Global Information Assurance Certification Paper

Bymer Worm: Post Mortem Analysis of Malicious Code

[GCIH - Exploit in Action – Version 2.1]
SANS 2002 Orlando, Florida

Monty McDougal

September 20, 2002

Table of Contents

Abstract

A month before the SANS conference, I was starting to think about what might make an interesting practical topic. The answer to this question cam totally by chance as the Bymer worm infected the home network of one of my close friends.

The Bymer worm is not a terribly malicious worm; it was written to cheat in the Distributed.net cracking contest by infecting unsuspecting user's PC and installing the Distributed.net client with the malicious user's credentials. The worm spreads via open Windows drive shares using the win.ini file as a means to start the worm after the infected PC is rebooted.

This paper is written with two goals in mind. First, it should serve as an informative account of why Bymer was born and how it spreads. This will include an in-depth analysis of exactly how the worm works along with a less than perfect incident response (especially given all the new information I learned in the SANS training). I hope this paper proves useful as an example of a "real world" response to a malicious mobile code incident.

The second goal of the paper is to give an in-depth example of how one might perform analysis on a piece of malicious code. While I don't claim to be an expert on the topic, I do think the reader will gain benefit from the techniques presented in this paper (especially Appendix A) for analyzing malicious code both statically (without running it) and dynamically (running it). Hopefully, the reader can use this paper as a stepping-stone in performing his or her own malicious code analysis.

Part 1 – The Exploit

Background

I should probably go into a brief discussion of Distributed.net[1] for those who may be unfamiliar with them. The origins of the Bymer worm are directly linked to their cracking contest because it was written to help one of their contestants cheat in the contest. It is also worth pointing out that Distributed.net is in no way responsible for Bymer or any of the other malicious code instances that are installing the Distributed.net client without a user's permission. Distributed.net has been very active in disqualifying people who have employed such methods in their contests.[2]

Distributed.net is a non-profit organization that is helping to organize millions of computers which are put to use to solve a complex problem (usually cryptographic in nature) by providing clients that cooperatively work with the Distributed.net servers to divide the work required to crack some really ambitious cryptographic challenges. The whole thing is often run as a contest with RSA Labs usually offering $10,000 to the group who finds the keys in their contests (but the winner will only see $1,000 - $2,000 of it with the rest going to Distributed.net and charity).[3] The way Distributed.net works is by splitting the computations among thousands of machines running on nearly every platform imaginable,[4] including all versions of Windows. These clients download a group of key blocks from a central key server, process them, and reports back the results when retrieving more keys. The client itself uses an email address as identifier so that credit for the work is given to the right individual or team. The client can be run in a "hidden" mode that is not visible by users of the system (which is important for discussion of the worm later).

Name

The most generic and common name used for the worm I am describing is the Bymer worm. It exists under many different names depending on the antivirus vendor, but the worm itself contains the string "bymer" as part of a host and email address used by the worm. The names McAfee uses for the two variants they document are *W32/Msinit.worm* and *W32/Msinit.worm.b*. Here is a list of the other aliases for this worm taken from the McAfee Virus Information Library.[5]

> *Bymer (Norman), Bymer.C (Panda), I-Worm.Msinit.A (Softwin), I-Worm.Msinit.B (Softwin), I-Worm/RC5.A (AVG) , I-Worm/RC5.B (AVG) , I-Worm/RC5.C (AVG), TR.Worm.RC5.WinInit (AntiVir), TROJ_BYMER (Trend), TROJ_MSINIT.A (Trend), TROJ_RC5.B (Trend), Trojan.Win32.Bymer, Trojan/Wln32.Msini.A (RAV), W32.Bymer.A (Ikarus), W32.Bymer.B (Ikarus), W32.Bymer.C (Ikarus), W32.HLLW.Bymer (NAV), W32/Bymer-A (Sophos), W32/Bymer-B (Sophos), W32/Bymer-C*

(Sophos), W32/Bymer.B (Norman), W32/MsInit.A (AntiVir) , W32/MSInit.A (Panda), W32/MSInit.B (Panda), W32/MSInit.D (Panda), W32/MsInit.worm.a, Win32.Bymer.A (CA/VET), Win32.Bymer.B (CA/VET), Win32.Bymer.C (CA/VET), Win32.HLLW.RC5 (DrWeb), Win32.MSInit.A@mm (Softwin), Win32.RC5.4096 (DrWeb), Win32/Bymer.Worm (CA/InoculateIt), Win32/Bymera.C.unp (RAV), Win32/Bymera.D@mm (RAV), Win32/MSInit.A (RAV), Win32/MSInit.A worm (ESET), Win32/MSInit.B worm (ESET), Win32/MsInit.C (ESET), Win32/MsInit.C worm (ESET), Win32/Rc5.B.Worm (CA/InoculateIT), Win32/Rc5.C.Worm (CA/InoculateIT), Win32:MSInit-A1 [Wrm] (Alwil), Win32:MSInit-A2 [Wrm] (Alwil), Win32:MSInit-B [Wrm] (Alwil), Worm-RC5 (Sophos), Worm.Bymer.a (KAV/AVP), Worm.Bymer.b (KAV/AVP), Worm.Bymer.c (KAV/AVP), Worm.Dnet.A (VirusBuster), Worm.Dnet.B (VirusBuster), Worm.Dnet.C (VirusBuster)[i]

The specific worm variant that was involved in the incident that I investigated was the *W32/Msinit.worm.b* (using the McAfee name).

The Bymer worm does not have a Common Vulnerability Exposure (CVE) number associated with it.[6] There are currently three CVE Candidates that are directly related to unprotected Windows drive shares used by the Bymer worm.

CAN-1999-0518 -- A NETBIOS/SMB share password is guessable.[7]

CAN-1999-0519 -- A NETBIOS/SMB share password is the default, null, or missing.[8]

CAN-1999-0520 -- A system-critical NETBIOS/SMB share has inappropriate access control.[9]

In addition to the CVE entries associated with unprotected Windows networking shares, there have been at least two other significant advisories I would like to point out.

SANS/FBI Top Twenty -- Open drive shares made the Top Twenty list (as W4 - NETBIOS - unprotected Windows networking shares).[10]

CERT IN-2000-02 – Exploitation of unprotected Windows networking shares.[11]

Operating Systems

The Bymer worm spreads across a network using an open file share named "*c*" with a "*windows*" directory and the *win.ini* file inside. Because the worm is needs

[i] McAfee.com, "Virus Information Library".

the file structure *c:\windows\win.ini*, it would be able to spread to default installations of the following OSes assuming they had the appropriate shares:

Windows 3.11
Windows 95
Windows 98
Windows ME
Windows XP

Additionally, the worm will work under other OSes (if run manually), but cannot infect them remotely for a default configuration (because they use "*WINNT*" as their default Windows directory). If a user performed a custom install using "*windows*" the worm could function normally.

Windows NT 3.5
Windows NT 4.0
Windows 2000

Protocols

The protocol being exploited by the Bymer worm is NetBIOS, but more specifically unprotected Windows network drive shares. It is not an attack on the NetBIOS protocol; it is simply an attack on the insecure use of the protocol.

Brief Description of Bymer

The Bymer worm is an example of malicious mobile code (malicious in the sense it uses computer resources without permission) that was written to help some individual (presumably Bymer) win the Distributed.net encryption cracking contest. It works by spreading to vulnerable machines that have unprotected Windows network shares and installing itself along with the Distributed.net client (*dnetc.exe* and *dnetc.ini*). Specifically the worm will copy itself to the *c:\windows\system* folder and then update the *c:\windows\win.ini* such that it is automatically started at boot up. Once the machine is rebooted, the worm will run taking three actions. It will add itself to the registry to ensure it restarts on boot up, install the Distributed.net client such that it will also restart "hidden" on boot up, and start randomly scanning the Internet looking for other hosts to infect. The actions taken by the worm are covered in much greater detail in Part 2 of this paper.

Bymer Variants

There are two major variants of the Bymer worm. I say major because if you look at the list of aliases provided on the McAfee web site[ii] (and included above) you will note that some of the antivirus vendors are reporting three variants of the

[ii] McAfee.com, "Virus Information Library".

worm. I was not able to find an adequate description of a third variant at any of the antivirus vendors sites. The minor variants seem to only differ in file naming conventions and choice of registry keys for startup. Therefore, the two major variants (using the McAfee names) of this worm are:

W32/Msinit.worm

The first major variant of the Bymer worm is known as *W32/Msinit.worm* and it is slightly less evolved than the second version of the worm. There are a few traits that make this major variant unique. First it is not a "dropper"[12] like the second one so the file size of the actual worm is significantly smaller (~22k). When this worm spreads, it will manually copy each file (*dnetc*.exe and *dnetc.ini*) needed by the worm individually to the victim machine. Second, the worm's file name is "random" because it will be either *msiXXX.exe* or *msXXX.exe* (depending on minor variant) where *XXX* is a number matching the first segment of the IP subnet for the remote machine. Finally, the first version can be identified by the email address used in the *dnetc.ini* that will be bymer@inec.kiev.ua.

W32/Msinit.worm.b

The second major variant, *W32/Msinit.worm.b*, is a more evolved version of the first worm. It is a "dropper" which makes the worm significantly larger than the first variant (~220k) because it caries the other files needed by the worm (*dnetc.exe* and *dnetc.ini*) inside the worm. When the worm is first executed, it will "drop" the other files into the *c:\windows\system* directory and execute *dnetc.exe*. The filename this version of the worm will use is *wininit.exe* (note there is already a valid Windows file with this filename in the *c:\windows* directory). Finally, the first version can be identified by the email address used in the *dnetc.ini* that will be bymer@ukrpost.net.

While there are only two known variants of the Bymer worm, there are fourteen known worms, Trojans, and other malicious code that have been written specifically for the purposes of cheating in the Distributed.net contests[iii]. While these malicious agents are generically variants of the Bymer worm because they were written for the same purpose, they will not be considered variants for the purpose of my analysis due to the differences in attack vectors used.

References

There are a few particularly useful sites for more information about the Bymer worm (most of which are also listed in my formal references because I have pulled information from them).

Associated Common Vulnerability Exposure (CVE) Entries

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0518
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0519

[iii] Distributed.net, "trojans, worms, viruses".

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0520

Associated CERT Advisory
http://www.cert.org/incident_notes/IN-2000-02.html

Antivirus Vendors
http://vil.mcafee.com/dispVirus.asp?virus_k=98844
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.bymer.html

Background on Bymer and Distributed.net Contests
http://www.distributed.net
http://www.distributed.net/trojans.html

NetBIOS Protocol Information
http://packetstorm.linuxsecurity.com/groups/rhino9/netbios.doc

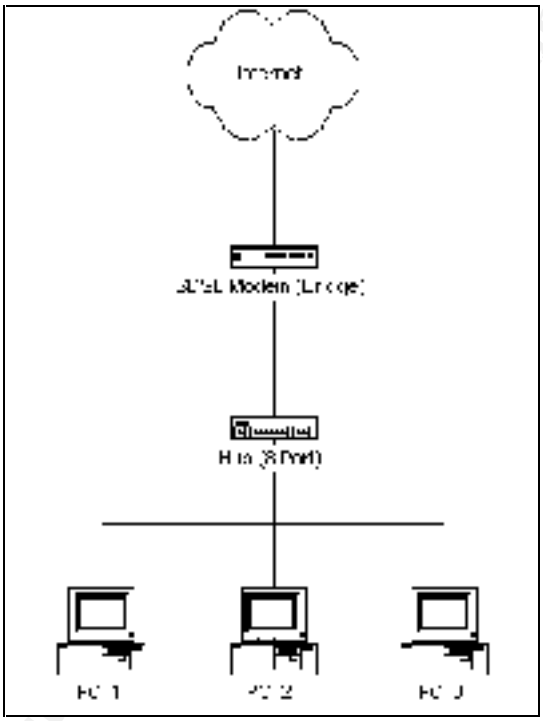
Honeynet Project's Bymer Paper "Know You Enemy: Worms at War"
http://project.honeynet.org/papers/worm/

Honeynet Project's Snort Logs and Bymer Worm Binaries (Both Variants)
http://stan.ksni.net/~lance/win98.tar.gz

Part 2 – The Attack

Description and Diagram of Network

The network involved in the Bymer worm infection that I am documenting could most likely be described as a variant of typical home network. At the time the incident occurred, it consisted of three Windows based machines, an eight-port network hub, and an SDSL modem (acting as a bridge). A diagram of the network at the time of the incident is included below.



A more detailed description of the devices involved in this network is included below.

PC_1 -- a typical home use machine running Windows 98. This machine had drive sharing enabled at the root level for both the C drive with full access and no passwords required. The only countermeasure in use on this machine was a "reasonably" up-to-date copy (less than 30 days old) of McAfee VirusScan. This machine was being used as a file and print server for PC_2 and PC_3.

PC_2 – another typical home use machine running Windows 98. This machine also had drive sharing enabled at the root level for the C drive but had a password required. This machine also had a "reasonably" up-to-date copy (less than 30 days old) of McAfee VirusScan. This machine had also been further hardened by the installation of Tiny Personal Firewall and was configured such that it only allowed access to the NetBIOS access to machines in the private

network 192.168.0.x. When the SDSL installers had re-IPed the machines, they had overlooked this (or more likely didn't care) and had actually broken drive and printer sharing for this machine because the other machines which accessed it now had a public IP.

PC_3 – another typical home use machine running Windows 98. This machine also had drive sharing enabled at the root level for the C drive with no password required. This machine also had a "reasonably" up-to-date copy (less than 30 days old) of McAfee VirusScan

SDSL Modem – this modem is a little different than the typical DSL modem. It is a SpeedStream 5251 Ethernet SDSL Modem manufactured by Efficient Networks, Inc. It is configured such that the modem is acting as a bridge and the three PC were assigned static IPs as part of the bridged network.

Hub – the hub in use is a typical home-network grade hub, in this case a NetGear EN 108 8-port unmanaged hub.

It is also relevant to this discussion of this incident, that this network had never been connected to the Internet as a whole prior to the incident that I am documenting. The network itself had been primarily setup as a convenience for the owner of the network in moving files between machines and allowing for shared printers. Internet access prior to SDSL had been strictly limited to PC_2 and PC_3 via dial–up modem accounts.

This incident occurred less than 24-hours after the introduction of SDSL Internet access into the network on 1 March 2002. As part of the installation process, the company that installed the SDSL reconfigured the IPs of each machine on the home network to use public IPs as opposed to the private IPs (192.168.0.x) that were previously in use on the system (which is an important fact in the case of PC_2).

NetBIOS Protocol Description[13]

The protocol the Bymer worm is using to make its connections to victim machines is known as NetBIOS (Network Basic Input / Output System). The NetBIOS protocol was originally developed by IBM and Sytek as a software API allowing client software to access network resources. NetBIOS has since been adopted as an industry standard for accessing network services. IBM later extended the NetBIOS protocol with NetBEUI (Network Extended User Interface) for use in its LAN Manager Server. Microsoft adopted both NetBIOS and NetBEUI for use in its networking products (including file and printer sharing). NetBIOS is supported on Ethernet, Token Ring, and IBM PC Networks. It can operate as a connection-oriented protocol (TCP) or as a connectionless one (UDP).

The NetBIOS protocol is an API designed to link network operating systems allowing inter-application communication and data transfer. Its intention is to isolate application programs from hardware dependencies. It also handles error recovery, low level addressing, and routing. In a NetBIOS LAN, computers are known to each other by a name (discussed in more detail below).

Machines on a NetBIOS LAN communicate by using the NetBIOS session, datagram, or broadcast method. One-on-one communication is handled through sessions and allows for larger messages and error detection / correction. Datagram and broadcast methods allow one-to-many communication, but are limited in message size and lack error detection / correction.

NetBIOS names are used to identify resources on a network and applications use these names to start and end sessions. NetBIOS names must be unique on a network and can consist of up to 16 alphanumeric characters (although Windows limits these name to 15 and uses the 16th character as a NetBIOS suffix). Before a machine using NetBIOS can fully function on a network it must register its NetBIOS name with the Master Browser on the network. This process is handled by the NetBIOS as the machine becomes active. The machine will broadcast the NetBIOS name it intends to use to the network (several times to ensure it is received). If any other machine on the network is using that name it will respond indicating the name is in use and the second machine must pick a new name. If no other machines on the network are using the given name then the second machine will finish the registration process.

