

## CHAPTER 8

# Using the Forensic Server Project

Collecting data from a potentially compromised system is relatively simple. There are several methodologies for collecting data that an investigator can adapt to her needs. Some investigators may simply go to the “victim” system and run native tools at the console and any installed anti-virus software. Others may download tools from the Internet or a network drive or bring those tools with them on a diskette. Still others may take a more stringent approach to their investigative techniques in an effort to preserve potential evidence on the system, realizing that their actions will leave footprints of some kind and attempting to minimize the effects of their actions on the “victim” system.

It should go without saying that collecting and analyzing data from a potentially compromised system is paramount. When a system is suspected to have been compromised in some manner, simply reinstalling the system from “clean” media or from a known-good image can be just as bad as ignoring the issue. Without determining the nature of the incident, there is no way for the administrator or investigator to know how to protect the system. Placing that system back into production may lead to it being compromised all over again. In addition, other systems within the infrastructure may also have been compromised, so the investigator needs to understand the complete nature and scope of the incident. This can only be done by collecting and analyzing data from potentially compromised systems. In order to accomplish this, the investigator needs a methodology and toolkit that will allow her to quickly and efficiently gather and correlate data from multiple systems, if necessary, so that she can then make decisions and provide guidance regarding follow-up actions. Shooting from the hip and speculating about what happened can lead to a complete misunderstanding and misrepresentation of the issue, and the actions taken to resolve the issue may end up being completely ineffective.

## 358 Chapter 8 Using the Forensic Server Project

---

This chapter will initially cover the collection of data, but we have to realize that collecting the data is the easy part. Correlating the data and understanding what it means requires an additional step. How does the investigator find files or processes that the attacker has taken great effort to hide? What constitutes “suspicious” activity? The primary focus of this chapter will be to address the forensically sound collection of data from a system, but this chapter will also discuss how to understand the data that has been collected.

The actual Perl code for the Forensic Server Project (FSP) server and client components will not be included in this chapter, as has been done in previous chapters. The code and its function will be described in detail, but the actual code itself is hundreds of lines long. The code for the server component and the two client components described in this chapter is included on the accompanying CD.

### The Forensic Server Project

---

The preferred method of obtaining volatile (and some non-volatile) data from a Windows system in a forensically sound manner is to use netcat or cryptcat (see the “Netcat” sidebar in Chapter 3, *Data Hiding*). This methodology lets the investigator pipe the output of commands run from a CD through the network connection provided by netcat/cryptcat to a waiting listener on a remote system. However, this process still requires that the investigator record a good deal of documentation by hand, making the process cumbersome and unlikely to be used in all cases.

The purpose of the Forensic Server Project (FSP) is to provide a framework for performing forensically sound data collection from potentially compromised systems. The project accomplishes this by collecting data and transporting it to a waiting server via the system’s network interface. This way, files are not written to the potentially compromised system, as doing so will overwrite deleted files and potentially compromise a follow-up litigious investigation. The general framework for the FSP not only allows for it to be run from removable media, such as a USB-connected thumb drive, but with minor modifications to the code, it can also write data to those thumb drives.

The FSP consists of a server component that resides on a system managed by the investigator and client components that the investigator places on a CD (or thumb drive) for use on “victim” systems. The client

components retrieve information from the “victim” systems and send it to the server. In the current version of the FSP, the communications between the client components and the server is not encrypted. However, the open source nature of the FSP makes this capability easy to add, and this capability will be included in future versions of the FSP.

The client components communicate with the server by using verbs, or action identifier keywords. When the client wants the server to take a particular action, it will send a keyword, and the server will perform a set of predefined actions based on that keyword. The FSP uses the following keywords:

- **DATA**—The client component sends the DATA keyword to the server when it wants to send data, such as data collected from a file (MAC times, hashes, etc.), the output of an external command (i.e., external to Perl, such as a standalone executable), or data collected using Perl functions. Once the data has been written to a file (using a specified filename), the server will compute MD5 and SHA-1 hashes (see the “Hashes” sidebar) on the file and record them in the case log file.
- **FILE**—The FILE keyword is used to let the server know that a file is being copied from the client system, and it is preceded by a DATA command. The file data sent by the DATA command includes the full path of the file, MAC times, owner, and MD5 and SHA-1 hashes. After the FILE command is sent, the server responds with an “OK.” When the “OK” is received, the client will copy the file to the server. Once the file has been copied, the server will recalculate MD5 and SHA-1 hashes for the file and compare them to the hashes calculated before the file was copied. The server does this to verify that there were no errors or changes to the file while it was being transferred.
- **LOG**—The client sends this command to the server when it wants to add an entry to the case log file.
- **CLOSELOG**—This is the last command that the client sends to the server, telling the server that it no longer has data to send. When this command is received, the server will add one final entry to the case log file and then compute MD5 and SHA-1 hashes for the case log file. If the first responder or investigator moves to another system to collect information without changing the case information, the log file will be reopened, and at the end of the data collection it will be closed again, and the hashes will be recomputed for the new file.

The client components communicate with the server using TCP/IP in order to provide a greater level of flexibility in diverse network environments. Other protocols, such as FTP or Microsoft file sharing, can limit the communications between the components by requiring specific ports to be open. In some cases, communications may be required to pass through firewalls that limit a wide range of communications protocols and ports. By allowing any port to be used, the FSP provides a great deal of flexibility for a variety of network environments.

### Hashes

Professor Ronald L. Rivest of MIT developed the MD5 message digest algorithm. This algorithm takes an input message of arbitrary length and produces a 128-bit message digest, or “fingerprint.” According to the executive summary of RFC 1321<sup>1</sup> describing the MD5 message digest, it should be computationally infeasible to produce two messages having the same message digest. This characteristic makes the MD5 hash a powerful tool for ensuring the integrity of data. For example, an MD5 hash generated for a file will be different if so much as a single bit within that file is changed. For this reason, applications that monitor file systems for changes use the MD5 algorithm.

As the MD5 message digest algorithm provides for one-way encryption (i.e., the resulting digest cannot be decrypted), it provides an excellent facility for protecting information such as passwords and for generating hashes to ensure the integrity of data such as strings and files. The MD5 message digest algorithm is implemented in Perl via the Digest::MD5 module. This module is not part of the standard ActiveState Perl distribution but is easily installed via the Perl Package Manager (PPM).

The Secure Hash Algorithm 1 (SHA-1) was developed by the National Institute of Standards and Technology (NIST)<sup>2</sup> and is described in RFC 3174<sup>3</sup>. Similar to the MD5 algorithm, the SHA-1 takes an input message of arbitrary length and computes a 120-bit message digest. Like the MD5 message digest algorithm, SHA-1 provides an excellent mechanism for ensuring the integrity of files. The algorithm is implemented in Perl via the Digest::SHA1 module, which is also installed via PPM.

