Reverse Engineering Malware

Lenny Zeltser

www.zeltser.com
May 2001

Abstract: This document discusses tools and techniques useful for understanding inner workings of malware such as viruses, worms, and trojans. We describe an approach to setting up inexpensive and flexible laboratory environment using virtual workstation software such as VMware, and demonstrate the process of reverse engineering a trojan using a range of system monitoring tools in conjunction with a disassembler and a debugger. Portions of this document are based on the paper that we submitted to GIAC as part of a practical assignment for obtaining a GCIH Certification. For more information see http://www.zeltser.com

Table of Contents

Section 1: Introduction

1.1   Overview

This paper attempts to document an approach to reverse engineering malicious software. The reason for highlighting the process itself, instead of concentrating solely on specifics of the program is two-fold. First, there are still many unanswered questions about the particular trojan discussed in this write-up (srvcp.exe); positioning our findings as comprehensive analysis would be misleading at best. Second, repeatable forensics steps should assist members of the defense community in developing a structured approach to understanding inner-workings of malicious software.

In the process of analyzing the trojan we have become impressed with programming efforts that attackers are willing to go through to create resilient and flexible malware. As the result of these efforts, the process of reverse engineering the program was time consuming yet fulfilling. We hope that our discussion will encourage security professionals to improve upon methodology suggested in this paper, and possibly fill in the gaps in our understanding of this particular trojan.

We would like to thank Joe Abrams for his insights regarding the trojan's assembly code, as well as for explaining the context of its existence in the wild. Many thanks to Slava Frid, who helped us in stepping through particularly cryptic areas of the program's code, and for making his mind available for our picking. Also, we thank Doug Kahler for sharing with us the trojan along with his observations. Finally, thanks to Jeremy Gaddis for bringing this trojan to the attention of the defense community.

1.2   Background Information

A variant of the srvcp.exe trojan, discussed in this document, was brought to the attention of the defense community by Jeremy L. Gaddis on 8 June 2000. In his posting to the Incidents mailing list, Jeremy reported noticing inbound connection attempts to TCP port 113 from an unknown host on the Internet, as well as unauthorized outbound connection attempts to a remote server on destination TCP port 6667.[JLG] Investigating the incident, Jeremy discovered the trojan's ties to an Internet Relay Chat (IRC) network, as described in his message:

> Knowing that 6667 is an often used port for IRC servers, I started up an IRC client and connected to that host. It was indeed an IRC server. A quick check showed 4 users online with the "Real Name" field set to "Im trojaned", one of which was my IP address.

For those not familiar with IRC, let us mention that IRC can be viewed as a network of interlinked servers that allows users to hold real-time online conversations. Participants of a conversation typically join a channel devoted to a particular topic or interest. This is accomplished by having the user's IRC client connect to a server that participates in the desired IRC peering network. When a user types a message meant to be seen by channel participants, it is relayed to the IRC server, and the server resends the message to participating clients, as specified in the Request for Comments (RFC) document 1459.[RFC1459]

A few days after Jeremy's initial message, Brandon Kittler also relayed his experience regarding this trojan in his posting to the mailing list on 10 June 2000.[BK] Brandon supplied details regarding the location of the program's executable and the associated registry entry. Based on his observations and examination of strings present in the executable, he concluded that the trojan was able to receive commands potentially dangerous commands via IRC:

```
0054C7 PRIVMSG %s :successfully spawned ftp.exe
0054F1 PRIVMSG %s :couldn't spawn ftp.exe
005515 PRIVMSG %s :no more...
00552D PRIVMSG %s :ready and willing...
```

Brandon also pointed out that the program listened on port 113 for Ident requests, and crashed when it received requests that did not follow the Ident protocol. The Ident service, whose protocol is defined in RFC 1413, allows a remote server to determine the login name of the user who initiated a TCP connection to the server.[RFC1413] In an attempt to control system abuse, many IRC servers do not accept IRC connections unless the connecting user's identity can be reconfirmed through the use of the Ident protocol.

Additional information regarding the program's behavior was reported by Doug Kahler a couple of days later on 12 June 2001.[DK] Doug mentioned that he ran into a similar trojan a few months earlier, and pointed out that at the time the program attempted to connect to an IRC server on the EFnet network to join a channel named "#mikag" with the key "soup". Doug also wrote:

> Just joined the channel today, and there are 55 people in there with nicks of random letters and numbers. I assume they are all infected.

Doug would later be kind enough to search through his archives and share with us a copy of the trojan that he based his analysis on. This was critically helpful for our research, since the executable posted to the mailing list in the beginning of the thread turned out to be a benign copy of what seemed to be a Windows screen saver.

As Nick FitzGerald pointed out in a message to the Incidents mailing list on 13 June 2001, the program seemed to be a variant of the relatively unknown Tasmer trojan.[NF] Nick confirmed that in his experience the trojan joined an IRC channel, and could be "remotely controlled via private messaging over IRC." According to Nick, some Tasmer variants were suspected of having distributed password cracking capabilities, and possible Denial of Service (DoS) attack agents.

Anti-virus vendors did not provide much information about this particular trojan, and later examination of virus databases disclosed lack of documented details regarding the trojan's capabilities, probably due to its relatively low profile. Since the discussion on the Incidents mailing list in June 2001, the trojan did not seem to catch the public's eye, and has remained relatively unexplored. However, our interest in this program was rekindled after a recent posting to the mailing list by Pete Schmitt on 10 March 2001.[PS] His brief description of the trojan that has infected his computer seemed to match the one discussed on the mailing list eight moths earlier. Pete also observed that the trojan attempted to connect to an IRC server using the name of "Joe Blow," which later allowed us to tie this incident to another variant of this trojan. Our research methodology and findings are documented in the following pages.

Section 2: Methodology

2.1   Controlled Environment

To facilitate efficient, inexpensive, and reliable research process, reverse engineers of malicious software should have access to controlled laboratory environment that is flexible and unobtrusive. In our research, we have come to rely on virtual workstation software available from VMware, Inc. VMware works by providing hardware emulation and virtual networking services, and allowed us to set up completely independent installations of operating systems on a single machine. With VMware, multiple operating systems can run simultaneously, and each virtual machine "is equivalent to a PC, since it has a complete, unmodified operating system, a unique network address, and a full complement of hardware devices."[VM1]

When setting up our laboratory environment, we installed VMware on a 500MHz laptop computer running Windows 2000 Professional with 20GB disk space and 256MB RAM. We decided to use a laptop, even though similarly priced desktop machines would offer much more computing power, to keep our laboratory as portable and lightweight as possible. Wishing to have a range of operating systems available for research, we created three virtual machines running within VMware: Red Hat Linux 7.0, Windows 98 Second Edition, and Windows NT 4.0 Workstation Service Pack 6a. Each guest operating system was allocated 1.6GB disk space and 64MB RAM, which was sufficient for our purposes. Because VMware emulates hardware, each guest operating system had to be installed in a way consistent with typical installation procedures. Figure 2-1 below illustrates our setup running two virtual machines simultaneously within the host operating system.
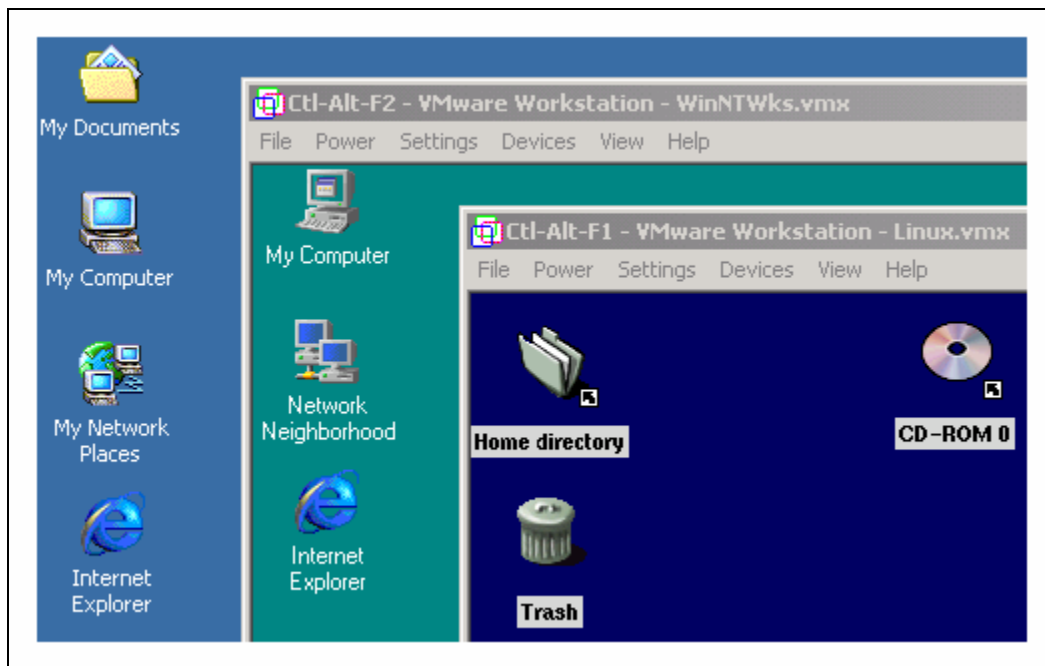

Figure 2-1

While each guest operating system was able to operate independently of each other, VMware allowed us to interconnect virtual machines using a virtual network that was completely enclosed within the hosting machine and did not have the ability to communicate with the outside world. To enable this functionality when using Windows 2000 as the host operating systems, we had to manually create a virtual host-only adapter by following directions from the VMware support site.[VM2] The adapter was assigned an IP address in a private RFC 1918 range, to ensure that it would not conflict with any of our existing connections.[RFC1918]

