Intrusion Detection in Practice

GCIA Practical Assignment
Version 3.3

Ernest Eustace
October 23 rd 2002

SANS – Ontario

May 13 – 18th, 2002
Toronto, ON, Canada

Table of Contents

Assignment 1 – "The State of Intrusion Detection" **3**
An Introduction to Nikto – A Handy Web Scanner 3
"Gort! Klaatu, Barada, Nikto" *3*
Obtaining Nikto *3*
Installation *4*
Usage and Sample Traffic *4*
Detecting Nikto at Work *6*
Conclusion *8*
References for Assignment 1 8

Assignment 2 – "Network Detects" **9**
Detect 1: Possible Scan by MS SQL Server Worm. 9
Detect 2: ICMP Host Unreachable – Administratively Prohibited 17
Detect 3: 'WEB-IIS ..\.. access' is it Nimda?. 26

Assignment 3 – "Analyze This" **35**
Executive Summary 35
Data Used in Analysis 35
Discussion of Detects 35
Most Frequent and False Positives *36*
Bandwidth Usage *41*
Web and FTP *47*
Virus or Worm *55*
Scan Logs *58*
OOS Logs *63*
Concluding Remarks 68
Analysis Methodology 69

References **71**

Assignment 1 – " The State of Intrusion Detection "

An Introduction to Nikto – A Handy Web Scanner

"Gort! Klaatu, Barada, Nikto"
That is a line from the 1951 classic science fiction movie "The Day the Earth Stood Still" that inspired the author of Nikto to name it what he did. Perhaps at the time of the movies making Nikto and the reason for its existence might have sounded more outlandish than their plot. Over a relatively short period of time telecommunications technology has grown in leaps and bounds. The most visible of these advances has been the growth of the World Wide Web and the underlying Internet. However as with all human endeavors, where there is success there will also be those who wish to prey upon that success.

This advance in technology and especially the ever-increasing accessibility to it has produced a new generation of troublemakers and vandals. One no longer sees the media pay attention to the youth spray-painting graffiti on subway walls, but instead it is the "script kiddies" vandalizing corporate web sites. Far more serious though are the professional criminals, for example those harvesting credit card numbers for resale from unprotected online stores. In this kind of environment those who are responsible for the security of these systems have to be ever vigilant. This is where Nikto comes in to the picture; it is a tool for performing vulnerability tests against web servers. Although it is a great tool for administrators, it can be as equally useful to the criminally inclined for reconnaissance. Thus the purpose of this paper is to act as an introduction to Nikto and to reveal some of the inner workings of this tool, giving enough knowledge to be useful for you, or to know when it is being used against you.

Obtaining Nikto

To download Nikto browse to the following link, currently it is at version 1.21:

http://www.cirt.net/code/nikto.shtml

There are certain steps that need to be followed when performing either a security analysis or an attack on an organization. Intelligence and information gathering is the first step. Using publicly available information such as the organizations web presence allows for the collection of relevant server information, such as IPs and server roles. After this other techniques such as port scanning will identify what services the various servers provide. Once this has been done the actually vulnerability assessment is performed, either with good or bad intent[1]. It is at this stage one would use Nikto when assessing web servers.

Installation

Since Nikto is written in Perl it can be used either on the Windows or Unix platforms with only minor differences in installation. To run on the Windows platform a Perl distribution such as ActivePerl from ActiveState Corporation[2] is required. Perl functionality is readily available with most Linux and Unix platforms. This paper focuses mostly on the Windows platform.

Installation consists or decompressing the package into a folder of your choice. Make sure to maintain the directory structure contained within the archive. Installations of Nikto on a Linux or Unix platform can make use of Fyodor's Nmap[3] to perform the port scans if available. If this tool is not available, or for Windows systems, Nikto has built in Perl code to perform network scans, but the author recommends Nmap. Nikto also relies on Rain Forest Puppy's LibWhisker[4]. If not already installed then the LibWhisker library, LW.pm, is included with Nikto, however, the author again recommends having the full LibWhisker module.

Usage and Sample Traffic

Nikto uses a set of tests that it runs against a target server. According to the documentation that comes with the software it will check for misconfigurations, default files and scripts, insecure files and scripts, and outdated software. The vulnerabilities that it tests for can be constantly updated through the software's website. There is also the option for users to add their own custom vulnerabilities that Nikto should check for. Creating a file named "user_scan_database.db" in the plugins directory and adding the required checks to the file will accomplish this[5]. The simplest command to run Nikto is shown here:

```
Nikto –h [target IP] [options]
```