On a Windows machine, NetBIOS and SMB (NetBIOS over TCP/IP) work closely together and both use ports 137 (UDP), 138 (UDP), and 139 (TCP). Newer versions of Windows (i.e. Windows 2000) also use port 445 (TCP). The SMB (Server Message Block) protocol, also known as the Common Internet File System (CIFS) enables file sharing over TCP/IP.

For more information on NetBIOS, see the paper “Understanding NetBIOS” by NeonSurge as released by the rhino9 Team.

http://packetstorm.linuxsecurity.com/groups/rhino9/netbios.doc

How Bymer Works

Bymer works by spreading across unprotected Windows networking drive shares. It is not exploiting the NetBIOS directly, just the insecure configuration of Windows drive shares commonly found on many PCs. For the Bymer worm to be able to successfully spread to a victim system, the victim must have drive sharing enabled, the name of the drive share must be “c”, must contain the Windows directory as *“\windows*”, and must allow read and write access without a password. Sadly, this configuration is pretty common in many home and small office networks.

The Bymer worm propagates to other systems by copying itself to the victim system and then modifying that systems c:\windows\win.ini file such that it will be loaded on startup via the load= line in the *win.ini*. The worm simply has to wait for a reboot and the worm will have been activated on the victim system. This process is documented extensively in the next section and Appendix A.

The worm itself is not doing anything magical, and this exploit can easily be carried out manually. The worm simply adds the element of doing it in an automated fashion by randomly scanning the Internet for vulnerable hosts to exploit. Assuming we have a vulnerable host 192.168.102 and a piece of malicious code *c:\foo.exe* that we want to execute on the victim system, we could use the following steps to achieve the same thing Bymer is doing.

First, copy the file *c:\foo.exe* to the victim system. Once this has been completed, retrieve the *c:\windows\win.ini* file from the victim system.

```
C:\>copy c:\foo.exe \\192.168.0.102\c\windows\system\foo.exe
        1 file(s) copied.

C:\>copy \\192.168.0.102\c\windows\win.ini c:\winini.txt
        1 file(s) copied.
```

Next make a quick edit to the local file *c:\winini.txt* in notepad to add the code to load *foo.exe* the next time the victim system starts. The relevant edits are shown below.

```
[windows]
load=foo.exe
```

Once the changes have been made, copy the new *win.ini* file back to the victim system.

```
C:\>copy c:\winini.txt \\192.168.0.102\c\windows\win.ini
Overwrite \\192.168.0.102\c\windows\win.ini? (Yes/No/All): y
        1 file(s) copied.
```

The next time the machine at address 192.168.0.102 is rebooted, the program *foo.exe* will be executed.

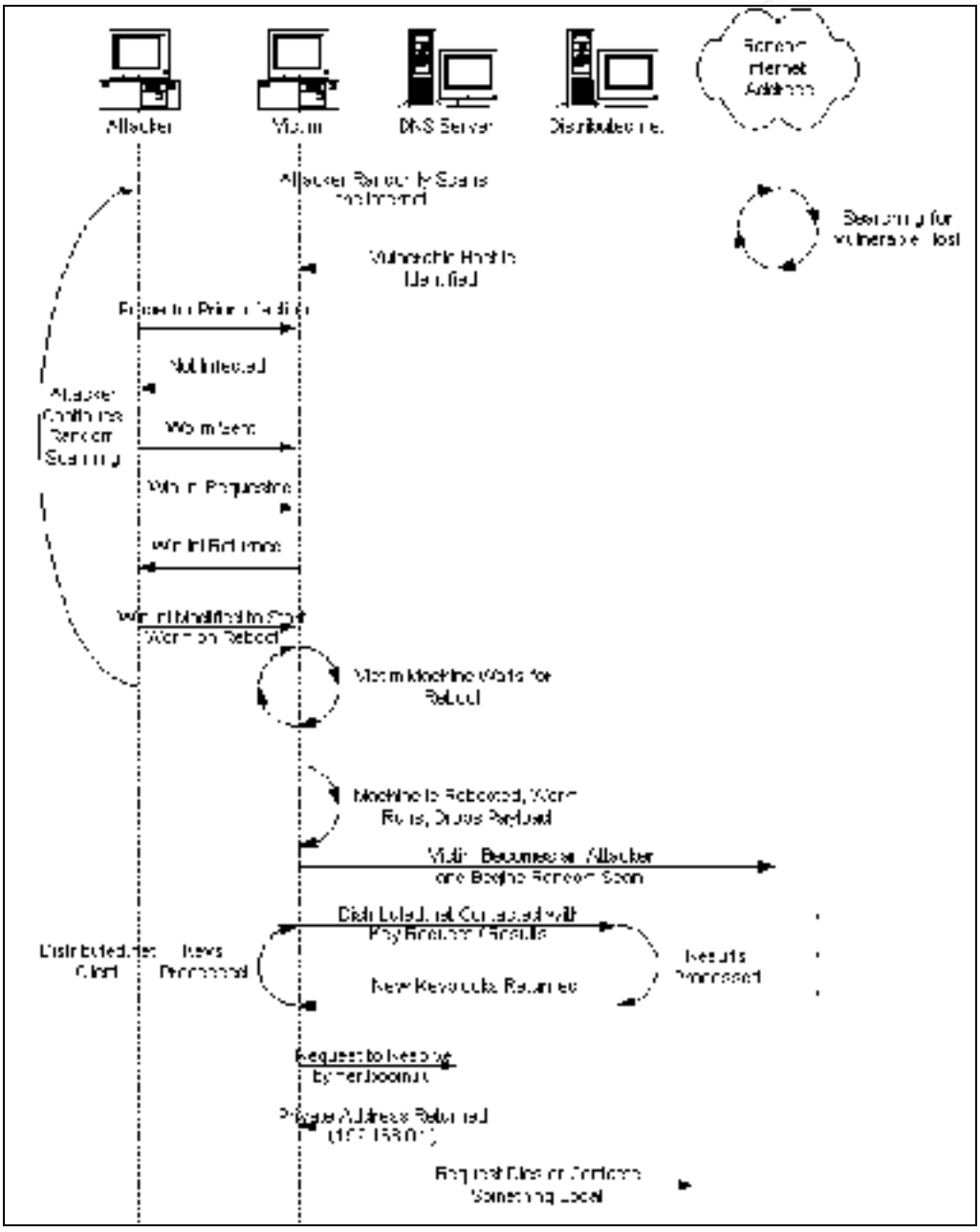
Description and Diagram of the Attack

In this section of the paper I am going to briefly try to show how the Bymer worm spreads at a network level. I have provided a much more thorough analysis of the Bymer worm as part of Appendix A which I would encourage interested readers to read. The excerpts from the Snort logs[iv] used in this section are taken

[iv] Honeynet Project, "win98.tar.gz".

from the Honeynet Project's web site. The sections I am highlighting are pulled from 4 days of entries documenting both Bymer variants attacking the same machine. Do to the complexities of simulating the Internet in a lab (because Bymer randomly chooses it's target IPs) and the fact that I had "real" logs available, I felt it was better to analyze these logs than the ones I could produce in a lab by using NAT to manipulate the packets.

The diagram below is a pictorial representation of how the Bymer worm typically spreads. Further details of each of these steps will be provided by the paragraphs that follow.



The Bymer worm's attack begins when it starts randomly scanning the Internet looking for a machine that has vulnerable Windows networking shares. The initial discovery the worm is trying to make during the scan is to determine if the worm is already installed on the target system. In the capture below you can see Bymer looking for a copy of itself in the *c:**windows**system*\ directory.

```
11/02-21:41:09.754218 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:36827  DF
*****PA* Seq: 0x21CC068   Ack: 0xCE67344   Win: 0x21AC
00 00 00 40 FF 53 4D 42 08 00 00 00 00 00 01 00  ...@.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 04 EE 00 1D 00 04 5C 57 49 4E 44 4F 57 53  ........\WINDOWS
5C 53 59 53 54 45 4D 5C 57 49 4E 49 4E 49 54 2E  \SYSTEM\WININIT.
45 58 45 00                                      EXE.
```

Assuming the worm is not already installed on the system, the worm will begin to copy itself to the target machine. In the excerpt below you can see the first part of this transfer in process.

```
11/02-21:41:17.287743 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:38619  DF
*****PA* Seq: 0x21CC0AC   Ack: 0xCE6736B   Win: 0x2185
00 00 00 5D FF 53 4D 42 2D 00 00 00 00 00 01 00  ...].SMB-.......
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 84 EE 0F FF 00 00 00 07 00 91 00 16 00 20  ...............
00 20 BB 01 3A 10 00 00 00 00 00 00 00 00 00 00  . ..:...........
00 00 00 1C 00 5C 57 49 4E 44 4F 57 53 5C 53 59  .....\WINDOWS\SY
53 54 45 4D 5C 77 69 6E 69 6E 69 74 2E 65 78 65  STEM\wininit.exe
00                                               .
```

Next the worm begins the process of actually copying itself on to the target system. Note the **MZ** – the first two characters of any valid Windows exe. The **PE** that I have highlighted at the end of the example shows that this is a Windows Portable Executable (PE) file.

```
11/02-21:41:17.632426 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:38875  DF
******A* Seq: 0x21CC10D   Ack: 0xCE673B0   Win: 0x2140
00 00 0B 68 FF 53 4D 42 1D 00 00 00 00 00 01 00  ...h.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 04 EF 0C 0E 00 F0 FF 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 2C 0B 3C 00 2D 0B 00  .........,.<.-..
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............

0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
EA 05 D3 58 AE 64 BD 0B AE 64 BD 0B AE 64 BD 0B  ...X.d...d...d..

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00  ........PE..L...
```

Once the worm is through copying itself to the victim system, it uses the *win.ini* file's load line to ensure that the worm is started when the machine is booted. In the example Snort logs below you can see that the worm is requesting the *c:\windows\win.ini* file from the victim system.

```
11/02-21:41:47.754427 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:19932  DF
*****PA* Seq: 0x220213C   Ack: 0xCE6751D   Win: 0x1FD3
00 00 00 52 FF 53 4D 42 2D 00 00 00 00 01 00  ...R.SMB-.......
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 84 F3 0F FF 00 00 00 07 00 A2 00 16 00 00  ................
00 3E BB 01 3A 01 00 00 00 00 00 00 00 00 00 00  .>..:...........
00 00 00 11 00 5C 57 49 4E 44 4F 57 53 5C 77 69  .....\WINDOWS\wi
6E 2E 69 6E 69 00                                n.ini.
```

Here we can see the victim system responding to the worm by sending a copy of its *win.ini* file. Note that the load line already has an entry in it. *Msi216.exe* is actually one of the other variants of the Bymer worm. In the case of the Honeynet computer, it had already been infected with the earlier variant of the worm.[14]

```
11/02-21:41:48.002536 172.16.1.105:139 -> 216.234.204.69:2021
TCP TTL:127 TOS:0x0 ID:9740  DF
******A* Seq: 0xCE67562   Ack: 0x22021C9   Win: 0x1E28
00 00 19 61 5B 77 69 6E 64 6F 77 73 5D 0D 0A 6C  ...a[windows]..l
6F 61 64 3D 63 3A 5C 77 69 6E 64 6F 77 73 5C 73  oad=c:\windows\s
79 73 74 65 6D 5C 6D 73 69 32 31 36 2E 65 78 65  ystem\msi216.exe
0D 0A 72 75 6E 3D 0D 0A 4E 75 6C 6C 50 6F 72 74  ..run=..NullPort
3D 4E 6F 6E 65 0D 0A 0D 0A 5B 44 65 73 6B 74 6F  =None.....[Deskto
70 5D 0D 0A 57 61 6C 6C 70 61 70 65 72 3D 28 4E  p]..Wallpaper=(N
6F 6E 65 29 0D 0A 54 69 6C 65 57 61 6C 6C 70 61  one)..TileWallpa
70 65 72 3D 31 0D 0A 57 61 6C 6C 70 61 70 65 72  per=1..Wallpaper
53 74 79 6C 65 3D 30 0D 0A 0D 0A 5B 69 6E 74 6C  Style=0.....[intl
```

The worm is done infecting the system once it has copied a newly modified version of the *win.ini* file back to the victim's computer. This ensures that the worm will be executed the next time the system is rebooted. This is shown in the example below. Note that in the Honeynet's case, the new worm did not remove the previous version of the worm.

```
11/02-21:41:48.538643 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:21212  DF
******A* Seq: 0x22021C9   Ack: 0xCE68EC7   Win: 0x1FA3
00 00 0B 68 FF 53 4D 42 1D 00 00 00 00 01 00  ...h.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 84 F4 0C 0F 00 7F 19 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 2C 0B 3C 00 2D 0B 00  ........,.<.-..
5B 77 69 6E 64 6F 77 73 5D 0D 0A 6C 6F 61 64 3D  [windows]..load=
63 3A 5C 77 69 6E 64 6F 77 73 5C 73 79 73 74 65  c:\windows\syste
6D 5C 77 69 6E 69 6E 69 74 2E 65 78 65 20 63 3A  m\wininit.exe c:
5C 77 69 6E 64 6F 77 73 5C 73 79 73 74 65 6D 5C  \windows\system\
```

```
6D 73 69 32 31 36 2E 65 78 65 0D 0A 72 75 6E 3D  msi216.exe..run=
0D 0A 4E 75 6C 6C 50 6F 72 74 3D 4E 6F 6E 65 0D  ..NullPort=None.
0A 0D 0A 5B 44 65 73 6B 74 6F 70 5D 0D 0A 57 61  ...[Desktop]..Wa
6C 6C 70 61 70 65 72 3D 28 4E 6F 6E 65 29 0D 0A  llpaper=(None)..
54 69 6C 65 57 61 6C 6C 70 61 70 65 72 3D 31 0D  TileWallpaper=1.
0A 57 61 6C 6C 70 61 70 65 72 53 74 79 6C 65 3D  .WallpaperStyle=
```

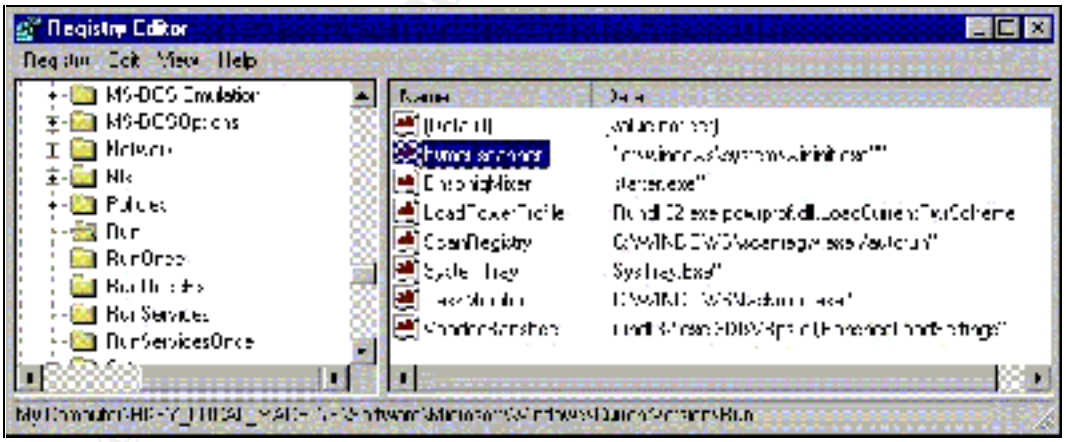
The next part of the worm's life cycle involves waiting for the machine to be rebooted. Assuming a machine that had not been previously infected by another Bymer variant (as shown above), the *win.ini* file would have a line that looks something like the following. Note: Anything on the load line of the *win.ini* file will be executed when Windows is started.

```
[windows]
load=c:\windows\system\wininit.exe
```

Once Windows has been restarted, Bymer will remove itself from the *win.ini* file's load line. This results in a *win.ini* file that will look something like the example below.