As part of the host-only adapter installation, VMware also installed a DHCP server service, dedicated to giving out IP addresses to hosts on the virtual network. To ensure that the virtual network is truly isolated from physical interfaces of the laptop, we installed ZoneAlarm Pro on the hosting machine, which offered us a comforting layer of protection, and warned us of any packets trying to leave the laboratory environment. This precaution, however, cannot guarantee complete isolation for the environment. Ideally, the VMware host should not be connected to a production network, and should be considered as dispensable as the virtual machines themselves. Figure 2-2 below illustrates virtual network infrastructure that resided within our laptop.
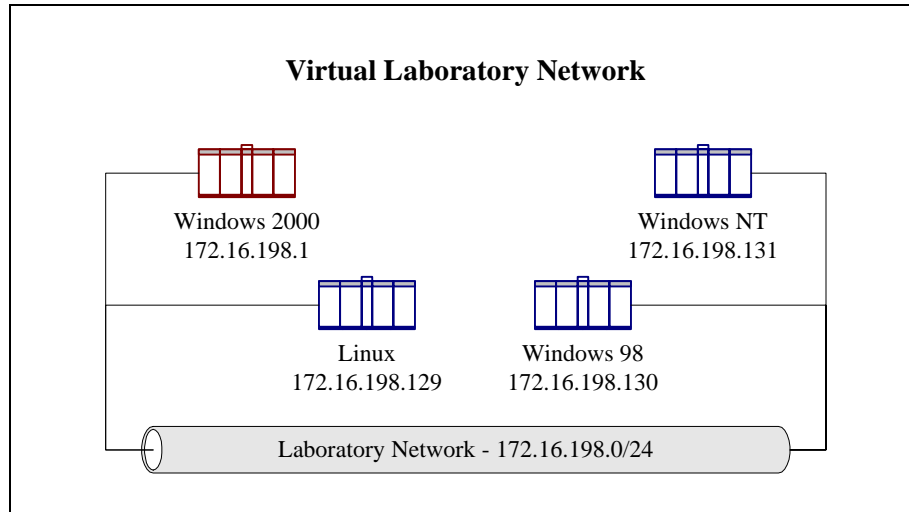

Virtual Laboratory Network

Windows 2000
172.16.198.1

Windows NT
172.16.198.131

Linux
172.16.198.129

Windows 98
172.16.198.130

Laboratory Network - 172.16.198.0/24

Figure 2-2

One of the advantages of using VMware instead of physically separate systems is the ability to backup and restore full systems in a matter of minutes. Each virtual machine is implemented using a several self-contained files located in the program's VMs subdirectory. Backing up the system can be accomplished by making a copy of the files that are used by VMware to represent the virtual machine. This is particularly useful when analyzing unknown malware, where unless the system is brought to a known state, repeated interactions with the virus or trojan might taint the environment. Additionally, the ease of making copies of virtual machines allows engineers to maintain a number of instances of an operating system with different patch levels.

Despite the high degree of isolation provided by VMware, there is some interaction between guest operating systems and the hosting machine. In particular, VMware provides VMware Tools drivers, which, when installed on a virtual machine will allow the user to move the mouse pointer between guest operating systems and the hosting machine. Additionally, the drivers allow virtual machines to share access to the hosting machine's clipboard. These features are enabled from within virtual machines, and the hosting system does not seem to be able to disable them. This

means that it is possible to craft a targeted attack against a user of a VMware-based laboratory that will achieve some level of access to the hosting system from within a virtual machine.

2.2   Behavioral Patterns

One of the ways to understand the threat associated with a malicious software specimen is to begin by examining its behavior in a controlled environment. This typically involves running the agent in a laboratory and studying its actions as it interacts with computer resources and responds to the various stimuli. VMware-based laboratory environment described in the previous chapter allowed us to use a single screen to monitor malware from perspectives of different systems.

We used a combination of process, disk, registry, and network monitoring tools to study the trojan's activity on a specific machine, as well as its attempts to connect to other systems. We were able to direct the trojan to laboratory machines when it attempted to communicate with systems on the Internet, so that we could replicate native environment for the trojan while maintaining full control over its interactions with the world.

We found freeware tools offered by Systernals to be very useful when monitoring the trojan's behavior. Specifically, we used Filemon for Windows NT/9x to observe which files the trojan attempted to access on the local system.[SI1] Similarly, Regmon for Windows NT/9x provided us with the ability to monitor registry-related read and write activity.[SI2] Additionally, the TCPView Pro utility, which costs $69, allowed us to obtain detailed listings of all TCP and UDP connections on the system, and tied network endpoints to processes that established them.[WI]

We used Winalysis, a program produced by SFullerton.com to detect changes made to the system by the trojan.[SF] Winalysis allowed us to create a baseline of the pristine system, and compare it to the system's state after the trojan ran. Using Winalysis in conjunction with monitoring tools from Systernals allowed us to be relatively certain that we would detect the trojan's affect on the file system. Winalysis costs $25 per copy for Windows 98/ME, and $35 per copy for Windows NT/2000, and is able to detect changes to the system's registry and file system. While Winalysis does not offer the level of detail and robustness associated with a more popular forensics tool Tripwire,[TR] its low price and ease of use made it a useful addition to our toolkit.

Malware may create temporary files as it executes, and delete them before the program exists. In this scenario Winalysis is unlikely to report evanescent existence of the transient file, while Filemon will report that it was created and deleted, but will not recover the file's contents. To account for this possibility we used the Undelete utility, available from Executive Software for around $45.[ES] Undelete replaces the native Windows Recycle Bin with a Recovery Bin, which is able to capture all deleted files, even those deleted by non-GUI processes. Unfortunately, at the time of this writing Undelete is only available for Windows NT/2000.

We relied on Snort, a freely distributed lightweight intrusion detection system, to monitor traffic on the laboratory network.[SN] Even though Snort is typically used to automatically detect network-based attacks, we only utilized its built-in sniffing capabilities to obtain details about network communications by invoking the program using the "`snort -v -d`" command, which told it to enter verbose mode and to capture data payload of packets. We preferred a Snort-based sniffer to other network capture utilities because of Snort's availability for Windows as well as UNIX operating systems, its large deployment base, as well as text-based logs and controls.

VMware comes with a tool called vnetsniffer, which can run on the hosting operating system and display network traffic between virtual machines, as well as between the hosting machine and its guests. However, the extent of information supplied by the program would not be sufficient for detailed forensics analysis. Even when executed with the "/e" switch, vmnetsniffer only logs packet size, source IP and MAC addresses, transport protocol type, as well as ARP and ICMP message types when appropriate. This is considerably less exhaustive than information provided by Snort, and most importantly lacks packet data payload.

For network monitoring purposes, the VMware virtual laboratory network can be considered to be hub-based, since every machine on the network is able to see all network traffic when its network interface is in promiscuous mode. However, the exception to this rule is the hosting operating system itself, which is only able to see traffic originating or targeting itself. To see all virtual network traffic from the hosting machine, one has no choice but to use the vmnetsniffer tool. To obtain the desired level of packet details, we monitored the network by running Snort on our Linux-based laboratory machine, which allowed us to see all virtual network traffic.

2.3   Code Analysis

Behavioral analysis described in the previous section concentrates on external aspects of malware as it interacts with its environment, but does not provide sufficient insight into the inner workings of the program. We utilized a debugger in conjunction with a disassembler to attempt reverse engineering the trojan's executable, since we did not have the luxury of looking at its source code. This process relied on the disassembler to understand the basic structure of the program, and proceeded by stepping through it with the debugger to study the trojan's workflow and to peak at its runtime memory contents.

We used DataRescue IDA Pro disassembler to decompose the trojan into assembly instructions. IDA Pro Standard can be purchased for $299 from the company's Web site.[DR1] Before purchasing the program, one might take advantage of its limited evaluation version, which is available for free and might prove to be sufficiently useful during early stages of the analysis.[DR2] DataRescue also provides IDA Pro Freeware, which is actually version 3.85B of the software, and lacks a Windows-based GUI that we found to be very useful in the newer versions of the program.[DR3] During our analysis we also relied on the Intel Architecture Software Developer's Manual for explanation of assembly instructions that we were not familiar with.[IN]

Because assembly is a low-level language, we encountered difficulties understanding the flow of the trojan's code without stepping through it with SoftICE, a powerful debugger that can be purchased as part of NuMega SoftICE Driver Suite for Windows 9x/NT/2000 for $999.[NM] We had the debugger running in the background of the laboratory system where we launched the trojan. Knowing the general structure of the trojan from its disassembled code as well as from system and registry calls intercepted by Systernals tools, we were able to set SoftICE breakpoints on code sections that seemed particularly interesting. Breakpoints allowed us to automatically invoke the debugger at specific workflow branches, and eliminated the need to step through every instruction in the trojan's program.

Once lauched, SoftICE ran in the background until invoked through the "Ctrl-D" key combination or until some program triggered a previously set breakpoint.[C4N] We usually set a breakpoint by executing the "bpx" command on the SoftICE command line by supplying an API function name or an instruction address as a parameter. We used the F10 key when in SoftICE to step through the program one step at a time while executing function calls as a single step, and

the F8 key to execute every instruction as an individual step.[MT] Finally, completing the list of our most frequently used SoftICE directives, is the "d" command, which displayed memory contents at a specific address or at a location stored in a particular register.

We installed SoftICE on Windows NT 4.0 and Windows 98 laboratory machines. Initially we had concerns over stability of the program when running in VMware environment. In particular, machines would sometimes freeze and fail to start when SoftICE was activated. However, most of the problems went away once we removed VMware video drivers from these virtual machines and installed standard Windows VGA video drivers in 640x480 mode with 16 colors. Under Windows NT we started SoftICE manually using the "net start ntice" command. Under Windows 98 we ran into problems trying to start the program manually, and had the system automatically launch it upon boot-up.

Another useful tool in our arsenal was the "strings" program, available in most UNIX distributions. This utility can extract text strings from executables, which is often helpful for assessing the program's purpose and mechanics. A strings snapshot of malware is considerably shorter than a complete listing of its disassembled code, but, of course, it is not as thorough. Similar functionality is available for Windows from the BinText program, which is freely distributed by Foundstone.[FS] BinText is a bit more flexible than most of its UNIX alternatives, and supports a range of advanced filtering options.