This will perform a basic scan against the specified server on port 80. The basic scan generally follows two steps in evaluating the server. Before proceeding one should pay attention to the warning provided in the Nikto documentation; since Nikto can perform a lot of requests to a web server there is the possibility that server instability may result. The scan first attempts to determine the type of web server and the HTTP methods that are supported by it, as seen in the excerpt here:

```
---------------------------------------------------------------------------------------------------
+ Target IP:       10.0.x.x
+ Target Hostname: xxxx
+ Target Port:     80
---------------------------------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Microsoft-IIS/4.0
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD
+ IIS/4 - May be able to bypass security settings using 8.3 file names
+ Microsoft-IIS/4.0 is outdated if server is Win2000 (4.0 is current for NT 4)
```

Next it checks for the existence of various directories on the web server. Directories that contain CGI scripts or other system files that can be either used in information gathering, or are susceptible to vulnerabilities. Some lines from the rest of the scan above are displayed below. Note that for this basic scan Nikto performs some 1800 plus checks against the server, so only a few key tests will be covered here. The specifics on all the tests performed can be obtained from the "scan_database.db" file found under Nikto's plugins folder.

The source is a system on my home network, and the target a test web server on my employers network accessed over via VPN. The target in question is running IIS 4 on a Windows NT 4.0 server with MS Exchange 5.5 and Outlook Web Access. The server has all the latest patches and service packs available from Microsoft. Windump logs of the query in action follow each line.

However, first an example of a check that did not find anything, here Nikto was checking for the existence of the /bin/ directory:

Nikto output:

```
None – Nikto only provides output for positive results.
```

Windump output:

```
20:06:28.970278 172.16.2.3.2391 > 10.0.x.x.80: P 812443960:812444078(118) ack 131596776 win 64400 (DF)
(ttl 128, id 45224, len 158)
0x0000    4500 009e b0a8 4000 8006 9193 ac10 0203       E.....@.........
0x0010    0a00 xxxx 0957 0050 306c e938 07d8 01e8       .....W.P0l.8....
0x0020    5018 fb90 b198 0000 4745 5420 2f62 696e       P.......GET./bin
0x0030    2f20 4854 5450 2f31 2e31 0d0a 486f 7374       /.HTTP/1.1..Host
0x0040    3a20 4d41 494c 310d 0a43 6f6e 6e65 6374       :.MAIL1..Connect
0x0050    696f                                          io

20:06:29.027150 10.0.x.x.80 > 172.16.2.3.2391: P 131596776:131597399(623) ack 812444078 win 8642 (DF)
(ttl 126, id 4883, len 663)
0x0000    4500 0297 1313 4000 7e06 2f30 0a00 xxxx       E.....@.~./0....
0x0010    ac10 0203 0050 0957 07d8 01e8 306c e9ae       .....P.W....0l..
0x0020    5018 21c2 77f4 0000 4854 5450 2f31 2e31       P.!.w...HTTP/1.1
0x0030    2034 3034 204f 626a 6563 7420 4e6f 7420       .404.Object.Not.
0x0040    466f 756e 640d 0a53 6572 7665 723a 204d       Found..Server:.M
0x0050    6963                                          ic
```

Next we have a successful attempt at locating the /cgi-bin/ directory on the server. This is accomplished by Nikto sending a HTTP GET ./cgi-bin/ request as shown in the windump output.

Nikto output:

```
+ /cgi-bin/ - Directory indexing of CGI directory should be disabled. (GET)
```

Windump output:

```
20:06:29.610653 172.16.2.3.2394 > 10.0.x.x.80: P 812713748:812713870(122) ack 131596802 win 64400 (DF)
(ttl 128, id 45240, len 162)
0x0000    4500 00a2 b0b8 4000 8006 917f ac10 0203       E.....@.........
0x0010    0a00 xxxx 095a 0050 3071 0714 07d8 0202       .....Z.P0q......
0x0020    5018 fb90 feca 0000 4745 5420 2f63 6769       P.......GET./cgi
0x0030    2d62 696e 2f20 4854 5450 2f31 2e31 0d0a       -bin/.HTTP/1.1..
```

```
0x0040    486f 7374 3a20 4d41 494c 310d 0a43 6f6e        Host:.MAIL1..Con
0x0050    6e65                                           ne

20:06:29.737918 10.0.x.x.80 > 172.16.2.3.2394: P 131596802:131597136(334) ack 812713870 win 8638 (DF)
(ttl 126, id 7955, len 374)
0x0000    4500 0176 1f13 4000 7e06 2451 0a00 xxxx        E..v..@.~.$Q....
0x0010    ac10 0203 0050 095a 07d8 0202 3071 078e        .....P.Z....0q..
0x0020    5018 21be 2292 0000 4854 5450 2f31 2e31        P.!."...HTTP/1.1
0x0030    2034 3033 2041 6363 6573 7320 466f 7262        .403.Access.Forb
0x0040    6964 6465 6e0d 0a53 6572 7665 723a 204d        idden..Server:.M
0x0050    6963                                           ic
```

Once all the available directories are discovered Nikto moves on to individual files than can be a danger to the system, for example the “htimage.exe” CGI.