The server component also has facilities for performing analysis and correlation of the collected data, making it easier for the investigator to review it and make decisions. Some of these facilities, such as scripts for correlating data, come with the server. However, using a little imagination

1. See <http://www.faqs.org/rfcs/rfc1321.html>

2. See <http://www.nist.gov>

3. See <http://www.faqs.org/rfcs/rfc3174.html>

and Perl programming skill, the investigator can extend these capabilities and even create new ones.

The FSP is intended for use by the investigator. The FSP is an open source project, written in Perl so that it can be easily extended. As the FSP is written in Perl, the server component can be run on either Windows or Linux systems, and client components can be written in Python, Visual Basic, or other scripting or programming languages. The investigator can use specially designed client components to collect specific information from various systems, or a single client component can be designed for use by the first responder to collect a wide range of data. Using Perl, for example, the Forensic Server Project client components consist of the following:

- The First Responder Utility (FRU), or fru.pl, which automatically collects a wide range of volatile (and some non-volatile) data from the “victim” system. This component does not have the flexibility inherent in the other components and must be fully configured by the administrator or investigator prior to being written to a CD. The reason for this is to reduce the amount of interaction the first responder has with the system by automating the collection of a wide range of information. All the first responder has to do is insert the CD into the “victim” system, launch a known-good, “clean” command prompt (cmd.exe retrieved from a “clean” system) for the appropriate system, and launch the FRU (i.e., the fru.pl script) via a batch file. Once the GUI dialog appears, the first responder will enter the IP address and port of the Forensic Server and hit the “GO” button. As long as the server component is configured and running and is reachable via TCP/IP communications, the FRU will begin automatically transferring the data it collects to the server. The server will record the activity and store the collected data for later analysis. The FRU can then be moved to another machine and run again without restarting the server.
- The Forensic Server Project includes a component for copying files from the “victim” system. The investigator first uses the GUI to select the files she wants to copy from the “victim” system (i.e., web server log files, suspicious executables, etc.). This component will then automatically collect information from the selected files, such as their MAC times, hashes, and other information, before copying the file to the server. Once on the server, the file’s hash is verified to ensure its integrity during transport. (Note: This component is available with the version of the FSP shipped with this book.)

Other components can be easily created using Perl or any other programming language the investigator (or developer) chooses. Possible components include:

- A volatile information collector for retrieving information from the memory on the “victim” system, such as clipboard contents, processes, network connections, etc. This is a subset of what the FRU collects from a system, and it can be extended to include items such as the contents of process memory, etc.
- A component for running commands external to the programming or scripting language. This is also a subset of the functionality available in the FRU, and it can provide additional flexibility in the toolset for the investigator. The purpose of such a component would be to provide the investigator with necessary framework for running arbitrary commands instead of using a preconfigured, hard-coded component such as the FRU.

The function of the FSP is not only to facilitate the collection of information in a forensically sound manner but also to automate the documentation of that collection and to allow for the analysis of that data. When collecting information from systems using tools such as netcat or cryptcat (netcat with TwoFish encryption) by piping the output of the commands through the tool to a waiting server, the investigator needs to document each of the commands used. When copying files, specific information about each file (MAC times, hashes, etc.) needs to be collected and documented. Once the files have been copied, the hashes need to be verified. This can be a time-consuming, laborious process that is also prone to mistakes. The server component of the FSP automates the collection of information as well as the creation of documentation. When the server receives data or a file, it automatically calculates and logs (to the case log file) MD5 and SHA-1 hashes for the files.

### **Collecting Data Using FSP**

---

The client components of the FSP make it extremely easy to collect data from a “victim” system. However, this ease of functionality is in part due to the preparation and configuration of the client components by the investigator prior to deployment. The FSP is written in an interpreted language

such as Perl in order to make it relatively easy for the administrator to modify it to suit her particular needs. The client components to the FSP, with the exception of the First Responder Utility (FRU), are intended to be flexible and easy to use. Ease of use and automation (i.e., restricting the amount of interface with the application that is required of the first responder) are the key aspects of the FRU.

### Launching the Forensic Server

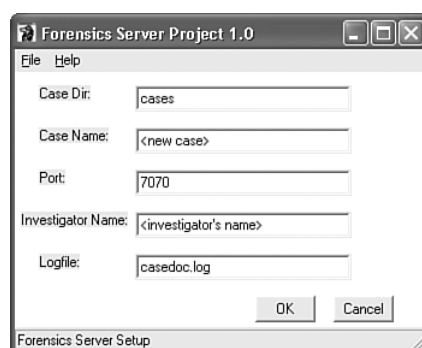
In order to use the FSP, the investigator launches the server component on the system where she wishes to store data. This system may be the investigator's workstation or a specific system set aside to be the FSP. This system should be physically secure, locked inside an office if the investigator herself needs to go on-site. The system should also be secured while on the network, with patches up-to-date and all unnecessary services disabled.

The investigator's system will also contain the various tools that the investigator will use to correlate and analyze the data she collects from the "victim" system. See the "Setting Up the FSP" sidebar.

Once the investigator's system is set up and prepared, the command to launch the FSP is:

```
C:\Perl\FSP>fsp.pl
```

This assumes, of course, that the investigator installed Perl on the C:\ drive and placed the scripts for the FSP in the C:\Perl\FSP directory. Once the investigator runs this command, she will be presented with a configuration dialog for setting up the server component, as illustrated in Figure 8-1.



**Figure 8-1** The initial configuration dialog for the FSP.

### Setting Up the FSP

The first step in setting up the Forensic Server Project (FSP) for deployment and use is to set up Perl on a system in accordance with Appendix A, *Installing Perl on Windows*. First install the ActiveState Perl distribution and then the Win32::GUI module (see Appendix A for the necessary instructions). Finally, install the Digest::MD5 and Digest::SHA1 modules using the following commands:

```
C:\perl>ppm install Digest-MD5
C:\perl>ppm install Digest-SHA1
```

These commands will install the modules necessary for computing MD5 and SHA-1 hashes. The only other module used by the FSP server component, IO::Socket (for handling TCP/IP communications), comes as part of the ActiveState Perl distribution.

The system used to set up the FSP server component does not need to have a CD-ROM writer installed, as the server component of the FSP does not need to be copied to a CD. It can be run from the system on which it is set up. The FSP should remain much more flexible and easily configured (the Perl scripts are more easily modified if they aren't copied to a CD), as it will generally only be used and controlled by the investigator. The investigator will want to be able to add any number of data examination and correlation tools to the FSP, using third-party tools, Perl, or any other programming language.

The version of the FSP included with this book makes use of relative paths. In essence, this means that the case directories created when using the FSP will be within the directory in which the FSP “lives,” where the FSP files are located. For example, the FSP was developed on a system in the D:\Perl\FSP directory. The main case directory, therefore, is D:\Perl\FSP\Cases, and all of the files collected from systems are in subdirectories.