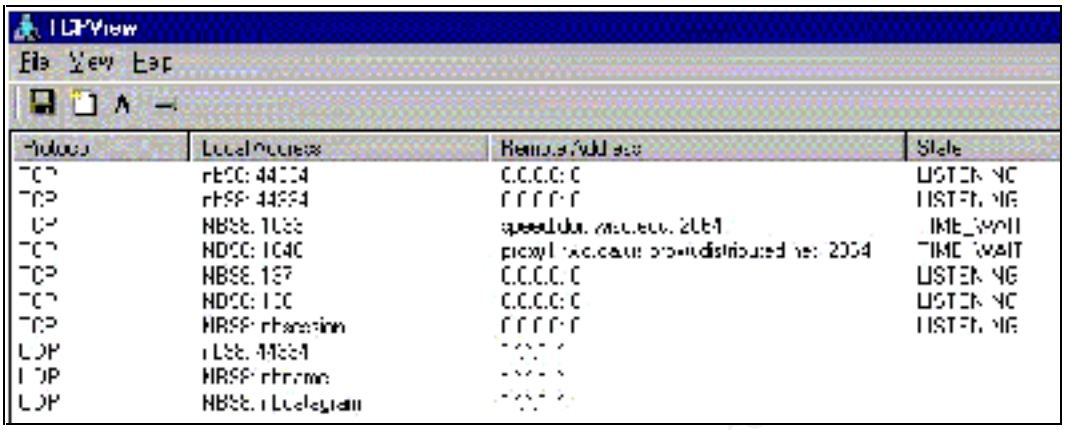
```
[windows]
load=
```

After Bymer has removed itself from the *win.ini*, it adds itself to the registry to start automatically when rebooted. You can see this in the screen capture from *regedit.exe* below.



The *wininit.exe* version of Bymer caries it's own copy of *dnetc.exe* and *dnetc.ini*, which it drops in the *c:\windows\system* directory. The worm launches *dnetc.exe* with the command line flags (-hide –install) that make *dnetc.exe* run hidden and install itself as a service to start automatically (*dnetc.exe* will create it's own registry entry to start itself automatically). Once *dnetc.exe* is fully configured, it will attempt to access the Internet and connect to one of the Distributed.net key servers to download a block of keys to start cracking. You

can see the *dnetc.exe* client attempting to contact these servers in the screen capture form *TCPView.exe*[15] below.



The complete *dnetc.ini* file that the worm dropped is shown below. Note the email address.

```
[parameters]
id=bymer@ukrpost.net

[misc]
project-priority=OGR,RC5,CSC,DES

[rc5]
fetch-workunit-threshold=64
randomprefix=222


[ogr]
fetch-workunit-threshold=16

[triggers]
restart-on-config-file-change=yes
```

The worm also creates a log file (*wininit.log*) that records how many systems the worm has scanned while looking for new hosts to compromise (maintained across reboots as shown below). You can see an example *wininit.log* file below.

```
Started at 10:36 3.03.2002
Stopped (scanned 5, found 0) at 10:38 3.03.2002
Started at 10:39 3.03.2002
Stopped (scanned 170, found 0) at 11:47 3.03.2002
Started at 11:48 3.03.2002
```

Finally, the worm will also try to connect to the web site *bymer.boom.ru*. This address now resolves to the private address 192.168.0.1 due to the worm author's attempt (as suggested by the Honeynet team) to deactivate some

feature of the original worm. You can see that the name *bymer.boom.ru* does indeed resolve to the private address shown in the partial nslookup output below.

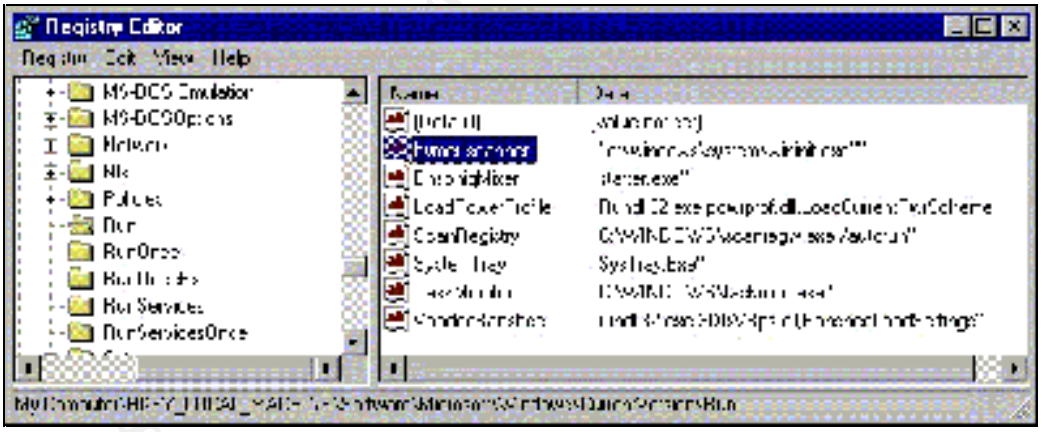
```
Non-authoritative answer:
Name:    bymer.boom.ru
Address:  192.168.0.1
```

This concludes the description of the Bymer worm's attack. There is a much more detailed explanation of Bymer in Appendix A.

Signature of Bymer

There are a number of things that could be used to conclusively identify the presence of the Bymer worm (*W32/Msinit.worm.b* variant) on a system, while attacking your network, or while present on your network.

First, identification of the worm on the local PC is trivial. Look for the presence of the file *wininit.exe* in the *c:\windows\system* folder of the PC. Note that there is a valid file of the same name in the *c:\windows* folder. The file size of the Bymer worm is 220,672 bytes. It is also conceivable that this file might be located somewhere else on the hard drive if it was infected by means other than the automated scanning of the worm (i.e. a user double-clicked the worm). It is probably safer to look for the presence of the Bymer worms registry key as shown below from *regedit.exe*.



The Bymer worm also has a number of unique strings in its body that could be used in a Snort[16] signature (none exist in the Snort database). Consider the highlighted section of the following Snort dump.

```
11/02-21:41:19.007539 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:42459  DF
******A* Seq: 0x21D0B35   Ack: 0xCE673D9   Win: 0x2117
E1 94 73 D9 5C 61 5F 61 01 6C C2 63 00 38 57 9E  ..s.\a_a.l.c.8W.
EA 95 DD 8D 38 74 2A 2C 57 50 27 C1 9C 61 4A 8E  ....8t*,WP'..aJ.
0D B2 18 9B 8D D0 45 72 0C B1 81 59 44 00 80 2C  ......Er...YD..,
```

```
45 81 00 07 95 FF AF 82 62 79 6D 65 72 2E 73 63  E.......bymer.sc
61 6E 6E DF FD FF 5D 5E 53 6F 66 74 77 61 72 65  ann...]^Software
5C 4D 69 63 72 6F 73 0D 5C 57 69 6E 64 FB DF DE  \Micros.\Wind...
FE 6F 77 73 5C 43 75 72 17 6E 74 56 27 73 69 6F  .ows\Cur.ntV'sio
6E 5C 52 75 6E 53 0A BB 51 D8 6D 76 26 65 73 37  n\RunS..Q.mv&es7
2F 6D 73 F7 0F 50 6F 1D 69 74 00 2A 5D 30 40 00  /ms..Po.it.*]0@.
71 03 DB 2E 80 6E 06 00 02 01 7F 03 06 09 02 FF  q....n..........
```

It would be pretty safe to create a Snort signature (for Snort v1.8.7) that was looking for NetBIOS traffic inbound to port 139 containing the string "bymer.scann". A possible Snort rule for this might be:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"WORM – Possible
Bymer Worm Attack [Incoming]"; content:"bymer.scann";
flow:to_server,established; classtype:misc-activity; reference:url,
vil.mcafee.com/dispVirus.asp?virus_k=98844; rev:1;)
```

This rule could of course be reversed to detect attacks originating from the internal network. Such a rule might be:

```
alert tcp $ HOME_NET any -> $ EXTERNAL_NET 139 (msg:"WORM – Possible
Bymer Worm Attack [Outgoing]"; content:"bymer.scann";
flow:to_server,established; classtype:misc-activity; reference:url,
vil.mcafee.com/dispVirus.asp?virus_k=98844; rev:1;)
```

A more generic (and useful) rule might be one that detected all NetBIOS traffic that originated from a system outside the internal network. Such a rule might be:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NetBIOS – Possible
NetBIOS Access from External Network"; flow:to_server,established;
classtype:misc-activity; reference:url, www.sans.org/top20.htm; rev:1;)
```

The final method of detecting Bymer (and probably the most effective) is to simply run an up-to-date virus scan. It will do the work for you.

How to Protect Against Bymer (and Other NetBIOS Share Attacks)

Because Bymer and many other NetBIOS attacks rely on the use of NetBIOS, just denying access to these services can mitigate it fairly effectively. Unfortunately, many users rely on these services for sharing files and printing, so this is not always an option.

If Windows drive sharing is needed, the user should take some preventive countermeasure to ensure it cannot be used to compromise the system. The paragraphs that follow will address the countermeasures I feel to be most beneficial in preventing these attacks.

First, the user should always require a strong password for their shares. This will prevent most automated worms (which generally are using open shares with no

password enabled) and will severely limit the speed at which more sophisticated worms can spread. Strong passwords really have little or no effect on the users of these shares because most versions of Windows will save the passwords and reconnect automatically. Additionally, it is important to ensure that the machine is patched to current levels because there are known exploits against Windows 9x (including ME) machines that allow malicious users to connect to NetBIOS shares with only part of the correct password.[17]

The Second thing that a user using drive shares should do is ensure that the shares are restricted to the fullest extent possible. This includes such things as sharing only a small portion of the hard drive that does not include the root of the file system (i.e. the C Drive) or any other important files or directories (i.e. Windows). This would also include limiting shares to read-only if the remote users have no reason to write to them. Many versions of Windows (NT and 2000 variants) allow for the limiting of share access based on user accounts and this option should be used when available.

Another option that is available for protecting Windows drive shares is personal firewalls. I consider a personal firewall to be required equipment for most machines in a small office or home environment (and highly recommend it in other environments). These firewalls should be configured such that they allow NetBIOS only to a "trusted" set of machines.

The next major step that can be taken to limit the spread of such worms and attacks is to block NetBIOS at the border router or firewall for a network. There are very few if any circumstances where the use of NetBIOS is appropriate across the Internet. Filtering NetBIOS at the network boundary will reduce the vulnerability to such attacks to machines that are located on the internal network.

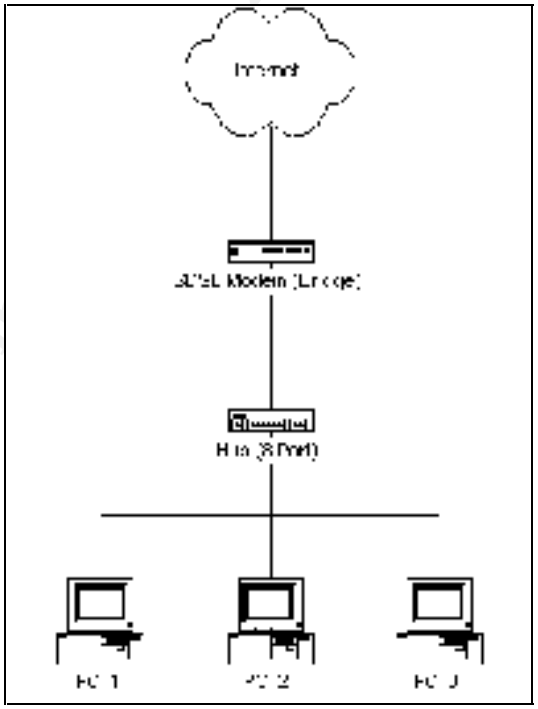
Finally, when all other mechanisms fail, we rely on detection tools to alert us of the problem. This could include things as sophisticated as Intrusion Detection Systems (IDS) deployed on the network, to the simpler antivirus software installed on the local machine. Up-to-date antivirus software will generally prevent the spread of these automated worms once they have been detected and included in the antivirus data files.

Part 3 – The Incident Handling Process

The SANS incident handling process is comprised of six primary phases designed to provide a roadmap for a complete incident handling methodology. This part of my paper is going to cover the actual steps involved in my handling of the Bymer worm. Because the incident that is being documented here occurred on a home network, most of the preparatory steps that should have been taken were absent. The actions shown here by no means are all inclusive of the SANS incident handling methodology, but it is a real world example of how an incident was handled. Also because the incident in question occurred at a home, the steps shown a good example of what a typical home or small office response might be to a malicious code attack (excluding my in-depth worm analysis).

Preparation

The network involved in this incident was a home network. I had personally been partially responsible for the building of the original network (the owner is a close friend of mine), which was later modified by the installation of SDSL and the owner of the network. The diagram below shows the network after SDSL was installed.



While there was not a formal incident handling process in place (which is probably appropriate for a home network), there were several steps taken by the owner of this network and myself prior to the incident that occurred which helped

to reduce the impact of the Bymer worm on the network. PC_1 (which was acting as a file and print server) had McAfee VirusScan installed locally and was reasonably up-to-date (less than 30 days old). Prior to installation of SDSL, this machine did not have Internet access and the owner was manually updating the McAfee data files. PC_2 was the machine the owner primarily used for Internet access. It had been also been protected by McAfee VirusScan (data file less than 30 days old). Additionally, it had a copy of Tiny Personal Firewall (TPF) that I had installed on it because it was the machine the owner was primarily using for Internet access. The rule set in TPF had been configured such that it prompted for nearly every access (incoming and outgoing), and had file sharing enabled for the local network (which prior to the SDSL install was 192.168.0.x). When the SDSL install had taken place, all the machines in the network were given public IPs and this rule was broken disabling file and print sharing for PC_2. Finally, PC_3 was also protected by a copy of McAfee VirusScan (data file less than 30 days old).

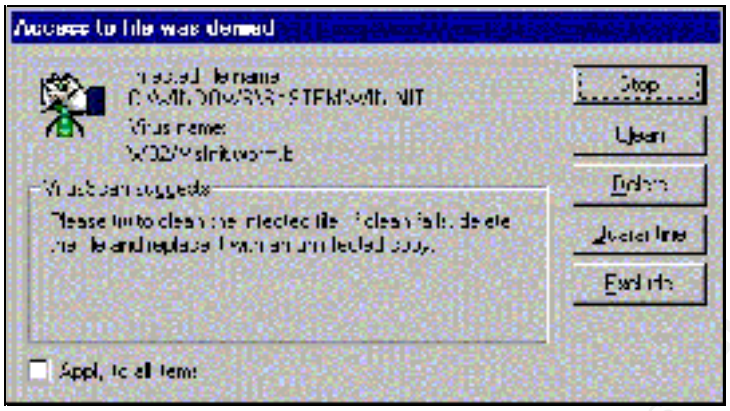
While there was no formal incident handling team, the owner of this network is reasonably computer knowledgeable and had identified a more skilled person (myself) to handle anything he did not feel comfortable with (including any "incidents" that might occur). The introduction of SDSL into this network did not happen without my knowledge, but sadly I did not realize how strong the threat against home broadband users had actually become. The SDSL installation had been scheduled for a Friday morning and I had told the owner of the network I would come over and help him reconfigure / secure the network sometime that weekend.

I thought surely, he would be safe for a couple of days until I had time to work on securing his network. Unfortunately, I was wrong in this regard and the Bymer worm came knocking less than 24 hours after the introduction of SDSL into the home network.

Identification

The owner of the network identified the infection of the Bymer worm very quickly because he happened to reboot all three machines on Saturday morning before he started using his computers for the day. Sometime between the time when SDSL was installed Friday morning and Saturday morning when the PC_1 was rebooted, the Bymer worm had silently installed itself to his machine and configured itself such that it would be started on boot up through the modification of the *win.ini* file on the machine. This action had gone undetected by the virus scan on PC_1 that was only configured to scan files as they were run (a definite oversight for a machine acting as a file server).

When the machine was rebooted, the owner was greeted with a pop-up from McAfee similar to the one shown below.



The owner of this machine immediately picked up his phone and called me (as his technical guru) to assess the situation. The first thing I ask to do for me when he called was read me the exact name of the virus so I could write it down (for further investigation). In this case, the name of the worm was "*W32/MsInit.worm.b*" which is McAfee's name for this particular variant of the Bymer worm (although I did not realize it was the Bymer worm at the time).

I also ask the owner to explain to me exactly what he was doing at the time he received the warning. When he told me he had received it when rebooting his system, I suspected this was some sort of file sharing worm, because I knew he was not running a web server on any of his PCs. I also ask him if he had rebooted the other machines in his network, and he indicated that he had, but had not received similar warnings when they were rebooted.

I decided I wanted to do a little research on "*W32/MsInit.worm.b*" to make sure I knew what I was dealing with before taking any further action. I figured if McAfee had detected the worm, then it would be included in their Virus Information database. The information that follows was taken directly from the Bymer entry from the McAfee page.