Nikto output:

```
+ /cgi-bin/htimage.exe - This CGI can give an attacker a lot of information. (GET)
```

Windump output:

```
20:11:10.874067 172.16.2.3.3554 > 10.0.x.x.80: P 939673937:939674070(133) ack 133017627 win 64400 (DF)
(ttl 128, id 51130, len 173)
0x0000    4500 00ad c7ba 4000 8006 7a72 ac10 0203        E.....@...zr....
0x0010    0a00 xxxx 0de2 0050 3802 4951 07ed b01b        .......P8.IQ....
0x0020    5018 fb90 feea 0000 4745 5420 2f63 6769        P.......GET./cgi
0x0030    2d62 696e 2f68 7469 6d61 6765 2e65 7865        -bin/htimage.exe
0x0040    2048 5454 502f 312e 310d 0a48 6f73 743a        .HTTP/1.1..Host:
0x0050    204d                                           .M
20:11:11.002962 10.0.x.x.80 > 172.16.2.3.3554: . 133017627:133018987(1360) ack 939674070 win 8627 (DF)
(ttl 126, id 50734, len 1400)
0x0000    4500 0578 c62e 4000 7e06 7933 0a00 xxxx        E..x..@.~.y3....
0x0010    ac10 0203 0050 0de2 07ed b01b 3802 49d6        .....P......8.I.
0x0020    5010 21b3 1ecb 0000 4854 5450 2f31 2e31        P.!.....HTTP/1.1
0x0030    2032 3030 204f 4b0d 0a53 6572 7665 723a        .200.OK..Server:
0x0040    204d 6963 726f 736f 6674 2d49 4953 2f34        .Microsoft-IIS/4
0x0050    2e30                                           .0
```

Nikto has the useful trait of including text descriptors with each line of its output. These provide summarized information on the vulnerability that was detected and possible sources for more information via bugtraq ID or CVE numbers, etc. The complete results of this test scan are attached as an appendix to this paper.

Detecting Nikto at Work

In the documentation the author himself notes that Nikto is by no means subtle in its activities[6], so it should be fairly easy to spot through web server logs and alerts generated by an IDS. However, there are ways to make Nikto operate in a stealthier manner, this will be covered further on in the paper.

Alert Name	Count
WEB-IIS scripts access	316
WEB-MISC 403 Forbidden	253
WEB-MISC /etc/passwd	70
WEB-MISC http directory traversal	25

Table 1.1 – Most frequent alerts triggered by Nikto.

Using Snort 1.87 with a default rule set as the IDS produced 279 unique alerts from running Nikto against the test server above. In total there were 1315 alerts recorded. These numbers were obtained by “greping”[7] the “alert.ids” file to extract only the alert names and then manipulating the resulting data in MS Excel to collect totals. The most frequent alerts are listed in the chart above.

These results are what one would expect, given the number of tests Nikto performs. Since Nikto checks for CGI directories and then attempts to access them there are quite a few script access and 403 Forbidden alerts. Thus these are clues to look for when studying web server or IDS logs. A lot of these alerts originating from one source IP would most likely indicate some sort of reconnaissance activity. Of course this does not necessarily mean it is Nikto, since there are other web scanning tools as well. The default installation of Nikto will also leave a fingerprint on the server via the 404 checks and the User-Agent header[8]. However, the values for these are set by two variables in the code and can be modified. The variables, found in the actual nikto.pl file, are listed below.

```
$NIKTO{fingerprint}
$NIKTO{useragent}
```

Apart from this change Nikto can also attempt various IDS evasion techniques, as listed below from the documentation. Rain Forest Puppy’s LibWhisker provides this capability[9].

Random URI encoding (non-UTF8)	1
Add directory self-reference /./	2
Premature URL ending	3
Prepend long random string to request	4
Fake parameters to files	5
TAB as request spacer instead of spaces	6
Random case sensitivity	7
Use Windows directory separator \ instead of /	8
Session splicing	9

To use these options the –evasion, or just –e, command line switch is required with the corresponding number above. These can be used in combinations as well, so a typical command would be as follows.

```
nikto –e 23 –h [target IP]
```

A second scan was attempted with all these options turned on, however the Snort IDS still picked up on the traffic. For example the session splicing attempt is picked up as follows by Snort.

```
Snort alert:
[**] [1:1087:4] WEB-MISC whisker tab splice attack [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/21-01:10:33.631792 172.16.2.3:1475 -> 10.0.0.11:80
```

```
TCP TTL:128 TOS:0x0 ID:30281 IpLen:20 DgmLen:44 DF
***AP*** Seq: 0xDE435815  Ack: 0x881CEEC  Win: 0xFB90  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS415]
[Xref => http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html]
```

This particular technique attempts TCP session fragmentation through the use of an unusually short packet[10]. Snort also picks up the various directory traversal methods used by Whisker.