Once you've installed Perl (version 5.8 from ActiveState was used for all development and testing for this book) and all necessary modules, copy all of the scripts used by the FSP into a directory on the system. For this book, the files were run from a directory named “FSP,” which is a subdirectory of the “Perl” directory.

The initial configuration dialog for the FSP has five text fields that need to be filled out by the investigator. These text fields allow the investigator to establish a case management structure in order to separate different cases. The first field, labeled “Case Dir,” refers to the directory in which the particular case directory will be created. A “case” refers to an event about which the investigator wishes to collect data. In this version of the FSP, the case directory is located immediately below the directory that FSP is

launched from, in this case, C:\Perl. The default setting for the “Case Dir” entry is simply “cases.”

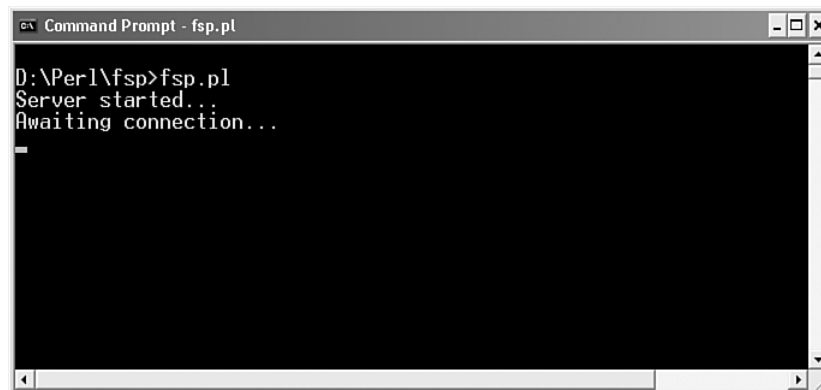
The second text field, with the label “Case Name,” allows the investigator to select a name for the case. This name will be given to a subdirectory within the case directory that is unique to the case. The default entry for the “Case Name” field is “<new case>.” The brackets are intended to remind the investigator to change the entry. The “Case Dir” and “Case Name” make up the case management component of the FSP, allowing the investigator to keep data collected from various systems and during various incidents separated.

The “Port” field is intended to allow the investigator to designate the port that the FSP listens on for connections. This field is specifically intended to be configurable, making it easy to use in a wide range of environments. For example, on an internal corporate infrastructure, the investigator may have no restrictions regarding network communications. However, when communicating with remotely connected offices, or even via the Internet, there may be firewall rulesets and router access control lists that come into play. The default port of 7070 may be acceptable when collecting data from a system located on the same local area network as the FSP. However, the investigator may be forced to configure the server to listen on port 80 if data is being collected from a system located in a remote location on a wide area network.

The “Investigator Name” field (“<investigator name>” by default) provides a place for the investigator to designate the name of the individual responsible for any portion of the investigation. When the server is started (i.e., when the investigator fills out the appropriate fields and clicks “OK”), the information in this field (as well as the “Port” field) is logged as part of the investigation documentation.

The final field, “Logfile,” is intended to provide the name of the file (“casedoc.log” by default) to which all of the case documentation will be logged. This file contains a time-stamped listing (based on the local time on the server) of all of the activity that occurs with the case while data is being collected. Once the investigator stops collecting data, logging stops.

Once the investigator fills out the fields appropriately, she launches the server by clicking the “OK” button. At that point, the GUI will disappear, and a record of activity will appear in the standard output (STDOUT) of the command prompt used to launch the server. This serves to give a visual record of what’s going on, while additional information is logged to the case logfile. Figure 8-2 illustrates the Forensic Server once it has been configured and is running, awaiting a connection.



```
Command Prompt - fsp.pl
D:\Perl\fsp>fsp.pl
Server started..
Awaiting connection...
_
```

**Figure 8-2** The Forensic Server running.

When all of the data has been collected, the investigator simply hits Ctrl+C to shut the server off. In order to do this, of course, the investigator needs to be seated at the console of the server.

As previously stated, the version of the FSP provided with this book does not support encrypted communications between the client components and the server. This functionality can be added by an investigator with the necessary Perl programming skills and will be included in future versions of the FSP.

### **Running the First Responder Utility**

In order to collect data using the Forensic Server, the investigator needs to run one or more of the client utilities on the “victim” system. One such client is the First Responder Utility, or FRU. The FRU is intended for use by first responders, which may be a system administrator. The FRU is deployed via CD-ROM (see Appendix A and the “Setting Up the FRU” sidebar), which the first responder will place in the CD-ROM drive of the “victim” system and then run the utility.

### Setting Up the FRU

The first step in setting up the First Responder Utility (FRU) for deployment and use is to set up Perl on a system in accordance with Appendix A. Once the ActiveState Perl distribution has been set up, install the following modules:

- Win32::GUI (see instructions in App. A)
- Win32::Lanman (see instructions in App. A)
- Win32::Perms (ppm install <http://www.roth.net/perl/packages/win32-perms.ppd>)
- Win32::API::Prototype (ppm install <http://www.roth.net/perl/packages/win32-api-prototye.ppd>)
- Win32::TaskScheduler (ppm install <http://taskscheduler.sourceforge.net/perl58/Win32-TaskScheduler.ppd>)
- Win32::DriveInfo (ppm install win32-driveinfo)
- Win32::IPConfig (ppm install win32-ipconfig)

All other modules used by the FRU are included with the ActiveState Perl distribution.

The system used to set up the FRU should have a CD-ROM writer installed with its associated software, or you should be able to copy the Perl directory to a system that has one, in order to create the First Responder CD. The same system that was used to set up the FSP can be used to set up the FRU.

The version of the FRU (as well as the FSP) included with this book makes use of relative paths. In essence, this means that all of the scripts that make up the client components and the tools used by the client components must be in the same directory.

Once you've installed Perl (version 5.8 from ActiveState was used for all development and testing for this book) and all necessary modules, create a directory within the "perl" directory called "fsp" and copy all of the scripts for the FRU into that directory. Also be sure to download all of the third-party utilities used by the FRU into that directory as well.

The version of the FRU included with this book uses the following third party utilities:

- Cmd.exe, the command interpreter on Windows systems (Note: You must ensure that you get copies of this program from "clean" systems. This means that if you are going to respond to incidents on Windows 2003 servers, you must get a copy of cmd.exe from this system.)
- Psloggedon.exe, pslist.exe, psloglist.exe, psinfo.exe, listdlls.exe, and handle.exe from SysInternals.com<sup>4</sup>
- Tlist.exe from the Microsoft Debugging Tools<sup>5</sup> (i.e., *not* the Resource Kit)

*(continued)*

4. See <http://www.sysinternals.com>

5. See <http://www.microsoft.com/whdc/ddk/debugging/default.mspx>

**Setting Up the FRU (cont.)**

- CmdLine.exe, iplist.exe, and openports.exe from DiamondCS<sup>6</sup> (Note: Licensing information on the DiamondCS web site states that openports.exe is free for personal use, as well as in public education institutes. A small licensing fee is required for use of this utility in commercial and business environments.)
- Rifiuti.exe and cygwin1.dll from FoundStone<sup>7</sup>
- Promiscdetect.exe and pstoreview.exe from NTSecurity.nu<sup>8</sup>
- Reg.exe and auditpol.exe from Microsoft

Each of these third party utilities is prefixed with “**fru\_**” when stored in the same directory as the fru.pl Perl script. This is hard-coded into the FRU Perl script (i.e., fru.pl) and intended to ensure that there are no issues with the tools or programs with the same names already being in the PATH on the “victim” system.