Finally, we used Perl as the scripting engine for automating minor tasks related to the analysis. For instance, we were able to model the decryption fragment of the trojan's code from assembly using a flexible routine implemented in Perl. Perl is built into most UNIX distributions, and is freely available for Windows platforms from ActiveState, which distributes it under the ActivePerl label.[AS]
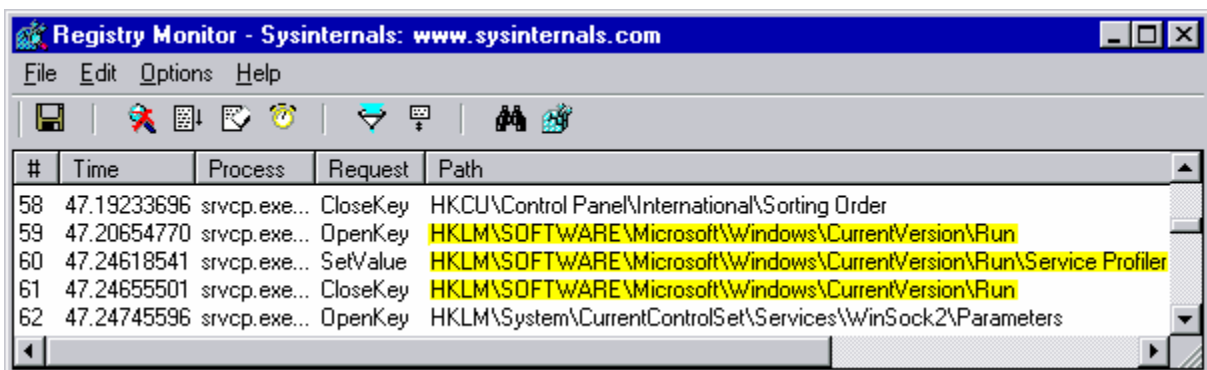
Section 3: Trojan Architecture

3.1   Local System Interaction

Before launching the srvcp.exe trojan on our Windows NT 4.0 laboratory machine, we enabled all monitoring tools that we had in our possession. We placed the srvcp.exe file in the arbitrarily chosen location on the local file system and ran the program. It went quietly into the background, and besides adding the "srvcp.exe" process to the process list, did not register any behavior that could be observed with a naked eye. We could see activity associated with the executable being logged by our monitoring utilities, and killed the srvcp.exe process after it lived for approximately 10 seconds. Examination of activity logs revealed behavior consistent with the discussion in the "unknown trojan" thread on the Incidents mailing list, which we discussed in the Background Information section of this document.

After the trojan was killed, we used Winalysis to scan the system for file system and registry changes. The program flagged several modifications related to normal Windows activity, and specified that the following registry key was created with the value of "srvcp.exe" – "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Service Profiler". This configured the system to attempt launching the trojan every time Windows started up. Note that the full path to the trojan's executable was not specified, which indicates that the author assumed that srvcp.exe would be in the path. This suggests that we did not possess the trojan's distribution mechanism, which would have either modified the system's path, or copied srvcp.exe into a directory that was in the path by default, such as C:\WINNT. In our case, the executable file remained in C:\Download, and no new files were created, which meant that the trojan would not start upon system boot-up. Filemon and Regmon logs confirmed Winalysis findings.
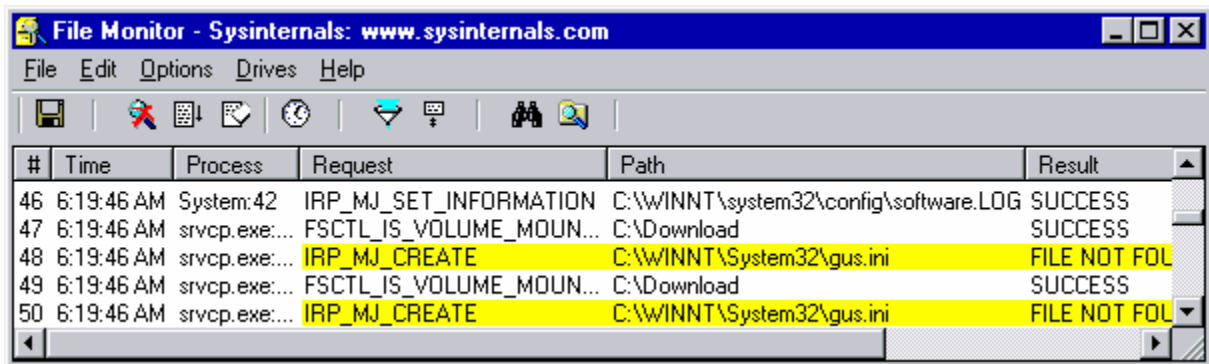
Figure 3-1 below illustrates creation of the trojan's registry key as witnessed by Regmon. (We manually highlighted relevant lines on the screen snapshot for emphasis.) Later experiments established that this key is set every time the trojan is executed, even if it has been created earlier. Note that Regmon does not display the value of the registry key – to obtain that information we had to either look at the registry using the regedit.exe utility, or compare system state using Winalysis. According to Regmon, the trojan also queried \Services\WinSock2 and Services\Tcp registry keys, which is typical for an executable utilizing TCP/IP for network communications.

Figure 3-1

As demonstrated in Figure 3-2 below, Filemon reported that the trojan attempted to access the `C:\WINNT\System32\gus.ini` file, which was not found on this system. Under Windows NT the request used the "`IRP_MJ_CREATE`" call with the option "`Open`", while in later experiments under Windows 98 Filemon simply labeled the request as "`Read`". In both cases, the trojan continued to function even though the file was not found. Further in the document we discuss the role of the gus.ini file. Furthermore, Filemon, captured access attempts to `C:\WINNT\System32\crtdll.dll`, which provides function calls such as fopen, fclose, and fscanf. Additionally, the trojan read `msafd.dll`, `wshtcpip.dll`, and `rnr20.dll`. Finally, the trojan read the hosts file, probably as part of the domain name resolution process that we have witnessed using Snort, as discussed in the next section.

Figure 3-2

3.2   Communication Protocols

Before launching the trojan on one of our laboratory machines, we configured the Linux machine to monitor network communications using Snort running in verbose mode. Because the VMware virtual network has hub-like properties, this system was able to capture all packets traveling across the laboratory network. Figure 3-3 below illustrates a Domain Name Service (DNS) request issued by the Windows NT machine running the srvcp.exe. We configured the Windows NT machine to use the VMware hosting system 172.16.1.98.1 as the DNS server, even though it was not running DNS software. An attempt to resolve irc.mcs.net is consistent with the behavior reported by Filemon, which registered the trojan's requests to read the local hosts file.

```
                        Domain Name Resolution Request

03/16-06:19:46.414790 172.16.198.131:1046 -> 172.16.198.1:53
UDP TTL:128 TOS:0x0 ID:4354 IpLen:20 DgmLen:57
Len: 37
00 01 01 00 00 01 00 00 00 00 00 00 03 69 72 63    .............irc
03 6D 63 73 03 6E 65 74 00 00 01 00 01             .mcs.net.....
```

Figure 3-3

Rather than bringing up a DNS server on the laboratory network, we added an entry to the infected machine's hosts file that resolved irc.mcs.net to 172.16.198.129. The purpose of this configuration step was to redirect the trojan to a server under our control so that we could observe

the nature of the connection that the program would make to the system whose name it was trying to resolve. Manipulating DNS records in laboratory environment was trivial in this case; had the trojan's author hard-coded an IP address into the program, we would have had to configure local routing tables and network parameters to redirect traffic to our system.

As demonstrated in Figure 3-4 below, the trojan attempted to connect to port 6667 on the system it thought was irc.mcs.net. This was not surprising, since port 6667 is commonly used for IRC communications, and is consistent with the host name and nature of irc.mcs.net, which is a popular EFnet IRC server. Note that our server responded with a RST packet to the trojan's SYN packet requesting the connection, because our server was not listening on port 6667.

```
                        Failed IRC Connection Attempt

03/16-06:44:10.522728 172.16.198.131:1060 -> 172.16.198.129:6667
TCP TTL:128 TOS:0x0 ID:47106 IpLen:20 DgmLen:44 DF
******S* Seq: 0x2D492  Ack: 0x0  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460

03/16-6:44:10.536857 172.16.198.129:6667 -> 172.16.198.131:1060
TCP TTL:255 TOS:0x0 ID:3 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0x2D493  Win: 0x0  TcpLen: 20
```

Figure 3-4

In response to observed behavior, we installed and configured ircd-hybrid software[IS] on the Linux machine 172.16.198.129 to provide IRC services to the trojan. After starting the trojan again, we were able to monitor its conversation with our IRC server. As shown in Figure 3-5 below, the trojan attempted to log in to the IRC channel "#daFuck" using the nickname "mikey". In this case the period in front of the "JOIN" command matches its "0A" hexadecimal counterpart, which represents the new line character.

```
                        Successful IRC Connection

03/16-07:59:47.630767 172.16.198.131:1030 -> 172.16.198.129:6667
TCP TTL:128 TOS:0x0 ID:18433 IpLen:20 DgmLen:102 DF
***AP*** Seq: 0x11B0D  Ack: 0xB1B0E8B2  Win: 0x2238  TcpLen: 20
4E 49 43 4B 20 3A 6D 69 6B 65 79 0A 55 53 45 52   NICK :mikey.USER
20 55 79 58 63 20 55 79 58 63 20 55 79 58 63 20    UyXc UyXc UyXc
3A 66 69 67 68 74 20 6D 65 2C 20 70 75 73 73 79   :fight me, pussy
0A 4A 4F 49 4E 20 23 64 61 46 75 63 6B 0A         .JOIN #daFuck.
```

Figure 3-5

The way in which the trojan connected to IRC is slightly different from the reports discussed earlier in the Background Information section of this document. In particular, Jeremy reported that the trojan's real name on the channel was "Im trojaned", while in our case the real name was set to "fight me, pussy". In both instances the names were suggestive of abuse, and it is possible that we were looking at a mutated version of the program. Additionally, Doug observed the trojan joining the channel "#mikag". In our case, we see a possible tie between Doug's channel name

"#**mik**ag" and our nickname "**mik**ey", even though our specimen joined a different channel. Later analysis, discussed further in our document, established a connection between the channel name and contents of the gus.ini file.

