target and is able to capture traffic in transit, the probability that the originating address is spoofed is very low.

A whois query at RIPE (Réseaux IP Européens) on the remote host gave the following result:

217.195.192.0 -	allocated to Teklan Internet Erisim Hizmetleri
217.195.198.31	Komunikasyon Elektronik San Ve Tic. A.S., Turkey

An nslookup query on the remote IP address (217.195.194.105) did not produce a hostname in return, while a query on the DShield database indicated that probes coming from this host had been reported in July 2002, ports targeted are not indicated.

4. Description of attack

The remote host is scanning two 255-addresses net blocks in an attempt to find hosts running the ssh service. The packets are an attempt to establish a TCP connection (SYN flag set). The first packet is received at 16:19:22 05 July, while the last packet of the first scan is received at 00:03:28, 06 July. The scan is resumed the next day (7 July) after 17 hours. This is a slow scan not detected by the portscan preprocessor with a packet received at the rate of about one packet every minute.
The packet show evident signs of crafting: the IP ID is changing in a random fashion, going from as low as 695 to as high as 61000. According to [1]: "The identification field uniquely identifies each datagram sent by a host. It normally increments by one each time a datagram is sent". Assuming that this host is generating other packets between one scan packet and the next, we should see the IP increment and wrap around when the maximum value allowed by the IP ID field (65535) is reached. Instead we see values incrementing and decrementing randomly. Another sign of packet crafting is the initial window size whose value is 0x3fff (decimal 16383) which does not seem to be used by any common operating system. According to [2] the closest value for the initial window size is used by AIX (16000-16100), however for AIX the initial TTL would be 60, while in these packets we have a TTL of 236 which makes us suppose that the initial TTL was 255 and that the remote host is 19 hops away from our network. 255 is a TTL value used by Solaris or Cisco 12.0. In conclusion, either the packets are crafted or the TCP/IP stack of the remote host has been modified against remote passive fingerprinting. Another sign of packet crafting is the source port, source ports have usually values greater than 1024. The use of port 20 as a source port is probably meant at bypassing some poorly configured firewall. Port 20 is used by ftp data connections. Ftp servers open a connection from source port 20 to a destination port specified by the ftp client to transfer data (active ftp). The slow speed of the scan can also be an attempt at making the scan go unnoticed.

5. Attack mechanism

The trace shown in this detect is an attempt at finding hosts running the ssh daemon. SSH is a widely used client-server application for authentication and encryption of network communications. It is normally used to substitute telnet for remote access, because it works on an encrypted channel. There are two main versions of the SSH protocol, v1 and v2. In 1998 Ariel Futoransky and Emiliano Kargieman [4] discovered a design flaw in the SSH1 protocol (protocol 1.5) that could lead an attacker to inject malicious packets into an SSH encrypted stream. The malicious traffic would allow execution of arbitrary commands on either client or server. They showed that this problem could not be fixed without breaking the semantic of the ssh protocol v1.5. A patch was devised that would detect an attack that exploited the vulnerability found. Unfortunately some time later, a vulnerability was found in the attack detection code that could lead to the execution of arbitrary code in SSH servers

and clients that incorporated this patch with the privileges of the SSH daemon, usually root. Non only SSH1 implementations are vulnerable, but also SSH2 implementations that implement also the SSH1 protocol for compatibility and can switch to SSH1 whenever requested by the client. However, for this vulnerability to be exploited it is necessary that server and attacking host successfully complete the key exchange negotiation and therefore that no access control restrictions are implemented on hosts allowed to connect to the ssh server (i.e. no "AllowHost" or "DenyHosts" set). In other words, sites that implement access control restrictions based on the incoming IP are not vulnerable.
Some tools have been made available, like scanSSH [9], that automatically scan for SSH servers and, based on the banner received, check whether the version running is vulnerable or not. As usual with vulnerability scanners, this information can be precious for system administrators to patch the system as well as for malicious users. However this tool does not allow for modification of the source port, which cannot be forced to 20 using this tool.
Based on the information obtained as a result of this scan, the scanner can launch a remote attack to exploit this vulnerability.
Examination of the tcpdump data shows that none of these reconnaissance attempt was successful, i.e. no TCP handshake was completed and therefore no useful data was gathered by the attacker.

Excerpt of tcpdump data:

```
18:40:02.294938 217.195.194.105.20 > x.y.z.191.22: S [tcp sum ok]
2767936284:2767936284(0) win 16383 (DF) (ttl 236, id 7389, len 40)
18:40:43.395290 217.195.194.105.20 > x.y.z.192.22: S [tcp sum ok]
1161287002:1161287002(0) win 16383 (DF) (ttl 236, id 48489, len 40)
18:41:42.187020 217.195.194.105.20 > x.y.z.193.22: S [tcp sum ok]
2451269476:2451269476(0) win 16383 (DF) (ttl 236, id 41743, len 40)
18:43:09.527843 217.195.194.105.20 > x.y.z.194.22: S [tcp sum ok]
1238813056:1238813056(0) win 16383 (DF) (ttl 236, id 63547, len 40)
18:45:05.384122 217.195.194.105.20 > x.y.z.195.22: S [tcp sum ok]
2182132973:2182132973(0) win 16383 (DF) (ttl 236, id 48325, len 40)

[snip - note how the ID changes greater than 30000 to 295 and back to nearly
30000]
20:00:10.174120 217.195.194.105.20 > x.y.w.14.22: S [tcp sum ok]
2365684774:2365684774(0) win 16383 (DF) (ttl 236, id 30951, len 40)
20:00:45.064262 217.195.194.105.20 > x.y.w.15.22: S [tcp sum ok]
3214464349:3214464349(0) win 16383 (DF) (ttl 236, id 295, len 40)
20:03:25.166863 217.195.194.105.20 > x.y.w.18.22: S [tcp sum ok]
1945027150:1945027150(0) win 16383 (DF) (ttl 236, id 29123, len 40)
20:04:54.174154 217.195.194.105.20 > x.y.w.20.22: S [tcp sum ok]
1697431168:1697431168(0) win 16383 (DF) (ttl 241, id 52857, len 40)
```

6. Correlations

The alert raised by snort is related to the usage of port 20 as source port associated to a destination port number less than 1024, and is described in ARACHNIDS[5], "IDS6/MISC_SOURCEPORTTRAFFIC-20-TCP": "This event indicates that an attacker is making a connection to a privileged port using the source port 20 (ftp-data). This should not normally occur. Old or misconfigured packetfilters may allow the connection if they allow all ftp response traffic."

[5] ARACHNIDS is available at http://www.whitehats.com

Attacks aimed at exploiting the SSH CRC32 vulnerability are described in [3], [5] and [6]. According to [3] in October 2001 intruders originating from network blocks in the Netherlands exploited this vulnerability to compromise a Red Hat linux box running OpenSSH 2.1.1 on the Washington University network. Having gained control of the system, they replaced a series of system commands with trojaned versions and introduced backdoors. The ssh server was also replaced and run on a different port (tcp 39999). The system was then used to scan other systems (a total of 47067 hosts were scanned) for the same vulnerability, 1244 of which were successfully exploited. Log files left behind by these tools indicate that they operate by looking for the banner displayed upon connection to the sshd service.

According to [10], vulnerable systems are:

Systems Affected	Vendor Status	Date Updated
Cisco	Not Vulnerable	13-Dec-2001
CORE SDI	Vulnerable	13-Dec-2001
Debian	Vulnerable	13-Dec-2001
FreeBSD	Vulnerable	13-Dec-2001
OpenSSH	Vulnerable	10-Dec-2001
SmoothWall	Vulnerable	14-Dec-2001
SSH Communications Security	Vulnerable	6-Nov-2001
SuSE	Vulnerable	13-Dec-2001

A lot of scans on ssh have also been reported to various security related mailing lists. Among them the one reported via snortsnarf at http://hvdkooij.xs4all.nl/snort/217/195/194/src217.195.194.105.html[6] is worth noting in this report as the scan is coming from the same remote host, with the same pattern, indicating that the remote host has actively scanned several networks for this vulnerability.

7. Evidence of active targeting

This is a scan directed at complete 255-addresses netblocks. No address is targeted in particular. It is likely that this network was included as part of a wider scan. Similar scans from the same address have been noticed on other networks.

8. Severity

The following formula is used to calculate the severity of the attack:

Severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Each aspect is ranked with a value from 1 to 5, 1 being the lowest and 5 being the highest.

Criticality: there are critical services among those probed, 4
Lethality: this is a reconnaissance attempt, 3
System Countermeasures: ssh is not enabled, 5

[6] This link seems to be not valid anymore at the date of the final version of this report (October 4th). I found this information early September.

Network Countermeasures: the firewall dropped all connections attempts, 5

Severity = (4+3) - (5+5) = -3

9. Defensive recommendation

Continue monitoring this type of traffic. If ssh is enabled, ensure, in addition to applying all necessary system patches, that IP based access control is enforced.

10. Multiple choice test question

```
18:40:02.294938 217.195.194.105.20 > x.y.z.191.22: S [tcp sum ok]
2767936284:2767936284(0) win 16383 (DF) (ttl 236, id 7389, len 40)
18:40:43.395290 217.195.194.105.20 > x.y.z.192.22: S [tcp sum ok]
1161287002:1161287002(0) win 16383 (DF) (ttl 236, id 48489, len 40)
18:41:42.187020 217.195.194.105.20 > x.y.z.193.22: S [tcp sum ok]
2451269476:2451269476(0) win 16383 (DF) (ttl 236, id 41743, len 40)
18:43:09.527843 217.195.194.105.20 > x.y.z.194.22: S [tcp sum ok]
1238813056:1238813056(0) win 16383 (DF) (ttl 236, id 63547, len 40)
```

The above trace indicates:
a) a normal ftp data transfer
b) a fast stealth scan for ssh service
c) an attempt to evade packet filtering using ftp data source port
d) a slow port scan

Correct answer is c)

References

[1] Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Inc, 1994

[2] List of fingerprints for passive fingerprint monitoring, URL: http://project.honeynet.org/papers/finger/traces.txt (Oct. 4th)

[3] Dittrich, David A. "Analysis of SSH crc32 compensation attack detector exploit", 15 Nov 2001
URL: http://staff.washington.edu/dittrich/misc/ssh-analysis.txt (Oct. 4th)

[4] Kargieman, Emiliano, Pacetti, Ariel and Futoransky, Ariel. "An attack on CRC-32 integrity checks on encrypted channels using CBC and CFB modes",
URL: http://www.core-sdi.com/common/showdoc.php?idx=115&idxseccion=11&CORE-ST=02aabbca5aa0b0c831ab51c1d7b310c5 (Oct. 4th)

[5] Sander, Rebecca. "One incident of remediating the CRC32 sshd1 vulnerability", January 12, 2002
URL: http://rr.sans.org/incident/CRC32.php (Oct. 4th)

[6] CERT. Advisory CA-2001-35 Recent Activity Against Secure Shell Daemons, Dec. 14 2001
URL: http://www.cert.org/advisories/CA-2001-35.html (Oct. 4th)

[7] CERT. Incident Note IN-2001-12 - Exploitation of vulnerability in SSH1 CRC-32 compensation attack detector, Nov 15 2001
URL: http://www.cert.org/incident_notes/IN-2001-12.html (Oct. 4th)

[8] Irwin, Vicki. "Handler's Diary 11/04/01, SSH CRC32 Vulnerability Exploitation Update",
URL: http://www.incidents.org/archives/intrusions/msg01631.html (Oct. 4th)

[9] Provos, Niels, Honeyman, Peter. “ScanSSH - Scanning the Internet for SSH servers”
URL: http://www.citi.umich.edu/techreports/reports/citi-tr-01-13.pdf (Oct. 4th)
[10] CERT Vulnerability Note VU#945216: SSH CRC32 attack detection code contains remote integer overflow
URL: http://www.kb.cert.org/vuls/id/945216 (Oct. 4th)

Detect #2: HTTP Metadata information gathering

This detect was posted to the incidents.org mailing list on Monday 23 September. No comments/questions were received from the list.
I received some comments/suggestions from one person. They were addressed to my address directly and not to the list. I did not receive any comment on the list and I noticed very little activity on the list during this week and no activity at all since Thursday 26. I do not know whether any comment was sent in that period. Basically The comments I received to my address directly suggested that I reviewed the references, add a fourth question to the multiple-choice question and that I check whether other sites were targeted by the same individuals. I modified my detect taking into account his suggestions. However I could not find evidence of other hosts being targeted by the same hosts in my detect.

Trace log

```
[**] [1:1171:6] WEB-MISC whisker HEAD with large datagram [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/18-05:23:06.944488 198.144.198.170:14304 -> 46.5.180.133:80
TCP TTL:47 TOS:0x0 ID:45757 IpLen:20 DgmLen:579 DF
***AP*** Seq: 0x299D6282 Ack: 0x2E758A6E Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 7852100 8378965
[Xref => http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]

[**] [1:1171:6] WEB-MISC whisker HEAD with large datagram [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/18-05:42:00.464488 198.144.198.170:14310 -> 46.5.180.133:80
TCP TTL:47 TOS:0x0 ID:51168 IpLen:20 DgmLen:579 DF
***AP*** Seq: 0x71994D29 Ack: 0x773D695A Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 7965440 8492307
[Xref => http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]

[**] [1:1171:6] WEB-MISC whisker HEAD with large datagram [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/18-14:16:36.714488 202.214.44.44:1677 -> 46.5.180.133:80
TCP TTL:46 TOS:0x0 ID:60402 IpLen:20 DgmLen:570 DF
***AP*** Seq: 0x8A1E6C65 Ack: 0xDCAE637 Win: 0x4470 TcpLen: 20
[Xref => http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]

[**] [1:1171:6] WEB-MISC whisker HEAD with large datagram [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/18-14:31:24.894488 202.214.44.44:1693 -> 46.5.180.133:80
TCP TTL:46 TOS:0x0 ID:61360 IpLen:20 DgmLen:569 DF
***AP*** Seq: 0x98033C83 Ack: 0x45EB5987 Win: 0x4470 TcpLen: 20
[Xref => http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]
```

1. Source of Trace

The detect used for this trace is http://www.incidents.org/logs/Raw/2002.6.18.