The source code for the FRU includes several “require” statements. These statements identify other Perl scripts that the FRU makes use of and depends upon, as these scripts contain additional code used by the FRU. When copying the FRU code from the CD that accompanies this book, be sure to include the following Perl scripts in the same directory as fru.pl:

- getos.pl (identifies the operating system)
- pclip.pl (retrieves the Clipboard contents)
- e\_cmd.pl (very important, as it provides a wrapper for running third-party executables)
- service.pl (retrieves service and device driver information)
- getsys.pl (retrieves system information)
- tasks.pl (retrieves information regarding Scheduled Tasks)
- regdump.pl (retrieves Registry information)
- mdmchk.pl (checks for installed modem drivers)
- shares.pl (retrieves information regarding available shares)
- dt.pl (retrieves drive information)
- ip.pl (retrieves IP configuration information)

Each of these additional scripts must be present for the FRU to function properly.

The command interpreter, cmd.exe, should be copied to the root directory when you are ready to create your CD. This way, when the CD is inserted into the CD-ROM drive (for example, F:\) of the “victim” system, the command prompt can be opened by clicking on the Start button, choosing Run, typing “F:\cmd.exe” and clicking the OK button. From there, type the following commands to launch the FRU:

6. See <http://www.diamondcs.com.au/>

7. See <http://www.foundstone.com>

8. See <http://www.ntsecurity.nu/toolbox/>

```
F:\>cd perl\fsp  
F:\perl\fsp>F:\perl\bin\perl.exe fru.pl
```

This assumes, of course, that the files for the FRU were placed in the FSP directory.

These commands can also be added to a batch file named “fru.bat”. This makes it easier for the first responder to launch the FRU. When the command prompt appears, the first responder must simply type “fru” to launch the FRU.

Once the FRU GUI is visible, the first responder simply enters the IP address and the correct port (if the default is not correct) of the FSP into the appropriate text fields, and clicks on the “Go” button.

Once the data collection is complete (i.e., the final command, “Close Log”, has been sent), the first responder simply clicks the “Exit” button, terminating the FRU, and withdraws the CD from the system. As long as the server is still running, the first responder can insert the CD into another “victim” system and repeat the process.

The FRU can be used from geographically remote locations, such as a wide area network in which there is TCP/IP connectivity between the “victim” system and server. The first responder places the CD-ROM containing the utilities in the CD drive of the “victim” system and then opens the appropriate command prompt for the operating system of the “victim” system. This is most easily done by clicking on the Start button, choosing “Run,” and then entering the path to the appropriate version of cmd.exe, located on the CD. When the prompt opens, the first responder will `cd` (i.e., change directories) to the appropriate directory and type `fru.pl` at the prompt in order to launch the FRU. Figure 8-3 illustrates the FRU GUI.

Once the FRU is running, the first responder has only to enter the IP address and port used by the Forensic Server and then click the “Go” button. The first responder may enter the IP address of the server as an argument at the command prompt, or she may enter it into the “Forensic Server IP” field in the GUI. The “Forensic Server Port” is set to 7070 by default, the same as the Forensic Server. However, in the case of both the server and clients, this port is configurable.

The FRU is completely automated in order to remove the first responder from the data collection process as much as possible. The FRU should be completely configured by the investigator prior to being copied to a CD (or USB thumb drive) so that the first responder won’t have to make any decisions regarding what information to collect.

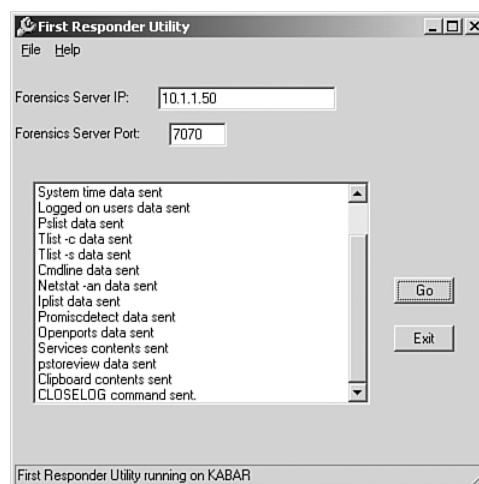
**370 Chapter 8 Using the Forensic Server Project****Figure 8-3** First Responder Utility GUI.

Once both the Forensic Server and the FRU have been correctly configured and the first responder clicks “Go,” the FRU will take over, limiting the interaction that the first responder has with the potentially compromised system.

The first thing the FRU does is attempt to ping the FSP server system itself. If the investigator’s FSP server system is Windows XP and has the Internet Connection Firewall (ICF) enabled, it will have to be disabled or configured to respond to ICMP pings and allow connections to the FSP server.

Once connected to the server, the FRU will automatically collect the data and send it out through the network to the waiting server. Throughout the data collection phase, as data is sent to the server, the server will compute MD5 and SHA-1 hashes of each of the files and record this information in the case log file. Figure 8-4 illustrates the FRU after it has completed collecting and sending data to the server.

As the FRU collects data and sends it to the server, the text area of the FRU GUI is updated so that the first responder will be able to observe the progress. The `CLOSELOG` command indicates to the first responder that the FRU has completed its data collection. All FSP clients send command verbs to the server in order to indicate what actions should be taken. The `CLOSELOG` command is always the last command to be sent, and when the server receives that command, it places a final entry in the case log file, closes it, and computes MD5 and SHA-1 hashes for the case log file. This



**Figure 8-4** The First Responder Utility after completing data collection.

provides a level of assurance that the log file integrity has been maintained, allowing the investigator to show that the log file has not been modified.

Figure 8-5 illustrates what the investigator sees at the Forensic Server as the FRU sends the data across the network.

After the server has been launched, it waits for a connection attempt from one of the client utilities. Once a connection has been created, the

```

c:\ Command Prompt - fsp.pl
D:\Perl\fsp>fsp.pl
Server started...
Awaiting connection...
Connection from 10.1.1.15
DATA command received: KABAR-psinfo.dat
KABAR-psinfo.dat closed.
Hashes computed for KABAR-psinfo.dat.
Awaiting connection...
Connection from 10.1.1.15
DATA command received: KABAR-systemtime.dat
KABAR-systemtime.dat closed.
Hashes computed for KABAR-systemtime.dat.
Awaiting connection...
Connection from 10.1.1.15
DATA command received: KABAR-loggedon.dat
KABAR-loggedon.dat closed.
Hashes computed for KABAR-loggedon.dat.
Awaiting connection...