Several packets after the trojan connected to the IRC server, we observed an Ident request from the server to the workstation running the trojan. The Ident mechanism is typically used by UNIX systems to obtain the name of the user who initiated a TCP connection. TCPView running on the infected machine reported that the trojan was listening on TCP port 113 typically used by Ident services, and Figure 3-6 below shows the program's Ident response. After this exchange the trojan was logged into the IRC server, as seen in the following response to the "/who #daFuck" command executed on our IRC server:

"#daFuck   mikey   H@ CaTiRk@172.16.198.131 (fight me, pussy)"

```
                          Trojan's Ident Response

03/16-07:59:47.701560 172.16.198.131:113 -> 172.16.198.129:1078
TCP TTL:128 TOS:0x0 ID:18945 IpLen:20 DgmLen:78 DF
***AP*** Seq: 0x11B21  Ack: 0xB1EAA1B9  Win: 0x222B  TcpLen: 20
31 30 33 30 20 2C 20 36 36 36 37 20 3A 20 55 53   1030 , 6667 : US
45 52 49 44 20 3A 20 55 4E 49 58 20 3A 20 43 61   ERID : UNIX : Ca
54 69 52 6B 0D 0A                                 TiRk..
```

Figure 3-6

Ident functionality was built into the trojan most likely to increase the likelihood of successfully connecting to an IRC server, since many IRC servers do not accept connections whose identity cannot be confirmed through the Ident mechanism. As soon as the trojan answered the Ident request, it stopped listening on TCP port 113, probably to avoid remote detection with a port scanner, as well as to simplify the flow of the program's code. Repeated launches of the trojan indicated that it generated the username reported via Ident in a pseudo-random manner. For instance, some of the other generated names were "MdSxJy", "HsSdLhG", "IyKw", "FgX" and "Ce". After joining the desired channel, the trojan continuously issued the "NICK mikey" command approximately every three seconds. This is demonstrated in Figure 3-7 below. (For clarity we did not include corresponding ACK packets from the IRC server.)

```
                        Trojan's Nickname Requests

03/16-09:01:30.651137 172.16.198.131:1030 -> 172.16.198.129:6667
TCP TTL:128 TOS:0x0 ID:21761 IpLen:20 DgmLen:51 DF
***AP*** Seq: 0x11B79  Ack: 0xB1B0EE89  Win: 0x2238  TcpLen: 20
4E 49 43 4B 20 6D 69 6B 65 79 0A                  NICK mikey.

03/16-09:01:33.818259 172.16.198.131:1030 -> 172.16.198.129:6667
TCP TTL:128 TOS:0x0 ID:22017 IpLen:20 DgmLen:51 DF
***AP*** Seq: 0x11B84  Ack: 0xB1B0EE89  Win: 0x2238  TcpLen: 20
4E 49 43 4B 20 6D 69 6B 65 79 0A                  NICK mikey.
```

Figure 3-7

Once the trojan joined the IRC channel, it remained connected, supposedly waiting for commands from its operator via the chat session. The nature of these communications is discussed further in the Code Analysis section of this document. The program also participated in periodic "PING" - "PONG" message exchanges as defined in the IRC protocol to ensure that the IRC client is alive.

In one of our experiments we launched two instances of the trojan simultaneously, one running on the Windows NT laboratory machine, and the other on the Windows 98 system. The first instance to connect to the IRC server acquired the nickname of "mikey". When the second instance connected, however, it was not allowed to use the same name, since the IRC protocol requires that nicknames be unique on the same IRC network. As illustrated in Figure 3-8 below, this forced the trojan to generate a different nickname for itself in a pseudo-random manner. (In the network trace, the name "irc.ticklabs.com" is the hostname we assigned to our Linux laboratory machine.) In this scenario both instances of the trojan continued attempting to acquire the same nickname by issuing the "NICK mikey" command every three seconds.

```
                        Nickname Conflict Resolution

03/16-09:19:48.633843 172.16.198.129:6667 -> 172.16.198.130:1025
TCP TTL:64 TOS:0x0 ID:233 IpLen:20 DgmLen:147 DF
***AP*** Seq: 0x3FCD1E87  Ack: 0x301A2  Win: 0x7D78  TcpLen: 20
3A 69 72 63 2E 74 69 63 6B 6C 61 62 73 2E 63 6F  :irc.ticklabs.co
6D 20 34 33 33 20 2A 20 6D 69 6B 65 79 20 3A 4E  m 433 * mikey :N
69 63 6B 6E 61 6D 65 20 69 73 20 61 6C 72 65 61  ickname is alrea
64 79 20 69 6E 20 75 73 65 2E 0D 0A 3A 69 72 63  dy in use...:irc
2E 74 69 63 6B 6C 61 62 73 2E 63 6F 6D 20 34 35  .ticklabs.com 45
31 20 2A 20 4A 4F 49 4E 20 3A 52 65 67 69 73 74  1 * JOIN :Regist
65 72 20 66 69 72 73 74 2E 0D 0A                 er first...

03/16-09:19:48.651467 172.16.198.130:1025 -> 172.16.198.129:6667
TCP TTL:128 TOS:0x0 ID:7936 IpLen:20 DgmLen:61 DF
***AP*** Seq: 0x301A2  Ack: 0x3FCD1EF2  Win: 0x2128  TcpLen: 20
4E 49 43 4B 20 59 76 0A 4A 4F 49 4E 20 23 64 61  NICK Yv.JOIN #da
46 75 63 6B 0A                                   Fuck.
```

Figure 3-8

Multiple instances of the trojan seemed to be designed to ensure that at least one of them possessed the name "mikey". If the trojan instance that held that name were to disconnect from the IRC server, another instance of the program would pick up the name within at most three seconds. The more instances of the trojan were connected, the greater the likelihood that one of them would be called "mikey". As suggested by Doug Kahler in his e-mail correspondence with us, one of the purposes of this trojan might be to reserve the nickname for its operator, or to prevent someone from obtaining it. Possibly the program's author can communicate with it via IRC messages to command the trojan to release the name so that the person may obtain it.

One of our experiments was aimed at examining the nature of communications between a potential attacker and multiple instances of the trojan. IRC is a wonderful channel for centrally controlling an army of distributed attack agents, which is one of the reasons that trojans such as srvcp.exe are often hypothesized to have distributed denial of service capabilities. The attacker has the ability to communicate with multiple trojan instances by issuing a single command on the IRC channel, leaving it up to the server to relay the message to connected trojans. This nature of

IRC communications makes it difficult to trace the attack to its origin. Additionally, IRC offers the ability to communicate with each instance of the trojan individually via private messages.

Both one-to-many, as well as one-to-one IRC messages are implemented using the "PRIVMSG" command from the underlying IRC protocol, as demonstrated in Figure 3-9 below. (We removed irrelevant packets from the trace for clarity purposes.) The first two packets are carrying a message meant to be seen by all channel participants; this message was submitted by simply typing "hi all" at the IRC client's prompt. Our nickname was set to "attacker", our real name was "lzeltser", and we were connected from "127.0.0.1". The server can be seen sending the message individually to each instance of the trojan, identified by different destination IP addresses. The third packet is carrying a message meant to be seen only by the instance of the trojan that possessed the nickname "mikey"; this message was sent by typing "/msg mikey just for you" at the IRC client's prompt. In either case, we were unable to elicit any response from the trojan by sending it IRC messages; analysis of the program's code later revealed the need to encrypt commands in a proper manner in order for trojan to understand them.

```
                           Relaying Messages to IRC Clients

03/16-09:43:15.650781 172.16.198.129:6667 -> 172.16.198.130:1025
TCP TTL:64 TOS:0x0 ID:962 IpLen:20 DgmLen:94 DF
***AP*** Seq: 0x46C1D0D2  Ack: 0x404F7  Win: 0x7D78  TcpLen: 20
3A 61 74 74 61 63 6B 65 72 21 6C 7A 65 6C 74 73   :attacker!lzelts
65 72 40 31 32 37 2E 30 2E 30 2E 31 20 50 52 49   er@127.0.0.1 PRI
56 4D 53 47 20 23 64 61 46 75 63 6B 20 3A 68 69   VMSG #daFuck :hi
20 61 6C 6C 0D 0A                                  all..

03/16-09:43:15.651144 172.16.198.129:6667 -> 172.16.198.131:1030
TCP TTL:64 TOS:0x0 ID:963 IpLen:20 DgmLen:94 DF
***AP*** Seq: 0x3498DF73  Ack: 0xFF37  Win: 0x7D78  TcpLen: 20
3A 61 74 74 61 63 6B 65 72 21 6C 7A 65 6C 74 73   :attacker!lzelts
65 72 40 31 32 37 2E 30 2E 30 2E 31 20 50 52 49   er@127.0.0.1 PRI
56 4D 53 47 20 23 64 61 46 75 63 6B 20 3A 68 69   VMSG #daFuck :hi
20 61 6C 6C 0D 0A                                  all..

03/16-09:43:49.946018 172.16.198.129:6667 -> 172.16.198.131:1030
TCP TTL:64 TOS:0x0 ID:989 IpLen:20 DgmLen:98 DF
***AP*** Seq: 0x3498DFE1  Ack: 0xFFB0  Win: 0x7D78  TcpLen: 20
3A 61 74 74 61 63 6B 65 72 21 6C 7A 65 6C 74 73   :attacker!lzelts
65 72 40 31 32 37 2E 30 2E 30 2E 31 20 50 52 49   er@127.0.0.1 PRI
56 4D 53 47 20 6D 69 6B 65 79 20 3A 6A 75 73 74   VMSG mikey :just
20 66 6F 72 20 79 6F 75 0D 0A                      for you..
```