Although we have no information about the network layout, by looking at the traffic contained in the raw data files and assuming that they were obtained using snort in binary logging mode we can observe some http and ftp traffic directed at the host involved in the traces (46.5.180.133). I assume only traffic that triggered an alert is logged (snort in binary mode) since no TCP session negotiation traffic is shown (for example, only ftp traffic where user is anonymous is logged).
The net block 46.0.0.0-46.255.255.255 is IANA reserved. Addresses in this block have probably been used to mask out the original addresses. This is the reason for the BAD checksum messages (the checksum field in the packet contains the value of the checksum computed using the original IP addresses).

2. Detect was generated by

The detect shown here was generated by snort 1.8.7 build 128 with the standard ruleset on the above mentioned file, using the -d option to dump the application layer payload. The format of the alert is the same as described in the previous detect.
The rule that triggered the alert is:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
MISC whisker HEAD with large datagram"; content:"HEAD"; offset: 0;
depth: 4; nocase; dsize:>512; flags:A+; classtype:attempted-recon;
reference:url,www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html;
sid:1171; rev:7;)
```

Basically this alert was generated because in a packet sent to one of the web servers on the monitored net on one of the HTTP ports, the following conditions occurred:

- the HTTP method used was "HEAD";
- the length of the application layer data content (i.e. the length of the HTTP request message) was greater than 512 bytes;
- the ack flag and possibly additional flags were set (actually in our case the ack and the push flags were set)

3. Probability the source address was spoofed

The packets logged by snort during this day and the previous ones show that there are replies from the 46.5.180.133 server on port 80, which means that a web server is really running on this host. This in turn suggests that, even though we do not see packets related to the connection establishment (TCP handshake: SYN - SYN/ACK - ACK) between this host and the two "attacking" hosts, the handshake has been completed and that the packets that we see in the trace are sent on an established TCP connection, which in turn suggests that the senders' addresses are not spoofed (unless the targeted web server is vulnerable to sequence number prediction attacks), otherwise the handshake phase could not have been completed.
A DNS query on the two "attacking" hosts gives the following results:

198.144.198.170: m198-170.dsl.rawbw.com
202.214.44.44: proxy2.sanritz.co.jp

A WHOIS query on the two addresses reveals that the first one belongs to the net block:

198.144.192.0 - 198.144.223.255 allocated to Raw Bandwidth

Communications, Inc., US

While the second one belongs to the net block:

202.214.44.40 - 202.214.44.47	allocated to Sanritz Automation Co., Ltd., Japan.

None of these addresses is contained in the Dshield database, indicating that no report were sent about attacks from these hosts.

4. Description of attack

snort detected an attempt at gathering information about a web server using the HEAD HTTP method from two different IP addresses. Each of the two “attacking” IP addresses sent two packets specifying a different URL. The “decoded” payload for the received messages is shown below:

From 198.144.198.170

```
HEAD /main/tools/discontinued/47u322.pdf HTTP/1.1\r\n
Host: www.XXXX.com\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0)
Gecko/20020529\r\n
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1\r\n
Accept-Language: en-us, en;q=0.50\r\n
Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n\r\n

HEAD /main/tools/discontinued/47u33x.pdf HTTP/1.1\r\n
Host: www.XXXX.com\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0)
Gecko/20020529\r\n
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1\r\n
Accept-Language: en-us, en;q=0.50\r\n
Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n\r\n
```

From 202.214.44.44

```
HEAD /main/datasheets/37c67x.pdf HTTP/1.1\r\n
Via: 1.1 - (DeleGate/7.5.3)\r\n
Host: www.XXXX.com\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; ja-JP; rv:1.0rc2)
Gecko/20020512 Netscape/7.0b1\r\n
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1\r\n
Accept-Language: ja_JP, ja;q=0.50\r\n
Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
Accept-Charset: Shift_JIS, utf-8;q=0.66, *;q=0.66\r\n
Pragma: no-cache\r\n
```

```
Cache-Control: no-cache\r\n\r\n

HEAD /main/tools/io-sch/67x.pdf HTTP/1.1\r\n
Via: 1.1 - (DeleGate/7.5.3)\r\n
Host: www.XXXX.com\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; ja-JP; rv:1.0rc2)
Gecko/20020512 Netscape/7.0b1\r\n
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1\r\n
Accept-Language: ja_JP, ja;q=0.50\r\n
Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
Accept-Charset: Shift_JIS, utf-8;q=0.66, *;q=0.66\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n\r\n
```

Google search for the files requested with the HEAD method in these http requests gave the following results:

```
www.smsc.com/main/tools/discontinued/47u322.pdf
www.smsc.com/main/datasheets/37c67x.pdf
www.smsc.com/main/tools/io-sch/67x.pdf
www.smsc.com/main/tools/discontinued/47u33x.pdf
```

Suggesting that the requested files exist.
The alert was triggered because the payload of the HTTP protocol data unit was longer than 512 bytes. The payload contains, as per the HTTP specification [2]

```
Method SP Request-URI SP HTTP-version CRLF <Request-header>
```

Where:

```
Method = HEAD
Request-URI = /main/tools/discontinued/47u322.pdf,
/main/tools/discontinued/47u33x.pdf,
/main/datasheets/37c67x.pdf, /main/tools/io-sch/67x.pdf
HTTP-version = HTTP/1.1
SP indicates the "space" character and CRLF the "carriage
return" and "linefeed characters".
```

And

```
Request-headers = Via, Host, User-Agent, Accept, Accept-
Language, Accept-Encoding, Accept-Charset, Pragma, Cache-
Control
```

The data in the HEAD message indicates that the OS of the two hosts are Linux and Windows 5.1 respectively. By observing the characteristics of the TCP packet (Window size, TTL, and TCP options) and comparing them to the fingerprints file of p0f [3], we can see that they are fairly consistent, indicating that there is no attempt at the "attacking" side to obfuscate information about the OS.

5. Attack mechanism

The snort rule that triggered the alert suggests usage of the whisker tool to evade IDS detection of information gathering activities against a web server.
Whisker is a tool that implements a series of "anti-IDS" tactics described in [1]. As described in this paper: "The goal of any anti-IDS tactic is to mutate a request so much that the ID systems will get confused, but the web server will still be able to understand it." The idea behind whisker is to modify the request in such a way that

the request is still syntactically correct for the web server, but does not match the signatures specified in the intrusion detection system. These tactics are especially effective against IDS which use a simple pattern matching method to examine incoming traffic. Examples of the modifications used by whisker include evasion of scans detection by using the HEAD method instead of the more common GET method (HEAD does not allow transfer of the resource, but nevertheless gives information about its availability and this information can be used later to actually exploit the target vulnerability), URL encoding, i.e. encode a URI with its escaped equivalent which is obtained by substituting a character with its hex value, use of double slashes or directory traversal, etc.
The HEAD method is used usually by links checking software and proxies and, in general, when actual retrieval of the resource is not needed. As it does not transfer the actual content but also the metadata, it is faster and less bandwidth consuming than the GET method. Quoting the HTTP specifications [2]: "The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification."
The HEAD method is generally used by proxy servers to test URIs, either to see whether an updated version is available or to ensure that the URI is available at all. Proxy servers are special server configurations that collect Web pages from standard servers, as though they were a Web client, and serve it back to Web clients, as though they were a conventional server. The second "attacker" has indeed a name that suggests it is in fact a proxy server.
The packets that triggered the alert are too few to be a scan for a vulnerable CGI using the HEAD method in an attempt to evade the IDS. In addition they are not targeted at any known vulnerable CGI script. This suggests that these packets are a false positive, being legitimate HTTP requests.

6. Correlations

CAN-2000-0899 describes a vulnerability of Small HTTP Server 2.01, which allows remote attackers to cause a denial of service by connecting to the server and sending out multiple GET, HEAD, or POST requests and closing the connection before the server responds to the requests.
The vulnerability is also described in:
BUGTRAQ:20001114 Vulnerabilites in SmallHTTP Server,
URL:http://marc.theaimsgroup.com/?l=bugtraq&m=97421834001092&w=2

BID:1942, URL:http://www.securityfocus.com/bid/1942

However, this is not the case here, since we do not see multiple GET, HEAD and POST request. No attempts from these hosts were observed in the previous days either.

7. Evidence of active targeting

The web server at 46.5.180.133 is evidently targeted as the repository of the documents specified in the URI contained in the HTTP HEAD request. However this

seems to be related to verifications of the availability of the documents specified in the URI rather than an attack on the web server.

8. Severity

The following formula is used to calculate the severity of the attack:
Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each aspect is ranked with a value from 1 to 5, 1 being the lowest and 5 being the highest.

Criticality: We do not have any information about this web server. Since, normally a web server is not a critical element of a network infrastructure (although it could be critical for a particular business), we will give it a 3
Lethality: this is not really an attack, so we will give it a 1
System Countermeasures: Again, we do not have any information about the countermeasures adopted on this particular server, assuming that all security patches have been applied both at the OS level and at the level of the web server (but not being sure about it...), we will give it a 4
Network Countermeasures: Again, we do not have any information about the countermeasures adopted on the network on which this server resides, since this network has at least an IDS system, we assume that it also has implemented some perimeter filtering system, we will give it a 4

Severity = (3+1)-(4+4) = -4

9. Defensive recommendation

Keep the system and the web server software up to date with patches, monitor constantly the IDS and web server logs and correlated the information contained therein.

10. Multiple choice test question

The HTTP HEAD method can be used
a) by proxies to retrieve and cache documents locally,
b) by IDS evasion tools to scan silently for known vulnerable CGI,
c) by web browsers to send user input data
d) by web browsers to maintain information about the status of the connection

The correct answer is b)

References

[1] Rain Forest Puppy, “A look at whisker’s anti-IDS tactics”, URL: http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html (October, 4th)
[2] “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616 URL: http://www.w3c.org/Protocols/rfc2616/rfc2616.html (October, 4th)
[3] P0f, URL: http://www.stearns.org/ (October, 4th)

Detect #3: Nmap TCP ping or load balancing device?

Trace log

```
 [**] [1:628:1] SCAN nmap TCP [**]
 [Classification: Attempted Information Leak] [Priority: 2]
 07/18-02:23:16.794488 163.23.190.2:80 -> 46.5.102.27:80
 TCP TTL:46 TOS:0x0 ID:37237 IpLen:20 DgmLen:40
 ***A**** Seq: 0x2EA Ack: 0x0 Win: 0x578 TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS28]

[snip]

 [**] [1:628:1] SCAN nmap TCP [**]
 [Classification: Attempted Information Leak] [Priority: 2]
 07/18-09:27:28.804488 12.164.64.41:80 -> 46.5.131.128:80
 TCP TTL:47 TOS:0x0 ID:61042 IpLen:20 DgmLen:40
 ***A**** Seq: 0xB4 Ack: 0x0 Win: 0x400 TcpLen: 20
 [Xref => http://www.whitehats.com/info/IDS28]

[snip]
```

tcpdump trace log (ordered by source IP address)

```
02:23:16.794488 163.23.190.2.80 > 46.5.102.27.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 46, id 37237, len 40, bad cksum d28!)
03:06:56.734488 163.23.190.2.80 > 46.5.147.201.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 46, id 15503, len 40, bad cksum 335f!)
03:08:14.424488 163.23.190.2.80 > 46.5.151.93.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 46, id 32675, len 40, bad cksum ebb8!)
03:08:17.424488 163.23.190.2.80 > 46.5.151.93.80: . [bad tcp cksum faf7!] ack 1 win 1400 (ttl 46, id 33296, len 40, bad cksum e94b!)
03:08:20.404488 163.23.190.34.80 > 46.5.151.93.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 46, id 33913, len 40, bad cksum e6c2!)
04:34:03.584488 163.23.190.2.80 > 46.5.35.191.80: . [bad tcp cksum f7fa!] ack 0 win 1400 (ttl 46, id 62064, len 40, bad cksum ef86!)
04:34:06.604488 163.23.190.2.80 > 46.5.35.191.80: . [bad tcp cksum f7fa!] ack 1 win 1400 (ttl 46, id 62438, len 40, bad cksum ee10!)
07:26:19.984488 163.23.190.2.80 > 46.5.39.76.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 46, id 18533, len 40, bad cksum 9507!)
09:05:28.474488 163.23.190.2.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 46, id 55443, len 40, bad cksum dbf0!)
09:05:31.474488 163.23.190.2.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 1 win 1400 (ttl 46, id 55953, len 40, bad cksum d9f2!)
09:05:34.694488 163.23.190.34.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 46, id 56415, len 40, bad cksum d804!)
09:05:37.594488 163.23.190.34.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 1 win 1400 (ttl 46, id 56895, len 40, bad cksum d624!)
09:05:54.034488 163.23.190.130.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 46, id 59335, len 40, bad cksum cc3c!)
09:05:57.544488 163.23.190.130.80 > 46.5.80.52.80: . [bad tcp cksum f8f8!] ack 1 win 1400 (ttl 46, id 59786, len 40, bad cksum ca79!)
09:06:42.254488 163.23.190.2.80 > 46.5.130.235.80: . [bad tcp cksum f7fa!] ack 0 win 1400 (ttl 46, id 1532, len 40, bad cksum 7ccf!)
09:06:45.264488 163.23.190.2.80 > 46.5.130.235.80: . [bad tcp cksum f7fa!] ack 1 win 1400 (ttl 46, id 1943, len 40, bad cksum 7b34!)
09:06:48.494488 163.23.190.34.80 > 46.5.130.235.80: . [bad tcp cksum f7fa!] ack 0 win 1400 (ttl 46, id 2389, len 40, bad cksum 7956!)
09:06:51.534488 163.23.190.34.80 > 46.5.130.235.80: . [bad tcp cksum f7fa!] ack 1 win 1400 (ttl 46, id 2872, len 40, bad cksum 7773!)
11:14:34.504488 163.23.190.2.80 > 46.5.248.197.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 46, id 19304, len 40, bad cksum bf89!)
12:20:05.034488 163.23.190.2.80 > 46.5.251.28.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 46, id 61150, len 40, bad cksum 18be!)
12:20:08.034488 163.23.190.2.80 > 46.5.251.28.80: . [bad tcp cksum faf7!] ack 1 win 1400 (ttl 46, id 61689, len 40, bad cksum 16a3!)
12:20:11.474488 163.23.190.34.80 > 46.5.251.28.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 46, id 62209, len 40, bad cksum 147b!)
15:07:18.184488 163.23.190.2.80 > 46.5.246.140.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 46, id 35962, len 40, bad cksum 80b0!)