Conclusion

Nikto is for the most part picked up by IDS systems like Snort and will also leave a large footprint in most web server logs. For this reason it is probably most useful to the legitimate user who wishes to evaluate the security of web servers under their control. As a reconnaissance tool Nikto in its default setup is quite loud and would not serve the purposes of the criminally inclined, who would wish their reconnaissance activities to go unnoticed. There are several more options available for use with Nikto in addition to the few covered above. These allow for more targeted scans, or other functions such as using a proxy server or targeting HTTPS instead of HTTP, etc. The details for these options can be found in the documentation for Nikto.

References

[1] van der Walt, Charl. “Assessing Internet Security Risk, Part Five: Custom Web Applications Continued.” 8 October 2002. URL: http://online.securityfocus.com/infocus/1632 (Oct 2002)

[2] ActiveState. “ActivePerl”. URL: http://www.activestate.com/Products/ActivePerl/ (Oct 2002)

[3] Fyodor. “Nmap”. URL: http://www.insecure.org/nmap/ (Oct 2002)

[4] Rain Forest Puppy. “Whisker information, scripts, and updates” 18 October 2002 URL: http://www.wiretrip.net/rfp/p/doc.asp/i2/d21.htm (Oct 2002)

[5] [6] [8] [9] Sullo. “Nikto readme”. URL: http://www.cirt.net/nikto/README_nikto.html (Oct 2002)

[7] Millington, Huw. “Welcome to the Windows Grep Home Page” URL: http://www.wingrep.com/ (Oct 2002)

[10] “IDS415 ‘HTTP-WHISKER-SPLICING-ATTACK-TAB’ “ URL: http://www.whitehats.com/info/IDS415 (Oct 2002)

Assignment 2 – " Network Detects "

Detect 1: Possible Scan by MS SQL Server Worm.

```
19:06:34.573967 64.18.109.241.3414 > 24.100.x.x.1433: S [tcp sum ok] 658034917:658034917(0) win 16384 <mss
1460,nop,nop,sackOK> (DF) (ttl 110, id 1040, len 48)
                                        4500 0030 0410 4000 6e06 c943 4012 6df1
                                        1864 xxxx 0d56 0599 2738 d0e5 0000 0000
                                        7002 4000 f89d 0000 0204 05b4 0101 0402

19:06:37.614745 64.18.109.241.3414 > 24.100.x.x.1433: S [tcp sum ok] 658034917:658034917(0) win 16384 <mss
1460,nop,nop,sackOK> (DF) (ttl 110, id 1144, len 48)
                                        4500 0030 0478 4000 6e06 c8db 4012 6df1
                                        1864 xxxx 0d56 0599 2738 d0e5 0000 0000
                                        7002 4000 f89d 0000 0204 05b4 0101 0402

19:06:43.565197 64.18.109.241.3414 > 24.100.x.x.1433: S [tcp sum ok] 658034917:658034917(0) win 16384 <mss
1460,nop,nop,sackOK> (DF) (ttl 110, id 1412, len 48)
                                        4500 0030 0584 4000 6e06 c7cf 4012 6df1
                                        1864 xxxx 0d56 0599 2738 d0e5 0000 0000
                                        7002 4000 f89d 0000 0204 05b4 0101 0402
```

Source of Trace:

This trace is from my home network, which is connected via cable modem to the Internet. The detect was made outside my firewall.

Detect was Generated b y:

It is by pure chance that this trace was captured; I noticed the attempt to TCP port 1433 while in the process of configuring a FreeBSD box as a Snort IDS. Tcpdump was running to make sure the NIC doing the sensing was seeing all traffic. The tcpdump command used was:

```
tcpdump –n –i xl1 –vv
```

A second instance of tcpdump was running in order to log the traffic, as below:

```
tcpdump –i xl1 –w  /[some-path]/2002.9.8-mynet
```

The traces above were generated from this log file, using the following tcpdump command:

```
tcpdump –n –x –vv –r /[some-path]/2002.9.8-mynet "dst port 1433" >
2002.9.8-mynet-1433-vv.txt
```

Probability the Source Address was Spoofed:

If we assume that this is an individual scanning for MS SQL servers then there would be no purpose in spoofing the source IP since the results of the scan