```

**Figure 8-5** The activity of the Forensic Server while the FRU sends data.

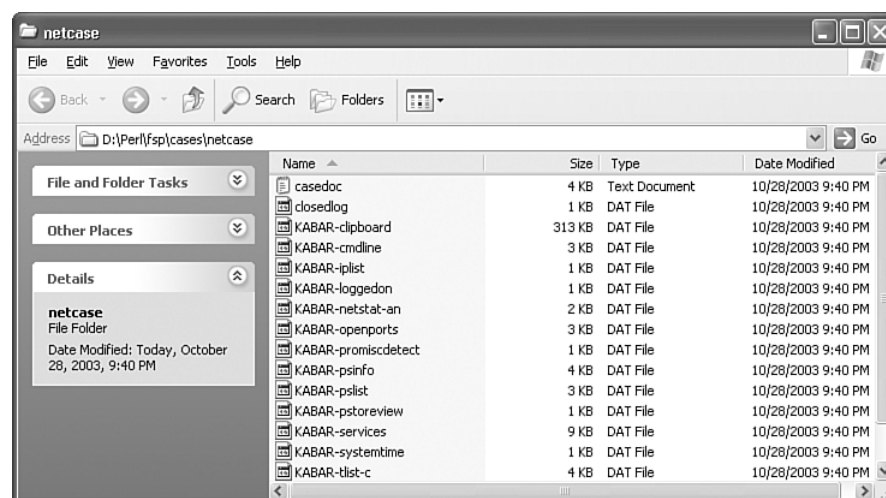
## 372 Chapter 8 Using the Forensic Server Project

server will receive and process the data that the client sends to it. In the case of the FRU, Figure 8-5 illustrates how the client connection is received (i.e., the FRU client is being run from the system using IP address 10.1.1.15), and how the command being sent is displayed. Again, this provides a visual indication to the investigator of the progress of the data collection. The activity status of the FSP that appears on the screen is also recorded in the case log file.

Figure 8-6 illustrates the contents of the case directory from an example case. All of the data is stored on the server in flat files. The naming convention for the files consists of the NetBIOS name of the “victim” system (i.e., “Kabar”), followed by the command that was run. The file extension is .dat to indicate that the files contain data.

Information collected by the FRU includes:

- Process information (tlist.exe, pslist.exe, listdlls.exe, handle.exe, cmdline.exe, openports.exe)
- Logged on user (psloggedon.exe)
- Network connection information (openports.exe, ip.pl, ipolist.exe, promiscdetect.exe)
- System information (psinfo.exe, getsys.pl)
- Protected storage information (pstoreview.exe)
- Clipboard contents (pclip.pl)



**Figure 8-6** Contents of the case directory after running the FRU.

- Service and device driver information (tlist.exe, service.pl)
- Scheduled tasks (tasks.pl)
- Registry information (regdump.pl, reg.exe)
- Modem drivers (mdmchk.pl)
- Information about shares (share.pl)
- Drive information (dt.pl)
- Audit policy (auditpol.exe)
- Event Logs (psloglist.exe)
- Recycle Bin contents (rifiuti.exe)

All of the third-party executables used by the FRU are stored in the same directory as the FRU Perl scripts. Each of the executables is prefixed with the tag “**fru\_**” (without the quotes) to avoid any issues that may occur if a malware file of the same name as one of the executables is located in the PATH. These executables allow the FRU to collect a variety of information from the “victim” system.

### File Client Component

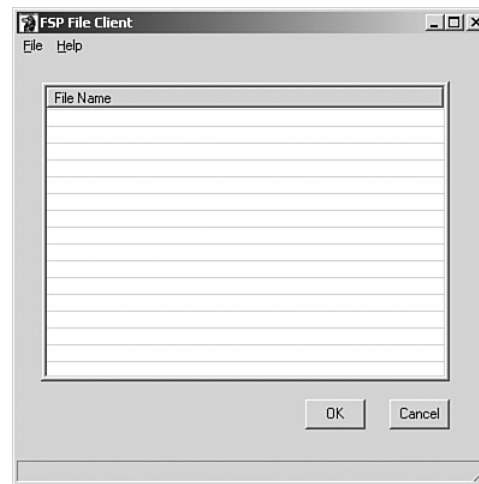
Setting up the file client component is similar to setting up the FSP server and FRU components. In fact, this component should be set up along with the FRU. The file client component uses the following modules that are not part of the ActiveState Perl distribution:

- Win32::GUI (see Appendix A for installation instructions)
- Win32::FileOp (ppm install Win32-FileOp)
- Digest::MD5 (ppm install Digest-MD5)
- Digest::SHA1 (ppm install Digest-SHA1)

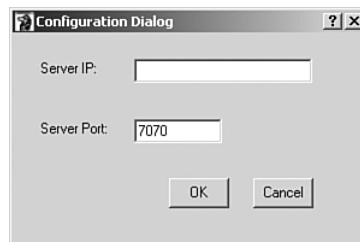
The file client component of the FSP provides a little more flexibility for the investigator with regards to the interface. The file client allows the investigator to copy files from the “victim” system to the FSP server. The initial interface to the file client (fcli.pl) is illustrated in Figure 8-7.

Once the interface is up, the investigator clicks on the File item in the menu bar and then on the Config item in the drop-down menu in order to configure the client to connect to the server. The Configuration Dialog will appear, as illustrated in Figure 8-8.

All the investigator needs to do is enter the IP address and port of the FSP server and click OK. As with the FRU and the FSP, the default port in the Configuration Dialog is 7070.

**374 Chapter 8 Using the Forensic Server Project**

**Figure 8-7** Initial interface to the file copy client of the FSP.

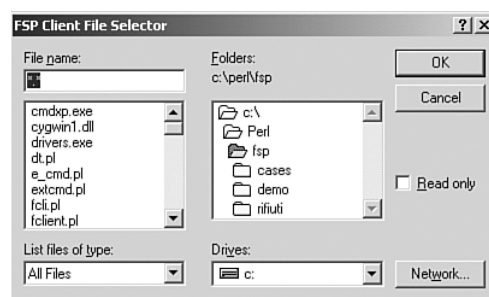


**Figure 8-8** Configuration Dialog of the file client component.

In order to select the files to be copied, the investigator chooses the Open item from the drop-down menu rather than the Config item. Clicking on the Open item opens the File Selector dialog, as illustrated in Figure 8-9.

The investigator can choose multiple files from each directory presented in the file selector dialog. When she does, she clicks OK, and the list of files in the main view of the file client component will be updated. If other files from other directories on the “victim” system also need to be copied, the investigator will reopen the file selector dialog and select those files.