Figure 3-9

As discussed earlier in the document, the trojan attempted to read the gus.ini file from the system directory upon start-up. When we placed the gus.ini file into the system directory, Filemon showed that the trojan successfully opened and read the file. Using Snort that was running on the Linux laboratory machine we were able to observe that the trojan attempted to connect to TCP port 6666 on the IRC server, instead of TCP port 6667 used earlier. This connection attempt failed because our server was not listening on this port. As shown in Figure 3-10 below, we observed that the trojan then attempted to resolve host names of several IRC servers. (We

removed TCP retry packets from the trace for clarity.) The trojan continued to attempt resolving a number of other hostnames that seemed to be associated with IRC until we killed the process.

```
                         Resolving IRC Server Names

 03/16-10:29:11.319877 172.16.198.131:1040 -> 172.16.198.1:53
 UDP TTL:128 TOS:0x0 ID:63756 IpLen:20 DgmLen:61
 Len: 41
 00 01 01 00 00 01 00 00 00 00 00 00 05 65 66 6E   .............efn
 65 74 02 63 73 03 68 75 74 02 66 69 00 00 01 00   et.cs.hut.fi....
 01                                                 .

 03/16-10:29:44.423230 172.16.198.131:1042 -> 172.16.198.1:53
 UDP TTL:128 TOS:0x0 ID:1037 IpLen:20 DgmLen:63
 Len: 43
 00 03 01 00 00 01 00 00 00 00 00 00 05 65 66 6E   .............efn
 65 74 05 64 65 6D 6F 6E 02 63 6F 02 75 6B 00 00   et.demon.co.uk..
 01 00 01                                           ...

 03/16-10:30:15.667072 172.16.198.131:1044 -> 172.16.198.1:53
 UDP TTL:128 TOS:0x0 ID:3341 IpLen:20 DgmLen:64
 Len: 44
 00 05 01 00 00 01 00 00 00 00 00 00 03 69 72 63   .............irc
 0A 63 6F 6E 63 65 6E 74 72 69 63 03 6E 65 74 00   .concentric.net.
 00 01 00 01                                        ....
```

Figure 3-10

In response to observed behavior, we configured the IRC daemon on our server to listen on TCP port 6666. Once the trojan was restarted, Snort logs showed that the program connected to the IRC server on TCP port 6666. However, as illustrated in Figure 3-11 below, the trojan now joined the channel "#mikag" with the key "soup". Before we made gus.ini available to the trojan, it used a different channel name, and did not supply a channel key. (A key is sometimes used on IRC to restrict access to a channel.) In fact, the trojan's behavior now matched observations reported by Doug and described earlier in the Background Information section of this document.

```
                       Connecting to a Different IRC Channel

 03/16-11:07:12.816149 172.16.198.131:1032 -> 172.16.198.129:6666
 TCP TTL:128 TOS:0x0 ID:34304 IpLen:20 DgmLen:112 DF
 ***AP*** Seq: 0xF46A  Ack: 0x28E76CC2  Win: 0x2238  TcpLen: 20
 4E 49 43 4B 20 3A 6D 69 6B 65 79 0A 55 53 45 52   NICK :mikey.USER
 20 49 66 4A 64 43 6E 20 49 66 4A 64 43 6E 20 49    IfJdCn IfJdCn I
 66 4A 64 43 6E 20 3A 66 69 67 68 74 20 6D 65 2C   fJdCn :fight me,
 20 70 75 73 73 79 0A 4A 4F 49 4E 20 23 6D 69 6B    pussy.JOIN #mik
 61 67 20 73 6F 75 70 0A                           ag soup.
```

Figure 3-11

3.3   Program Code

When searching the Web for information relating to the srvcp.exe trojan we came across a paper by Joe Abrams, in which he analyzed several code sections from one of the variants of this trojan. Joe pointed out a number of strings embedded in the trojan's executable that were encrypted with an XOR-based algorithm, and described a way to decrypt them.[JA1] Even though most of the strings mentioned in the paper were absent from our copy of the executable, Joe decrypted one of the strings to have a clear text value of "gus.ini". Additionally, one of the deciphered values was "joeblow", which suggested that Joe's version of the trojan was similar to the one reported by Pete Schmitt in the posting discussed in the Background Information of this document.

To understand the decryption algorithm so that we could decipher strings embedded in our copy of srvcp.exe, we loaded the executable into the IDA Pro disassembler. (The reader may wish to refer to the program's assembly code by looking at our Adobe Acrobat print-out of the disassembled program at http://www.zeltser.com.) Following Joe's analysis, we searched for the string "nhl*pwf", which according to him was the encrypted value of "gus.ini". This brought us to a section of code presented in Figure 3-12 below. In this section the program repeatedly pushes to the stack a string that looks encrypted and calls the same routine, which suggests that "sub_4012C6" might be a decryption routine. Joe described this process as well, although his data offsets did not match ours, probably because of subtle differences in versions of the srvcp.exe executable.

```
               Probable Calls to String Decryption Routine

.text:0040141D    push    offset aNhlPwf    ; "nhl*pwf"
.text:00401422    call    sub_4012C6
.text:00401427    push    offset aAhkl      ; "|ahkli"
.text:0040142C    call    sub_4012C6
.text:00401431    push    offset aWtwgr     ; "wtwgr"
.text:00401436    call    sub_4012C6
.text:0040143B    push    offset aCdkk      ; "|cdkk"
.text:00401440    call    sub_4012C6
.text:00401445    push    offset aMfqece    ; "mfqEce"
.text:0040144A    call    sub_4012C6
```

Figure 3-12

By following Joe's analysis of the decryption process, and by looking through its assembly code in IDA Pro, we were able to create a Perl-based routine that mimicked its counterpart in the trojan's code. The assembly routine was labeled in IDA Pro as "sub_4012C6" and started at the "text:004012C6" offset. Our Perl routine to decrypt embedded strings is presented in Figure 3-13 below. The routine obtains the length of the encrypted string, and then iterates through it backwards by XOR'ing the ordinal value of each character with its position from the right of the encrypted string. One of the ways to ensure that our implementation of the routine works properly was to decrypt the "nhl*pwf" string present in ours as well as Joe's version of the executable, and make sure that the resulted value is "gus.ini". Our Perl routine was invoked using the format of "&decryptEmbeddedString($encryptedString)" and returned the deciphered string.

Decrypting Embedded Strings

```
sub decryptEmbeddedString {
  my($encrypted) = @_;
  my(@encrypted) = split(//, $encrypted);
  my($length) = $#encrypted;
  my($plain) = "";
  my($counter);
  for ($counter=0; $counter<=$length; $counter++) {
    $plain .= chr(ord($encrypted[$length-$counter]) ^ ($counter+1));
  }
  return($plain);
}
```

Figure 3-13

One of the ways to obtain plain-text versions of embedded strings was to call our Perl routine on each string that the program passed through its "sub_4012C6" routine. Additionally, to ensure that we have not missed any encrypted strings, we parsed srvcp.exe using the UNIX-based strings program, and used a Perl script to automatically attempt deciphering each string. We then looked through the resulted list of strings to manually single out those that seemed to be English-based. The combination of both approaches resulted in decrypted strings presented in Figure 3-14 below.

Embedded Strings Decrypted

Encrypted Value	*Decrypted Value*
nhl*pwf	gus.ini
|ahkl	mikey
wtwgr	setpr
|cdkk	jiggy
mfqEce	daFuck
~h`PmfqEce	daFuckWhat
v}~y{*%mj&qldkg	fight me, pussy
og&teh*`ph	irc.mcs.ne
O_ATU@VDE@	AGGRESSIVE
5|3u1v/k-h+i)j'g%JLQH	ISON a b c d e f g h

Figure 3-14

Some of the embedded strings that we decrypted were already seen in our analysis, as discussed in earlier sections of this document. In particular, "gus.ini" was the name of the file that the trojan attempted to locate upon start-up to change several aspects of its behavior. The "mikey" string matched the nickname that was used when connecting to an IRC server. The "daFuck" string, prefixed with "#", was the name of the IRC channel that the trojan joined. The "fight me, pussy" string matched the real name property of the trojan's IRC user as seen by the IRC server. Finally, "irc.mcs.ne", with the "t" character tagged on, was the host name of the IRC server that the trojan attempted to connect to; the final character is missing from the embedded string probably because

our strings program was unable to extract it properly. The purpose of other strings became a bit clearer after we analyzed contents of the gus.ini file.

A copy of the gus.ini file that we obtained seemed to be encrypted with an algorithm different from the one used to encode strings embedded into srvcp.exe. Based on our observations, we surmised that the trojan decrypted the file during runtime. We ventured to understand the decryption process in order to decipher the file. Looking through the trojan's assembly code in IDA Pro, we found a number of calls to the fopen function, which is typically used to access a file on disk. Only one of those calls specified the "r" attribute, which opened a file in read-only mode, as shown in Figure 3-15 below. This is the first fopen call that is made when the program starts up, which, combined with Filemon logs discussed earlier, suggested to us that this was an attempt to read in the gus.ini file. The "arg_0" parameter pushed onto the stack before calling fopen represents the name of the file to open, which in this case should be "gus.ini".

Opening gus.ini File

```
.text:00403662      push      offset aR          ; "r"
.text:00403667      push      [ebp+arg_0]
.text:0040366A      call      fopen
```

Figure 3-15

To ensure that we were correct in our understanding of how the gus.ini file is opened, we used SoftICE to examine program runtime when fopen is called. This was accomplished by starting SoftICE, then pressing "Ctrl-D" to access the SoftICE command shell. On the command prompt we entered the "bpx CRTDLL!fopen" command to set a breakpoint when the fopen function is invoked. We then entered the "x" command on the shell to put SoftICE in the background, and launched the srvcp.exe executable. In a matter of seconds SoftICE intercepted a call to fopen and interrupted the program's execution by bringing us back into the SoftICE command shell. We then used the "F10" key to step through several assembly instructions to return from the fopen function. Figure 3-16 below shows a section of the SoftICE screen as soon the trojan executed its first fopen call. We used the "d *(EBP+08)" command to display memory contents of the first parameter to fopen, which was "C:\WINNT\System32\gus.ini", as we expected.

Figure 3-16

Looking at the trojan's assembly code in IDA Pro, we see a single call to the fscanf function with the parameter of "%[^\n]\n", as shown in Figure 3-17 below. This is one of the ways to read in a whole line from the file, which is how the trojan reads in the gus.ini file line-by-line. Further in the program, we saw the executable jumping to another section of the code using the "jnz loc_4036B2" instruction, after which memory was cleaned up and the file handle closed. This suggested that file contents were processed using code located at the offset of 4036B2.