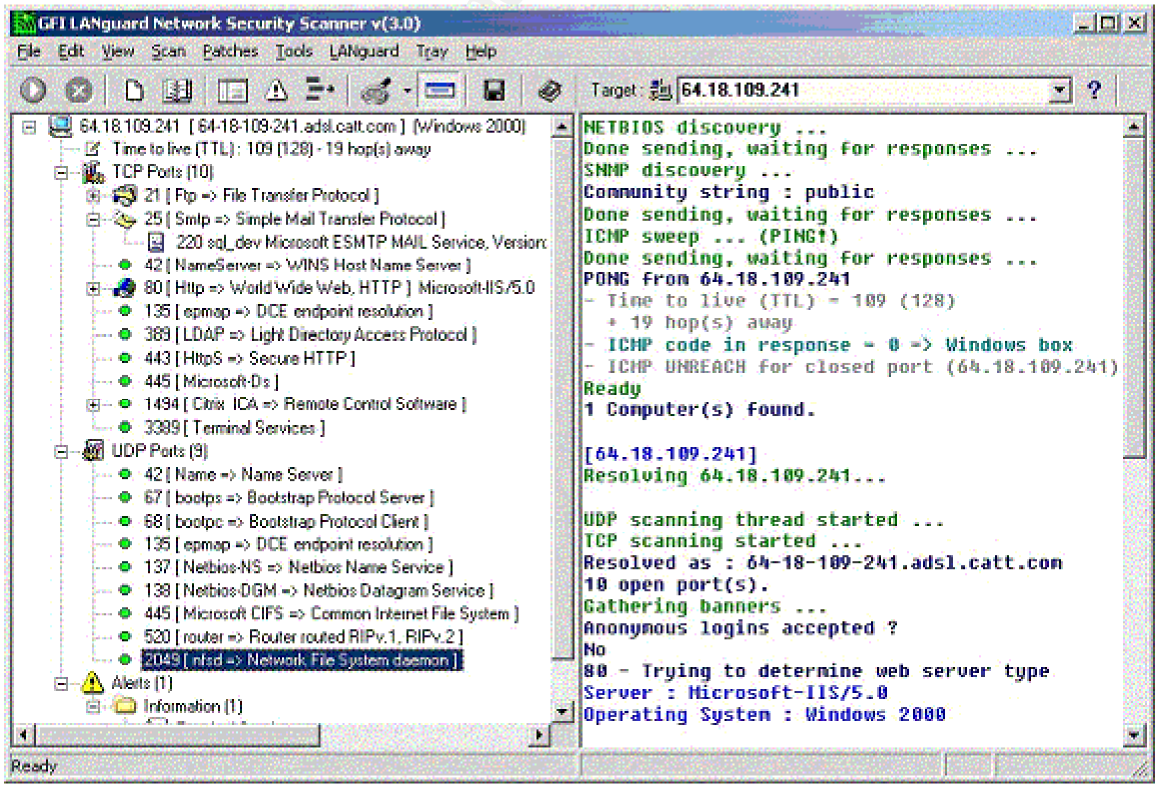
would be lost.
If it is not a scan driven directly by human intervention and rather the result of the SQL worm, then it is even less likely the address would be spoofed, since a major part of the code for the worm involves finding and enumerating possible future targets in order to propagate itself. A ping sent to the IP gives us the following, showing the system is still live:

```
Reply from 64.18.109.241: bytes=32 time=169ms TTL=109
```

This seems to tally with the trace above, since the trace above was from outside my firewall the extra hop that I pass through would account for my packets final TTL of 109 where as the trace has a final TTL of 110. The machine used to perform the ping runs Windows XP Pro. The TCP/IP stack for XP uses an initial TTL value of 128; the same is true for Windows 2000. So from this we see that the alleged attacker is about 19 hops from my home network.

Thus if the IP is not spoofed it seems our attacker is also using Windows 2000 or XP, as previous versions of windows all used an initial TTL of 32 and other operating systems have differing initial TTLs.

To confirm this hypothesis and to collect further information on the attacker the LANguard network scanner tool by GFI Software Ltd. was used[1]. This tool does OS fingerprinting, checks for open ports and also detects vulnerabilities in services on the system.