Once all the files that the investigator is interested in have been selected (and the information in the Configuration Dialog has been verified, if need be), she then clicks the OK button in the main window, and the



**Figure 8-9** File Selector dialog of the file client component.

file client component takes care of the rest. As the file client moves through the list of files, it first collects information about each file, specifically the MAC times, full path, size in bytes, and MD5 and SHA-1 hashes. This information is then sent to the FSP server and saved in a file with the name of the original file and a .dat extension. Listing 8-1 illustrates the contents of an example .dat file created on the server.

**Listing 8-1** Contents of an example .dat file

```
ctime:Thu Aug 14 21:14:41 2003
sha1:bbe028069169da0f3b86b6b6ceb6cc58e1088ec8
mtime:Thu Aug 14 21:21:43 2003
name:C:\WINNT\system32\LogFiles\W3SVC1\ex030815.log
atime:Wed Mar 10 14:00:04 2004
md5:0195c718f03bca769189bc2694ba5875
size:359
```

The values of the .dat file are colon-separated so that they can be easily parsed and loaded into a Perl hash data structure for ease of use. The MAC times stored in the .dat file are based on the system time available on the “victim” system and are not converted or modified in any way once they appear on the server.

The file is then opened in binary mode and copied to the FSP server. Once the file has been copied, the FSP server component opens the associated dat file and attempts to verify the MD5 and SHA-1 hashes. All of this activity is logged. Listing 8-2 illustrates an excerpt of the case log file, showing the logged activity when a file is copied to the server.

**376 Chapter 8 Using the Forensic Server Project**

---

**Listing 8-2** Excerpt of case log file from file copy

---

```
Wed Mar 10 15:36:34 2004;DATA command received: ex030815.dat
Wed Mar 10 15:36:34 2004;HASH
ex030815.dat:d053eb8b0527691db3de041d3d173d18:aa47c9f63f1dc5eb86873b5a5e33db5a35
1ca5a7
Wed Mar 10 15:36:34 2004;FILE command received: ex030815.log
Wed Mar 10 15:36:34 2004;ex030815.log created and opened.
Wed Mar 10 15:36:34 2004;ex030815.log closed. Size 359
Wed Mar 10 15:36:34 2004;MD5 hashes confirmed for ex030815.log.
Wed Mar 10 15:36:34 2004;SHA-1 hashes confirmed for ex030815.log.
```

---

As illustrated in Listing 8-2, the server receives the initial DATA command, telling it that data is being sent. The file is then created on the server, and MD5 and SHA-1 hashes are generated for the newly created file. The values on the second line of Listing 8-2 are separated by colons so that they can be easily parsed and used for verification later. Each of the entries in the file is prefixed with a time stamp in order to document the progression of the activities. These time stamps are derived from the system time of the server.

The server then receives the FILE command, indicating that a file will be sent. The server receives all of the bytes sent by the client component and places them in a file on the server. Once the file is closed, the size of the file is recorded in the log file, and the MD5 and SHA-1 hashes are verified in order to ensure the integrity of the file, proving that it hasn't been modified in any way.

Once all of the files have been sent to the server, the file client (as with the FRU) sends the CLOSELOG command. Again, a final entry is added to the log file, after which it is closed. The server then generates hashes for the case log and saves those to a separate file. At that point, it is up to the investigator to maintain the physical security of the server system in order to ensure the integrity of the data saved on it.

Once the investigator has collected all of the files she's interested in and copied them to the server, she can use several of the tools (i.e., third-party tools and Perl scripts) listed in Chapter 5, *Incident Response Tools*, to analyze those files. For example, let's say the investigator uses the FRU or a similar tool to collect information from a system she thinks may have been compromised. Based on the information she collected, she decides to copy several files from the system via the file client component. Once she has

copied those files, she can use tools such as strings.exe, sigs.pl (from Chapter 3), and ver.pl to retrieve information from them and get a better idea of the extent of the issue. If any of the files copied are MS Office documents, such as Word documents, she can use tools such as wd.exe and meta.pl (discussed in Chapter 3) to locate hidden data within those documents.

The FRU and the file client are simply two components that can be used with the FSP server component. As all of the components are written in Perl and are open-source, they can be modified and extended to suit the needs of the investigator.

## **Correlating and Analyzing Data Using FSP**

Using the Forensic Server Project, collecting data from a “victim” system is relatively easy. However, the issue of correlating the data for analysis still needs to be addressed. Now that all of this data has been collected, what do we do with it?

The client components of the FSP are capable of collecting a wide range of data from Windows systems. Much of the data collected by these components is the result of external third-party utilities that send their output to the screen (i.e., standard output, or STDOUT) when run from the command prompt. All of these utilities send their output to STDOUT with their own formatting. When their output is captured and sent to the FSP, the result is many files, all with their data in different formats. This data needs to be parsed and put into a format that can be easily reviewed and understood.

Some of the data collected by the FRU can be easily understood. For example, the output of the promiscdetect tool (i.e., appears on the server as “<systemname>-promiscdetect.dat”) is fairly straightforward, in that if the network interface card (NIC) of the system were in promiscuous mode, indicating that a network sniffer is running, then the output would display information to that effect. The investigator can open the output file and quickly determine whether the network interface card is in promiscuous mode or not.

Other information isn’t quite as easy to pull from the data collected from the “victim” system. For example, a complete view of the process information from a Windows system is only possible by using several tools. Fortunately, one strength of the Perl programming language is in quickly

handling flat text files such as those produced by the FSP and FRU. The different ways that these files can be parsed and the information within them presented depends only on the programming skills and needs of the investigator. One of the primary needs of the investigator when trying to track down malware is to be able to correlate processes to the resources they're using, such as ports, as well as the command line used to launch the process. This information can be very useful to the investigator in locating malware or a suspicious process. However, the information required by the investigator is spread across several files. By making use of Perl, the investigator can quickly locate processes that may be harboring malware.

One example for the power of Perl when correlating data from across various files is a script entitled `procdmp.pl`. This script is a process dumper utility, in that it parses the contents of several files (all containing information specific to processes) and correlates that information, listed by process identifier (PID) in a single HTML file. The script correlates the process data from the various files and presents it in an easy-to-view HTML file. The `procdmp.pl` Perl script is used in the following Windows 2003 case study and is included on the CD that accompanies this book.

In order to illustrate the use of both the data collection and correlation aspects of the FSP, let's take a look at two case studies. The first is a Windows 2003 system that is "infected" with a network backdoor. The backdoor is, in reality, a copy of netcat renamed to `inetinfo.exe`, listening on port 80. The second case study is of a Windows 2000 system "infected" with the AFX Rootkit 2003 that was discussed in Chapter 7, *Knowing What To Look For*.

### **Infected Windows 2003 System**

In order to demonstrate the use of the FRU and FSP, a Windows 2003 system was "infected" with a network backdoor. The FSP is installed on a Windows XP laptop connected to a network via a wireless adapter. The FRU was created and tested on a Windows 2000 system and copied to a CD along with "clean" copies of `cmd.exe` and `netstat.exe` taken from a Windows 2003 system. None of the other utilities used with the FRU are native to the Windows 2003 system.