Reading Lines from gus.ini File

```
.text:00403738     push     eax
.text:00403739     push     offset asc_4083D1 ; "%[^\n]\n"
.text:0040373E     push     ebx
.text:0040373F     call     fscanf
.text:00403744     add      esp, 0Ch
.text:00403747     cmp      eax, 0FFFFFFFFh
.text:0040374A     jnz      loc_4036B2
```

Figure 3-17

Several lines after the offset of 4036B2 the program invokes the sscanf function, commonly used to parse strings, with the "%[^=]=%[^" parameter. This string pattern often represents format of typical .ini files whose lines follow the convention of "PARAMNAME=value". Since the encrypted gus.ini file did not follow the standard .ini file format for separating parameter names and values with equal signs, the program must be decrypting each line from the file before invoking sscanf. As demonstrated in Figure 3-18 below, the only routine called after reading in the line with fscanf and parsing it with sscanf is sub_405366, which is the probably decryption routine.

Parsing gus.ini Lines

```
.text:004036B2     lea      eax, [ebp+var_414]
.text:004036B8     push     eax
.text:004036B9     push     esi
.text:004036BA     call     sub_405366
.text:004036BF     mov      edi, eax
.text:004036C1     lea      eax, [ebp+var_9F0]
.text:004036C7     push     eax
.text:004036C8     lea      eax, [ebp+var_14]
.text:004036CB     push     eax
.text:004036CC     push     offset asc_4083C5 ; "%[^=]=%[^"
.text:004036D1     push     edi
.text:004036D2     call     sscanf
```

Figure 3-18

In order to examine the trojan's runtime environment when the decryption routine is called, we restarted the srvcp.exe process, leaving the SoftICE CRTDLL!fopen breakpoint mentioned earlier. Once SoftICE intercepted the call, we added a breakpoint at offset 4036BA, which is where sub_405366 is called from, by entering "bpx 4036BA" at the SoftICE command line. We then put SoftICE in the background using the "x" command, and waited until the debugger

interrupted the trojan at our breakpoint. The reason for waiting for the first fopen call before setting the second breakpoint was to let SoftICE calculate the absolute offset for the sub_405366 call's relative offset with respect to the trojan's runtime stack.

The state of the program before it invoked the decryption routine is shown in Figure 3-19 below. In this section of the SoftICE screen we observed that the executable pushed two values onto the stack before calling sub_405366, as seen earlier in IDA Pro. The value stored in the memory location pointed to by the EAX register was "JexO215WuK60H7HgI.j11vh1", which is actually the first line of the encrypted gus.ini file. We were able to view these memory contents by executing the "d EAX" command in SoftICE. We viewed the value associated with the ESI register in a similar manner, and determined that it was set to "EcbJer8\0dx.CJVJsAlmIZ" (note the null character in the middle, presented here as "\0"). The nature of this second string and its role in the decryption process became clearer later in our analysis process.

```
4036B2   LEA      EAX,[EBP+FFFFFBEC]
4036B8   PUSH     EAX
4036B9   PUSH     ESI
4036BA   CALL     00405366
4036BF   MOV      EDI,EAX                                          ◀ ▶
———KTEB(804B6600)-TID(0077)———srvcp!.text+26A2—————————————————
D8EAEC 4A 65 78 4F 32 31 35 57-75 4B 36 30 48 37 48 67  JexO215WuK60H7Hg
D8EAFC 49 2E 6A 31 31 76 68 31-00 00 00 00 00 00 00 00  I.j11vh1........
```

Figure 3-19

We used the "F10" key in SoftICE to let the trojan execute the sub_405366 call, after which we looked at memory contents associated with the EAX register again. As shown in Figure 3-20 below, the memory location contained "NICK=mikey", which seemed to be a decrypted version of the gus.ini line that was being processed by the trojan.

```
4036B2   LEA      EAX,[EBP+FFFFFBEC]
4036B8   PUSH     EAX
4036B9   PUSH     ESI
4036BA   CALL     00405366
4036BF   MOV      EDI,EAX                                          ◀ ▶
———KTEB(804AE020)-TID(0027)———srvcp!.text+26A2—————————————————
136518 4E 49 43 4B 3D 6D 69 6B-65 79 00 00 00 00 00 00  NICK=mikey......
136528 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Figure 3-20

Going through this process for other lines of gus.ini allowed us to confirm that decryption of the file occurred in the sub_405366 routine. We were also able to obtain the deciphered version of the gus.ini file by monitoring memory contents pointed to by the EAX register after the call to the decryption routine without knowing the actual decryption algorithm. Decrypted contents of the gus.ini file are presented in Figure 3-21 below. We abridged the file's contents for added clarity.

```
                              Decrypted gus.ini File

 NICK=mikey
 MODE=AGGRESSIVE
 SETCOMMAND=setpr
 COMMAND=fuckedup
 CHANNEL=mikag soup
 SOUPCHANNEL=alphasoup ah
 SERVER0=irc.mcs.net:6666
 SERVER1=efnet.cs.hut.fi:6666
 SERVER2=efnet.demon.co.uk:6666
 SERVER3=irc.concentric.net:6666
 SERVER4=irc.etsmtl.ca:6666
 SERVER5=irc.fasti.net:6666
          ... cut for brevity ...
```

Figure 3-21

Contents of gus.ini seem to overwrite default values embedded into srvcp.exe at compile time. The decrypted version of the file provides parameter names in addition to parameter values, which offered clues as to the purpose of the strings seen before. We already knew that the "NICK" parameter defined the nickname used when connecting to an IRC server. While our gus.ini file set the name to "mikey", the trojan's operator seems to be able to set it to an arbitrary value by manipulating contents of the gus.ini file. The "CHANNEL" parameter defines the name of the channel that the trojan joins. Our gus.ini overwrites the program's built-in channel name, which is consistent with behavior that we observed earlier. The file also provides the trojan with the list of servers and port number that the program will use when joining an IRC network, which overwrites the default value of TCP port 6667 on irc.mcs.net. Our gus.ini file, whose decrypted version can be downloaded from http://www.zeltser.com/, defined thirty-four such servers. These host names are consistent with observed DNS resolution requests discussed in the Communication Protocols section of this document.

Judging by parameter names, the trojan is able to join an alternate channel, defined in gus.ini by the "SOUPCHANNEL" parameter. This parameter's value is related to the value defined by the "CHANNEL" parameter through a common substring "soup". The built-in channel name "daFuck" and the unknown built-in parameter with the value of "daFuckWhat" are also related through a common substring, which suggests that "daFuckWhat" is the built-in alternate channel name. Details regarding the use of the alternate channel are unclear to us at the time of this writing.

The value defined by the "SETCOMMAND" parameter in gus.ini matches the "setpr" string embedded into the executable. This seems to be a keyword used as the basis for authenticating to the trojan over IRC. This hypothesis is reinforced by Joe Abrams's analysis, even though the value embedded into his version of the trojan was different from ours.[JA2] Joe suggested that there are several access levels to the trojan, which indicates that the our trojan's second password is set in gus.ini by the "COMMAND" parameter, which is "fuckedup". Note that this is different from what we believe to be the embedded counterpart of this parameter, which is set to "jiggy".

The purpose of the "MODE" parameter, set to "AGGRESSIVE" in the gus.ini file, as well as encrypted within the executable, is unknown at the time of this writing. This could be a way to control the tenacity of the trojan's attacks, or to fine-tune its behavior on IRC channels. We

presume that the alternative value for this parameter is "PASSIVE", since both are defined in clear-text form in the executable next to each other on the stack. The program compares clear-text versions of these values to a variable in adjacent sections of the code; this might be the process of comparing the decrypted value of the "MODE" parameter to known values, and acting accordingly.

In an attempt to understand the algorithm used to obfuscate lines in the gus.ini file, we turned our attention to the second parameter that was passed to the decryption routine. This string was set to "EcbJer8\0dx.CJVJsAlmIZ", and seemed to be used as a key during the deciphering process, but was not embedded as a plain string into the executable. We found that this key was stored as a collection of characters located next to each other on the stack in locations 4080C8 to 40811F. The final fragment of this data stack is shown in Figure 3-22 below, and includes the terminating null character. The key is not stored as a single string probably to avoid easy detection.

```
                      Fragment of Embedded Decryption Key

                    ... cut for brevity ...
        .data:00408110    db    6Dh ; m
        .data:00408111    db      0 ;
        .data:00408112    db      0 ;
        .data:00408113    db      0 ;
        .data:00408114    db    49h ; I
        .data:00408115    db      0 ;
        .data:00408116    db      0 ;
        .data:00408117    db      0 ;
        .data:00408118    db    5Ah ; Z
        .data:00408119    db      0 ;
        .data:0040811A    db      0 ;
        .data:0040811B    db      0 ;
        .data:0040811C    db      0 ;
        .data:0040811D    db      0 ;
        .data:0040811E    db      0 ;
        .data:0040811F    db      0 ;
```

Figure 3-22

Note that each actual character is followed by three null characters. Upon start-up, the trojan assembles the key into a single string using code presented in Figure 3-23 below. This section of the program reads characters one-by-one and aligns them in memory as adjacent bytes.

```
                      Assembling the Decryption Key

 .text:00403698    mov     edx, dword_4080C8[edi*4]
 .text:0040369F    mov     [esi+edi], dl
 .text:004036A2    inc     edi
 .text:004036A3    cmp     edi, 2Ch
 .text:004036A6    jb      short loc_403698
 .text:004036A8    mov     byte ptr [edi+esi+1], 0
```

Figure 3-23

We attempted to step through the trojan's decryption routine in order to comprehend its algorithm. While time constraints prevented us from completing this process, we were able to understand some of the underlying principles of the procedure. As part of the decryption process, the program splits the key mentioned above into two strings using the null character as the delimiter. We followed the workflow of the program as it mutated a copy of the first portion of the key, which was "`EcbJer8`", by adding 7 to the ordinal value of each character. Assembly code performing this function is presented in Figure 3-24 below. As the result of this code fragment, the initial part of the key was transformed into "`LjiQly?Ck`" followed by an unprintablable character represented by the hexadecimal value of 7F.