13:17:25.374488 199.197.130.21.80 > 46.5.180.133.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 55, id 59474, len 40, bad cksum 751e!)
13:17:25.394488 199.197.135.21.80 > 46.5.180.133.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 50, id 59477, len 40, bad cksum 751b!)

04:53:01.394488 218.96.62.2.80 > 46.5.183.70.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 44, id 13908, len 40, bad cksum 5fd6!)
04:53:06.424488 218.96.62.2.80 > 46.5.183.70.80: . [bad tcp cksum faf7!] ack 1 win 1400 (ttl 44, id 14350, len 40, bad cksum 5e1c!)
09:03:57.304488 218.96.62.2.80 > 46.5.252.204.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 44, id 57700, len 40, bad cksum 703d!)
09:04:02.364488 218.96.62.2.80 > 46.5.252.204.80: . [bad tcp cksum f9f9!] ack 1 win 1400 (ttl 44, id 58126, len 40, bad cksum 6e93!)

04:52:51.294488 202.96.52.99.80 > 46.5.183.70.80: . [bad tcp cksum faf7!] ack 0 win 1400 (ttl 47, id 13062, len 40, bad cksum 79c3!)
04:52:56.364488 202.96.52.99.80 > 46.5.183.70.80: . [bad tcp cksum faf7!] ack 1 win 1400 (ttl 47, id 13484, len 40, bad cksum 781d!)
09:03:47.244488 202.96.52.99.80 > 46.5.252.204.80: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 47, id 56790, len 40, bad cksum 8a6a!)
09:03:52.274488 202.96.52.99.80 > 46.5.252.204.80: . [bad tcp cksum f9f9!] ack 1 win 1400 (ttl 47, id 57240, len 40, bad cksum 88a8!)

02:52:56.114488 194.52.177.9.80 > 46.5.78.7.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 38, id 47112, len 40, bad cksum f484!)
```

```
02:53:01.104488 194.52.177.9.80 > 46.5.78.7.80: . [bad tcp cksum f8f8!] ack 1 win 1400 (ttl 38, id 47406, len 40, bad cksum f35e!)
04:56:01.254488 194.52.177.9.80 > 46.5.57.85.80: . [bad tcp cksum f8f8!] ack 0 win 1400 (ttl 38, id 41810, len 40, bad cksum 1ded!)
04:56:06.234488 194.52.177.9.80 > 46.5.57.85.80: . [bad tcp cksum f8f8!] ack 1 win 1400 (ttl 38, id 42166, len 40, bad cksum 1c89!)

14:48:55.544488 64.152.70.68.53 > 46.5.180.250.53: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 49, id 38105, len 40, bad cksum 9121!)
14:48:55.544488 64.152.70.68.80 > 46.5.180.250.53: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 49, id 38104, len 40, bad cksum 9122!)
14:48:55.604488 63.211.17.228.80 > 46.5.180.250.61424: . [bad tcp cksum f9f9!] ack 0 win 1400 (ttl 49, id 62213, len 40, bad cksum 681a!)

09:27:28.804488 12.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 47, id 61042, len 40, bad cksum a413!)
09:27:33.884488 12.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 1 win 1024 (ttl 47, id 61606, len 40, bad cksum a1df!)
09:27:38.904488 38.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 48, id 62198, len 40, bad cksum 848f!)
09:27:43.944488 38.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 1 win 1024 (ttl 48, id 62680, len 40, bad cksum 82ad!)
09:27:48.964488 207.106.237.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 51, id 63276, len 40, bad cksum 2792!)
09:27:53.994488 207.106.237.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 1 win 1024 (ttl 51, id 63852, len 40, bad cksum 2552!)
```

1. Source of Trace

The detect used for this trace is http://www.incidents.org/logs/Raw/2002.6.18.
We do not have any information about the structure of this network, the services that are provided or the perimeter defences in place. A look at the binary logs of the previous days on the hosts targeted by this attack did not provide any useful information, i.e. no evidence was found that publicly accessible services are run on these hosts (e.g. DNS, http, ftp, ssh, etc.) apart from 46.5.180.133.80 for which data transfers on port 80 and 21 were found, suggesting this host is running http and ftp services.
The net block 46.5.0.0-46.255.255.255 is IANA reserved. Addresses in this block have probably been used to mask out the original addresses. This is the reason for the BAD checksum messages (the checksum field in the packet contains the value of the checksum computed using the original IP addresses).

2. Detect was generated by

There are two types of logs shown in this detect: the first part is an excerpt of the alerts generated by snort 1.8.7 build 128 with the standard ruleset on the above mentioned file, using the -d option to dump the application layer payload.
The rule that triggered the alert is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "SCAN nmap TCP";
flags:A,Ack:0; reference:arachnids,28; classtype:attempted-recon;
sid:628; rev:1;)
```

Basically this alert was generated on TCP packets with the ACK bit set and with a value 0 in the acknowledge field.
Snort associates this signature to the use of nmap to scan remote hosts using TCP.
The snort alert log contained 13 different source IP's generating this alert, the information dumped by snort for all these packets was very similar except for the window size for which there were two different values (1400 and 1024). Snort alerts are shown only for two packets, while the rest of the traffic is shown as tcpdump output obtained by filtering on the IP addresses that triggered the alert.

3. Probability the source address was spoofed

This is a reconnaissance attempt where the attacker is interested in receiving results from his probes. The probability that source IP is spoofed is low, unless the attacker can capture return traffic from the target to the spoofed address.
There are 13 hosts involved in this probe.
A DNS query on the "attacking" hosts gives the following results:

IP address	Fully qualified domain name
163.23.190.2	-
163.23.190.34	-
163.23.190.130	-
218.96.62.2	-
202.96.52.99	-
194.52.177.9	lp.adept.se
64.152.70.68	proximitycheck2.allmusic.com
63.211.17.228	proximitycheck1.allmusic.com

12.164.64.41	-
207.106.237.41	41-237-106-207.thompcomp.net
38.164.64.41	-
199.197.130.21	-
199.197.135.21	-

A whois query on ARIN[7] (American Registry for Internet Numbers) shows that the first three addresses (163.23.190.2, 163.23.190.34, 163.23.190.130) belong to 163.13.0.0 - 163.32.255.255 assigned to Ministry of Education Computer Center, Changhua Country Education Network. None of them resolves to a name and scans from all of them have been reported to Dshield (respectively 673, 734 and 620 targets)

A whois query on APNIC (Asia Pacific Network Information Center)[8] shows that 218.96.62.2, belongs to 218.96.0.0 - 218.97.127.255, assigned to China Enterprise Network Communication Technology Co. Ltd. The address is present in the Dshield database with 263 targets.

Analogously it shows that 202.96.52.99 belongs to 202.96.52.0 - 202.96.52.127, assigned to National Economic Trade Institute, China. This host too is present in the Dshield database with 369 targets.

A whois query on RIPE (Réseaux IP Européens)[9] shows that 194.52.177.9 belongs to 194.52.177.0 - 194.52.177.255, assigned to Adept AB, Sweden. This host too is present in the Dshield database with 240 targets.

A whois query on ARIN shows that 64.152.70.68 belongs to 64.152.0.0 - 64.159.255.255, assigned to Level 3 Communications, Inc. This host is present in the Dshield database with 740 targets. 63.211.17.228 too belongs to a network range (63.208.0.0 - 63.215.255.255) allocated to Level 3 Communications, Inc. and is present in the Dshield database with 775 targets.

The whois query for 12.164.64.41 shows that it belongs to 12.164.64.0 - 12.164.64.255, assigned to EARTHSTATION NETAXS. Attacks from this host too have been reported to Dshield (50 targets).
207.106.237.41 too belongs to a network range (207.106.0.0 - 207.106.255.255) assigned to Netaxs. This host too is in the Dshield database (4 targets)

38.164.64.41 belongs to the net-range 38.0.0.0 - 38.255.255.255, assigned to Performance Systems International Inc. This host is in the Dshield database with 9 targets.

199.197.130.21 and 199.197.135.21 belong to the network range 199.197.128.0 - 199.197.255.255, assigned to Corning Incorporated, US and both are present in the Dshield database respectively with 252 and 7 targets.

[7] http://www.arin.net
[8] http://www.apnic.org
[9] http://www.ripe.net

Therefore we can group the attackers, according to the organization to which they are assigned, as follows:

IP address	Organization associated to the Netblock of the IP address
163.23.190.2 163.23.190.34 163.23.190.130	Ministry of Education Computer Center, Changhua Country Education Network
64.152.70.68 63.211.17.228	Level 3 Communications
199.197.130.21 199.197.135.21	Corning Inc.
12.164.64.41 207.106.237.41	Netaxs
38.164.64.41	Performance Systems International Inc.
218.96.62.2	China Enterprise Network Communication Technology
202.96.52.99	National Economic Trade Institute, China
194.52.177.9	Adept AB, Sweden

4. Description of attack

TCP packets with the ACK flag set and acknowledge number value 0 are received by various hosts in our network (46.5.0.0/16). Some of these packets appear to be generated on networks belonging to the same organization, in most cases each source address is sending two such packets to each destination and the time delta between two packets to a destination IP from the same IP address and sometimes from different IP addresses in the same organization is very similar (3 seconds for the source addresses in the 163.23.190.x block, 5 seconds for 218.96.62.2, 202.96.52.99, 194.52.177.9, 12.164.64.41, 38.164.64.41, 207.106.237.41) with a few exceptions. Most such packets are addressed at port 80 and have port 80 also as source port, with the exception of packets from Level 3 communication, which are addressed to port 53 and port 61424 and originate from ports 53 and 80.

5. Attack mechanism

This attack is described by snort as a portscan using the nmap scanning tool [2]. Nmap has a number of varied options for scanning remote hosts. One of these options (-sA) allows sending a TCP packet with the ACK bit set to a specified port. This option is useful in determining whether the remote port is filtered by a stateful firewall or by a packet filter that blocks only incoming SYN packets. However, this option sets the value of acknowledgement number to a random-looking value. Nmap has also another option that allows sending a TCP packet with the ACK bit set to determine if a host is up and running, instead of using an ICMP echo request (-PT). The default destination port used in this type of probing is 80, while the source port can be set to any value using the "-g Port_number" option. So, it is possible that these two options were used to forge these packets. Usage of port 80 as the source port would mask traffic as "return" traffic from a web site. After a test with nmap, with the following options:

```
#nmap -sP -PT -g 80 <destination_address>
```

I discovered that, while the ACK value is set, the acknowledgement number is not 0 as in the packets in the trace. Apparently only old versions of nmap had this "0 value problem", newer version set a random-looking value just like the -sA option (I was using nmap-v3.00).
Another tool which could have been used in this type of reconnaissance probing is hping2 [4]. Hping is a tool that can send "custom TCP/IP packets" and can also be used to transfer files under supported protocols. It can perform traceroute-like functions using different protocols, remote OS fingerprinting, TCP/IP stack auditing, and other similar functions.
Hping2 with the following options gives exactly the same packets as the ones in the trace examined in this detect:

```
#hping -c 2 -I 5 -s 80 -p 80 --keep -A -L 0 -w 1400 <destination_address>
```

The "c" option stops after sending the specified number of packets
The "i" option sets the interval, in seconds, between each packet sent
The "s" option sets the source port
The "p" option sets the destination port
The "A" option sets the ACK bit
The "L" option sets the value of the acknowledgement number
The "w" option sets the TCP window size
The "--keep" option keeps the source port constant

Here is the trace of the hping2 tool executed with the options above:

```
13:06:59.256030 10.0.0.1.80 > 10.0.0.2.80: . [tcp sum ok] ack 0 win 1400
(ttl 64, id 53614, len 40)
13:07:04.256030 10.0.0.1.80 > 10.0.0.2.80: . [tcp sum ok] ack 1 win 1400
(ttl 64, id 21565, len 40)
```

In conclusion, we can rule out nmap, but hping2 could have been used in these scans. However, there is another interesting possibility. According to [5], which shows a trace similar to what presented in this detect, these packets could be part of the traffic generated by a load balancing device (LinkProof by Radware). This device tries to calculate the best route in terms of load and response time to a target server. It chooses the target server based on an internal table which is updated regularly. In [5], they describe the activity of a LinkProof device against their name server which includes a UDP packet, followed by an ICMP Echo Request, a TCP ACK, TCP SYN, TCP RST. They also provide a trace that is very similar to what presented in this trace (same window size, two TCP ACK packets. A similar activity is also mentioned in [6], [7] and [8]. Apparently, the LinkProof device can be recognized by the fact that it sends a UPD packet to port 37852. However for the detect in this practical we do not have all the traffic and therefore are unable to verify that such packets have been received.
An additional piece of information that we can use in analysing this detect is the information that we got with the name resolution. At least in two of these addresses (proximitycheck2.allmusic.com, proximitycheck1.allmusic.com) we get two names that would make us think that this is a reconnaissance probe aimed at load balancing or finding the optimum route.

6. Correlations

The nmap TCP scan is described in Arachnids[10]:
IDS28/SCAN_PROBE-NMAP_TCP_PING
"This event indicates that a remote user has used the NMAP portscanning tool to probe the server. An NMAP TCP ping was sent to determine if a host is reachable. This event is specific to a particular exploit, but the packet payload is not considered as part of the signature to detect the attack."
This attack has also a CVE number: CVE: CAN-1999-0523
Attacks triggered by the same snort rule, related to the nmap TCP scan, were described in the GCIA practicals by James Conz, Vernon Stark, and Roderick Campbell[11]. However all of them reached the conclusion that the packets were effectively generated by nmap.
Similar activity related to the use of the LinkProof Radware load-balancing device is described in [5], [6], [7] and in the GCIA practical of William Stearns.

7. Evidence of active targeting

Some hosts in the 46.5.0.0/16 network are directly targeted. This is not a scan involving the whole network. In absence of information about the services running on the targeted hosts (no useful information was found looking at the logs from previous days, which seem to be the binary generated by snort and hence contain only packets that generated some alerts and no logs are available with all the traffic), we can only make an assumption that these hosts were targeted because they run services useful for the load balancing measures of the remote host.

8. Severity

The following formula is used to calculate the severity of the attack:
Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each aspect is ranked with a value from 1 to 5, 1 being the lowest and 5 being the highest.