GFI LANguard Network Security Scanner v(3.0)
File Edit View Scan Patches Tools LANguard Tray Help
Target : 64.18.109.241
Ready

```
64.18.109.241 [ 64-18-109-241.adsl.catt.com ] (Windows 2000)
  Time to live (TTL) : 109 (128) - 19 hop(s) away
  TCP Ports (10)
    21 [ Ftp => File Transfer Protocol ]
    25 [ Smtp => Simple Mail Transfer Protocol ]
      220 sql_dev Microsoft ESMTP MAIL Service, Version:
    42 [ NameServer => WINS Host Name Server ]
    80 [ Http => World Wide Web, HTTP ] Microsoft-IIS/5.0
    135 [ epmap => DCE endpoint resolution ]
    389 [ LDAP => Light Directory Access Protocol ]
    443 [ HttpS => Secure HTTP ]
    445 [ Microsoft-Ds ]
    1494 [ Citrix ICA => Remote Control Software ]
    3389 [ Terminal Services ]
  UDP Ports (9)
    42 [ Name => Name Server ]
    67 [ bootps => Bootstrap Protocol Server ]
    68 [ bootpc => Bootstrap Protocol Client ]
    135 [ epmap => DCE endpoint resolution ]
    137 [ Netbios-NS => Netbios Name Service ]
    138 [ Netbios-DGM => Netbios Datagram Service ]
    445 [ Microsoft CIFS => Common Internet File System ]
    520 [ router => Router routed RIPv.1, RIPv.2 ]
    2049 [ nfsd => Network File System daemon ]
  Alerts (1)
    Information (1)
```

```
NETBIOS discovery ...
Done sending, waiting for responses ...
SNMP discovery ...
Community string : public
Done sending, waiting for responses ...
ICMP sweep ... (PING!)
Done sending, waiting for responses ...
PONG from 64.18.109.241
- Time to live (TTL) - 109 (128)
  + 19 hop(s) away
- ICMP code in response - 0 -> Windows box
- ICMP UNREACH for closed port (64.18.109.241)
Ready
1 Computer(s) found.

[64.18.109.241]
Resolving 64.18.109.241...

UDP scanning thread started ...
TCP scanning started ...
Resolved as : 64-18-109-241.adsl.catt.com
10 open port(s).
Gathering banners ...
Anonymous logins accepted ?
No
80 - Trying to determine web server type
Server : Microsoft-IIS/5.0
Operating System : Windows 2000
```

Figure 2.1.1 – LANguard network security scanner at work.

Though I do not know every method the software uses to perform these various tasks, one technique I am aware of is to send a correct ICMP Echo message but with an incorrect ICMP code, windows systems will reply to this packet and reset the ICMP code to 0, no other OS will do this[2]. The LANguard scanner uses this and other techniques, as can be seen in the screen capture above, to identify the attacker's system as a window 2000 machine.

Hence we know for sure that the initial TTL value would have been 128, and it seems very unlikely that the IP was spoofed, unless someone already had this information and went through a lot of trouble to make it look like it was this system causing the scans, but this seems unlikely.

Description of attack:

The attack, if it can be really called that, consists of a scan from IP 64.18.109.241 to the outside IP of my home firewall on TCP port 1433. This is a well-known port used by MS SQL.

Also it should be noted that the TCP sequence numbers do not change, so this would indicate that the packet was being retransmitted, since my firewall was doing what it should and not replying to the scan.

Another piece of information to note is the change in the IP ID numbers, although the change is not as dramatic as in some detects of the SQL worm, there are other detects that show similar behaviour to this one, these are discussed further under the correlations section.

Packet	Timestamp	Time Diff	IP ID	ID Diff
1	19:06:34.573967	N/A	1040	N/A
2	19:06:37.614745	3.04s	1144	104
3	19:06:43.565197	5.95s	1412	268

Table 2.1.1 – Change in IP ID numbers over time.

The scan itself does not have a CVE number; however the vulnerability targeted by worms such as SQLsnake is currently under review as a candidate for entry into the CVE list. It is listed as CAN-2000-1209[3].

Attack Mechanism:

Stimulus or Response: Stimulus as the infected remote system initiates the scanning.

Affected Service: MS SQL Server, listening on TCP port 1433.

Known Vulnerability/Exploit: Default installations of MS SQL have a system administration, “sa”, account with a null password. This account cannot be removed.

Attack Intent: To discover MS SQL servers listening on TCP port 1433 and to gain privileges on those servers.

Details:

As read from the candidate CVE listing above, the “sa” account on MS SQL Server and several products that use some version of MS SQL has a null password by default. Furthermore this account cannot be removed. Thus this worm scans for systems listening on TCP port 1433, indicating an MS SQL server is present, and then attempts to gain access to the server using the default “sa” account.

If successful there are a series of steps the worm will follow to copy itself and other files it needs in order to extract data and system information. This information is then e-mailed to an address in Singapore, ixltd@postone.com. This is followed by attempts to scan for other vulnerable systems.

These steps are clearly explained in an archived article from the Handler’s Diary at incidents.org. Of particular interest in this case, as there was no actual compromise, are the scanning characteristics of the worm. It uses a file named services.exe which is a copy of Foundstone ‘fscan’ to perform its scanning. A look at the code shows that the worm uses a set of class A network address as a starting point to generate random IPs to scan. One of those class A networks is of course 24.0.0.0. In this phase it uses 100 threads to perform the scans and the results are recorded in a text file. This file is then parsed to extract IPs of systems that responded on port 1433. From this list the cycle repeats with the exploit being sent out to each of the ‘working’ IPs obtained by the scan. The final step for the worm is to clean up after itself[4].