> *Virus Characteristics:*
> *W32/Msinit has been seen with the filenames, "MSINIT.EXE" and MS*.EXE [where * represents the first segment of the victim's IP subnet, ie. MS216.EXE]. This worm spreads through open network shares like the VBS/Netlog worm. It scans random IP address over NetBIOS for computers that have shares named "C" and a Windows folder called "Windows". When it finds one, it copies itself and the files "dnetc.exe" and "dnetc.ini" to the "c:\windows\system\" folder of the remote computer. The file "dnetc.exe" is an encryption-cracking program from www.distributed.net, which is not the author of this worm. The samples received by AVERT are packed with the UPX file-compression utility.*
>
> *Method of Infection:*
> *When it finds a computer with an open share, it copies itself directly to the unprotected computer, and modifies the win.ini load= line to run the worm*

> *on the next bootup. The next bootup, it creates the registry key HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices\msinit so the worm and encryption-cracking program runs without any user intervention. The worm them runs the command "dnetc -hide -install" which causes the distributed.net client to install itself in the background.*[v]

The above information along with the aliases for the worm (Bymer) was all I needed to take action. I have been a long time supporter of the Distributed.net, and I was aware of the Bymer worm, but not McAfee's name for it. As indicated by the information above, there was really no immediate threat to the owner's system being posed by this worm.

Containment

Based on this information gathered during identification, I provided him with my initial plan of action until I could get there to assess the situation in person. The initial action I had him take was to disconnect the affected PC (PC_1) from the home network by removing its network cable from the hub. This action was taken to prevent the further spread of the worm to his other PCs and the Internet as a whole. Additionally, I told him not to do anything to this PC until I was there. I also suggested to him that the other PCs did not seem to be infected at this point, but I highly recommended that he run a complete virus scan on them as well. This scan reported no virus. I also suggested that he might want to consider disconnecting his other machines from the network until I could respond. He decided he would do that as well.

I decided I would gather the resources necessary to properly secure his home network for the threat of outside attack. This included gathering the tools necessary to both deal with the current problem and prevent future ones.

The specific steps that I wanted to perform as part of my response were:

1. Capture a copy of the Bymer Worm for later offline analysis
2. Remove Bymer from the affected system (PC_1)
3. Verify that all three PCs were virus free
4. Add a hardware-based router firewall to the home network to protect the perimeter and restore private addressing for the internal network using NAT for Internet access
5. Install and configure Tiny Personal Firewall on all three machines to further protect against future incidents
6. Restore Internet access for the network
7. Analyze the worm at my leisure for "fun" on my home test network

[v] McAfee.com, "Virus Information Library".

Following a quick trip to CompUSA to buy a hardware-based router/firewall, my "jump kit" contained all the tools I needed to perform the seven steps indicated above.

The specific contents of my "jump kit" for the initial response (steps 1 through 6 as indicated above) included the following:

- Write protected Windows 98 boot disk from a known clean system
- Command line version of McAfee VirusScan on a write protected disk (just incase cleaning from within Windows failed)
- Blank floppy disk (for copying the virus)
- CD containing up-to-date copies of the McAfee data files
- CD containing most recent copy of Tiny Personal Firewall
- NetGear Web Safe Router (Model RP114)

The procedure to remove the Bymer worm was considered low risk so a backup of the system was not made. Additionally, because this was an automated worm that had attacked a home Windows 98, there was no evidence to preserve. This decision was made based on the scope of the incident (a common worm infection) and the fact that logging in Windows 98 is virtually non-existent. Because there was no further containment necessary at this point, the decision was made to move to the Eradication step.

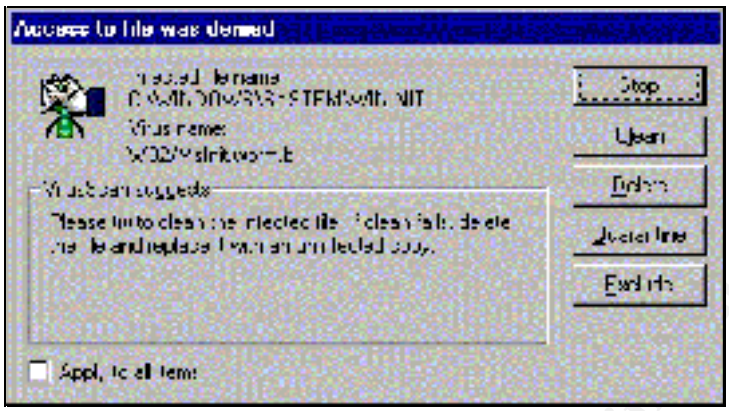
Eradication

Removal of the worm was actually more difficult than I first expected. I wanted to make sure I got a copy of the worm for analysis so I started my removal operation by logging out of Windows and rebooting to DOS mode (by hitting *F8* during the boot sequence). The commands I issued at the DOS prompt to get a copy of the worm on the blank floppy I had brought are shown below.

```
C:\>cd windows

C:\WINDOWS>cd system

C:\WINDOWS\SYSTEM>copy wininit.exe a:\wininit.mmc
        1 file(s) copied
```

Changing the extensions to *mmc* is a convention I use anytime I store malicious mobile code for later analysis. Additionally, I clearly labeled the disk as containing the Bymer worm.

Once I had captured a copy of the worm for later analysis, I was ready to remove it. I considered doing this manually, but I wanted to see how well McAfee would handle it so I went ahead and rebooted the machine and let it enter into Windows normally. When Windows was fully loaded, I again got a warning from McAfee that looked like the one shown below.



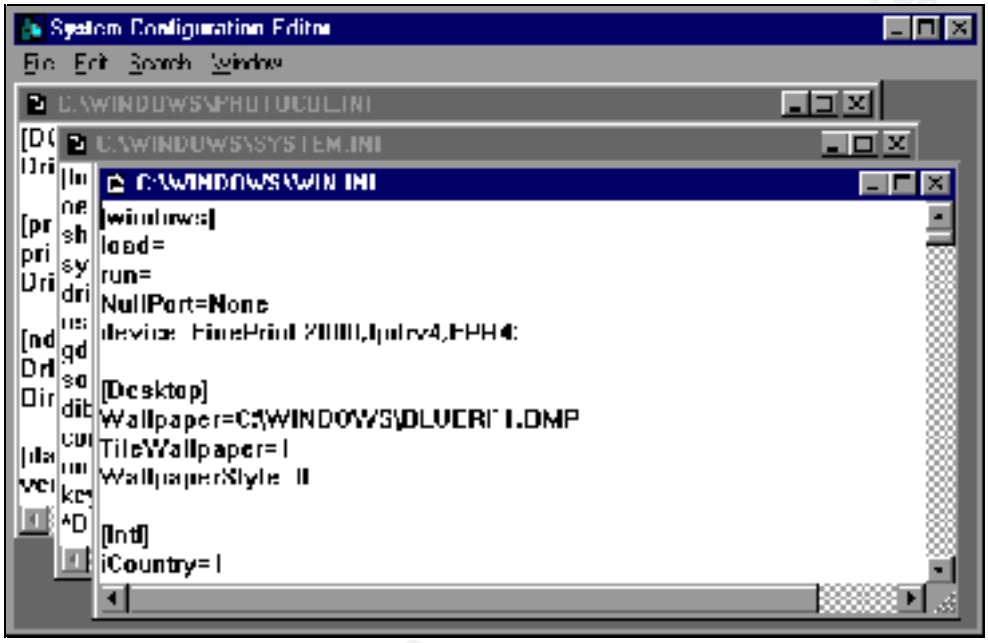
This time I planned on letting McAfee get rid of the file for me, so I clicked the *Delete* button. McAfee responded with the window shown below indicating that the worm had been removed.

Unfortunately, McAfee is not capable of removing this worm (at least not while Windows is running). McAfee must remove the virus and it's startup entries and then issues a signal to terminate the running program. The problem with this approach is whenever Bymer is unloaded, it checks to ensure it is configured to restart. If it is not, it will create another registry entry, copy itself out to the hard drive, and then run itself again before exiting. When the worm takes this action, the whole process starts over and McAfee again pops-up warning the user. Based on my observations of this worm in Windows 98, it is impossible to kill this worm manually from within Windows due to the fact it is not visible in the task bar.

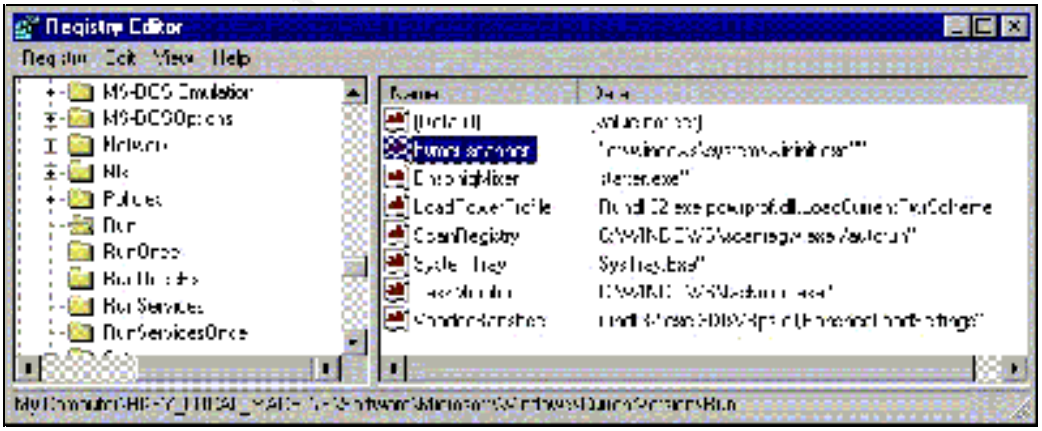
While I am quite sure McAfee could have removed this worm from DOS automatically, I decided to just deal with it myself. I again rebooted the machine and entered DOS mode issuing the commands shown below.

```
C:\>cd windows

C:\WINDOWS>cd system

C:\WINDOWS\SYSTEM>del wininit.exe
```

I rebooted into Windows, and verified that McAfee did not pop-up again alerting me of the worm. I also wanted to verify that none of the mechanisms the worm had used to start were still present. The first thing I wanted to check was the *win.ini* to verify that the load= line was clean. I did this by running *sysedit.exe* (by clicking *Start* | *Run* and typing *sysedit.exe*) to get the window shown below. As you can see, the load= line is blank.



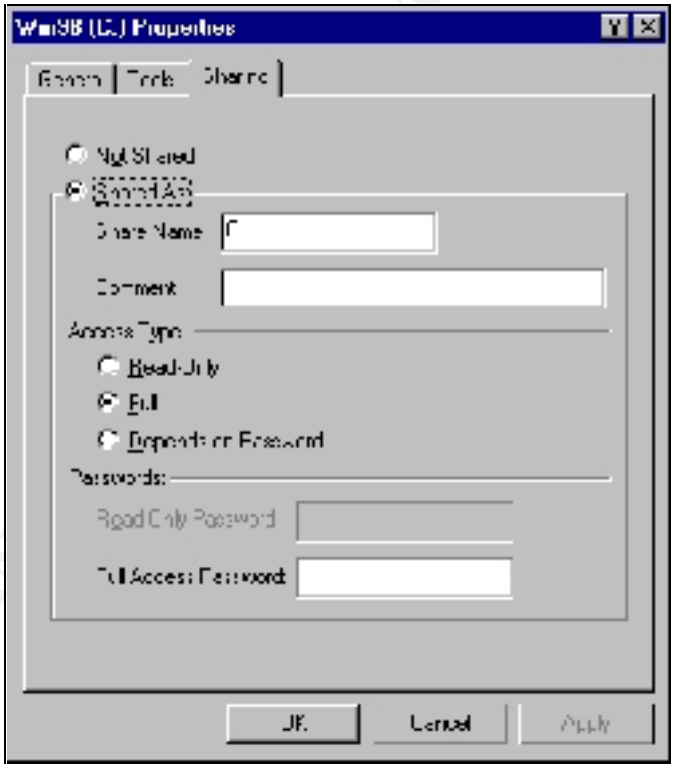
I also wanted to check to make sure the worm did not have an entry in the registry. I did this by running *regedit.exe* (by clicking *Start* | *Run* and typing *regedit.exe*). The window show below indicates that the worm's registry entry is still present. This line was removed by highlighting it and hitting the *Delete* key.



At this point, I had removed the worm from the computer but I had done nothing to verify the method of infection and had taken no actions to prevent the infection from reoccurring. The first thing that I need to do was verify that the system was indeed vulnerable to the suspected infection method. I did this by opening

Windows Explorer and highlighting the C drive as shown below. Note the hand below the drive indicates that this drive is indeed being shared.

Once the C drive was selected, I right-clicked it and went down to *Sharing*. This opened the file share dialog as seen below.



You can see from this dialog that the C drive is shared as “C” with full access and no password required. This is obviously an insecure setting that needs to be changed during the Recovery phase.

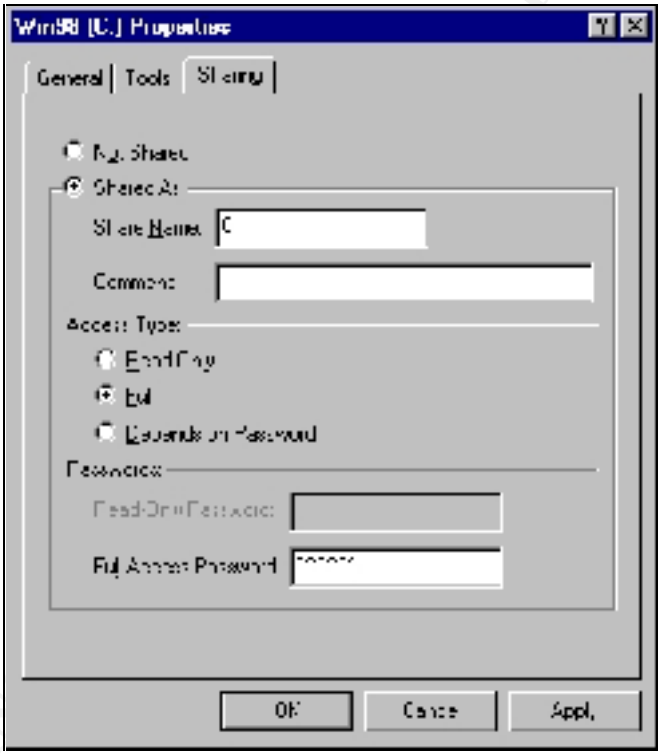
The final step of the Eradication phase was to update the McAfee VirusScan signatures on all the machines and to run a complete system scan of all the

machines on the network. This scan was completed without identifying any further instance of malicious code.

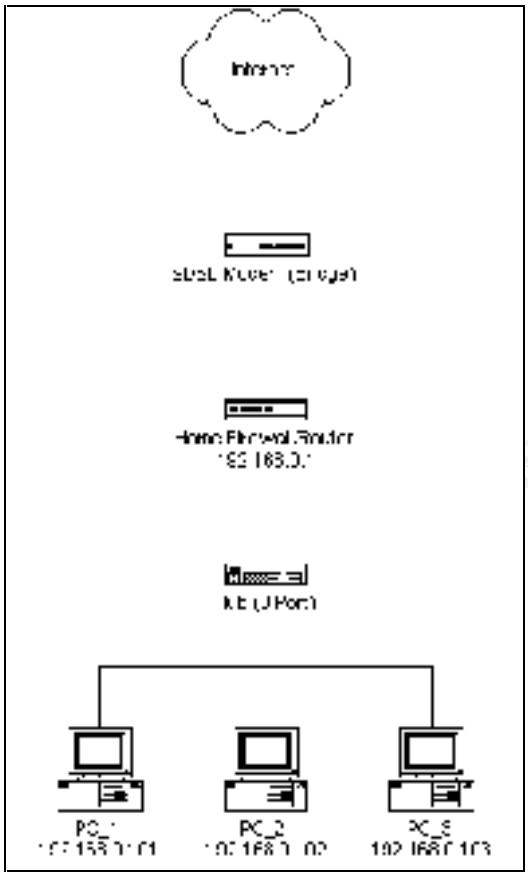
Recovery

At this point of the incident response, I was satisfied that the machines had been returned to normal, but I needed to address some of the major security holes which were open before reconnecting the machines back to the Internet.

The first thing that was needed was to ensure that all available drive shares were properly secured. This was handled by going to the file sharing screen (using the method above) and enabling a strong password on the full access share. This can be seen below and was repeated for each machine on the network. Note you will be prompted to confirm this password, but it is not shown here.