Criticality: We assumed the targeted systems include systems that run services that might be critical (web sites, DNS), 4
Lethality: this is a reconnaissance, apparently only aimed at gaining timing information, 2
System Countermeasures: We assume all critical systems have been patched and secured, but we are not sure, 4
Network Countermeasures: Since this network has an IDS system in place, we assume it also has in place efficient security countermeasures at the network level, but we are not sure, 4

Severity = (4+2) - (4+4) = -2

9. Defensive recommendation

If the assumptions about the countermeasures at the system level are wrong (i.e. systems not up to date with patches, etc.) we recommend checking the vulnerability

[10] ARACHNIDS is available at http://www.whitehats.com
[11] GIAC practicals are available at http://www.giac.org/cert.php

status of these servers with a vulnerability scanner, applying the necessary patches and run all services in chroot environment.

10. Multiple choice test question

09:27:28.804488 12.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 47, id 61042, len 40, bad cksum a413!)
09:27:38.904488 38.164.64.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 48, id 62198, len 40, bad cksum 848f!)
09:27:48.964488 207.106.237.41.80 > 46.5.131.128.80: . [bad tcp cksum faf7!] ack 0 win 1024 (ttl 51, id 63276, len 40, bad cksum 2792!)

The above trace shows clear signs of packet crafting because
a) the TTL value is changing
b) the checksum value does not match
c) the ACK bit is set and the value of the acknowledgement field is 0
d) the length of the packet is 40

Correct answer is c)

References

[1] Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Inc, 1994
[2] Fyodor. "Remote OS detection via TCP/IP stack fingerprinting" URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html (October, 4th)
[3] "List of fingerprints for passive fingerprint monitoring" URL: http://project.honeynet.org/papers/finger/traces.txt (October, 4th)
[4] Hping man page, URL: http://www.hping.org/manpage.html (October, 4th)
[5] Global Incidents Analysis Center, Report Date: March 14, 2001, URL: http://www.sans.org/y2k/031401.htm (October, 4th)
[6] Posting at incidents.org, URL: http://www.incidents.org/archives/intrusions/msg03560.html (October, 4th)
[7] Posting on incidents.org, URL: http://www.incidents.org/archives/intrusions/msg08110.html (October, 4th)
[8] William Stearns, GIAC Practical, URL: http://www.sans.org/y2k/practical/william_stearns_gcia.html

Detect #4: A fast scan for ftp servers

Trace log

```
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:48.095592 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.2:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x2FA49E87 Ack: 0x27D62430 Win: 0x404 TcpLen: 20

[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:48.118323 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.3:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x2FA49E87 Ack: 0x27D62430 Win: 0x404 TcpLen: 20
```

```
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:48.158553 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.5:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x2FA49E87 Ack: 0x27D62430 Win: 0x404 TcpLen: 20

[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:48.161564 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.1:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x2FA49E87 Ack: 0x27D62430 Win: 0x404 TcpLen: 20

[snip 31 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:48.661811 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.30:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x5CF1C357 Ack: 0x5BFC2F01 Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:49.661168 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.80:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0xB2F93F1 Ack: 0x1121BE91 Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:50.658178 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.130:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x38BA7B73 Ack: 0x5DA6EB0 Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:51.660152 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.180:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x671DC684 Ack: 0x7B3E9782 Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:52.655895 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.z.230:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1496D5A8 Ack: 0x2FBF6190 Win: 0x404 TcpLen: 20

[snip 35 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:53.964456 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.w.40:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x42D11EA4 Ack: 0x64FB32B8 Win: 0x404 TcpLen: 20

[snip 34 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:54.657947 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.w.75:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x30AC65F6 Ack: 0x19F297CF Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:55.655847 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.w.125:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x5E0B3807 Ack: 0x4E64AC3C Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:56.665207 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.w.175:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x4C1CE29B Ack: 0x33564D9 Win: 0x404 TcpLen: 20

[snip 49 pkts deleted]
[**] [111:13:1] spp_stream4: STEALTH ACTIVITY (SYN FIN scan) detection [**]
08/19-14:00:57.657172 aa:bb:cc:dd:ee:ff -> gg:hh:ii:jj:kk:ll type:0x800 len:0x3C
211.57.212.220:21 -> x.y.w.225:21 TCP TTL:21 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x3A4D2EE8 Ack: 0x785A64D2 Win: 0x404 TcpLen: 20

[snip 29 pkts deleted]
```

Excerpts from tcpdump log file

```
14:00:48.095616 211.57.212.220.21 > x.y.z.2.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
```

```
14:00:48.118341 211.57.212.220.21 > x.y.z.3.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.158576 211.57.212.220.21 > x.y.z.5.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.161582 211.57.212.220.21 > x.y.z.1.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.163073 211.57.212.220.21 > x.y.z.4.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.178565 211.57.212.220.21 > x.y.z.6.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.198737 211.57.212.220.21 > x.y.z.7.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.218567 211.57.212.220.21 > x.y.z.8.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.247278 211.57.212.220.21 > x.y.z.9.21: SF [tcp sum ok] 799317639:799317639(0)
win 1028 (ttl 21, id 39426, len 40)
14:00:48.259037 211.57.212.220.21 > x.y.z.10.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.280628 211.57.212.220.21 > x.y.z.11.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.296346 211.57.212.220.21 > x.y.z.12.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.318428 211.57.212.220.21 > x.y.z.13.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.336140 211.57.212.220.21 > x.y.z.14.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
[snip]
```

1. Source of Trace

NIDS sensor on a network that I manage.

2. Detect was generated by

This detect was generated by snort 1.8.7 build 128 in full alert mode with the standard ruleset.
Each alert generated by snort contains the following information:
1st line:
[**] ... [**] = snort-signature,
2nd line:
MM/DD-hh:mm:ss.cccccc = timestamp,
aa:bb:cc:dd:ee = data link source address
-> = traffic direction
gg:hh:ii:jj:kk:ll = data link destination address
Data link type
Length of the data link frame
3rd line:
A.B.C.D:X = IP source address and source port
-> = traffic direction
E.F.G.H:Y = IP destination address and destination port
Protocol (TCP, UDP, etc.)
Time to live (TTL) - as specified in the IP header
Type of Service (TOS) - as specified in the IP header
IP Identification number - as specified in the IP header
Length of the IP header
Length of the IP datagram
4th line:
List of TCP flags in the TCP header ("*" indicates that the flag corresponding to the option in that position is not set)
TCP Sequence number
TCP Acknowledge number

TCP Window size
Length of the TCP header

The destination addresses have been sanitized, and represented as x.y.z.N, and x.y.w.N (N is a number that ranges in the trace from 1 to 255) to indicate two different 255-addresses blocks.
The alert was generated by snort's stream4 preprocessor with the option detect_scans. This preprocessor detects statefully (taking into account the phases of a TCP connection) various port scans. In this case it detected a fast portscan (about 50 packets per second) each packet of which contained both the SYN and FIN flags set. This should never occur in any phase of a TCP connection, as SYN indicated the intention to establish a new connection (transition from the CLOSED state to the SYN_SENT state) while FIN indicates the intention to close a connection already established (transition from the ESTABLISHED to the FIN_WAIT_1 state) [1].

3. Probability the source address was spoofed

The probability that the source address is spoofed is low. The "attacking" host is trying to determine if the ftp service is running on any host or simply mapping the network, scanning almost completely two 255-addresses netblocks, probably in an information gathering attempt for a possible future exploit. For this attempt to succeed, the remote system needs to receive replies to its probes, for this reason, unless it is positioned between the possibly spoofed address and the target and is able to capture traffic in transit, the probability that the originating address is spoofed is very low.
A whois query at APNIC (Asia Pacific Network Information Center) on the remote host gave the following result:

211.57.212.0 - 211.57.213.127	allocated to OFFICE OF EDUCATION KYONGNAM KEOCHANG, 1303-4 Daepyong-ri Keochang-eup Keochang-gun, KYONGNAM, Korea

An nslookup query on the remote IP address (211.57.212.220) did not produce a hostname in return, while a query on the DShield database indicated that probes coming from this host had been reported against 72581 hosts, ports targeted are not indicated.

4. Description of attack

The remote host is scanning two 255-addresses netblocks in an attempt to find hosts running the ftp service or to simply to map the network for alive hosts. The packets are an attempt to go undetected by firewalls and/or packet filtering devices relying on the fact that connection attempts with an out of spec flags combination set are not logged by some perimeter defence devices. The first packet is received at 14:00:48 19 August, while the last packet of the first scan is received at 14:00:58, 19 August. 479 packets are received in 10 seconds. This is really a fast scan.
The packet show evident signs of crafting: the same Sequence Number is repeated for 50 times before changing to a different value. The same happens with the acknowledge number, while, strangely enough there is no attempt at rendering the IP Identification number "apparently" random, as its value is the same for all packets

(39426), a sure sign that the packets are crafted. Other interesting values in the packets received are the TTL (21) which suggests that the initial TTL could have been 64 and that the attacking host is 43 hops away (in effect a traceroute to the attacking address stops after 23 hops at an address on the same subnet but different from the attacker, 211.57.212.217) and the window size (0x404) which, according to [2] does not correspond to any common operating system.
An additional sign of crafting is the fact that both source and destination port are the same (reflexive scan).

5. Attack mechanism

This is a fast scan that is targeted at finding hosts running the ftp service. Information gathered via this type of probing activities can be used in subsequent attacks. The attacking host scans for hosts listening on port 21, which is associated to ftp (file transfer protocol) using an illegal combination of TCP flags (SYN and FIN, respectively used to open and to close a connection, they should never be found in the same packet). In the past, the SYN-FIN flags combination was considered stealth because some packet-filtering devices did not use to log illegal connection attempts. However, nowadays this is a known and detected combination, it is surprising that it is still used. In fact, the IP ID value of 39426 and windows size of 1028 (0x404) are clear signs that the tool used for the scanning is synscan. [3] provides also an explanation of the fact that sequence numbers (SeqNo) and acknowledge numbers (AckNo) change after approximately 50 packets. The random function called to obtain random values for SeqNo and AckNo is seeded with UNIX time function which, under Linux returns a value rounded to the nearest second. As a consequence, in the timeframe of one second this function and hence the random function will return the same value, or, in other words, the SeqNo and AckNo will change after one second. In one seconds about 50 packets are sent, each of them will have the same AckNo and SeqNo values. Synscan is also used by the Ramen Linux worm to scan for vulnerable versions of wu-ftpd [4], if it finds target systems responsive to its SYN-FIN probe (some OS would reply with a SYN/ACK to a SYN/FIN!), it connects to the system to grab the ftp banner. So we cannot exclude that the attacking host is infected with the Ramen worm or a possible modification, since, according to [4], Ramen probes random class B address space.

6. Correlations

The alert raised by snort is related to the fast 21/TCP scan with SYN and FIN bits both set. This is described in ARACHNIDS[12], "IDS441/SCAN_PROBE-SYNSCAN-PORTSCAN", which also relates this types of scan to synscan.
Synscan probes have been recently reported in the following posts:

http://cert.uni-stuttgart.de/archive/incidents/2001/01/msg00189.html:
http://cert.uni-stuttgart.de/archive/incidents/2001/06/msg00161.html:

Synscan is also described in GIAC practical[13] by Donald J. Smith, while GIAC practical by Wade Walker describes a similar scan but targeted at port 111 (portmap). Information about ftp vulnerabilities can be found in [5], [6] and [7] among the others. In particular, [6] is related to any system running wu-ftpd 2.6.0 or earlier. wu-ftpd is a

[12] ARACHNIDS is available at http://www.whitehats.com
[13] GIAC practicals are available at http://www.giac.org/cert.php

common package used to provide file transfer protocol (ftp) services and is the one targeted by Ramen.

7. Evidence of active targeting

This is a scan directed at complete 255-addresses netblocks. No address is targeted in particular. It is likely that this network was included as part of a wider scan. Similar scans from the same address have been noticed on other networks.

8. Severity

The following formula is used to calculate the severity of the attack:
Severity = (criticality + lethality) - (system countermeasures + network countermeasures)
Each aspect is ranked with a value from 1 to 5, 1 being the lowest and 5 being the highest.

Criticality: there are critical services among those probed, 4
Lethality: reconnaissance stealth scan, 3
System Countermeasures: ftp is not enabled, 5
Network Countermeasures: the firewall dropped all connections attempts, 5

Severity = (4+3) - (5+5) = -3

9. Defensive recommendation

Continue monitoring this type of traffic. If ftp is enabled, ensure, in addition to applying all necessary system patches, that vulnerabilities bulletins are checked regularly.