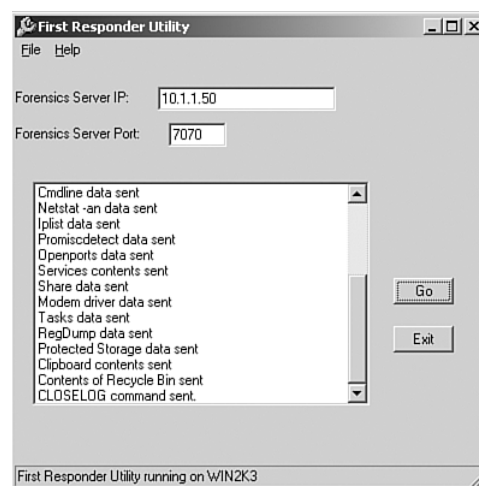
Once the FRU CD is available, it is inserted into the CD-ROM drive of the "infected" system. The command prompt from the CD is launched via the Run box (as described earlier), and the FRU is launched (also as described earlier). The IP address of the waiting FSP is entered into the "Forensic Server IP" field and the "Go" button is clicked.

Figure 8-10 illustrates what the FRU looks like once it has completed sending all data to the server.

The server was configured to place all files in a directory named “net-case2k3.” Now all the investigator needs to do is parse through all of these files, with the exception of the log files, looking for something unusual.

Since we have the advantage of knowing that the system was “infected” with a network backdoor, we have an idea that we may be looking for malware of some kind. With that in mind, the first thing we’ll do is run the procdmp.pl Perl script and examine the output HTML file. Figure 8-11 illustrates an excerpt from that file, showing a suspicious process.

The full HTML file, procdmp.html, is included on the accompanying CD-ROM.



**Figure 8-10** FRU after sending data from a Windows 2003 system.

<b>PID</b>	436		
<b>Process</b>	inetinfo		
<b>Command Line</b>	inetinfo -L -d -p 80 -e c:\windows\system32\cmd.exe		
<b>Context</b>	WIN2K3\Administrator		
<b>Ports</b>	TCP	80	0.0.0.0:LISTENING

**Figure 8-11** Excerpt from the procdmp.html file.

## 380 Chapter 8 Using the Forensic Server Project

---

On many Windows systems, finding a process called “inetinfo” running in the Task Manager is not unusual, as this is the name of the process that manages the Microsoft Internet Information Server (IIS) web server. Seeing the process name and noting in the output of `netstat -an` (add the `-o` switch for Windows XP and 2003) that port 80 is open and in `LISTENING` mode is not unusual. However, in this case it is unusual because the service usually associated with the IIS server (specifically the World Wide Web Publishing Service, or `W3SVC`) is not running. Not only that, the command line used to launch the process is *very* unusual and definitely not something one would normally see being used to launch IIS.

In fact, this is the network backdoor process. If it’s not already obvious from the command line for the process, this backdoor is really `netcat.exe` renamed to `inetinfo.exe`. The command line arguments indicate that the process is to listen for connections on port 80 and, when a connection is received, to launch a command prompt. Using `netcat.exe` in client mode to connect to the “infected” system on port 80 will bear this fact out. It is also important to note that the user context of the process is that of the local Administrator, meaning that anyone connecting to the system will have that same level of privileges.

### A Rootkit on a Windows 2000 System

The FRU was run on a Windows 2000 system infected with the AFX Rootkit 2003, described in Chapter 7. The server component was run on a Windows XP system where the FRU data was collected and stored. The `pd.pl` Perl script, illustrated in Listing 8-3, was used to parse through the output of several of the files created by the FRU, looking for discrepancies.

---

#### Listing 8-3 Perl script to parse the output of `tlist`, `pslist`, and `openports` from the FRU

---

```
#! d:\perl\bin\perl.exe
# pd.pl
# Parse output of pslist, tlist, openports from FRU
# Looking for discrepancies
use strict;

my $dir = shift || die "You must enter a directory name.\n";
$dir = $dir."\" unless ($dir =~ m/\\$/);
my @files;
my %systems;
```

```
if (-e $dir && -d $dir) {
    opendir(DIR,$dir) || die "Could not open $dir: $!\n";
    @files = readdir(DIR);
    close(DIR);
# First determine how many computers are in the case dir
    foreach (@files) {
        next unless ($_ =~ m/\.dat$/);
        my $sys = (split(/-/, $_, 2))[0];
        $systems{$sys} = 1;
    }
}

foreach my $sys (keys %systems) {
    my $pslist = $dir.$sys."-pslist\.dat";
    my $op      = $dir.$sys."-openports-fport\.dat";
    my $tlist   = $dir.$sys."-tlist-c\.dat";

    my %tlist = parse_tlist($tlist);
    my @op     = parse_op($op);
    my %pslist = parse_pslist($pslist);

    print "-----\n";
    print "$sys report\n";
    print "-----\n";
# Check to see what PIDs are in tlist, but not pslist
    print "Processes in pslist but not tlist\n";
    foreach my $key (keys %pslist) {
        print "PID: $key Process: $pslist{$key}\n" unless (exists
$tlist{$key});
    }
    print "\n";
    print "Processes in tlist but not pslist\n";
    foreach my $key (keys %tlist) {
        print "$tlist{$key} = $key\n" unless (exists $pslist{$key});
    }
    print "\n";
# Check to see what PIDs are in openports
    print "Processes in openports but not pslist/tlist\n";
```

*(continued)*

**382 Chapter 8 Using the Forensic Server Project**

---

**Listing 8-3** Perl script to parse the output of tlist, pslist, and openports from the FRU (*cont.*)

---

```
    foreach (@op) {
        my ($pid,$process,$port,$proto,$path) = split(/;/,$_,5);
        print "PID: $pid Port: $port Path: $path\n" unless (exists
$pslist{$pid});
        print "PID: $pid Port: $port Path: $path\n" unless (exists
$tlist{$pid});
    }
    print "\n";
}

#-----
# parse_tlist()
# parse the file generated by tlist -c
#-----

sub parse_tlist {
    my $file = $_[0];
    my @lines;
    my %data;
    open(FH,$file) || die "Could not open $file: $!\n";
    while (<FH>) {
        chomp;
        s/^\s+//;
        push(@lines,$_);
    }
    close(FH);
    my $fini = (scalar @lines) - 1;
    foreach my $i (0..$fini) {
        next unless ($lines[$i] =~ m/^\d+//);
        my($pid,$name,$title) = split(/\s+/, $lines[$i],3);
        my $cli = (split(/:/,$lines[$i+1],2))[1];
        $data{$pid} = $cli;
    }
    return %data;
}
```

```
#-----
# parse_op()
# parse the file resulting from "openports -fport"
#-----
sub parse_op {
    my $file = $_[0];
    my @data;

    open(FH, $file) || die "Could not open $file: $! \n";
    while(<FH>) {
        chomp;
        next unless ($_ =~ m/^\d/);
        my @proc = (split(/\s+/, $_, 6))[0,1,3,4,5];
        my $str = join(';', @proc);
        push(@data, $str);
    }
    close(FH);
    return @data;
}

#-----
# parse_pslist()
# parse the output of "pslist"
#-----
sub parse_pslist {
    my $file = $_[0];
    my %data;

    open(FH, $file) || die "Could not open $file: $! \n";
    while(<FH>) {
        chomp;
        my ($process, $pid) = (split(/\s+/, $_))[0,1];
        next unless ($pid =~ m/^\d+$/);
        next if ($pid =~ m/^\d\.\d+$/);
        $data{$pid} = $process;
    }
    close(FH);
    return %data;
}
```