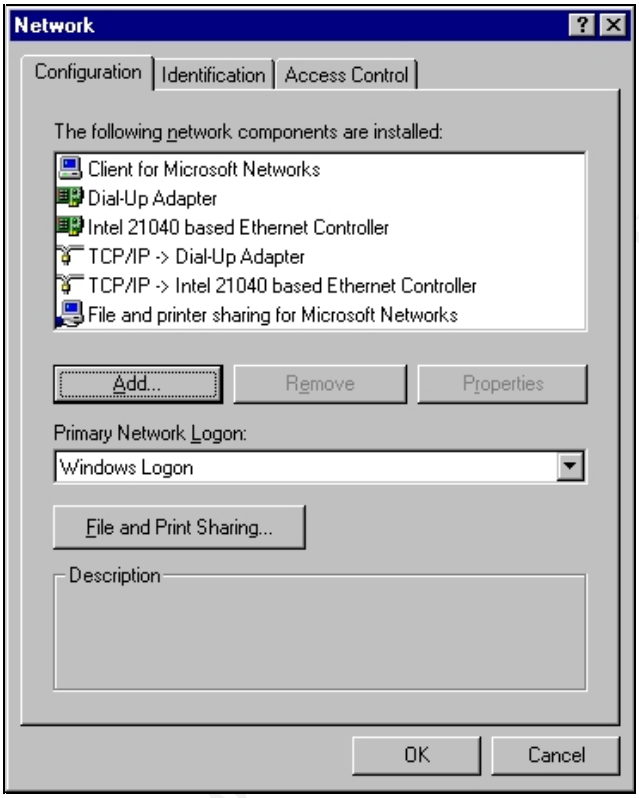


The machines on this network had also been reconfigured to use public IP addresses by the SDSL installer. I preferred to go back to the private addressing scheme, and hide the whole network behind the NetGear Web Safe Router purchased for this purpose. This device also serves as a NAT enabled firewall allowing all machines to connect to the outside world through a single IP address while virtually eliminating the threat of outside attacks on the network. The diagram of the new network that was built is shown below.



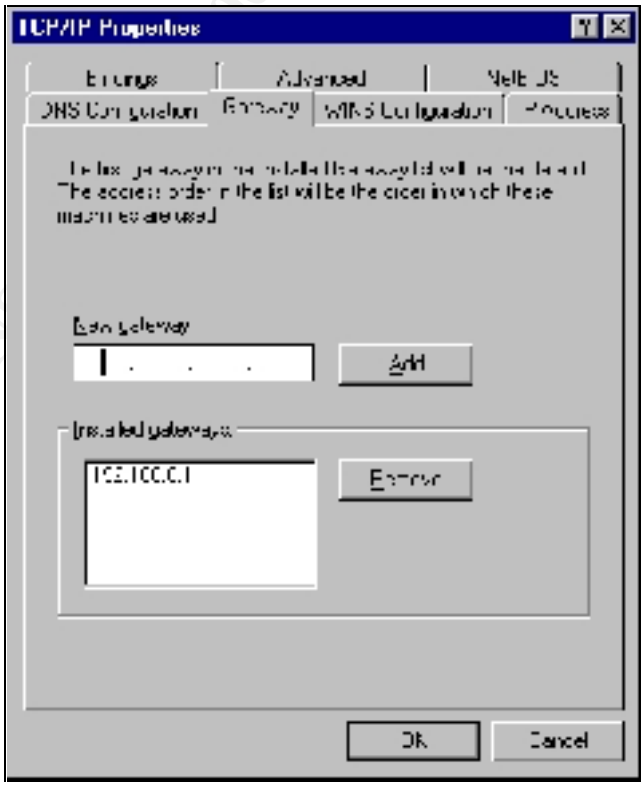
While I am not going to document the process required to configure the NetGear Router (very straightforward), I will document briefly the changes made to one of the PCs network configuration. I changed the network settings on each computer by going to *Start* | *Settings* | *Control Panel* and then opening *Control Panel* as shown below.

From this window, I double-clicked *Network* to open the *Network Configuration* dialog shown below.

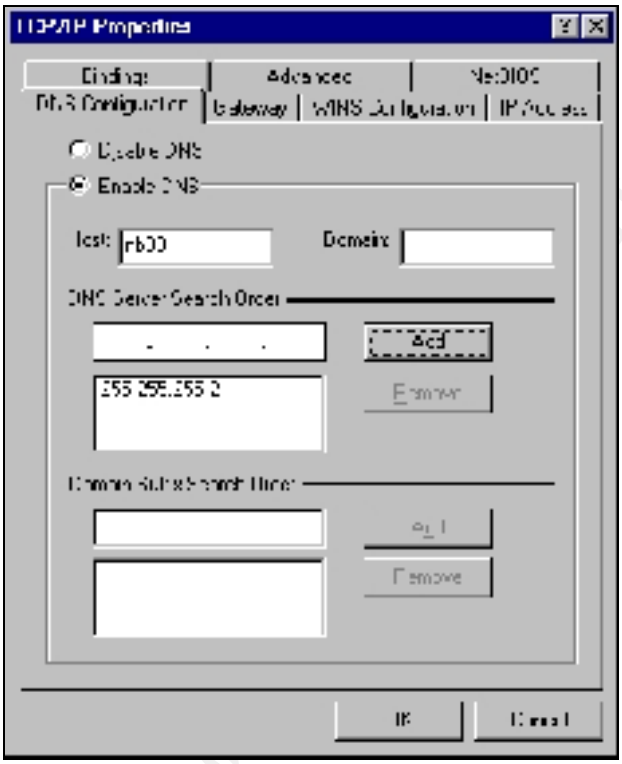


From this screen, I selected the *TCP/IP* settings for the Ethernet adapter (*TCP/IP -> Intel 21040 based Ethernet Controller*) and hit the *Properties* button. On the *IP Address* tab, I set the network properties for each machine as shown. The IPs used for the three machines were 192.168.0.101 (PC_1), 192.168.0.102 (PC_2), 192.168.0.103 (PC_3).

The *Gateway* for each of the machines was configured to be the IP address of the inside interface of the NetGear router. The IP of this interface is 192.168.0.1 as shown below.

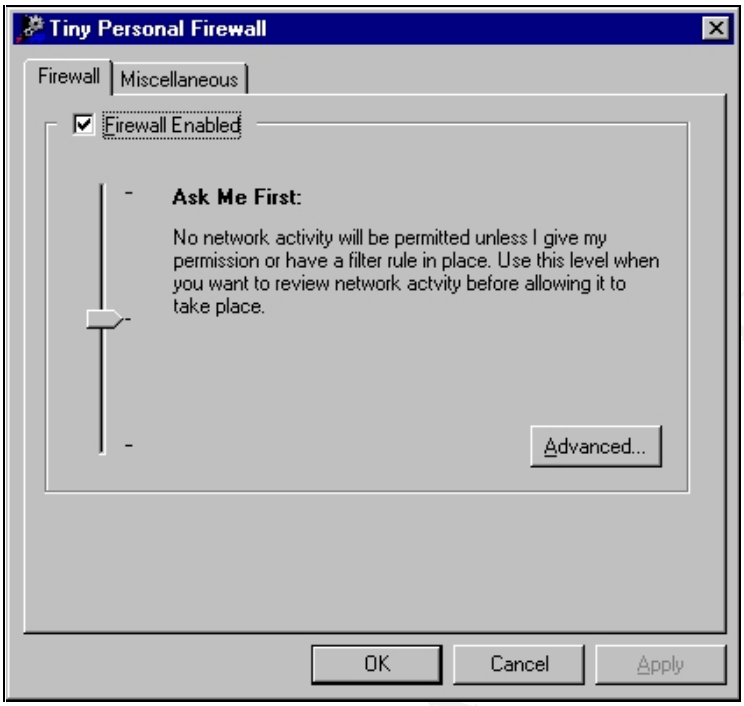


The *DNS Configuration* for each of the machines was also set to point to the owner's ISP for DNS resolution. Note the values here have been changed from those actually used (and is not even valid).

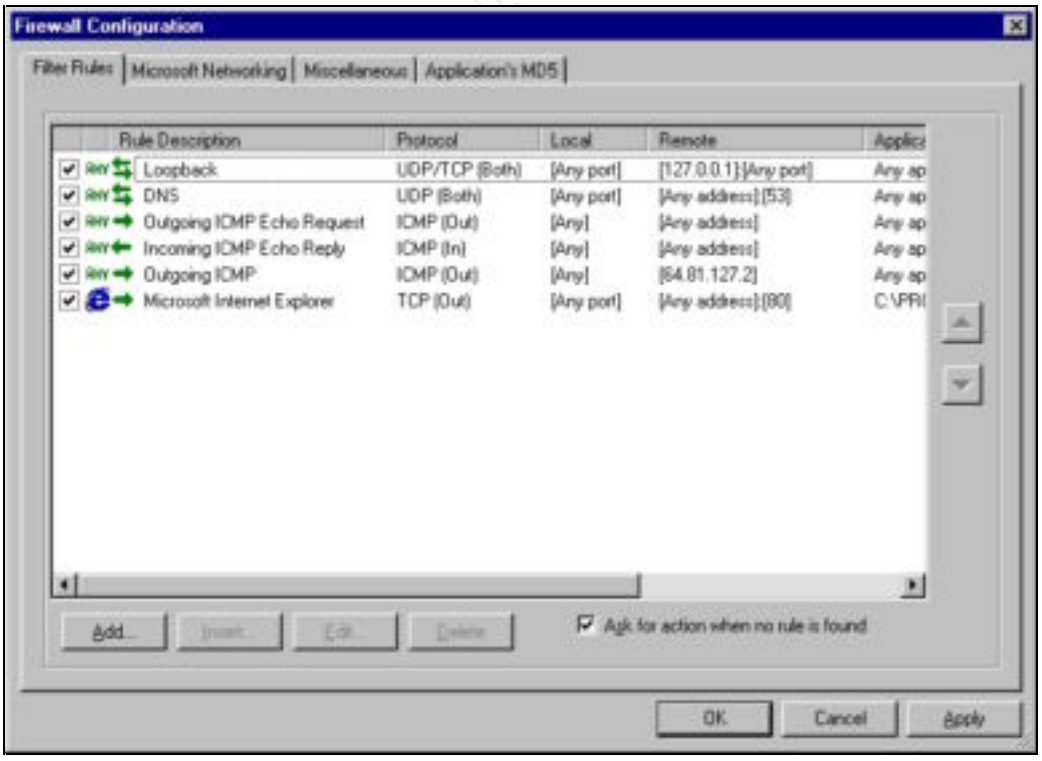


Now that the machines had been properly reconfigured, the machines were rebooted to allow the networking changes to take place.

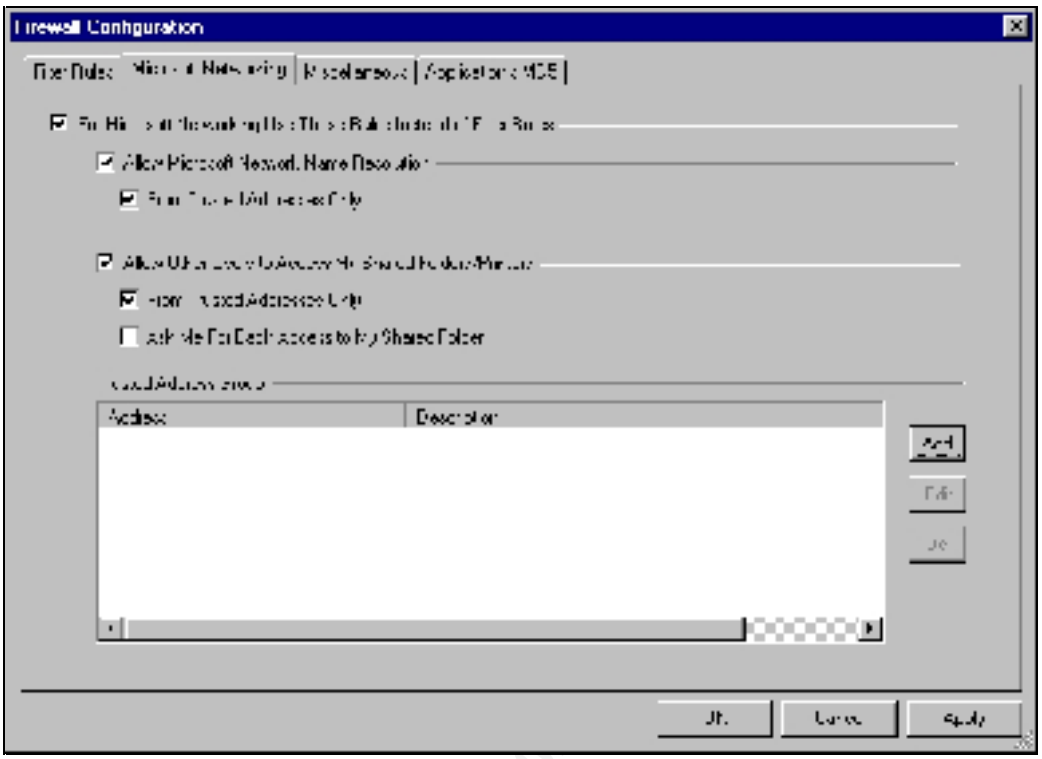
Third, I installed Tiny Personal Firewall on each of the three machines. I am not going to document the install process for the firewall, but it is relevant to discuss the rules that were put in place to further protect the Windows drive sharing exploited by the Bymer worm. Windows file sharing was protected using the steps I am about to outline. The process starts by right-clicking the Tiny Personal Firewall icon in the system tray (the are in the right hand corner with the clock) and selecting *Firewall Administration* from the menu. When this is done, the following screen will be displayed.



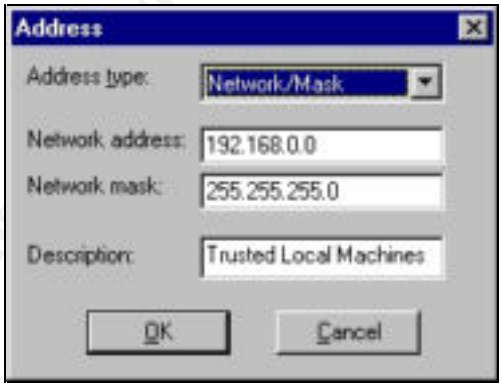
The next step is to click the *Advanced* button which takes the user to the *Firewall Configuration* dialog as shown below.



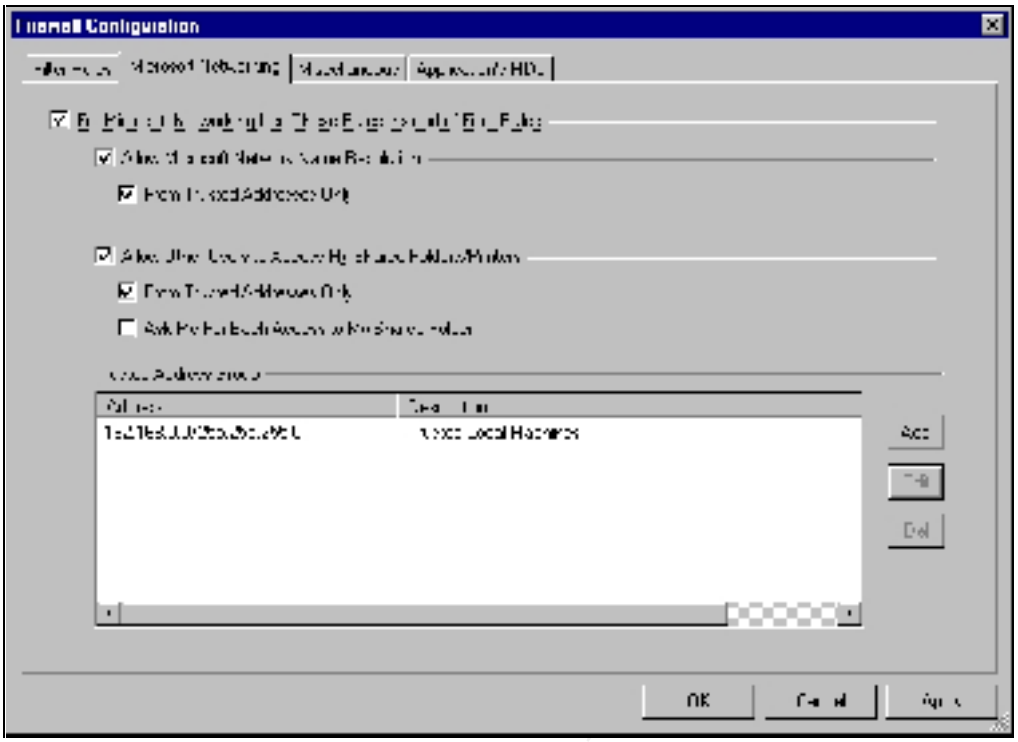
From this screen the *Microsoft Networking* tab should be selected resulting in the screen shown below.