10. Multiple choice test question

```
14:00:48.095616 211.57.212.220.21 > x.y.z.2.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.118341 211.57.212.220.21 > x.y.z.3.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.158576 211.57.212.220.21 > x.y.z.5.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.161582 211.57.212.220.21 > x.y.z.1.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
14:00:48.163073 211.57.212.220.21 > x.y.z.4.21: SF [tcp sum ok]
799317639:799317639(0) win 1028 (ttl 21, id 39426, len 40)
```

The above trace indicates:
a) a slow scan
b) a noisy port scan
c) normal TCP connection establishment attempts
d) a synscan scan

The correct answer is d)

References

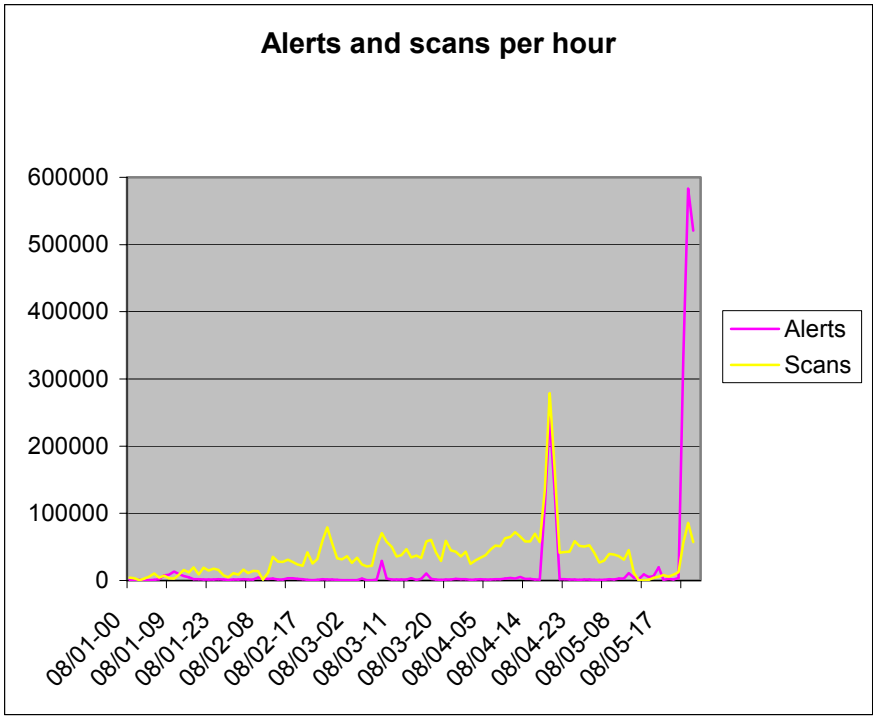
[1] Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Inc, 1994

[2] "List of fingerprints for passive fingerprint monitoring"
URL: http://project.honeynet.org/papers/finger/traces.txt (October, 4th)
[3] Smith, Donald. "Mscan, Sscan and Synscan - the evolution of worm - enabling vulnerability scanners that span two millenniums" URL: http://www.whitehats.ca/main/publications/external_pubs/scanner_fingerprints/scanner_fingerprints.html (October, 4th)
[4] Vision, Max. Ramen Internet Worm Analysis,
URL: http://www.whitehats.com/library/worms/ramen/index.html
[5] CIAC Information Bulletin, E-17: FTP Daemon Vulnerabilities, April 14, 1994
URL: http://ciac.llnl.gov/ciac/bulletins/e-17.shtml (October, 4th)
[6] CERT Advisory CA-2000-13: Two input validation Problems in FTPD, July 7, 2000 URL: http://www.cert.org/advisories/CA-2000-13.html (October, 4th)
[7] CERT Advisory CA-1999-03 FTP Buffer Overflows, February 11, 1999
URL: http://www.cert.org/advisories/CA-1999-03.html (October, 4th)

Part 3 – Analyze This

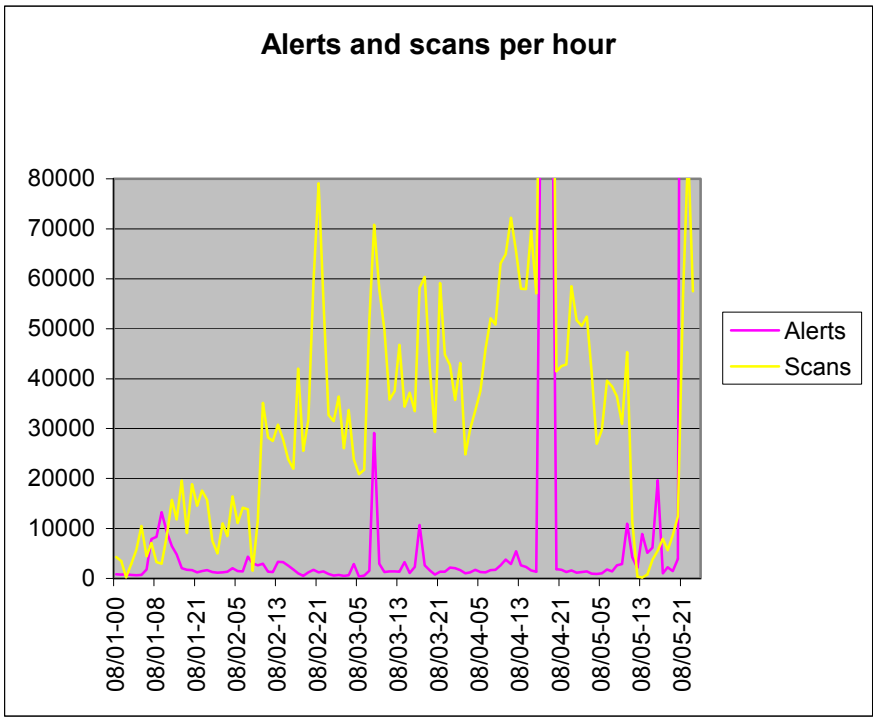
Executive Summary

This section contains the analysis of the anomalous traffic logs generated by a University during five consecutive days. Scan reports, alert logs and out-of-spec logs were analysed. The network topology and the ruleset used to generate the alerts were not available.
The amount of data collected was quite large over 500 MB uncompressed data including alerts, scans and oos). The analysis focused on the events and hosts that generated the largest number of alerts or scans.
The two graphs presented here show an overview of the events registered in the files examined for this report. They show the number of alerts and scans received on the network MY.NET.0.0/16 during the days from the first to the fifth of August 2002. The second graph is obtained by limiting the number of alerts to 80000 to eliminate the peaks and view the trend more clearly.



The graph shows a burst of activity related to NIMDA and CodeRed on the fourth and fifth days of the analysis. No particular trend can be associated to the day of the week (the two peaks occur on a Sunday and on a Monday).
Analysis of the alert files showed that some hosts have been compromised by CodeRed and NIMDA while there are some hosts with suspicious trojan activity. University hosts are very active in scanning outside hosts. Most of this activity is aimed at finding peer-to-peer servers (Gnutella, WinMx, AudioGalaxy, etc.) and at finding servers for online games (HalfLife, Starsiege Tribes, etc.).
We do not know where in the University network the sensor was located. There are quite extensive logs, but, in fact we do not know if all the inbound attempts (scans) have been successful or have been blocked by perimeter defenses. However, there are signs that little attention is paid to security at the hosts level (e.g. web servers compromise and trojan activity). Apparently little attention has been paid to securing

web servers against the vulnerabilities exploited by NIMDA and CodeRed (one year



old).
The following sections present the detailed analysis for the most recurring events. Analysis of the compromised machines and recommendations are given in each paragraph in relation to the alert or scan being analysed.

Data files used for analysis

The files used for this analysis were downloaded from http://www.incidents.org/logs :

Alert Files		Scans Files		OOS Files	
Filename	**Size**	**Filename**	**Size**	**Filename**	**Size**
alert.020801.gz	844,437	scans.020801.gz	1,344,265	oos_Aug.1.2002.gz	544
alert.020802.gz	1,069,475	scans.020802.gz	4,391,619	oos_Aug.2.2002.gz	35,863
alert.020803.gz	1,150,676	scans.020803.gz	6,595,155	oos_Aug.3.2002.gz	17,080
alert.020804.gz	9,358,581	scans.020804.gz	12,577,283	oos_Aug.4.2002.gz	205
alert.020805.gz	13,769,129	scans.020805.gz	4,940,845	oos_Aug.5.2002.gz	194

These log files contain alerts, scans and out of spec packets data for the days from the first to the fifth of August and represent a five day period including working days and the week-end. They were chosen because oos data were available for these days only at the time of starting this analysis.
These logs were generated by snort. The version, build and rulesets used to generate these logs are not known.

Total number of alerts (excluding the port scans entries)	2236962
Total number of scans	410204
Total number of OOS packets received	1637

The alerts files contained some spurious entries where more than one alert were "joined" together. Some information was duplicated in these lines (usually the description of the alert) while some information was missing (usually the last characters, and hence part of the destination address and port numbers).
As these entries could not be automatically processed like the others (duplicate alert descriptions, ports missing, etc.) they were deleted from the file processed for the analysis and saved into a separate file.

Total number of entries in the processed file (without the spurious entries)	2230007
Total number of entries in the "spurious entries file"	6955

Subsequently, the alert description of the spurious entries was examined in order to see whether they had an impact on the overall percentage of the various alerts counted in the processed alert file. None of the lines in the spurious entries file contained more than two alerts descriptions, on a theoretical maximum number or alerts equal to 13910, 4674 entries were related to the first most recurring alert, 2896 to the second, 1317 to the third, 986 to the fourth and 78 to the fifth. As a consequence, the data eliminated with this file does not impact the analysis on the "correct" data.

Analysis of the Alerts files

The following alerts were triggered by the activity on the University network during the 5 days examined:

Number of occurrences	Alert Description
874199	NIMDA - Attempt to execute cmd from campus host
492452	spp_http_decode: IIS Unicode attack detected
481125	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
122835	NIMDA - Attempt to execute root from campus host
106847	UDP SRC and DST outside network
53560	spp_http_decode: CGI Null Byte attack detected
30074	SMB Name Wildcard
24212	TFTP - External UDP connection to internal tftp server
14576	External RPC call
11916	Watchlist 000220 IL-ISDNNET-990517
4113	Possible trojan server activity
2543	SUNRPC highport access!
2053	IRC evil - running XDCC
1305	Watchlist 000222 NET-NCFC
1293	EXPLOIT x86 NOOP
1120	Queso fingerprint
927	SNMP public access
788	connect to 515 from outside
730	Attempted Sun RPC high port access
679	Samba client access
628	High port 65535 udp - possible Red Worm - traffic
314	IDS552/web-iis_IIS ISAPI Overflow ida nosize
260	ICMP SRC and DST outside network
236	SMB C access
173	TFTP - Internal UDP connection to external tftp server
166	beetle.ucs
147	Port 55850 tcp - Possible myserver activity - ref. 010313-1
136	Incomplete Packet Fragments Discarded
106	Null scan!
88	NMAP TCP ping!
58	EXPLOIT x86 setuid 0
53	Tiny Fragments - Possible Hostile Activity

48	EXPLOIT x86 stealth noop
44	High port 65535 tcp - possible Red Worm - traffic
42	STATDX UDP attack
38	EXPLOIT x86 setgid 0
18	Port 55850 udp - Possible myserver activity - ref. 010313-1
13	TCP SRC and DST outside network
13	SMB CD...
11	HelpDesk MY.NET.70.50 to External FTP
11	External FTP to HelpDesk MY.NET.70.50
11	MY.NET.30.4 activity
9	HelpDesk MY.NET.70.49 to External FTP
8	External FTP to HelpDesk MY.NET.70.49
6	TFTP - External TCP connection to internal tftp server
5	EXPLOIT NTPDX buffer overflow
4	HelpDesk MY.NET.83.197 to External FTP
3	RFB - Possible WinVNC - 010708-1
3	DDOS shaft client to handler
3	Back Orifice
2	Traffic from port 53 to port 123
2	SYN-FIN scan!
1	MY.NET.30.3 activity

Top 10 most occurring attacks

The most frequent attacks counting more than 10000 events recorded in the alerts file are indicated in the following table:

Attack Description	Number of occurrences	Percentage on total
NIMDA - Attempt to execute cmd from campus host	874199	39.2%
spp_http_decode: IIS Unicode attack detected	492452	22.0%
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	481125	21.5%
NIMDA - Attempt to execute root from campus host	122835	5.5%
UDP SRC and DST outside network	106847	4.8%
spp_http_decode: CGI Null Byte attack detected	53560	2.4%
SMB Name Wildcard	30074	1.3%
TFTP - External UDP connection to internal tftp server	24212	1.1%
External RPC call	14576	0.6%
Watchlist 000220 IL-ISDNNET-990517	11916	0.5%
Total percentage for the top 10 attacks		**98.9**

Top 10 most active hosts

The following table shows the list of hosts that generated the highest number of alerts. In this table and in the following ones, the fully qualified domain name was not available where not shown.

IP address	Fully Qualified Domain name	Alerts generated by this host	Number of alerts generated by the address	Percentage on the total number of alerts
MY.NET.100.208		1. NIMDA - Attempt to execute cmd from campus host 2. spp_http_decode: IIS Unicode attack detected 3. NIMDA - Attempt to execute root from campus host 4. TFTP - Internal UDP connection to external tftp server	1433246	64.3%

MY.NET.84.234		1. IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 2. Possible trojan server activity	481130	21.5%
3.0.0.99		UDP SRC and DST outside network	51359	2.3%
63.250.213.12	dal-qcwm213012.broadcast.com	UDP SRC and DST outside network	32115	1.4%
MY.NET.81.37		1. spp_http_decode: CGI Null Byte attack detected 2. spp_http_decode: IIS Unicode attack detected	27085	1.2%
194.98.189.139		1. External RPC Call 2. STATDX UPD attack	8375	0.4%
MY.NET.85.74		1. spp_http_decode: IIS Unicode attack detected 2. Possible trojan server activity	6990	0.3%
80.137.90.34	p50895A2B.dip.t-dialin.net	1. spp_http_decode: IIS Unicode attack detected 2. beetle.ucs	6898	0.3%
MY.NET.111.230		TFTP - External UDP connection to internal tftp server	6089	0.3%
MY.NET.111.231		TFTP - External UDP connection to internal tftp server	6059	0.3%
Total percentage of attacks from the top ten most active hosts				**92.3%**

Top 10 most targeted hosts

The following table shows the list of hosts against which the highest number of alerts was recorded. Interestingly enough, the most targeted hosts are outside the University network, indicating that most of alerts are generated by activity initiated on the University network or some network device misconfiguration. (10.0.0.1 is a private address and events involving this address are generated by 3.0.0.9, an address outside the University IP addresses range).

IP address	Fully Qualified Domain name	Alerts generated by activity addressed to this host	Number of alerting packets sent to this address	Percentage on the total number of alerts
10.0.0.1		UDP SRC and DST outside network	51359	2.3%
216.241.219.28		spp_http_decode: CGI Null Byte attack detected	39484	1.8%
233.28.65.148		UDP SRC and DST outside network	32115	1.4%
192.168.0.216		1. TFTP - External UDP connection to internal tftp server 2. spp_http_decode: IIS Unicode attack detected	24208	1.1%
233.2.171.1		UDP SRC and DST outside network	17945	0.8%
152.163.210.84		1. spp_http_decode: CGI Null Byte attack detected 2. spp_http_decode: IIS Unicode attack detected	6457	0.3%
233.28.65.173		UDP SRC and DST outside network	4975	0.2%
207.200.86.97	myns-v1.websys.aol.com	spp_http_decode: IIS Unicode attack detected	4758	0.2%
MY.NET.104.204		1. SMB Name Wildcard 2. Watchlist 000220 IL-	4492	0.2%

	ISDNNET-990517 3. NMAP TCP ping! 4. Null scan! 5. EXPLOIT x86 setuid 0 6. Incomplete Packet Fragments Discarded 7. High port 65535 udp - possible Red Worm – traffic 8. EXPLOIT x86 setgid 0 9. Queso fingerprint		
209.10.239.135	spp_http_decode: CGI Null Byte attack detected	3631	0.2%
Total percentage of attacks on the top ten most attacked hosts			**8.9**

233.28.65.173, 233.2.171.1 and 233.28.65.148 are multicast addresses. It is very likely that the traffic generated towards these hosts is due to the fact that the University is part of a multicast group.