Correlations:

Looking at the scan results from the LANguard software gives a good indication that the SQL worm could infect this system. For instance the system name seems to be “sql_dev” suggesting that it is or was used as an SQL server. Secondly this system has so many services wide open and exposed to the internet that it seems the operator has not done much at all to secure it. Just to name a few there is WINS, Citrix ICA, terminal services, and Netbios services all exposed to the Internet. Thus it is quite conceivable that the worm could have easily infected the system. The report below for the attacking IP was obtained from the Dshield database[5].

```
Created: 9 Sep 2002, 22:07 EST Started: Tue Sep 10 06:55:01 2002
Finished: Tue Sep 10 06:55:01 2002

sourceport= 0 to 65536  targetport = 0 to 65536 protocol=0 to 99 source='064.018.109.241'
Rows returned: 40
```

date	time (gmt)	count	sourceip	sourceport	targetport	protocol	flags
8/28/2002	6:08:43	1	064.018.109.241	4764	1433	6	S
8/28/2002	6:08:46	1	064.018.109.241	4764	1433	6	S
8/28/2002	6:08:52	1	064.018.109.241	4522	1433	6	S
8/28/2002	6:08:52	1	064.018.109.241	4764	1433	6	S
8/28/2002	18:44:32	2	064.018.109.241	3363	1433	6	S
8/30/2002	9:31:32	1	064.018.109.241	4377	1433	6	S
8/30/2002	9:31:35	1	064.018.109.241	4377	1433	6	S
8/30/2002	9:31:41	1	064.018.109.241	4377	1433	6	S
8/30/2002	13:21:14	1	064.018.109.241	4271	1433	6	S
9/4/2002	20:41:21	1	064.018.109.241	3812	1433	6	S
9/4/2002	20:41:21	1	064.018.109.241	3813	1433	6	S
9/4/2002	20:41:24	1	064.018.109.241	3812	1433	6	S
9/4/2002	20:41:24	1	064.018.109.241	3813	1433	6	S
9/4/2002	20:41:30	1	064.018.109.241	3812	1433	6	S
9/4/2002	20:41:30	1	064.018.109.241	3813	1433	6	S
9/6/2002	19:08:09	1	064.018.109.241	4139	1433	6	S
9/6/2002	19:08:09	1	064.018.109.241	4140	1433	6	S
9/6/2002	19:08:09	1	064.018.109.241	4141	1433	6	S
9/6/2002	19:08:09	1	064.018.109.241	4142	1433	6	S
9/6/2002	19:08:09	1	064.018.109.241	4143	1433	6	S
9/6/2002	19:08:12	1	064.018.109.241	4139	1433	6	S
9/6/2002	19:08:12	1	064.018.109.241	4140	1433	6	S
9/6/2002	19:08:12	1	064.018.109.241	4141	1433	6	S
9/6/2002	19:08:12	1	064.018.109.241	4142	1433	6	S
9/6/2002	19:08:12	1	064.018.109.241	4143	1433	6	S
9/6/2002	19:08:18	1	064.018.109.241	4139	1433	6	S
9/6/2002	19:08:18	1	064.018.109.241	4140	1433	6	S
9/6/2002	19:08:18	1	064.018.109.241	4141	1433	6	S
9/6/2002	19:08:18	1	064.018.109.241	4142	1433	6	S
9/6/2002	19:08:18	1	064.018.109.241	4143	1433	6	S
9/6/2002	19:27:19	1	064.018.109.241	3928	1433	6	
9/6/2002	19:27:22	1	064.018.109.241	3928	1433	6	
9/6/2002	19:27:28	1	064.018.109.241	3928	1433	6	
9/6/2002	19:27:30	3	064.018.109.241	4016	1433	6	
9/6/2002	19:27:36	1	064.018.109.241	4016	1433	6	
9/6/2002	19:27:47	1	064.018.109.241	4237	1433	6	
9/6/2002	19:27:50	1	064.018.109.241	4237	1433	6	
9/6/2002	19:27:56	1	064.018.109.241	4237	1433	6	
9/8/2002	6:54:04	1	064.018.109.241	3271	1433	6	S
9/8/2002	6:56:08	1	064.018.109.241	4395	1433	6	S

Table 2.1.2 – Results from Dshield.org report for 64.18.109.241.