The next step in this process is to check the boxes as indicated above and then click the *Add* button. The *Address* dialog shown below will be displayed.



This dialog shows the settings to allow file and printer sharing for all local machines (192.168.0.x). This could be further restricted, but in this instance I chose to open up the entire private class C. When *OK* is clicked, the user is returned to a dialog box as indicated below.



By clicking *OK*, the Tiny Personal Firewall is fully configured to only allow file and printer sharing to the trusted local network.

The final thing I wanted to do is make sure that the machine's McAfee VirusScan was properly configured. Consider the "*System Scan"* tab shown below (available after right-clicking the VShield icon in the tray, selecting *Status*, then clicking the *Properties* button) which shows a "secure" configuration.

The original VirusScan settings were not configured to scan files on "*Copy"*, "*Delete"*, or "*Rename"* (just "*Run"*). The new settings are more secure. I also changed "*What to scan"* to "*All files"* and unchecked the "*System scan can be disabled"* option. Once these changes were complete, I clicked the next tab -- "*E-Mail Scan"*.

I checked "*Enable Scanning of e-mail attachments*" and enabled "*Internet Mail*" before moving to the next tab – "*Download Scan*".

On "*Download Scan*", I configured VirusScan to "*Enable Internet download scanning*" for "*All files*" including compressed files. Next, I moved on to the "*Internet Filter*" tab.

Finally, to finish the VirusScan configuration, I clicked "*Enable Java & ActiveX filter*" including "*ActiveX Controls*" and "*Java-classes*". The other two options are enabled but not being used at this time. When configuration was complete, I clicked *OK* to enable the changes.

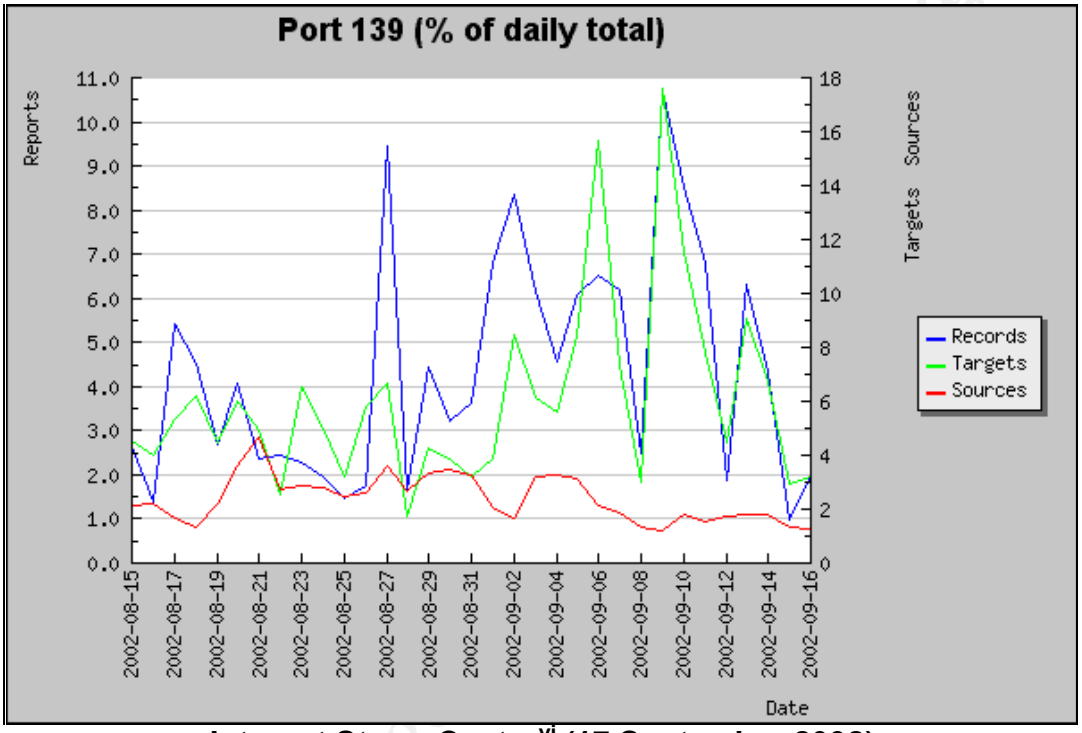
Lessons Learned

There were numerous lessons learned in the handling of the incident, the analysis of the worm, and in preparing this paper. Probably, the single most important lesson I learned from handling this incident is you can never let your guard down – not even for a couple of days or you can live to regret it.

I was aware of the fact that SDSL was being installed well before (at least a couple of weeks) before the installation occurred. Had this been a "paying" job, I would have definitely been more apt to be proactive in making sure the network was secured in preparation of the broadband installation. I was also ignorant of the extreme dangers posed by the Internet to the home user community and how fast these threats can become real attacks.

While I would hardly consider two compromises to be enough to generate a statistical average, both the Honeynet Project's network and the one involved in this incident were compromised in less than 24 hours of being online. The fact that the incident I am documenting occurred on 1 March 2002 shows the worm

had been going strong for 17 months (as of March). The sheer speed at which these machines were infected leads me to believe there is a fairly significant installed base of these worms already on the Internet. The following graph[18] from the Internet Storm Center (17 September 2002) shows that NetBIOS scans (ranking 4th) account for between 5 and 10 percent of the daily reported activity. Automated worms, like Bymer, account for most of this traffic.



Internet Storm Center[vi] (17 September 2002)

This event taught me that the Internet threat is real, even for the home user. You have to take action to secure the network now, because a few days from now when you have the time, it could be too late. It was lucky the Bymer worm was not too malicious, but it could have just as easily been a worm that was more malicious.

I hope you as the reader have enjoyed reading this paper as much as I did writing it. I learned a lot along the way and hope that in some way I have helped you to have a better understanding of how the Bymer worm (and other similar malicious mobile code) operates. The analysis, techniques, and tools presented in Appendix A will hopefully help you to better understand the Bymer worm and also provide an example how one might go about analyzing malicious mobile code.

[vi] Internet Storm Center. "Port Reports". 17 Sep 2002.

Appendix A: Malicious Code Analysis

One of my goals with the capture of the Bymer worm was to use it as an opportunity to further my understanding of malicious code analysis by taking it through a full analysis of the worm (or at least as much as I have the facilities to do). In doing so, I planned to do both a Static (without running the code) and Dynamic (running the code but not allowing to get to the Internet) Analysis of the worm. In doing this analysis I have relied on the works of others (notably the Snort[vii] logs captured by the Honeynet Project).[19] I also had a pretty good idea of what this code was going to be doing based on the information gathered from McAfee[viii] and Norton[20] summaries of the worm. I would also like to caveat this analysis by saying I am not a professional malicious code analyzer, but I hope this example will prove useful to someone who is getting started in malicious code analysis.

Static Analysis

There are a number of things you can find out about malicious code by hitting the search engines and the major anti-virus vendor websites. In the case of the Bymer worm it had been around for a while, so finding information on it using a search engine to be pretty effective.

Search Engine Analysis

I started my static analysis of the worm by doing additional research on the Internet about the worm. The best sources of information on the Bymer worm ended up being the McAfee and Norton websites, the Honeynet Project, and the Distributed.net site. Information from all these sites has been liberally referenced in completing this paper.

Strings Analysis

The second step in my static analysis was to run *strings.exe* against the *wininit.exe* binary to dump any strings that might be found in the executable that might provide clues as to how the code operates. The specific command I used was as follows and dumps the output to a file named *strings.txt*.

strings.exe wininit.exe > strings.txt

I went through the output of the previous command manually to see if there was anything "interesting" in the output that might provide further clues or guide my further analysis. I am going to show some of this output here that I felt was relevant to this investigation and discuss the implications of each.

[vii] Sourcefire, "Snort The Open Source Network Intrusion Detection System".
[viii] McAfee.com, "Virus Information Library".

This block of output from *strings* provides a wealth of information to the investigator. The first line tells us this is indeed a Windows program (but there are more accurate ways to identify this which I will describe later). The text indicating this executable has been packed with UPX 0.84 should alert an investigator to two things. First, because this executable has been compressed, a *strings* analysis is not going to be as beneficial to us as it would have been if the code were not compressed. We will be able to pick a few useful strings out of the output (as I will show), but most of the useful text is going to be obscured through the compression (but we will address this limitation later). Second, if the program that compressed the executable is known, then we can often go get the same executable and use it to decompress the malicious code. At this point, I just made a note of the UPX compression in the example below and continued on with my analysis using *strings*.

```
!This program cannot be run in DOS mode.
Rich
UPX0
UPX1
.rsrc
$Info: This file is packed with the UPX executable packer $
$Id: UPX 0.84 Copyright (C) 1996-1999 Laszlo Molnar & Markus Oberhumer
$
$Id: NRV 0.61 Copyright (C) 1996-1999 Markus F.X.J. Oberhumer $
$License: NRV for UPX is distributed under special license $
UPX!
```

This next section of output from *strings* is also useful. Here we can see a partial registry key entry that the Bymer worm is going to be using to start itself up when Windows loads.

```
bymer.scann
]^Software\Micros
\Wind
ows\Cur
ntV'sion\RunS
```

Here we can see a possible URL that may be important later in our investigation.

```
http://
boom.ru/
```

Next, we can see another header for a program that has also been compressed with UPX. It is highly unusual to see this kind of behavior in a “normal” program. This is a good indication that this program may also function as some sort of dropper (the program has another program inside of it that it will extract and run when the dropper is executed). Because we already know the Bymer worm installs a copy of the *dnetc.exe* program, we have a pretty good idea that this is the file is being carried inside this worm.

```
LThisV
qbe
DOS mode.
T@*}
Rich
NUPX0
w `3
Id: , 0.72 Copyv
h5(C) 1996-
zlo Molnar &
```

Here we can see some of the DLLs being used by the worm. We can make an educated guess about two things based on this list. First, Bymer is probably written in a high level language like C++ (no VB runtime files in this list) and that it will be accessing the Internet (*WININET.dll* and *WS2_32.dll*).

```
KERNEL32.DLL
ADVAPI32.dll
USER32.dll
WININET.dll
WS2_32.dll
```

UPX Analysis

The third step in my static analysis of the worm was to follow up on the information acquired in step two. Specifically, I knew from previous experience that UPX has a decompress option. If I could decompress the worm, I knew I might be able to find much more useful information using *strings*. I figured I would have the best luck with UPX version 0.84 (the one it was compressed with) so I found a copy of version 0.84 using a search on the Internet. I ran it with no arguments to get the command line options and have shown the output below.

```
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996, 1997, 1998, 1999
UPX v0.84        Markus F.X.J. Oberhumer & Laszlo Molnar         Oct
4th 1999

Usage: upx_84 [-123456789dlthVL] [-qvfk] [-o file] file..

Commands:
  -1     compress faster                   -9    compress better
  -d     decompress                        -l    list compressed exe
  -t     test compressed exe               -V    display version number
  -h     give more help                    -L    display software
license
Options:
  -q     be quiet                          -v    be verbose
  -oFILE write output to `FILE'
  -f     force overwrite of output files
  -k     keep backup files
  file.. executables to (de)compress
```

```
This version supports: dos/exe, dos/com, dos/sys, djgpp2/coff,
watcom/le,
                       win32/pe, rtm32/pe, tmt/adam, atari/tos,
linux/i386

UPX comes with ABSOLUTELY NO WARRANTY; for details type `upx -L'.
```

Seeing the –d flag was great. Now I knew that I would be able to extract the original file from the compressed worm. I ran the command to extract the original exe.

upx.exe –d –o wininit2.exe wininit.exe

```
                       Ultimate Packer for eXecutables
                     Copyright (C) 1996, 1997, 1998, 1999
UPX v0.84        Markus F.X.J. Oberhumer & Laszlo Molnar         Oct
4th 1999

        File size        Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx_84: wininit.exe: CantUnpackException: not yet implemented

Unpacked 1 file: 0 ok, 1 error.
```

Well, as you can see, decompression had not been implemented in UPX 0.84. I went to the UPX website[21] and downloaded the latest version of UPX (which was version 1.20). I again tried to decompress the worm with the following results.

```
                       Ultimate Packer for eXecutables
              Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001
UPX 1.20w        Markus F.X.J. Oberhumer & Laszlo Molnar        May
23rd 2001

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx_120: wininit.exe: CantUnpackException: this program is packed with
an obsole
te version and cannot be unpacked

Unpacked 0 files.
```

Not giving up, I went back to the UPX website and started reading their FAQs and forums. Buried several levels down in their website under the downloads section for obsolete versions, I found the following message. “Unpacking of win32/pe programs packed with the obsolete versions has never been implemented, so they won't help you here either”.[22] At this point, the static analysis portion of my investigation had pretty much ended. In order to gain any further data that would be useful, I needed to perform my dynamic analysis.

Dynamic Analysis

A dynamic analysis is very important to understanding the specifics of what a piece of malicious code does. Sure the search engines can provide a lot of useful information, but unless someone has done an in depth analysis of the worm and posted their results, it is probably going to be lacking the details. The first thing I want to document was the method the Bymer worm was using to spread. To fully document this, I really wanted to show the output from Snort as the worm was copying itself between two machines. Unfortunately, I do not have access to an adequate test network where I could simulate a network large enough to get the worm to pick one of my own random IPs (and I wasn't going to let this worm loose on the Internet). I considered using NAT to manipulate the packets, but didn't see the point with real world Snort logs available (see below).

Network Analysis

Fortunately for me, the Honeynet Project[ix] had been compromised by the Bymer worm(s) and had posted their logs and the worm binaries[23] to their website, so I was able to use four days of their Snort logs in my analysis. They have posted an excellent, but somewhat limited, analysis and paper[x] of their experience with the worms on their website. Note that the Honeynet Project was actually attacked by two variants of the Bymer worm. The version that attacked the network used in my incident is the second and more sophisticated of the two – the one named *wininit.exe* that carries a copy of *dnetc.exe* and it's *dnetc.ini* file inside the worm. Additionally, I verified that the worm I am analyzing is the same as the one that attacked them by verifying the MD5 checksum for the worm.