Details on the most frequent alerts

The top 5 alerts account for more than 93% of the total alerts logged for the five days examined. All of them were reported more than 100000 times are analysed in this section. As we can see from the summary graph in the first section of this report, most of these alerts were generated on the 4th and 5th of August.

"NIMDA - Attempt to execute cmd from campus host"

This alert is not generated by a standard snort rule. The rule was probably defined by the University using the "content" rule option to specify the content of the payload as one of the strings reported below for the NIMDA worm containing the "cmd.exe". This event was reported 874199 times, all but 5 occurring on one single day (August 5th).
A sample of the event logged is provided below:

```
08/02-17:44:33.001172  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.70.16:2142 -> 65.54.250.121:80
08/03-19:55:53.607645  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.83.176:1345 -> 207.68.132.9:80
08/04-14:23:02.748399  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.111.30:1092 -> 207.46.235.150:80
08/04-14:45:06.913040  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.165.19:1085 -> 65.54.250.120:80
08/05-09:14:50.113555  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.70.169:1103 -> 65.54.250.121:80
08/05-13:22:36.967751  [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.70.144:1116 -> 207.46.235.150:80
```

These alerts are coming from 10 hosts only, all of them in the University's network. In fact almost all of them are coming from one host only, the other 9 generating only one alert each:

Number of alerts	Source IP address in the alert
874190	MY.NET.100.208
1	MY.NET.105.10
1	MY.NET.111.30
1	MY.NET.130.20
1	MY.NET.165.19
1	MY.NET.70.144

1	MY.NET.70.16
1	MY.NET.70.169
1	MY.NET.82.87
1	MY.NET.83.176

This alert is a known signature of the NIMDA worm.
NIMDA is a worm that started spreading last September 2001. It propagates via four distinct mechanisms and infects hosts running any version of the Windows Operating System. According to [2], NIMDA scans the Internet looking for web services and attempts to exploit a number of vulnerabilities of the Windows IIS software. Network attacks include exploitation of the "IIS/PWS Extended Unicode Directory Traversal Vulnerability" and of the "IIS/PWS Escaped Character Decoding Command Execution Vulnerability". NIMDA also exploits backdoors left behind by previous Code Red and sadmind infections (root.exe and mapping of C: and D: drives to virtual IIS folders which allow the execution of cmd.exe).
The following signatures are only some of those which indicate a NIMDA attack:

```
GET /scripts/root.exe?/c+dir
GET /MSADC/root.exe?/c+dir
GET /c/winnt/system32/cmd.exe?/c+dir
GET /d/winnt/system32/cmd.exe?/c+dir
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir
GET /_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
GET /_mem_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir
GET /msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../
winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
```

Clearly the alerts reported here have been triggered by packets containing this data in the payload. [2] also indicates that NIMDA targets IP classes with the same first octet with 25% probability. In the data examined for this report, over 100000 hosts have been attacked by host MY.NET.100.208. Once the worm gains access to a vulnerable IIS web server, it uses TFTP to download a copy of a file called Admin.dll from the infecting host to the vulnerable host. So, if this host is infected with NIMDA and is scanning for vulnerable hosts we could also see some TFTP connections from this host when the worm finds a vulnerable IIS server and attempts to install a copy of Admin.dll. The alert files contain a log of all attempt at connecting from the outside to internal tftp servers and from internal hosts to external tftp servers. If MY.NET.100.208, succeeded in infecting other hosts with NIMDA, we should see some alerts indicating a connection from external hosts to this host. However tftp connections between hosts in the University network seem not to be logged, so we don't know whether this host succeeded in infecting other hosts in the campus.
A detailed analysis about the spread of the worm from this host can be performed with the web server data. Host MY.NET.100.208 is clearly infected with the NIMDA worm, it should be disconnected from the network and "cleaned".

"spp_http_decode: IIS Unicode attack detected"

This alert is generated by the http_decode snort preprocessor. This preprocessor normalises HTTP requests by converting any Unicode character (denoted by %xx) into their ASCII equivalent. These alerts indicate that a request has been received by a web server which contained Unicode escaped characters in the URL. The Unicode

exploit uses malformed URLs, with a Unicode representation of the directory delimiter, to traverse directories and execute arbitrary commands on vulnerable web servers. This vulnerability is a variation of the 'dot dot' directory traversal attack. It affects unpatched Microsoft IIS 4.0 and 5.0 web servers. This vulnerability can be exploited by typing the malformed URL in the address bar of the web browser, however automated scripts exist that scan for vulnerable servers. The Unicode exploit is described in [1].
This alert is reported 492452 times of which 459102 are recorded on the 5th of August.
A sample of the events logged for this alert is provided below:

```
08/01-00:10:57.452228  [**] spp_http_decode: IIS Unicode attack detected [**]
64.86.155.118:2672 -> MY.NET.109.87:80
08/01-00:10:59.910726  [**] spp_http_decode: IIS Unicode attack detected [**]
64.86.155.118:2710 -> MY.NET.109.87:80
08/01-00:21:07.087017  [**] spp_http_decode: IIS Unicode attack detected [**]
211.91.255.154:51337 -> MY.NET.53.84:80
08/01-00:21:08.346841  [**] spp_http_decode: IIS Unicode attack detected [**]
211.91.255.154:51343 -> MY.NET.53.84:80
08/01-00:22:55.947869  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.183.25:4413 -> 64.12.42.116:80
08/01-00:22:55.947869  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.183.25:4413 -> 64.12.42.116:80
```

This event is reported most often as being generated by activity on these hosts:

Number of alerts	Source IP address in the alert
436058	MY.NET.100.208
6982	MY.NET.85.74
6888	80.137.90.34
2885	MY.NET.152.19
2826	MY.NET.153.145
2475	151.203.178.36
2003	MY.NET.153.168
1855	MY.NET.153.143
1642	MY.NET.91.103
1592	MY.NET.91.105

By examining closer the alerts with this description for the host most active, we discover that packets are addressed at more than 85000 different hosts, each of them receiving from 15 to 1 offending packets, they are in fact different fast scans. By examining the scans file, we discover that host MY.NET.100.208 accounts for more than 10% of the total number of scans.
This alert is a well-known signature of CodeRed (and its variants) and NIMDA. However, according to the snort FAQ, this alert can also be generated by "normal surfing" by some web browsers [3]. Combining this result with the observation for the previous alert we have further indications about the fact that host MY.NET.100.208 is infected with the NIMDA worm. This type of alert, generated by packets sent from this host, is also triggered by the NIMDA scan.
Rick Yuen and Joe Ellis also report this alert in their practicals and attribute it to CodeRed and Nimda.[14].

"IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize"
This alert is generated by a snort rule that is a slight modification of the following standard snort rule

[14] GIAC practicals are available at www.giac.org

```
alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype:
system-or-info-attempt; reference: arachnids,552;)
```

available from whitehats.com at the URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552&view=signatures, the modification being on the source of the offending packet (the internal network) which is reflected in the word "INTERNAL" added to the alert description. This alert is related to an unchecked buffer vulnerability in Microsoft IIS Index Server ISAPI Extension which could enable a remote intruder to gain SYSTEM access to the web server.
This event is reported as being generated by activity on one host only in the University network, all of them on the 4th of August.

Number of alerts	Source IP address in the alert
481125	MY.NET.84.234

A sample of the events logged for this alert is provided below:

```
08/04-17:30:00.336917  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**]
MY.NET.84.234:4736 -> 62.58.155.117:80
08/04-17:30:00.344443  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**]
MY.NET.84.234:4737 -> 160.193.184.87:80
08/04-17:30:00.369439  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**]
MY.NET.84.234:4740 -> 188.146.27.103:80
08/04-17:30:00.374757  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**]
MY.NET.84.234:4741 -> 80.119.211.169:80
08/04-17:30:00.381321  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize [**]
MY.NET.84.234:4742 -> 148.27.15.207:80
```

This alert is a signature for CodeRed. CodeRed exploits the Index Server (.ida) buffer overflow vulnerability reported in CIAC Bulletin L-098 [5]. The buffer overflow allows the worm to execute code within the IIS server to deface the server's home page, to compromise other vulnerable hosts and to run a denial of service attack on www.whitehouse.gov. The worm arrives at the web server as a GET /default.ida request. This request exploits the .ida vulnerability and starts the execution of the worm code. Hosts infected by these worms scan port 80, looking for other web servers with the same vulnerability to infect. Like MY.NET.100.208, this host too should be disconnected from the network and "cleaned". Joe Ellis reports this alert in his practical and attributes it to CodeRed and Nimda.

"NIMDA - Attempt to execute root from campus host"
This alert is not generated by a standard snort rule. The rule was probably defined by the University using the "content" rule option to specify the content of the payload as one of the strings reported above for the NIMDA worm containing the "root.exe" (discussion on alert "NIMDA - Attempt to execute cmd from campus host").
This event was reported 122835 times all occurring on the same day (August 5th).
A sample of the event logged is provided below:

```
08/05-21:21:55.661920  [**] NIMDA - Attempt to execute root from campus host [**]
MY.NET.100.208:2008 -> 130.95.40.191:80
08/05-21:21:55.664339  [**] NIMDA - Attempt to execute root from campus host [**]
MY.NET.100.208:2010 -> 130.7.64.55:80
08/05-21:21:55.670339  [**] NIMDA - Attempt to execute root from campus host [**]
MY.NET.100.208:2009 -> 130.178.180.123:80
```

```
08/05-21:21:55.670567  [**] NIMDA - Attempt to execute root from campus host [**]
MY.NET.100.208:2011 -> 130.91.203.243:80
08/05-21:21:55.677068  [**] NIMDA - Attempt to execute root from campus host [**]
MY.NET.100.208:2015 -> 130.62.62.95:80
```

This event is reported by activity generated by a unique host:

Number of alerts	Source IP address in the alert
122835	MY.NET.100.208

This is the same host that generated alert "NIMDA - Attempt to execute cmd from campus host" and is part of the same scanning.

"UDP SRC and DST outside network"

This alert is reported 106847 more or less evenly distributed in all the five days examined.
A sample of the events logged for this alert is provided below:

```
08/01-00:00:05.732948  [**] UDP SRC and DST outside network [**] 3.0.0.99:137 ->
10.0.0.1:137
08/01-00:00:08.733589  [**] UDP SRC and DST outside network [**] 3.0.0.99:137 ->
10.0.0.1:137
08/01-00:00:11.738004  [**] UDP SRC and DST outside network [**] 3.0.0.99:137 ->
10.0.0.1:137
08/01-00:00:13.240134  [**] UDP SRC and DST outside network [**] 3.0.0.99:137 ->
10.0.0.1:137
08/01-00:00:14.742414  [**] UDP SRC and DST outside network [**] 3.0.0.99:137 ->
10.0.0.1:137
```

This alert is generated because UDP traffic is logged with both source IP and destination IP address outside the network. This kind of traffic could be generated by a mis-configured network device or by a mis-configured snort that does not include all local networks in HOME_NET.
Ports in these alerts are mostly 137 (Windows Netbios Name Service). Most of this traffic is addressed to a private address (10.0.0.1). The other port in the traffic that raised this alert that is also worth noting is port 54646. All traffic having 54646 as destination port was addressed to the same address 233.2.171.1 which is a multicast address. UDP port 54646 and IP address 233.2.171.1 are associated to the Multicast Beacon, an active measurement software that monitors the performance of a multicast session. [4]. 233.28.65.148 and 233.28.65.173 are also multicast addresses. According to the Internet Multicast Addresses list published by IANA [7], the address block 233.0.0.0-233.255.255.255 is assigned to GLOP block.
The configuration of snort should be revised to verify that all local subnets are included. Egress filtering should be applied at the perimeter of the University network, incoming traffic with the same addresses as local addresses should be denied, as well as outgoing traffic with source address different from local addresses. Local area network services, like Netbios Name Service should not be allowed into the University network from the outside.
Similar activity was noted in the practical from Andrew Windsor, however, his case, all traffic that triggered this alert was addressed to a multicast address and the port number was associated to a multicast service, both different from the multicast address and port noted here.

Analysis of the scans files

The scan activity is more "regular" than the alert activity during the examined days, in that there are no peaks like those reported in the alerts.
Most of the scan activity is SYN and UDP scans. Examination of the scan files was useful to understand some of the activity going on in the university network. In particular, scans for ports associated to peer-to-peer services and to online games indicate that this type of activity is allowed and frequent.

Type of scan	Occurrences
UDP Scans	3112461
SYN Scans	996371
SYN Scans with the reserved bits set	1151
Other types of scans	198
Scans initiated from internal addresses towards external addresses	1023378
Scans initiated from external addresses towards internal addresses	200560

As a general comment we can notice that scans initiated from the University network to external hosts are more numerous than scans on the University network, indicating that University users are quite active in scanning. Surprisingly enough, no scans are logged from University hosts to other hosts in the same net.
Not only the most active port scanner is an internal address (MY.NET.70.200 with 2437159 scans) but also the ten most active scanners are all internal hosts.
The table below shows the most active scanners and the most scanned ports by them.