---

**384 Chapter 8 Using the Forensic Server Project**

---

The `pd.pl` Perl script takes a path to a case directory as its only argument. The first thing the script does is parse through the case directory in order to determine from how many systems data was retrieved. Remember that the FRU can be run on multiple machines, all with the same instance of the FSP server component, and the case directory can contain data from multiple systems.

After getting the name of each system with data in the case directory, the script parses three specific files—the output of `tlist -c`, `pslist`, and `openports -fport`, as collected by the FRU. The purpose of parsing these files is to locate any discrepancies in process information among these three files in the hopes of locating rootkits. By using multiple tools that retrieve process information using different APIs, the hope is to locate processes that may be hidden from one tool but not from another. The script first looks for process identifiers (PIDs) that are listed in the output of `pslist` but not `tlist -c`. The script then switches this check around and looks for PIDs that are listed in `tlist -c` but not in `pslist`. Finally, the script checks to see what PIDs are listed in the output of `openports -fport` but not in either `pslist` or `tlist -c`. Again, the idea is that if a process or PID is visible in the output of `openports -fport`, then it should also be visible in other process enumeration tools. If it isn't, then it may indicate a process that has been hidden, possibly by a rootkit. In any case, any information that appears in the output of the Perl script should be investigated.

Listing 8-4 illustrates the output from running the `pd.pl` Perl script against the data collected by running the FRU against a Windows 2000 system infected with the AFX Rootkit. The script's output is sent to the screen (i.e., `STDOUT`), rather than to a fancy HTML file. The system in question was described in Chapter 7. The data collected from this system using the FRU is located on the accompanying CD in a zipped archive named `afx_2k.zip`.

**Listing 8-4** Result of `pd.pl` run on Windows 2000 system infected with a rootkit

---

```
-----  
KABAR report  
-----
```

```
Processes in pslist but not tlist  
PID: 1280 Process: afx_nc
```

```
Processes in tlist but not pslist
```

```
Processes in openports but not pslist/tlist  
PID: 1280 Port: 8180 Path: C:\afx_nc.exe
```

---

The output of the `pd.pl` Perl script clearly demonstrates why multiple tools should be used to enumerate process information from Windows systems that the investigator or administrator suspects may have been compromised or infected.

### **A Compromised Windows 2000 System**

For purposes of a reader exercise, a Windows 2000 system was compromised, and a network backdoor was installed. The FRU was run on this system, and the files created on the server are included on the CD in a zipped archive named `win2k.zip`. The reader is encouraged to explore and analyze these files. How was the system compromised? What did the attacker do to gain access to the system? What actions did the attacker take once he had gained access? How could the incident have been prevented?

---

## **Future Directions of the Forensic Server Project**

---

The Forensic Server Project is an open source project and can be modified in any way the investigator may see fit. The server can be run not just on Windows but on any platform that supports Perl. Clients can be created for specific platforms, with functionality that meets the specific needs of the investigator. All that is required is some Perl programming ability. The current version of the FSP provides the base functionality onto which additional capabilities can be built. Functionality such as the following can be added:

- Encrypted communications—The communications between the client and server can be encrypted to provide an added layer of protection, using Perl modules such as `Crypt::TripleDES` or `Crypt::TwoFish`.
- Running the server as a service—Using a variety of means, one of which is the `Win32::Daemon` module from Dave Roth, the server component can be run as a Windows service. This may be something that makes the FSP server more convenient for the investigator and perhaps for first responders.

**386 Chapter 8 Using the Forensic Server Project**

---

- Authentication—As an additional level of protection, the investigator may want to add authentication so that only specific users can send data to the server.
- Remote setup of the server—In addition to running the server as a service and providing authentication, the investigator may want to provide the functionality of allowing for remote setup of the server. This will require some modification to both the server and the client, but allowing remote setup provides a level of flexibility to the first responder.
- Support for multiple processes—Using the `fork()` functionality in Perl, the investigator can provide support for multiple processes running simultaneously. This would make things much easier in instances in which first responders are conducting data collection activities from multiple systems and from multiple locations all to the same server.
- Additional client components—The two client components addressed in this chapter provide a great deal of functionality but only serve as the basis for what's possible. Not only can additional client components be created, but also components can be developed for additional platforms.
- Additional analysis capability—As stated, data collection is usually the easy part when it comes to incident response and forensic audit investigations. Analyzing the data is can be difficult, particularly when the investigator is looking for specific activity. When dealing with large amounts of data, using some form of automation makes analysis of the data more accurate and more efficient. Investigators can craft tools to meet their specific needs or the specific needs of the investigation.

However, keep in mind that the Forensic Server Project is simply a framework that is implemented in Perl. Neither the server nor the client components need to be written in Perl. Other languages, such as Visual Basic, Python, or Ruby, can be used. In addition, the basic functionality outlined in the FSP can be expanded to include other capabilities or functionality as desired by the investigator.

---

## Summary

---

Not all investigations are litigious in nature. In fact, many investigations are conducted with no intention to prosecute the offender(s). Many times, the investigator is most interested in determining what happened, how to fix it, and how to prevent it from happening to other systems. A stringent methodology should still be used, but that methodology will need to meet several criteria. While the methodology must retrieve data in a forensically sound manner, it must also be quick, efficient, and easy to use. It should also require very little interaction from the first responder in order to collect the data but provide a degree of flexibility to the investigator when it comes to correlating and analyzing the data. The Forensic Server Project meets these needs.

The FRU, used in conjunction with FSP server component, provides an automated collection, transport, and documentation mechanism for the use of a variety of third-party and native tools. The Perl programming language not only acts as the “glue” language to encapsulate the necessary functionality but also provides a quick and easy means for parsing the collected data for information of interest, such as discrepancies in process information.

Up to now, we’ve focused on finding evidence of an incident on a single host. The Forensic Server Project can quickly and easily be used to collect and analyze data from several hosts. Chapter 9, *Scanners and Sniffers*, will take this a step further by demonstrating tools and processes for retrieving additional data from the network itself.