The Bymer worm begins it's attack by scanning the Internet using random IP addresses looking for machines that have open C drive shares. In the Honeynet Snort logs shown below, you can see the *wininit.exe* variant of the worm using an open drive share to connect to a vulnerable system. In this example, the worm is looking to see if the worm is already installed in the *c:\windows\system* directory.

```
11/02-21:41:09.754218 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:36827  DF
*****PA* Seq: 0x21CC068   Ack: 0xCE67344   Win: 0x21AC
00 00 00 40 FF 53 4D 42 08 00 00 00 00 00 01 00  ...@.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 04 EE 00 1D 00 04 5C 57 49 4E 44 4F 57 53  ........\WINDOWS
5C 53 59 53 54 45 4D 5C 57 49 4E 49 4E 49 54 2E  \SYSTEM\WININIT.
45 58 45 00                                      EXE.
```

If the worm is not already installed on the system, it will copy itself on to the system using the open drive share. In the example below, you can see the worm starting the process of copying itself to the victim system.

```
11/02-21:41:17.287743 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:38619  DF
```

[ix] Honeynet Project, "The Honeynet Project".
[x] Honeynet Project, "Know Your Enemy: Worms at War".

```
*****PA* Seq: 0x21CC0AC   Ack: 0xCE6736B   Win: 0x2185
00 00 00 5D FF 53 4D 42 2D 00 00 00 00 00 01 00  ...].SMB-.......
00 00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  ..............O.
00 00 84 EE 0F FF 00 00 00 07 00 91 00 16 00 20  ...............
00 20 BB 01 3A 10 00 00 00 00 00 00 00 00 00 00  . ..:...........
00 00 00 1C 00 5C 57 49 4E 44 4F 57 53 5C 53 59  .....\WINDOWS\SY
53 54 45 4D 5C 77 69 6E 69 6E 69 74 2E 65 78 65  STEM\wininit.exe
00                                               .
```

Next the worm begins the process of actually copying itself on to the target system. In the next example we can see a number of things. First, note the **MZ** – the first two characters of any valid Windows exe. You can also see the DOS headers indicating this program cannot be run in DOS. The **PE** that I have highlighted at the end of the example shows that this is a Windows Portable Executable (PE) file.

```
11/02-21:41:17.632426 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:38875  DF
******A* Seq: 0x21CC10D   Ack: 0xCE673B0   Win: 0x2140
00 00 0B 68 FF 53 4D 42 1D 00 00 00 00 00 01 00  ...h.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  ..............O.
00 00 04 EF 0C 0E 00 F0 FF 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 2C 0B 3C 00 2D 0B 00  .........,.<.-..
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............

0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
EA 05 D3 58 AE 64 BD 0B AE 64 BD 0B AE 64 BD 0B  ...X.d...d...d..

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00  ........PE..L...
```

Also note that you can see all the strings that were captured earlier in the static analysis in the Snort logs. I have put three examples of the strings being shown in the Snort logs as the binary worm is being transferred to the victim system below.

```
00 00 00 00 00 00 00 00 0A 00 24 49 6E 66 6F 3A  ..........$Info:
20 54 68 69 73 20 66 69 6C 65 20 69 73 20 70 61   This file is pa
63 6B 65 64 20 77 69 74 68 20 74 68 65 20 55 50  cked with the UP
58 20 65 78 65 63 75 74 61 62 6C 65 20 70 61 63  X executable pac
6B 65 72 20 24 0A 00 24 49 64 3A 20 55 50 58 20  ker $..$Id: UPX
30 2E 38 34 20 43 6F 70 79 72 69 67 68 74 20 28  0.84 Copyright (
43 29 20 31 39 39 36 2D 31 39 39 39 20 4C 61 73  C) 1996-1999 Las
7A 6C 6F 20 4D 6F 6C 6E 61 72 20 26 20 4D 61 72  zlo Molnar & Mar
6B 75 73 20 4F 62 65 72 68 75 6D 65 72 20 24 0A  kus Oberhumer $.
00 24 49 64 3A 20 4E 52 56 20 30 2E 36 31 20 43  .$Id: NRV 0.61 C
6F 70 79 72 69 67 68 74 20 28 43 29 20 31 39 39  opyright (C) 199
36 2D 31 39 39 39 20 4D 61 72 6B 75 73 20 46 2E  6-1999 Markus F.
58 2E 4A 2E 20 4F 62 65 72 68 75 6D 65 72 20 24  X.J. Oberhumer $
```

```
11/02-21:41:19.007539 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:42459  DF
******A* Seq: 0x21D0B35   Ack: 0xCE673D9   Win: 0x2117
E1 94 73 D9 5C 61 5F 61 01 6C C2 63 00 38 57 9E  ..s.\a_a.l.c.8W.
EA 95 DD 8D 38 74 2A 2C 57 50 27 C1 9C 61 4A 8E  ....8t*,WP'..aJ.
0D B2 18 9B 8D D0 45 72 0C B1 81 59 44 00 80 2C  ......Er...YD..,
45 81 00 07 95 FF AF 82 62 79 6D 65 72 2E 73 63  E.......bymer.sc
61 6E 6E DF FD FF 5D 5E 53 6F 66 74 77 61 72 65  ann...]^Software
5C 4D 69 63 72 6F 73 0D 5C 57 69 6E 64 FB DF DE  \Micros.\Wind...
FE 6F 77 73 5C 43 75 72 17 6E 74 56 27 73 69 6F  .ows\Cur.ntV'sio
6E 5C 52 75 6E 53 0A BB 51 D8 6D 76 26 65 73 37  n\RunS..Q.mv&es7
2F 6D 73 F7 0F 50 6F 1D 69 74 00 2A 5D 30 40 00  /ms..Po.it.*]0@.
71 03 DB 2E 80 6E 06 00 02 01 7F 03 06 09 02 FF  q....n..........
```

```
52 22 04 00 00 00 00 00 74 00 00 80 00 00 00 00  R"......t.......
4B 45 52 4E 45 4C 33 32 2E 44 4C 4C 00 41 44 56  KERNEL32.DLL.ADV
41 50 49 33 32 2E 64 6C 6C 00 55 53 45 52 33 32  API32.dll.USER32
2E 64 6C 6C 00 57 49 4E 49 4E 45 54 2E 64 6C 6C  .dll.WININET.dll
00 57 53 32 5F 33 32 2E 64 6C 6C 00 00 00 4C 6F  .WS2_32.dll...Lo
61 64 4C 69 62 72 61 72 79 41 00 00 47 65 74 50  adLibraryA..GetP
```

Once the worm is through copying itself to the victim system, it needs some way of making sure it gets run. In the case of the Bymer worm, it uses the *win.ini* file's load line to ensure that the worm is started when the machine is booted. In the example Snort logs below you can see that the worm is requesting the *c:\windows\win.ini* file from the victim system.

```
11/02-21:41:47.754427 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:19932  DF
*****PA* Seq: 0x220213C   Ack: 0xCE6751D   Win: 0x1FD3
00 00 00 52 FF 53 4D 42 2D 00 00 00 00 00 01 00  ...R.SMB-.......
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 84 F3 0F FF 00 00 00 07 00 A2 00 16 00 00  ................
00 3E BB 01 3A 01 00 00 00 00 00 00 00 00 00 00  .>..:...........
00 00 00 11 00 5C 57 49 4E 44 4F 57 53 5C 77 69  .....\WINDOWS\wi
6E 2E 69 6E 69 00                                n.ini.
```

Here we can see the victim system responding to the worm by sending a copy of its *win.ini* file. Note that the load line already has an entry in it. *Msi216.exe* is actually one of the other variants of the Bymer worm. In the case of the Honeynet computer, it had already been infected with the earlier variant of the worm[xi].

```
11/02-21:41:48.002536 172.16.1.105:139 -> 216.234.204.69:2021
TCP TTL:127 TOS:0x0 ID:9740  DF
******A* Seq: 0xCE67562   Ack: 0x22021C9   Win: 0x1E28
00 00 19 61 5B 77 69 6E 64 6F 77 73 5D 0D 0A 6C  ...a[windows]..l
6F 61 64 3D 63 3A 5C 77 69 6E 64 6F 77 73 5C 73  oad=c:\windows\s
79 73 74 65 6D 5C 6D 73 69 32 31 36 2E 65 78 65  ystem\msi216.exe
0D 0A 72 75 6E 3D 0D 0A 4E 75 6C 6C 50 6F 72 74  ..run=..NullPort
```

[xi] Honeynet Project, "Know Your Enemy: Worms at War".

```
3D 4E 6F 6E 65 0D 0A 0D 0A 5B 44 65 73 6B 74 6F  =None....[Deskto
70 5D 0D 0A 57 61 6C 6C 70 61 70 65 72 3D 28 4E  p]..Wallpaper=(N
6F 6E 65 29 0D 0A 54 69 6C 65 57 61 6C 6C 70 61  one)..TileWallpa
70 65 72 3D 31 0D 0A 57 61 6C 6C 70 61 70 65 72  per=1..Wallpaper
53 74 79 6C 65 3D 30 0D 0A 0D 0A 5B 69 6E 74 6C  Style=0....[intl
```

Bymer is done infecting the system once it has copied a newly modified version of the *win.ini* file back to the victim's computer. This ensures that the worm will be executed the next time the system is rebooted. This is shown in the example below. Note that in the Honeynet's case, the new worm did not remove the previous version of the worm.

```
11/02-21:41:48.538643 216.234.204.69:2021 -> 172.16.1.105:139
TCP TTL:113 TOS:0x0 ID:21212  DF
******A* Seq: 0x22021C9   Ack: 0xCE68EC7   Win: 0x1FA3
00 00 0B 68 FF 53 4D 42 1D 00 00 00 00 00 01 00  ...h.SMB........
00 00 00 00 00 00 00 00 00 00 00 00 D0 4F 1F  .............O.
00 00 84 F4 0C 0F 00 7F 19 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 2C 0B 3C 00 2D 0B 00  .........,.<.-..
5B 77 69 6E 64 6F 77 73 5D 0D 0A 6C 6F 61 64 3D  [windows]..load=
63 3A 5C 77 69 6E 64 6F 77 73 5C 73 79 73 74 65  c:\windows\syste
6D 5C 77 69 6E 69 6E 69 74 2E 65 78 65 20 63 3A  m\wininit.exe c:
5C 77 69 6E 64 6F 77 73 5C 73 79 73 74 65 6D 5C  \windows\system\
6D 73 69 32 31 36 2E 65 78 65 0D 0A 72 75 6E 3D  msi216.exe..run=
0D 0A 4E 75 6C 6C 50 6F 72 74 3D 4E 6F 6E 65 0D  ..NullPort=None.
0A 0D 0A 5B 44 65 73 6B 74 6F 70 5D 0D 0A 57 61  ...[Desktop]..Wa
6C 6C 70 61 70 65 72 3D 28 4E 6F 6E 65 29 0D 0A  llpaper=(None)..
54 69 6C 65 57 61 6C 6C 70 61 70 65 72 3D 31 0D  TileWallpaper=1.
0A 57 61 6C 6C 70 61 70 65 72 53 74 79 6C 65 3D  .WallpaperStyle=
```

Run-Time Analysis

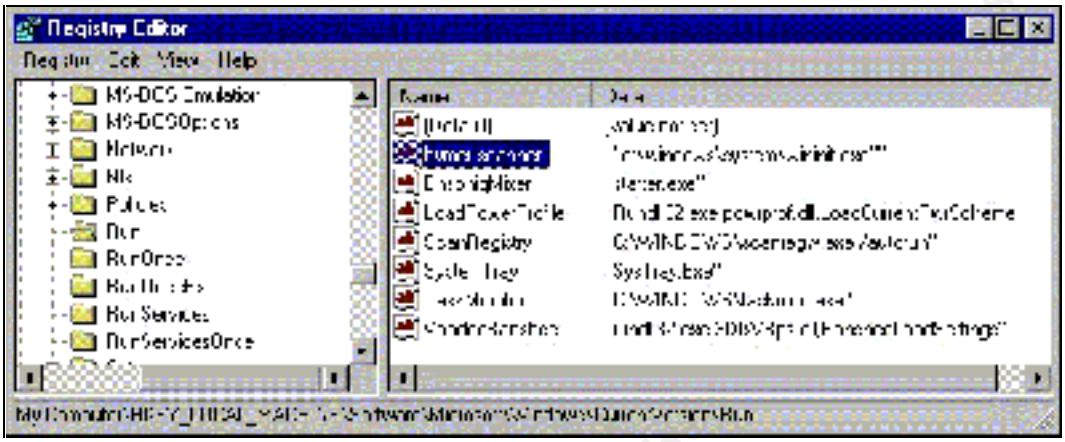
The next step of my analysis of the worm was to determine what the worm does once it is activated on the target system. In the case of the Bymer worm, it is easy to simulate this event. I simply copied *wininit.exe* to *c:\windows\system\wininit.exe* just like the worm would have done. I added my own *win.ini* entry to start the worm automatically and rebooted. The change I made is documented below.

```
[windows]
load=c:\windows\system\wininit.exe
```

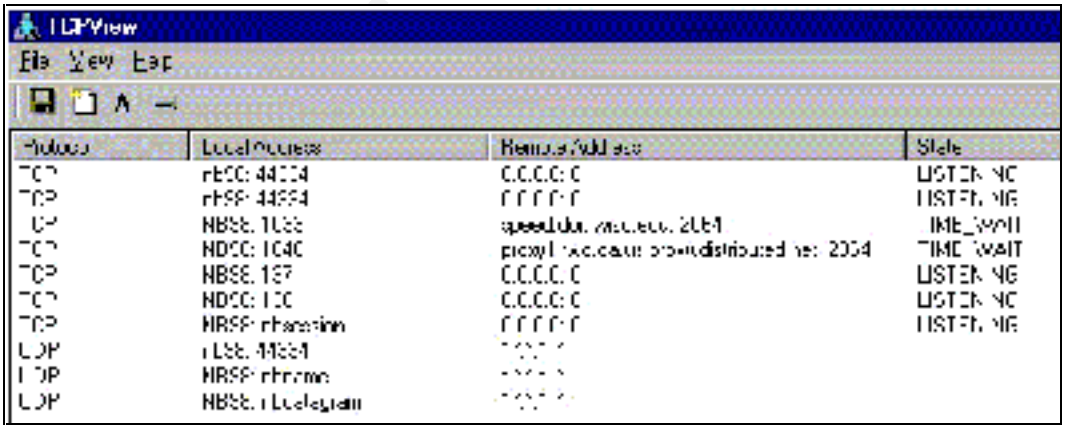
When the machine rebooted, the worm made a number of changes to the system. The first thing I noticed it did was removing itself from the *win.ini* file's load line. When the worm was done, the *win.ini* file looked like the example below.

```
[windows]
load=
```

The worm is not going to disable its only way of starting, so it also added itself to the registry to start automatically when rebooted. You can see where the worm added itself in the screen capture from *regedit.exe* below. Note the name of the registry key that is highlighted.



As mentioned previously, the *wininit.exe* version of Bymer caries it's own copy of *dnetc.exe* and *dnetc.ini*, which it then drops in the *c:\windows\system* directory. The worm launches *dnetc.exe* with the command line flags (shown later in analysis) that make *dnetc.exe* run hidden and install itself as a service to start automatically (*dnetc.exe* will create it's own registry entry to start itself automatically). Once *dnetc.exe* is fully configured it will attempt to access the Internet and connect to one of the Distributed.net key servers to download a block of keys to start cracking. You can see the *dnetc.exe* client attempting to contact these servers in the screen capture form *TCPView.exe*[xii] below.



If you examine the *dnetc.ini* file that the worm dropped, you can see another one of the reasons this worm is called the Bymer worm. Notice the email address that is in the *dnetc.ini* file below. This is the person who will get credit for the

[xii] Sysinternals, "TCPview".

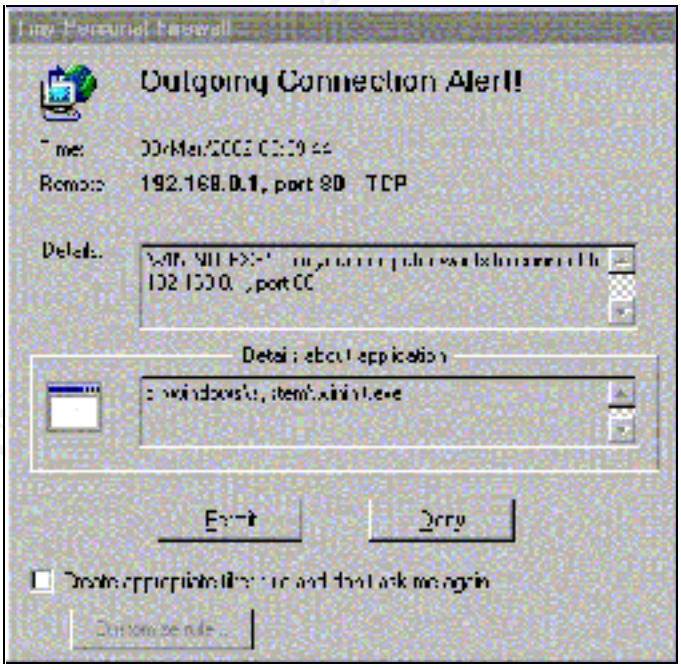
work done by any of the Distributed.net clients (at least until Distributed.net banned his / her email address for life).[xiii]

```
[parameters]
id=bymer@ukrpost.net
```

Another thing the worm did, was create a log (*wininit.log*) that would record how many systems the worm scanned while looking for new hosts to compromise (maintained across reboots as shown below). You can see a capture from my *wininit.log* file below. Note the worm cannot find hosts to compromise because it's Internet connection is being blocked by Tiny Personal Firewall[24] running on my test machine.

```
Started at 10:36 3.03.2002
Stopped (scanned 5, found 0) at 10:38 3.03.2002
Started at 10:39 3.03.2002
Stopped (scanned 170, found 0) at 11:47 3.03.2002
Started at 11:48 3.03.2002
```

Another very curious behavior I saw exhibited by the worm was that it tries to make a connection to 192.168.0.1 on port 80 (which happens to be the default gateway for my home network's DSL firewall / router). The popup from Tiny Personal Firewall is shown below.