Some of the discussions in the incidents.org archives provided more insight[6]. There are several traces that show very similar behaviour, all attributed to the

SQL worm. For example this Snort trace, posted by John Sage, has similar characteristics to those seen in my trace above. It shows the attempts to resend the packet, as seen by the unchanging TCP sequence numbers, the difference in times between resends is almost the same, 3 seconds then 6 seconds, this of course is dependant on the TCP/IP stack. Finally we also see the IP ID increment in a similar fashion as with my trace.

```
05/27-16:52:00.355864 4.17.203.8:57915 -> 12.82.140.27:1433
TCP TTL:117 TOS:0x0 ID:33802 IpLen:20 DgmLen:48 DF
******S* Seq: 0x38875CBC  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1380 NOP NOP SackOK

05/27-16:52:03.566173 4.17.203.8:57915 -> 12.82.140.27:1433
TCP TTL:117 TOS:0x0 ID:33953 IpLen:20 DgmLen:48 DF
******S* Seq: 0x38875CBC  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1380 NOP NOP SackOK

05/27-16:52:10.136801 4.17.203.8:57915 -> 12.82.140.27:1433
TCP TTL:117 TOS:0x0 ID:34238 IpLen:20 DgmLen:48 DF
******S* Seq: 0x38875CBC  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1380 NOP NOP SackOK
```

It is known that the worm's scanning engine can use 100 threads as seen in the attack mechanism section. This might account for the difference in rates of change of IP ID between traces. For example the Snort trace below, also posted by John Sage, show very large changes in IP ID over relatively similar time frames. In fact the IP IDs seem to have wrapped around again when we see the second packet in this trace.

```
05/23-02:31:55.332640 63.119.243.5:3645 -> 12.82.128.10:1433
TCP TTL:114 TOS:0x0 ID:58622 IpLen:20 DgmLen:44 DF
******S* Seq: 0x6555E93A  Ack: 0x0  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460

05/23-02:31:58.512995 63.119.243.5:3645 -> 12.82.128.10:1433
TCP TTL:114 TOS:0x0 ID:1791 IpLen:20 DgmLen:44 DF
******S* Seq: 0x6555E93A  Ack: 0x0  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460

05/23-02:32:05.073682 63.119.243.5:3645 -> 12.82.128.10:1433
TCP TTL:114 TOS:0x0 ID:6912 IpLen:20 DgmLen:44 DF
******S* Seq: 0x6555E93A  Ack: 0x0  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460
```

To further add to this possibility if one looks again at the report from Dshield, you will notice on certain days, September 4th for example, there are two traces at the same time, but from different source ports. Looking further the system attempts to resend both traces after failure in much the same way as the detects above. However having two different sets of traces going at the same time may indicate multiple instances of the worm. Unfortunately I do not know of any logs for that date for this attacker IP that I could use to verify the changes in IP IDs. With so many similar detects, it would appear this is indeed the SQL worm at work.

Evidence of Active Targeting:

Now that it has been determined that the SQL worm is responsible for this scan, we can safely say that no active scanning is occurring. As discussed in the attack mechanism section a part of the worm generates random lists of IPs to scan from a set of seed class A networks.

Severity:

	Value	Explanation
Criticality	1	This scan was aimed at my home network that has no MS SQL servers.
Lethality	5	If successful the worm would infect the MS SQL server, gather and then send out sensitive information, and finally proceed to attack other systems.
System Countermeasures	5	System is patched with latest fixes for all software. Also no MS SQL running
Network Countermeasures	5	Hardware firewall blocked inbound TCP port 1433.

Table 2.1.3 – Severity data.

Severity = (Criticality + Lethality) – (System + Network Countermeasures)
= 6 – 10 = -4

Defensive Recommendation:

Several steps can be taken to defend against this threat.

First TCP port 1433 should be blocked on boundary firewalls.

Secondly any MS SQL Servers should be patched and brought up to date.

Lastly as a precaution in case of a successful penetration by this worm, block any e-mails to the address used for data collection, ixltd@postone.com.

Multiple Choice Test Question:

When studying network traces of a scan to a TCP port produced by tcpdump using the –vv option, what information could tip you off that this was not active targeting directed specifically at your IP?

a. The TCP Sequences numbers are the same.

b. The source ports keep changing.

c. The IP ID numbers vary greatly over a very short time period.

d. The –vv option of tcpdump does not provide enough information to answer this question.

Answer: C

A is incorrect, as TCP sequence numbers being the same are the result of a resend of the packet, but this does not give you any information on whether you a actively being targeted or not.

B is also incorrect as the source ports changing could be the result of the scan being performed at separate times, or of different scanning tools running at the same time, however, again it does not provide information as to whether you were specifically targeted for the scan or not.

C Is the correct answer, because a large variance in IP ID numbers over very short periods of time would indicate that the scanning system was sending out a lot of network traffic, which could indicate the actions of one of those scanning tools as it went through a range of IPs, thus possible evidence against active targeting.

D is also incorrect as the –vv option does give extra information, including the IP ID.