```
                    Mutating the Decryption Subkey

.text:00405401     mov      eax, edi
.text:00405403     add      eax, [ebp+var_C]
.text:00405406     add      byte ptr [eax], 7
.text:00405409     inc      edi
.text:0040540A     mov      eax, [ebp+var_C]
.text:0040540D     lea      ecx, [eax]
.text:0040540F     or       eax, 0FFFFFFFFh
.text:00405412     inc      eax
.text:00405413     cmp      byte ptr [ecx+eax], 0
.text:00405417     jnz      short loc_405412
.text:00405419     cmp      edi, eax
.text:0040541B     jb       short loc_405401
```

Figure 3-24

We also noticed that the trojan processed a large number of character values embedded into the executable starting with offset 4086CC by aligning them next to each other in memory. This was done in routine sub_404E78 that was invoked at offset 405435 soon after mutating the key prefix. At this point the logic became somewhat cumbersome to follow, and we lost track of the program's workflow. Those interested in tracing our steps may wish to set a breakpoint at the referenced offset and step through the code. (This can be accomplished by first breaking at the fopen call using the "`bpx CRTDLL!fopen`" command in SoftICE, and then creating a new breakpoint at the desired offset using a command such as "`bpx 405435`".)

After setting up the environment in a way that is still somewhat unclear to us, the decryption process continued by looping through a subset of characters in the beginning of the encrypted line and calling the sub_40517A routine from offset 405473. The sub_40517A routine takes a character from the encrypted line and returns the index indicating the location of this character in the string that is embedded into srvcp.exe at offset 409718. The index starts from 0 and the string is "`./0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`". After the sub_40517A routine returns, the index value gets shifted to the left by varying amounts using commands at offset 405489, and is masked into memory using the OR instruction located at offset 40548B. The shifting process is continued with minor variations until the end of the decryption routine sub_405366 that we discussed earlier, which returned a deciphered version of the line from the gus.ini file.

According to Joe Abrams, similar encryption techniques were used to encrypt commands sent to the trojan via IRC. Commands have to be properly encrypted in order for the program to interpret

them. Joe was able to control his copy of the trojan in a limited fashion by replaying some of the encrypted commands that he witnessed being used on the trojan's IRC channel. This suggests that the algorithm used to encrypt commands and the gus.ini files does not take into account characteristics specific to an instance of the trojan.

For a properly encrypted command to be honored by the trojan, the command needs to be prefixed by a proper password in each communication attempt. This means that the operator trying to control the trojan would need to send a message to the IRC channel in the form "*password command*". As we discussed earlier, at least two passwords are hard-coded into the program. These can be overwritten using "SETCOMMAND" and "COMMAND" parameters in the gus.ini file, and probably provide the operator with different access levels.

As Joe described in his paper, the password has to be properly encrypted using its own algorithm. This algorithm protects the trojan against replay attacks by taking into account the IP address of the host trying to control the program. Joe described the password encryption process in his paper, and points out that some of the characters comprising the encrypted password may not be printable. The need to properly encrypt password and command strings suggests there exists a trojan controlling program that operates outside, or in conjunction with, the standard IRC client.

Functionality provided by the srvcp.exe trojan to the attacker is still unclear. As we mentioned in the Background Information section of the document, a number of FTP-related strings are embedded into the program, which suggests that its operator has the ability to transfer files to and from the infected system. As shown Figure 3-25, the trojan seems to rely on the ftp.exe program built into Windows for FTP-based file transfer capabilities.

Embedded FTP Messages

```
.data:004084B9 db 'PRIVMSG %s :successfully spawned ftp.exe',0Ah,0
.data:004084E3 db 'PRIVMSG %s :couldn',27h,'t spawn ftp.exe',0Ah,0
```

Figure 3-25

Several strings embedded into srvcp.exe suggest that the trojan is able to perform denial of service attacks. As shown in Figure 3-26 below, "sacker" and "jacker" could be references to such attacks. Note that "sacker" takes time and port numbers as parameters, which might allow the attacker to flood a target with network traffic for a specified period of time. The "spawn" command might allow the attacker to launch an arbitrary program on the infected machine.

Possible Network Flood Commands