Internal "scanner" IP address	Number of scans from this address	Ports scanned by this address
MY.NET.70.200	2439514	80/tcp (HTTP)and ports in the range 41000-41300 (audiogalaxy satellite MP3 sharing application)
MY.NET.84.234	478408	80/tcp (HTTP), (this is the host that raised the "web-iis_IIS ISAPI Overflow ida" alert)
MY.NET.100.208	170345	80/tcp (HTTP), 3722/udp, 3269/udp (IBM Dial Out), 3267/udp (Microsoft Global Catalog with LDAP/SSL)
MY.NET.70.207	137226	11492 different UDP ports
MY.NET.82.2	127725	12528 different UDP ports
MY.NET.165.24	104553	Various UDP ports, mainly 6257/udp 16257/udp and 6699/udp (WinMX peer-to-peer)
MY.NET.83.150	90049	Various UDP ports, mainly 6257/udp 16257/udp and 6699/udp (WinMX peer-to-peer)
MY.NET.137.7	49208	Various UDP ports, 80/tcp (HTTP), 25/tcp (SMTP), 53/udp (DNS)
MY.NET.70.133	42744	22/tcp (SSH), 53/udp (DNS), 7000,7001,7002,7003,7004/udp (AFS)
MY.NET.81.27	31926	Various UDP ports, 28800/udp (On line game Starsiege tribes)

If we exclude hosts MY.NET.84.234 and MY.NET.100.208, whose scanning activity is mostly related to the NIMDA and CodeRed worms, and MY.NET.70.200, MY.NET.165.24, MY.NET.83.150, MY.NET.81.27 whose scanning activity is mostly related to the search for peer-to-peer hosts for file sharing or online games servers, the remaining hosts whose activity is worth commenting are MY.NET.137.7 and MY.NET.70.133. The activity of MY.NET.137.7 is mostly related to 25/tcp and

53/udp, which indicates that this host might be a mail server opening many frequent 25/tcp connections and 53/udp connections to send messages. In fact, analysing the 53/udp connections we discover that they are addressed to DNS servers (among them 192.5.6.30-a.gtld-servers.net, 192.52.178.30 - k.gtld-servers.net, 198.133.199.100-arrowroot.arin.net) while 25/tcp connections were addressed to mail servers (among them 208.45.133.107-xmxpita.excite.com, 65.54.254.129- mc1.law16.hotmail.com, 64.157.4.89-mta580.mail.yahoo.com, 204.127.134.23- mtiwgwc14.worldnet.att.net, etc.).[15]
The scanning activity of external hosts towards the University network is much lower compared to the scans generated from internal hosts:

External "scanner" IP address	Number of scans from this address	Ports scanned by this address
216.228.171.81	25940	445/tcp, 139/tcp, 137/tcp (Netbios and Windows DS)
24.138.61.171	21019	80/tcp (HTTP)
161.132.205.100	20330	80/tcp (HTTP)
211.232.192.153	17730	1433/tcp (MS SQLServer)
67.104.84.142	16264	1433/tcp (MS SQLServer)
219.96.171.20	15741	80/tcp (HTTP)
80.137.90.34	15693	80/tcp (HTTP)
24.101.152.5	12593	21/tcp (FTP)
202.98.223.86	10739	80/tcp (HTTP)
66.224.37.26	10139	80/tcp (HTTP)

A part from the 80/tcp scans that can be related most frequently to NIMDA and CodeRed worms, which have already been discussed in the previous sections, it is worth noting that most of the scans are addressed to 1433/tcp and 21/ftp. Ftp vulnerabilities, the search for which can be the cause for the ftp scans have already been discussed in part 2 of this practical. Port 1433/tcp is used by Microsoft SQL Server. Some serious vulnerabilities were discovered in this product which allow remote attackers to compromise the database server, to alter the database content and in some cases to compromise the server host too. SQL Server's vulnerabilities are third in the list of MS Windows related vulnerabilities in [6] and, according to [6] these vulnerabilities are well publicized and actively under attack.

Most scanned ports by University hosts		**Most scanned ports on University hosts**	
Number of occurrences	***Port and service***	***Number of occurrences***	***Port and service***
581292	80/tcp, http	94382	80/tcp, http
133651	41170/udp Audiogalaxy	44910	1433/tcp, MS SQLserver
78959	6257/udp WinMx Peer-to-peer tool	22980	21/tcp
2894	6346/udp|tcp Gnutella	9428	111/tcp SUN Remote Procedure Call
2756	53/udp DNS – host receiving most of these requests is a DNS server (192.5.6.30-a.gtld-servers.net). This might be legitimate activity	5261	139/tcp Netbios Session Service
1173	139/tcp Netbios	5204	445/tcp Microsoft DS
1028	25/tcp SMTP, host receiving	5199	23/tcp Telnet

[15] DNS servers were verified by using nslookup and setting the host addressed by these scans on port 53 as the server. SMTP servers were verified by connecting on TCP port 25 using telnet.

	most of these requests is a mail server (see discussion above)		
931	28800/udp On line game Starsiege tribes	2225	13000/tcp SennaSpy Trojan Generator (on udp!)
885	12300/udp MOHAA GameSpy Monitoring Port	2172	6112/tcp and udp Starcraft/Blizzard Battlenet (udp)
870	27020/udp Halflife	643	137/udp Netbios Name Service

In addition to 1433/tcp and 21/tcp a frequently scanned port is 111/ucp-tcp (RPC). Remote Procedure Call (RPC) is the first in the SANS [5] list of the top vulnerabilities for Unix systems. RPCs allow a program on one host to run a procedure on another host, they are used in many distributed network services, such as network file sharing or remote administration. Recently many vulnerabilities have been found in many RPC services (e.g. rpc.mountd, rpc.statd, sadmind, cachefsd, etc.).
Scans types other than SYN and UDP are reported in the table below. They are invalid combinations of the TCP flags meant at being stealthy:

Number of occurrences	"name" and flag combination	Description found in the scans file
61	VECNA ****P***	Series of stealth scans, coded in nmap by vecna (http://online.securityfocus.com/archive/1/42136)
59	NULL ********	Null scan: none of the TCP flags is set.
26	INVALIDACK ***A*R*F	Ack, Reset and Fin flags are set, Ack is invalid
7	NOACK *****RS*	Reset and Syn flags are set
6	INVALIDACK ***A*RS*	Ack, Reset and Syn flags are set Ack is invalid
5	NOACK **U**RS*	Urgent, Reset and Syn flags are set
4	XMAS **U*P**F	Subset of the Christmas scan, abbreviated in Snort as XMAS
4	VECNA **U*P***	Urgent and Push flags are set
3	NOACK **U**R*F	Urgent, Reset and Fin flags are ser
3	INVALIDACK **UA*RS*	Urgent, Ack, Reset and Syn flags are set
3	FIN *******F	Fin flag is set
2	NOACK **U**R**	Urgent, and Reset flags are set
2	INVALIDACK ***AP*S*	Ack, push and Syn flags are set
2	FULLXMAS **UAPRSF	The combination of all TCP flags set is known as "Christmas Tree", abbreviated in Snort as FULLXMAS:
1	VECNA **U****F	Urgent and Fin flags are set
1	VECNA **U*****	Urgent flag is set
1	NOACK ****PRS*	Push, Reset and Syn flags are set
1	NOACK ****PR*F	Push, Reset and Fin flags are set
1	NOACK *****R*F	Reset and Fin flags are set
1	INVALIDACK **UAPRS*	Urgent, Ack, Push, Reset and Syn flags are set
1	INVALIDACK **UAPR*F	Urgent, Ack, Push, Reset and Fin flags are set
1	INVALIDACK **UA*RSF	Urgent, Ack, Reset, Syn and Fin flags are set
1	INVALIDACK **UA*R**	Urgent, Ack and Reset flags are set
1	INVALIDACK ***APRSF	Ack, Push, Reset, Syn and Fin flags are set
1	INVALIDACK ***A**SF	Ack, Syn and Fin flags are set

Scans for 1433, 139, 111 and 21 are clearly meant at finding vulnerabilities in these services. Disable access to these services from outside whenever this service is not meant for public access. If SQL servers and ftp servers must be accessed from the outside, check that all patches have been applied. Netbios services and RPC should

not be accessed from the outside. Although not malicious in nature, scans for peer-to-peer sites and online games servers should be investigated (copyright issues for exchanged material and misuse of the network bandwidth).

Analysis of the OOS files

There were 1537 out of spec packets in the oos files. All these packets are generated from external sources only.
The following table indicates the most active hosts in sending out of spec packets:

IP address	Fully qualified domain name	Number of OOS packets	Destination address and port
68.32.126.64	pcp01823532pcs.howard01.md.comcast.net	652	MY.NET.6.7, 110/tcp
62.76.241.129	clamas.uni.udm.ru	345	MY.NET.97.217, MY.NET.96.238, 113/tcp
209.116.70.75	vger.kernel.org	214	Mostly MY.NET.100.217, 25/tcp
212.35.180.17		83	MY.NET.253.20, 21/tcp
65.210.154.210		48	MY.NET.111.198, 4662/tcp (edonkey, file sharing program).

Interestingly enough, all packets received from these hosts have only the two reserved bits set, and the same window size 0x16d0.
Packets from 209.116.70.75 were addressed to several hosts, but MY.NET.100.217 was the most recurring one (95 occurrences). The packets have the same characteristics as the others and were sent on 1st and 2nd of August. This host is also logged in the scans and in the alert file as the source address in the packets that raised the queso alert, probing port 25. These packets too are addressed at port 25.

An excerpt of the packets logged in the oos file, related to this host, is provided below:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/01-01:51:09.333121 209.116.70.75:46541 -> MY.NET.100.217:25
TCP TTL:51 TOS:0x0 ID:40167  DF
21S***** Seq: 0xA7D51545   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 770738594 0 EOL EOL EOL EOL
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/01-01:59:25.542188 209.116.70.75:49223 -> MY.NET.100.217:25
TCP TTL:51 TOS:0x0 ID:38528  DF
21S***** Seq: 0xC766DDF6   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 770788214 0 EOL EOL EOL EOL
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/01-02:09:21.062675 209.116.70.75:55608 -> MY.NET.100.217:25
TCP TTL:51 TOS:0x0 ID:9764  DF
21S***** Seq: 0xEDA21233   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 770847767 0 EOL EOL EOL EOL
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Using the p0f fingerprint database, and assuming that the host is 13 hops away (it is a likely figure) we can see that this host is most likely a Linux 2.4.2 or 2.4.1-14 box (Window size=5840, initial TTL=64, SackOK, MSS=1460, DF bit set). There are no clear sign of packet crafting. A SYN packet with both the last two reserved bits set

(CWR and ECN-echo flags) is a signature for the queso port scanner. In fact this address is also present in the alerts file under the description related to queso fingerprinting. However, it is still possible that these two bits have been set by the sending host to indicate congestion. According to [8], in order to distinguish between a queso SYN packets with both reserved bits set from a legitimate use of the ECN fields, we should have a look at the following packets and check whether the TOS field in the IP header is set, which would indicate that the SYN packet with the reserved bits set was a legitimate packet. Unfortunately we do not have here the following packets for further analysis.

Moving on from the "top talkers" in terms of OOS packets, we can have a look at the other types of OOS packets received, which are summarised in the table below:

Number of occurrences	TCP flags combination	Number of occurrences	TCP flags combination
1604	21S*****	1	21S**P**
2	21S*R***	1	21S***AU
2	21S***A*	1	21*F****
2	21*FRPAU	1	21**RPAU
2	21*FR***	1	2*SFRPA*
2	2*SFR**U	1	2*SFR*A*
2	2*SF****	1	2*SF**A*
1	21SFRPAU	1	*1SFR***
1	21SFR*AU	1	*1SF*P*U
1	21SFR*A*	1	*1SF**AU
1	21SFR**U	1	*1SF**A*
1	21SF*P*U	1	**SFRPAU
1	21S*R*AU	1	**SFR*A*
1	21S**PAU	1	**SF***U

A part from the illegal combination of TCP flags, it is difficult to make further analysis on these packets (for instance on the variation of the sequence number or the IP ID), since in most cases there is only one packet per each combination. So each of these packets would deserve the same right to be examined carefully.
An excerpt of some of these packets is shown below:

```
07/31-19:44:50.801909 209.163.19.41:43028 -> MY.NET.88.162:51450
TCP TTL:106 TOS:0x0 ID:55554  DF
21S*R*** Seq: 0xD09E0BD9   Ack: 0xE04ABC0A   Win: 0xC50A
TCP Options => EOL EOL EOL EOL EOL EOL SackOK
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
08/01-06:13:00.731738 211.154.85.159:1893 -> MY.NET.111.140:80
TCP TTL:107 TOS:0x0 ID:59605  DF
21S*R*** Seq: 0x20DB060   Ack: 0x94F7   Win: 0x5010
TCP Options => EOL EOL EOL EOL EOL EOL
```

EOL means "End of Options List" it is used to indicate where the receiving end should stop processing the options list. What is strange in these packets, a part from the weird combination of SYN and RST flags (SYN="I want to initiate a connection", RST="I want to terminate abruptly this connection"), is that a number of EOL are specified (isn't one sufficient?). Moreover, in the second packet EOL is useless since there are no options, while in the first packet the only option (sackOK) would not be processed by a correct TCP implementation (which is to ignore options specified after EOL).

External source addresses information

Scanning and more importantly stealth scanning are often the first step in launching an attack. Therefore it is useful to keep an eye on those hosts that are attempting to scan the University network more often or extensively.