I was actually quite confused by this behavior until I saw the analysis made by the Honeynet team[xiv] (I had not figured out how to do a memory dump yet). The

[xiii] Honeynet Project, "Know Your Enemy: Worms at War".
[xiv] Honeynet Project, "Know Your Enemy: Worms at War".

worm is not actually trying to connect to 192.168.0.1, it trying to connect to bymer.boom.ru (you may remember we saw the boom.ru in the strings output earlier). This address now resolves to the private address 192.168.0.1 due to the worm author's attempt (as suggested by the Honeynet team) to deactivate some feature of the original worm. You can see that the name *bymer.boom.ru* does indeed resolve to the private address shown in the partial nslookup output below.

```
Non-authoritative answer:
Name:    bymer.boom.ru
Address:  192.168.0.1
```

Memory Analysis

I continued my analysis of the Bymer worm by taking a look at what the exe was doing in memory. Because the worm was compressed, a lot of useful information was not available for static analysis but it could be available in memory while the worm is running. The first thing I tried was to get a list of all the DLLs that the worm was using the freeware utility *ListDLLs.exe*.[25] This list is shown below.

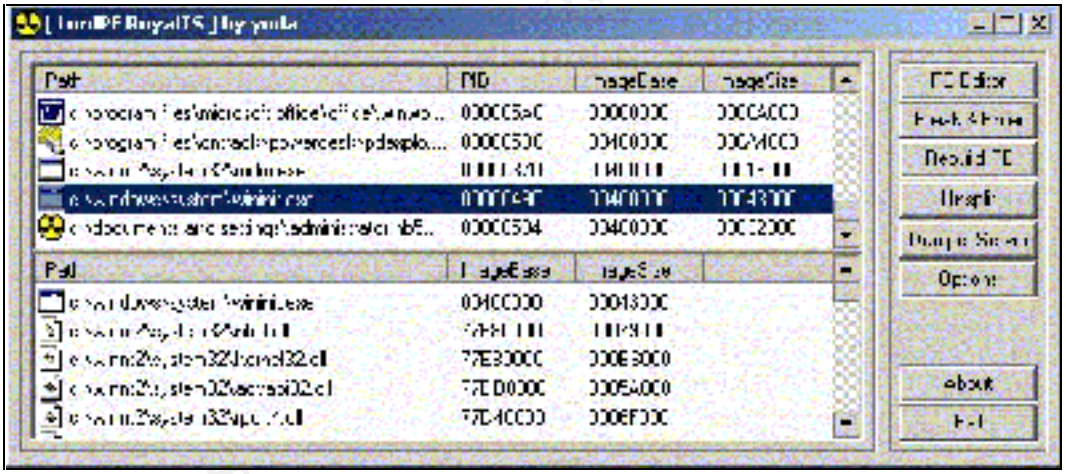
```
C:\WINDOWS\SYSTEM\WININIT.EXE pid: FFFD0EC5
  Base        Size      Version             Path
  0x7f990000  0x5000    4.10.0000.1998  C:\WINDOWS\SYSTEM\NETAPI32.DLL
  0x7f840000  0x8000                    C:\WINDOWS\SYSTEM\NETBIOS.DLL
  0x7fb90000  0x52000   4.71.2900.0002  C:\WINDOWS\SYSTEM\RPCRT4.DLL
  0x77c50000  0x9000    5.00.2614.3500  C:\WINDOWS\SYSTEM\SHFOLDER.DLL
  0x7b410000  0xb000    4.10.0000.1998  C:\WINDOWS\SYSTEM\MSAFD.DLL
  0x75fa0000  0xa000    4.10.0000.1998  C:\WINDOWS\SYSTEM\WSOCK32.DLL
  0x794d0000  0x15000   4.10.0000.2222  C:\WINDOWS\SYSTEM\MSWSOCK.DLL
  0x783c0000  0xf000    4.10.0000.2222  C:\WINDOWS\SYSTEM\RNR20.DLL
  0x00400000  0x43000   1.00.0000.0001  C:\WINDOWS\SYSTEM\WININIT.EXE
  0x76000000  0x12000   4.10.0000.2222  C:\WINDOWS\SYSTEM\WS2_32.DLL
  0x75fe0000  0x6000    4.10.0000.1998  C:\WINDOWS\SYSTEM\WS2HELP.DLL
  0x78000000  0x40000   6.00.8397.0000  C:\WINDOWS\SYSTEM\MSVCRT.DLL
  0x76280000  0x70000   5.00.2614.3500  C:\WINDOWS\SYSTEM\WININET.DLL
  0x70bd0000  0x44000   5.00.2614.3500  C:\WINDOWS\SYSTEM\SHLWAPI.DLL
  0xbff50000  0x11000   4.10.0000.2222  C:\WINDOWS\SYSTEM\USER32.DLL
  0xbff20000  0x26000   4.10.0000.1998  C:\WINDOWS\SYSTEM\GDI32.DLL
  0xbfe80000  0x10000   4.80.0000.1675  C:\WINDOWS\SYSTEM\ADVAPI32.DLL
  0xbff70000  0x73000   4.10.0000.2222  C:\WINDOWS\SYSTEM\KERNEL32.DLL
```

I also attempted to dump the contents of memory from the worm in Windows 2000 using the freeware utility *pmdump.exe*.[26] This proved to be of some value, but the file it dumped was over 12 megs in size. Most of this was Microsoft DLLs in use by the program. I was able to extract some interesting information using strings and *egrep* – notably a number of the host IPs the worm was trying to attack and the Windows connection strings it was using in the attacks. A partial listing of some of the information I extracted is in the example below.

```
C:\Windows\System\WININIT.EXE
\??\UNC\100.140.34.212\c\windows\system\wininit.exe
```

```
\??\UNC\110.21.239.81\c\windows\system\wininit.exe
\??\UNC\192.94.81.176\c\windows\system\wininit.exe
\??\UNC\228.182.57.27\c\windows\system\wininit.exe
\??\UNC\55.57.142.171\c\windows\system\wininit.exe
\\100.140.34.212\c\windows\system\wininit.exe
\\110.21.239.81\c\windows\system\wininit.exe
\\192.94.81.176\c\windows\system\wininit.exe
\\228.182.57.27\c\windows\system\wininit.exe
\\55.57.142.171\c\windows\system\wininit.exe
bymer.scanner
2\dnetc.ex
dnetc
dnetc.exe
distributed.net: dnetware
```

I also finally figured out how to dump the running program pack to a PE file so that I could view the file without the compression (it is not executable after dumping). The tool I used to do this is LordPE.[27] When it was done dumping the running program, I had a file named *dumped.exe* that was 274,432 bytes in size (compared to the original *wininit.exe* that was 220,672 bytes in size). I have included a capture of the LordPE window showing the running *wininit.exe* before it was dumped.



There is some really useful information that makes up these additional bytes retrieved by using LordPE. I ran the strings command

strings.exe dumped.exe > dumped.txt

and some of the more interesting output is shown in the excerpts below. I would like to comment on a few of these lines and what they mean. First, you can see two registry keys in the exe. I have never seen it register itself as a service, but it obviously has some capability to do so.

```
bymer.scanner
Software\Microsoft\Windows\CurrentVersion\RunServices
Software\Microsoft\Windows\CurrentVersion\Run
```

```
msinit
```

Also, we see the Visual C++ Runtime Library header – that is a pretty good indicator that the worm was written in Visual C++.

```
Microsoft Visual C++ Runtime Library
Runtime Error!
Program:
```

You can see the *dnetc.ini* file is contained within the executable (and it is not shown here but the other UPX block shown earlier is the compressed version of *dnetc.exe*).

```
[parameters]
id=bymer@ukrpost.net
[misc]
project-priority=OGR,RC5,CSC,DES
[rc5]
fetch-workunit-threshold=64
[ogr]
fetch-workunit-threshold=16
[triggers]
restart-on-config-file-change=yes
```

The two http lines are pretty interesting. It appears this worm is trying to download some type of .ini file (project.ini) as well. There was obvious some other capability this worm had before the author disabled the DNS names being used by the worm. I have never seen the worm try to connect to the second address (*xq.chat.ru*), but this name now also resolves to a private address, in this case 127.0.0.1 (which is the private loop-back address for a machine).

```
http://bymer.boom.ru/project.ini
http://xq.chat.ru/project.ini
```

You can also see the command line switches being sent in to *dnetc.exe* to make it run hidden and install itself as a service.

```
dnetc.exe -hide
 -install
```

The dump also has the complete path to the *wininit.log* file that is produced by the worm along with the string formatting instructions for the log file entries.

```
%s at %d:%02d %d.%02d.%d
Stopped (scanned %d, found %d)
\\%d.%d.%d.%d\c\windows\
C:\WINNT2\System32\wininit.log
```

Acknowledgements

I would like to give credit to two people who helped me as I was learning the basics of malicious code analysis. While I didn't rely on any of their information directly in writing my paper, the techniques and tools that I picked up from them proved invaluable to me in doing my analysis.

Lenny Zeltser

Lenny Zelester is actively involved in the SANS community (including co-authoring on of the newest SANS books). He holds several SANS certifications, and his GCIH practical "Reverse-Engineering Malware" (http://www.zeltser.com/sans/gcih-practical/) is an extremely useful paper on for continuing Malware analysis beyond what my paper presents. I was fortunate enough to be in attendance at his SANS 2002 (Orland, Florida) presentation, also titled "Reverse-Engineering Malware", and was able to talk to him in person after the presentation. I would personally like to thank him for turning me on to LordPE and answering a few other questions I had about dealing with compressed/encrypted code analysis; without his suggestions for tools to use, much of my analysis would have been impossible.

Professor Thomas C. Ervin

Professor Ervin teaches the NS677 Malicious Software as part of the Capitol College (http://www.capitol-college.edu/academics/grad/) Masters of Science in Network Security curriculum. His lectures, presentations, and labs provided incredible insight into the inner workings of malicious code and malicious code analysis.

References

[1] Distributed.net. “the organization”. 20 Jun 2002. URL: http://www.distributed.net/ (20 Sep 2002).

[2] Distributed.net. “trojans, worms, viruses”. 17 Jul 2002. URL: http://www.distributed.net/trojans.html (20 Sep 2002).

[3] Distributed.net. “project rc5”. 3 Oct 2000. URL: http://www.distributed.net/rc5/ (20 Sep 2002).

[4] Distributed.net. “client download”. 7 Dec 2001. URL: http://www.distributed.net/download/clients.html (20 Sep 2002).

[5] McAfee.com. “Virus Information Library”. 28 Sep 2000. URL: http://vil.mcafee.com/dispVirus.asp?virus_k=98844 (20 Sep 2002).

[6] MITRE Corporation. “Common Vulnerabilities and Exposures”. 6 Sep 2002. URL: http://www.cve.mitre.org/ (20 Sep 2002).

[7] MITRE Corporation. “CAN-1999-0518 (under review)”. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0518 (20 Sep 2002).

[8] MITRE Corporation. “CAN-1999-0519 (under review)”. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0519 (20 Sep 2002).

[9] MITRE Corporation. “CAN-1999-0520 (under review)”. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0520 (20 Sep 2002).

[10] SANS Institute. “The Twenty Most Critical Internet Security Vulnerabilities (Updated): The Experts’ Consensus”. 2 May 2002. URL: http://www.sans.org/top20.htm (20 Sep 2002).

[11] CERT Coordination Center (Carnegie Mellon University). “Exploitation of Unprotected Windows Networking Shares”. 7 Apr 2000. URL: http://www.cert.org/incident_notes/IN-2000-02.html (20 Sep 2002).

[12] Grimes, Roger A. “Malicious Mobile Code”. Sebastopol. O’Reilly & Associates, Inc. 2001. 1 - 128, 180 – 225.

[13] NeonSurge. “Understanding NetBIOS”. URL: http://packetstorm.linuxsecurity.com/groups/rhino9/netbios.doc (20 Sep 2002).

[14] Honeynet Project. “Know Your Enemy: Worms at War”. 9 Nov 2000. URL: http://project.honeynet.org/papers/worm/ (20 Sep 2002).

[15] Sysinternals. “TCPview”. 10 Aug 2002. URL: http://www.sysinternals.com/ntw2k/source/tcpview.shtml (20 Sep 2002).

[16] Sourcefire. “Snort The Open Source Network Intrusion Detection System”. URL: http://www.snort.org (20 Sep 2002).

[17] Microsoft Corporation. “Microsoft Security Bulletin (MS00-072)”. 10 Oct 2000. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-072.asp (20 Sep 2002).

[18] Internet Storm Center. “Port Reports”. 17 Sep 2002. URL: http://isc.incidents.org/port_details.html?port=139 (17 Sep 2002).

[19] Honeynet Project. “The Honeynet Project”. URL: http://project.honeynet.org (20 Sep 2002).

[20] Symantec. “Symantec Security Response: W32.HLLW.Bymer”. 9 Oct 2000. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.bymer.html (20 Sep 2002).

[21] Oberhumer, Markus F.X.J. & Molnár, László. “UPX: the Ultimate Packer for eXecutables”. 6 Sep 2002. URL: http://upx.sourceforge.net (20 Sep 2002).

[22] Oberhumer, Markus F.X.J. & Molnár, László. “Index of /download/00-OLD-VERSIONS/obsolete”. URL: http://upx.sourceforge.net/download/00-OLD-VERSIONS/obsolete/ (20 Sep 2002).

[23] Honeynet Project. “win98.tar.gz” (Snort Logs and Worm Binaries). 9 Nov 2000. URL: http://stan.ksni.net/~lance/win98.tar.gz (20 Sep 2002).

[24] Tiny Software. “Tiny Personal Firewall”. http://www.tinysoftware.com (20 Sep 2002).

[25] Sysinternals. “ListDLLs”. 27 Jun 2000. URL: http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml (20 Sep 2002).

[26] Ntsecurity.nu. “PMDump”. URL: http://ntsecurity.nu/toolbox/pmdump/ (20 Sep 2002).

[27] Reverser’s Fortress. “Utilities: EXE utilities” URL: http://linux20368.dn.net/protools/utilities.htm#lordpe (20 Sep 2002).

Last Updated: October 19th, 2017

Upcoming Training

Community SANS Columbus SEC504	Columbus, OH	Oct 23, 2017 - Oct 28, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, United Arab Emirates	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, Netherlands	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, Italy	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS New York SEC504^	New York, NY	Nov 06, 2017 - Nov 11, 2017	Community SANS
Mentor Session AW - SEC504	Houston, TX	Nov 06, 2017 - Jan 29, 2018	Mentor
SANS Miami 2017	Miami, FL	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Nov 06, 2017 - Nov 11, 2017	Community SANS
Pen Test Hackfest Summit & Training 2017	Bethesda, MD	Nov 13, 2017 - Nov 20, 2017	Live Event
Community SANS Toronto SEC504	Toronto, ON	Nov 13, 2017 - Nov 18, 2017	Community SANS
Mentor Session SEC504	Houston, TX	Nov 13, 2017 - Dec 11, 2017	Mentor
SANS Sydney 2017	Sydney, Australia	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS San Francisco Winter 2017	San Francisco, CA	Nov 27, 2017 - Dec 02, 2017	Live Event
Community SANS Detroit SEC504~	Detroit, MI	Nov 27, 2017 - Dec 02, 2017	Community SANS
SANS London November 2017	London, United Kingdom	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS Austin Winter 2017	Austin, TX	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Frankfurt 2017	Frankfurt, Germany	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Cyber Defense Initiative 2017 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Washington, DC	Dec 14, 2017 - Dec 19, 2017	vLive
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
Community SANS Honolulu SEC504	Honolulu, HI	Jan 08, 2018 - Jan 13, 2018	Community SANS
Mentor Session - SEC504	San Antonio, TX	Jan 09, 2018 - Mar 13, 2018	Mentor
SANS Amsterdam January 2018	Amsterdam, Netherlands	Jan 15, 2018 - Jan 20, 2018	Live Event
Community SANS Ottawa SEC504	Ottawa, ON	Jan 15, 2018 - Jan 20, 2018	Community SANS
Northern VA Winter - Reston 2018	Reston, VA	Jan 15, 2018 - Jan 20, 2018	Live Event
Community SANS St Louis SEC504	St Louis, MO	Jan 15, 2018 - Jan 20, 2018	Community SANS
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	SEC504 - 201801,	Jan 16, 2018 - Feb 22, 2018	vLive
SANS Dubai 2018	Dubai, United Arab Emirates	Jan 27, 2018 - Feb 01, 2018	Live Event