```
.data:00408541 db 'PRIVMSG %s :ctcp <nick> PING 848348, help, '
.data:00408541 db 'getnick <nick>, getnonick, rnick <nick>!!, '
.data:00408541 db 'sacker time low_port high_port addy, jacker'
.data:00408541 db ' time ip ip ip etc, stopsack, stopjack, spa'
.data:00408541 db 'wn filename, ftpget EVERYTHING, randnick, c'
.data:00408541 db 'lone, clonedie',0Ah,0
```

Figure 3-26

Section 4: Defensive Measures

4.1   Propagation Mechanisms

We found no evidence that srvcp.exe is capable of spreading or replicating without help of an external program. As mentioned in the Local System Interaction section of the document, the executable assumes that it is located in a directory that is included in the system path, such as C:\WINNT, however it does not actually copy or move itself there upon start-up. This suggests that we were missing another program that was actually responsible for placing srvcp.exe into the proper directory. The victim probably received an infected carrier file via an e-mail message, or downloaded the infected program from the Internet.

According to information found on a HackFix page related to srvcp.exe and dated June 2000, the trojan was being was being spread as a DivX_e3.exe program, which claimed to be a registered version of a DivX video decoder.[HF] Additionally, the trojan was reportedly spreading via serials.2000.v6.0.zip, CDRWin3.8.zip, and PSXCopy.v6.0.zip files. The trojan is continuing to spread, as witnessed by a recent incident reported by Pete Schmitt and mentioned in the Background Information section of this document; however, file names of current carriers are unknown to us at the time of this writing. Once the machine is infected, the attacker is able to install other malware on the victim's system through the trojan's file transfer capabilities.

It is unclear whether the carrier program also installs a copy of the gus.ini file along with srvcp.exe. Since the trojan is able to operate without gus.ini, it is possible that the file is created after the initial infection. The attacker has the ability to upload gus.ini to the infected machine via IRC file transfer as well as FTP capabilities. Note that most of these connections are initiated by the infected machine, which makes it less likely that they will be stopped by the firewall. It is also possible that the attacker is able to instruct the trojan to create the gus.ini file with specific values by sending appropriate commands via the IRC control channel.

4.2   Trojan Variants

Looking through public databases of anti-virus product vendors, we found a number of records related to variants of the srvcp.exe trojan. Vendors do not seem to agree on the name for this trojan, however. Symantec calls it "IRC.SRVCP.Trojan", but does not supply any information about it besides characterizing the trojan as "wild". [SY] Trend Micro offers a brief description of the trojan that matches our variant closely in both behavioral characteristics as well as file size; they refer to the trojan as "TROJ_SRVCP".[TM1] Computer Associates provides sufficient information about the trojan to indicate that their variant closely matches the executable described in this document; they refer to this program as "Tasmer.B" and "Win32.Tasmer.B"[CA1]

Computer Associates also describes a variant of the trojan that uses tskmngr.exe as the name of its executable and "Task Manager" as its registry key; they refer to this program as "Tasmer.A", "Win32.Tasmer.A", "Backdoor.Tasmer", and "Troj/Narnar" and report seeing it in the wild as early as April 2000. Trend Micro describes a tskmngr.exe variant of the trojan as well, but refers to it as "TROJ_TASMER.B".[TM2] The "Troj/Narnar" name for the tskmngr.exe variant is also used by Sophos; they point out that the trojan connects to an DALnet IRC server, which is an

alternative network to EFnet used by our version of the trojan.[SP] This variant is also discussed by Network Associates, which refers to it as "IRC/Randy".[NA]

Clearly, there are at least two variants of this trojan – one operating on EFnet, and the other on DALnet. According to Joe Abrams, the EFnet version of the trojan is significantly younger than the one operating on DALnet. Judging by the number of infected users on DALnet channels discussed in Joe's paper, the EFnet trojan variant is not as popular as the DALnet one. Joe's version of the trojan joined the "#KimmiTheB" channel on DALnet, where it was controlled by members of the group that called itself "divide".

Joe also pointed out that he came across a new version of the trojan that was packed/compressed using the NeoLite utility. NeoLite encapsulates Windows executables and their resources to protect them from reverse engineering using decompilers.[NL] After NeoLite compresses the executable, it extracts it into memory when the executable is launched. This kind of protection can be bypassed using SoftICE by setting a breakpoint on the NeoLite decryption routine, letting the program decompress itself, and examining the decrypted program in memory using SoftICE.

4.3   Trojan Signatures

As illustrated in Figure 4-1 below, Norton AntiVirus was able to automatically detect our version of the srvcp.exe trojan. Detecting the trojan manually by examining the infected machine is relatively easy as well – one should look for the presence of the srvcp.exe executable on the file system, examine the registry for presence of the key
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Service Profiler", and check for existence of the gus.ini file in the machine's system directory. We have not seen any of these three characteristics attributed to a program other than srvcp.exe, although there is some chance for encountering a false positive scenario.
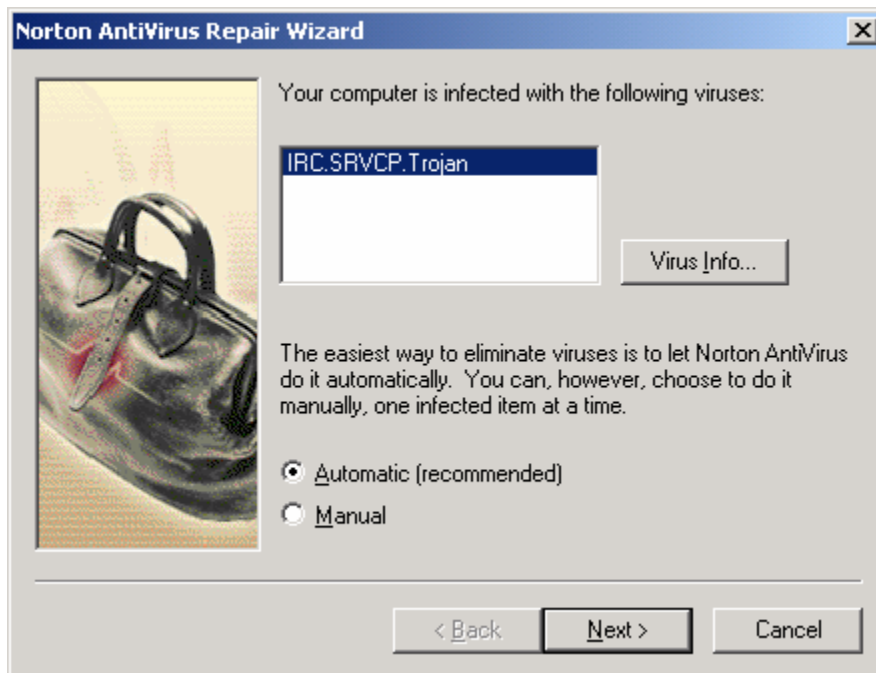

Figure 4-1

The trojan can be easily mutated to make its detection more difficult even without having access to its source code. For instance, the program will continue to function even if its file does not have the name of "srvcp.exe". The executable can also be edited with a regular file editor to use a different a different registry key. Similarly, the executable can be edited to use a file name other than "gus.ini" for its resources.

As a proof of concept we used a plain text editor to modify the compiled executable by changing the name of the "gus.ini" file – this required searching for the string "`nhl*pwf`" since the file name is stored in the encrypted format. We then modified one of the letters in the encrypted string so that the new string became "`nhl*pwg`". The resulted data stack, as seen by disassembling the modified executable using IDA Pro, is shown in Figure 4-2 below. Running the modified version of the trojan resulted in the program using the name "fus.ini" for the file that it used to know as "gus.ini". Most disturbingly, modifying the executable in this simple manner caused Norton AntiVirus to stop detecting it as malware. Note that an arbitrary file name can be constructed for this file by reversing the embedded string decryption algorithm that we described in the Program Code section of this document.

Modifying Name for Initialization File

Before modification:
```
.data:00408034      db 'nhl*pwf',0
```

After modification:
```
.data:00408034      db 'nhl*pwg',0
```

Figure 4-2

One of the ways of detecting the trojan is by detecting its network communications. For instance running the "`netstat -a`" command on the infected system will show the system listening on TCP port 113 for Ident requests if the trojan has not connected to an IRC server yet. Scanning the network for hosts unauthorized to run Identd is a possible way of detecting the trojan in this state remotely. Once the trojan successfully connected to an IRC server, the "`netstat -a`" is likely to show a TCP connection to an external server on either port 6667 or 6666. This kind of communication can be detected using a network-based Intrusion Detection System (IDS) if the organization's workstations do not normally use IRC.

A more reliable way to detect the trojan with a network-based IDS is to scan packet contents for strings that are associated with the trojan's operation. Common variants of the trojan can be detected by looking for strings such as "`NICK mikey`" in the network stream. Unfortunately, this approach is unlikely to be more effective than using anti-virus software, since the signature can be easily changed. A more reliable indicator of the trojan's activity is presence of nickname change requests issued by a single IRC client every three seconds or so. Unfortunately, implementing this kind of logic is likely to be resource intensive, since it would require that the IDS maintain state information regarding potentially offensive traffic across multiple packets. Alternatively, the IDS could be tuned to scan packets for encrypted command strings used to operate the trojan, since even in the encrypted form these commands remain the same across multiple instances of the trojan unless the encryption algorithm is changed. Obtaining these command strings will require a more thorough analysis of the trojan. We encourage the reader to partake in such quest and to share his or her findings with the defense community.

Section 5: References

[JLG] Jeremy L. Gaddis. Incidents Mailing List Archive. "unknown trojan (attached)." 8 June 2000. URL: http://www.securityfocus.com/archive/75/64241. 21 March 2001.

[RFC1459] Jarkko Oikarinen, Darren Reed. RFC 1459. "Internet Relay Chat Protocol." May 1993. URL: http://www.faqs.org/rfcs/rfc1459.html. 21 March 2001.

[BK] Brandon Kittler. Incidents Mailing List Archive. "Re: unknown trojan (attached)." 10 June 2000. URL: http://www.securityfocus.com/archive/75/64380. 21 March 2001.

[RFC1413] Michael St. Johns. RFC 1413. "Identification Protocol." February 1993. URL: http://www.faqs.org/rfcs/rfc1413.html. 21 March 2001.

[DK] Doug Kahler. Incidents Mailing List Archive. "Re: unknown trojan (attached)." 12 June 2000. URL: http://www.securityfocus.com/archive/75/64849. 21 March 2001.

[NF] Nick FitzGerald. Incidents Mailing List Archive. "Re: unknown trojan (attached)." 13 June 2000. URL: http://www.securityfocus.com/archive/75/64847. 22 March 2001.

[PS] Pete Schmitt. Incidents Mailing List Archive. "new (?) windows irc ddos trojan." 10 March 2001. URL: http://www.securityfocus.com/archive/75/167985. 22 March 2001.

[VM1] VMware, Inc. "VMware Workstation FAQs." URL: http://www.vmware.com/products/desktop/ws_faqs.html. 22 March 2001.

[VM2] VMware, Inc. "Host-Only Networking Configuration Notes for Windows 2000 Hosts." URL: http://www.vmware.com/support/ws2/doc/hostonly_w2k_ws_win.html. 4 April 2001.

[RFC1918] RFC 1918.URL: http://www.faqs.org/rfcs/rfc1918.html. 22 March 2001.

[SI1] Mark Russinovich, Bryce Cogswell. Filemon for Windows NT/9x. 26 December 2000. URL: http://www.sysinternals.com/ntw2k/source/filemon.shtml. 22 March 2001.

[SI2] Mark Russinovich, Bryce Cogswell. Regmon for Windows NT/9x. 7 November 2000. URL: http://www.sysinternals.com/ntw2k/source/regmon.shtml. 22 March 2001.

[WI] Winternals Software. TCPView Professional Edition. URL: http://www.winternals.com/products/monitoringtools/tcpviewpro.shtml. 22 March 2001.

[SF] SFullerton.com. Winalysis Version 2.50. 30 December 2000. URL: http://www.sfullerton.com/products.htm. 22 March 2001.

[TR] Tripwire Home Page. URL: http://www.tripwire.com. 4 April 2001.

[ES] Executive Software. Undelete 2.0. URL: http://www.execsoft.com/undelete/undelete.asp. 22 March 2001.

[SN] Martin Roesch. Snort – The Open Source Network Intrusion Detection System. URL: http://www.snort.org. 22 March 2000.

[DR1] DataRescue. IDA Pro by Ilfak Guilfanov. URL: http://www.datarescue.com/idabase/idaorder.htm. 22 March 2001.

[DR2] DataRescue. IDA Pro Evaluation Download. URL: http://www.datarescue.com/idabase/ida4down.htm. 22 March 2001.

[DR3] DataRescue. IDA Pro Freeware. URL: http://www.datarescue.be/downloadfreeware.htm. 22 March 2001.

[IN] Intel Corporation. Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual. 13 January 1997. URL: http://developer.intel.com/design/pentium/manuals/243191.htm. 22 March 2001.

[NM] Compuware NuMega. "Can I still buy SoftICE separately?" URL: http://www.numega.com/drivercentral/FAQs/dsq29.shtml. 22 March 2001.

[C4N] CoRN2. "#Cracking4Newbies SoftIce Tutorial." URL: http://whateverhosting.com/krobar/beginner/04.htm. 22 March 2001.

[MT] "Mammon_'s Tales to His Grandson" URL: http://newdata.box.sk/neworder/cracking/ice.html. 22 March 2001.

[FS] Foundstone. BinText v3.0. URL: http://www.foundstone.com/rdlabs/proddesc/bintext.html. 22 March 2001.

[AS] ActiveState Corporation. ActivePerl. URL: http://www.activestate.com/Products/ActivePerl. 21 March 2001.

[IS] IRCD-Hybrid. URL: http://www.ircd-hybrid.net. 4 April 2001.

[JA1] Joe Abrams. "Reversing a Trojan." 1 October 2001. URL: http://freeshell.org/~abrams/troj.txt. 22 March 2001.

[JA2] Hack in the Box. Joe Abrams. "Reversing a trojan." 19 November 2001. URL: http://www.hackinthebox.org/article.php?sid=1138. 22March 2001.

[HF] HackFix. "srvcp.exe" June 2000. URL: http://www.hackfix.org/ircfix/srvcp.shtml. 22 March 2001.

[SY] Symantec Virus Encyclopedia. "IRC.SRVCP.Trojan." URL: http://www.symantec.com/avcenter/cgi-bin/virauto.cgi?vid=18552. 22 March 2001.

[TM1] Trend Micro Virus Encyclopedia. "TROJ_SRVCP." URL: http://antivirus.com/vinfo/virusencyclo/default5.asp?VName=TROJ_SRVCP&VSect=T. 4 April 2001.

[CA1] Computer Associates Virus Encyclopedia. "Tasmer.B." URL: http://ca.com/virusinfo/encyclopedia/descriptions/tasmerb.htm. 22 March 2001.

[TM2] Trend Micro Virus Encyclopedia. "TROJ_TASMER.B" URL: http://antivirus.com/vinfo/virusencyclo/default5.asp?VName=TROJ_TASMER.B&VSect=T. 22 March 2001.

[SP] Sophos Virus Info. "Troj/Narnar". URL: http://www.sophos.com/virusinfo/analyses/trojnarnar.html. 22 March 2001.

[NA] Network Associates. "IRC/Randy". 12 April 2000. URL: http://vil.nai.com/villib/dispVirus.asp?virus_k=98569&EY=y. 22 March 2001.

[NL] NeoWorks. "NeoLite: Program Encryption & Compression for Developers." URL: http://www.neoworx.com/products/neolite/default.asp. 4 April 2001.