Whois query for: 209.116.70.75 (queso scanning and OOS packets-possibly a false positive) – This IP appears 57 times in the Dshield database

```
CustName:   Red Hat, Inc.
Address:    4518 South Miami Blvd. Suite #100 Durham NC 27703
Country:    US
RegDate:    2002-09-23
Updated:    2002-09-23

NetRange:   209.116.70.64 - 209.116.70.95
CIDR:       209.116.70.64/27
NetName:    INFLOW-18773-5591
NetHandle:  NET-209-116-70-64-1
Parent:     NET-209-116-68-0-1
NetType:    Reassigned
Comment:
RegDate:    2002-09-23
Updated:    2002-09-23
```

Whois query for 148.64.21.23 (This host is scanning hosts in the University network using the VECNA combination of TCP flags) – This IP does not appear in the Dshield database

```
OrgName:    Spacenet, Inc.
OrgID:      SPAN
NetRange:   148.62.0.0 - 148.78.255.255
CIDR:       148.62.0.0/15, 148.64.0.0/13, 148.72.0.0/14, 148.76.0.0/15,
148.78.0.0/16
NetName:    SPACENET-SPAN
NetHandle:  NET-148-62-0-0-1
Parent:     NET-148-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1-MCL.STARBAND.COM
NameServer: NS2-MCL.STARBAND.COM
NameServer: NS1-MAR.STARBAND.COM
NameServer: NS2-MAR.STARBAND.COM
Comment:
RegDate:    2000-05-31
Updated:    2001-07-26

TechHandle: FM173-ARIN
TechName:   Miller, Fred
TechPhone:  +1-703-848-1108
TechEmail:  fred.miller@spacenet.com
```

Whois query for 218.47.166.219 (This host is scanning hosts in the University network using NULL Scan) - This IP does not appear in the Dshield database

```
inetnum:    218.40.0.0 -
218.47.255.255
netname:    JPNIC-NET-JP
descr:      Japan Network Information
Center
country:    JP
admin-c:    JNIC1-AP
tech-c:     JNIC1-AP
rev-srv:    ns0.nic.ad.jp

rev-srv:    ns.wide.ad.jp
```

```
inetnum:    218.47.164.0 -
218.47.255.255
netname:    PLALA
descr:      Plala Networks Inc.

country:    JP
admin-c:    MN2905JP
tech-c:     HS3694JP
remarks:    This information has been
partially mirrored by APNIC from
remarks:    JPNIC. To obtain more
specific information, please use the
```

```
rev-srv:    ns0.iij.ad.jp

rev-srv:    dns0.spin.ad.jp

rev-srv:    ns-jp.sinet.ad.jp

rev-srv:    ns-jp.ntt.net

mnt-by:     APNIC-HM

mnt-lower:  MAINT-JPNIC

changed:    hostmaster@apnic.net
20010531
status:     ALLOCATED PORTABLE

source:     APNIC
```

```
remarks:    JPNIC whois server at
whois.nic.ad.jp. (This defaults to
remarks:    Japanese output, use the
/e switch for English output)
changed:    apnic-ftp@nic.ad.jp
20020129
remarks:    This information has been
partially mirrored by APNIC from
remarks:    JPNIC. To obtain more
specific information, please use the
remarks:    JPNIC whois server at
whois.nic.ad.jp. (This defaults to
remarks:    Japanese output, use the
/e switch for English output)
changed:    apnic-ftp@nic.ad.jp
20020925
source:    JPNIC
```

Whois query for 3.0.0.9 (this host is generating quite a number of alerts due to the attempted UDP communications with host 10.0.0.1-an address in the private range) - This IP does not appear in the Dshield database

```
OrgName:    General Electric Company
OrgID:      GENERA-9
NetRange:   3.0.0.0 - 3.255.255.255
CIDR:       3.0.0.0/8
NetName:    GE-INTERNET
NetHandle:  NET-3-0-0-0-1
Parent:
NetType:    Direct Assignment
NameServer: ns.ge.com
NameServer: ns1.ge.com
NameServer: ns2.ge.com
Comment:
RegDate:    1988-02-23
Updated:    2002-09-26

TechHandle: GET2-ORG-ARIN
TechName:   General Electric Company
TechPhone:  +1-518-612-6672
TechEmail:  genictech@ge.com
```

Whois query for 211.232.192.153 (this host is looking for vulnerable MS SQLServers) – This IP appears 2001 times in the Dshield database

```
inetnum:    211.232.192.0 -
211.232.192.255
netname:    CABLELINE-CATV-KR
descr:      BANDOCABLELINE
descr:      906-3 Inhu1dong Dukjin-ku
descr:      CHONBUK
descr:      561-230
country:    KR

admin-c:    JJ2128-KR
tech-c:     BK1504-KR
remarks:    This IP address space has
been allocated to KRNIC.
remarks:    For more information,
using KRNIC Whois Database
remarks:    whois -h whois.nic.or.kr
mnt-by:     MNT-KRNIC-AP
remarks:    This information has been
partially mirrored by APNIC from
remarks:    KRNIC. To obtain more
specific information, please use the
remarks:    KRNIC whois server at
```

```
country:    KR

phone:      +82-63-900-9000
fax-no:     +82-63-900-9000
e-mail:     catv@catvnet.co.kr
nic-hdl:    JJ2128-KR
mnt-by:     MNT-KRNIC-AP
changed:    hostmaster@nic.or.kr
20020923
source:     KRNIC



person:     Byungduk Kim

descr:      BANDOCABLELINE
descr:      906-3 Inhu1dong Dukjin-ku
descr:      CHONBUK

descr:      561-230

country:    KR
```

whois.krnic.net. changed: hostmaster@nic.or.kr 20020923 source: KRNIC person: Jehong Jung descr: BANDOCABLELINE descr: 906-3 Inhu1dong Dukjin-ku descr: CHONBUK descr: 561-230	 phone: +82-63-900-9000 fax-no: +82-63-900-9000 e-mail: ip@cableline.com nic-hdl: BK1504-KR mnt-by: MNT-KRNIC-AP

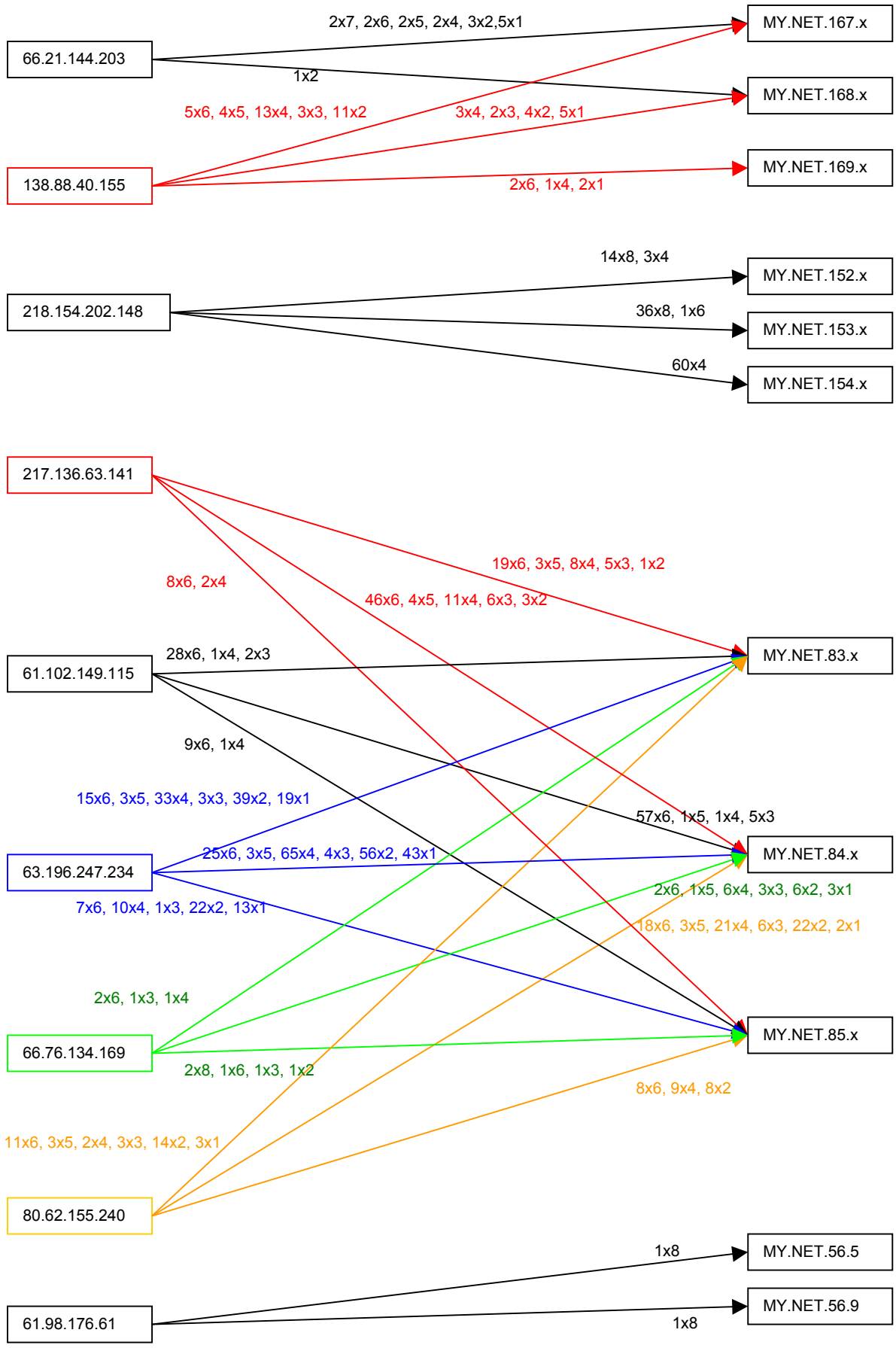
Link graph

The alerts file indicates that there is some suspicious activity that would suggest that some hosts are running Trojan servers. We do not know exactly what the rule that records the alert checks but we can see packets from an ephemeral port to port 27374 and back. If we identify a connection with the quadruple: (source IP, source port, destination IP, destination port), where the destination port is always 27374 we can build a diagram which shows these connections. 27374 is a port associated with SubSeven and its variants, to the Ramen worm and to a number of other different trojans.
The diagram indicates the originating hosts on the left side and the hosts that "listen" on port 27347 on the other side. "listening" hosts are concentrated on a number of subnets of the campus net, in some cases the number of hosts involved in this activity is quite high. The arrows in the diagram indicate the direction of the connection (the initiator) and the numbers (a x b) indicate the number of connections ("a") towards hosts in the subnet. For each of these connections (a connection is the quadruple defined above) the transit of "b" packets is logged in the alerts file (colors are used to differentiate packets and connections coming from different hosts).
External hosts that initiate these connections are:

66.21.144.203	adsl-21-144-203.mia.bellsouth.net
138.88.40.155	pool-138-88-40-155.res.east.verizon.net
217.136.63.141	141.63-136-217.adsl.skynet.be
218.154.202.148	none found
61.102.149.115	none found
63.196.247.234	adsl-63-196-247-234.dsl.lsan03.pacbell.net
66.76.134.169	cdm-66-134-169-newp.cox-internet.com
80.62.155.240	0x503e9bf0.odnxx4.adsl-dhcp.tele.dk
61.98.176.61	none fount

All of them come from dial-up/ADSL lines. In some cases the activity (1 packet to many different hosts) looks like a scan, in other cases, the record of up to 8 packets on the same connection indicates that some activity worth more investigations on the involved hosts is going on.
The link graph in the following page was obtained by identifying the connections in the alert file and eliminating all those connections that, even though logged as suspicious trojan activity, were related to other activities involving use of the port 27374 as source port. What is interesting in this graph is the clustering around subnets MY.NET.83.x, MY.NET.84.x and MY.NET.85.x, which seem to be targeted by many different hosts located worldwide. It would be interesting to investigate whether in the past days there has been any scanning activity aimed at port 27374 from any of this

host which could have taken advantage of a trojan previously installed. Definitely these hosts should be removed from the network and investigated.


66.21.144.203
138.88.40.155
218.154.202.148
217.136.63.141
61.102.149.115
63.196.247.234
66.76.134.169
80.62.155.240
61.98.176.61
MY.NET.167.x
MY.NET.168.x
MY.NET.169.x
MY.NET.152.x
MY.NET.153.x
MY.NET.154.x
MY.NET.83.x
MY.NET.84.x
MY.NET.85.x
MY.NET.56.5
MY.NET.56.9
2x7, 2x6, 2x5, 2x4, 3x2,5x1
1x2
5x6, 4x5, 13x4, 3x3, 11x2
3x4, 2x3, 4x2, 5x1
2x6, 1x4, 2x1
14x8, 3x4
36x8, 1x6
60x4
19x6, 3x5, 8x4, 5x3, 1x2
8x6, 2x4
46x6, 4x5, 11x4, 6x3, 3x2
28x6, 1x4, 2x3
9x6, 1x4
15x6, 3x5, 33x4, 3x3, 39x2, 19x1
57x6, 1x5, 1x4, 5x3
25x6, 3x5, 65x4, 4x3, 56x2, 43x1
7x6, 10x4, 1x3, 22x2, 13x1
2x6, 1x5, 6x4, 3x3, 6x2, 3x1
18x6, 3x5, 21x4, 6x3, 22x2, 2x1
2x6, 1x3, 1x4
2x8, 1x6, 1x3, 1x2
8x6, 9x4, 8x2
11x6, 3x5, 2x4, 3x3, 14x2, 3x1
1x8
1x8

Defensive recommendations

In addition to the recommendation given in the previous sections, as general recommendations:

- Remove the infected hosts (worms and trojans) from the network, investigate the incidents analysing the logs and reinstall;
- Develop a procedure (for each operating system used) to automatically install "hardened" version of the OS on all hosts in the campus network;
- Use egress filtering to filter outbound packets; ensure antispoofing is enabled on all network devices;
- Develop a policy to determine whether or not peer-to-peer sharing application and online gaming should be allowed;
- Filter traffic that should be allowed only on a LAN/campus network basis (netbios, lpd, NFS, RPC);
- Do not install any service unless needed (ftp, http should not be allowed on all machines, many hosts showed signs of being compromised via various web exploits; use of tftp should be investigated);
- Monitor security bulletins and keep up to date with patches and fixes;
- Update regularly the snort (or any other IDS) rule base;

Analysis Process

Initially I intended to process the alert files using SnortSnarf to get an overall picture of the alerts triggered by the activity on the University network on the five days I examined. However, the amount of data was so huge that SnortSnarf was not able to process it. So I used only Unix (Solaris 8 and Linux) commands to process the data. Firstly, I joined all the alert files, the scans files and the oos files into one file only using `cat`. Then, I processed the data using `sed` to obtain a file in a ";" delimited fields format. This was done for the alerts file, for the scans file and for the oos file. Subsequently I did all the searches and processing using `nawk`, `sort`, `cut`, `uniq` and `egrep`.

The graphs showing trends in the number of alerts and scans were made using MS excel.

References

[1] "Microsoft IIS Unicode exploit", URL: www.lucent.com/livelink/197020_Whitepaper.pdf (Oct 4th)

[2] "NIMDA Worm/Virus Report – Final", October 3, 2001, URL: http://www.incidents.org/react/nimda.pdf (Oct 4th)

[3] "Snort FAQ", URL: http://www.snort.org/docs/faq.html (Oct 4th)

[4] Multicast Beacon *Server* v0.8.X (Perl), URL: http://dast.nlanr.net/Projects/Beacon/ (Oct 4th)

[5] "CIAC Bulletin L-098, Microsoft Index Server ISAPI Extension Buffer Overflow", URL: http://www.ciac.org/ciac/bulletins/l-098.shtml (Oct 4th)

[6] "The Twenty Most Critical Internet Security Vulnerabilities (Updated) ~ The Experts' Consensus", October 1, 2002, http://www.sans.org/top20/ (Oct 4th)

[7] "Internet Multicast Addresses – Last updated 2002-05-02". URL: http://www.iana.org/assignments/multicast-addresses (Oct 4th)

[8] Miller, Toby. "ECN and it's impact on Intrusion Detection" URL: http://www.sans.org/y2k/ecn.htm (Oct 5th)