File Marshal: Automatic Extraction of Peer-to-Peer Data

By

Frank Adelstein and Rob Joyce

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2007 USA

Pittsburgh, PA (Aug 13th - 15th)

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

File Marshal: Automatic extraction of peer-to-peer data

*Frank Adelstein**, *Robert A. Joyce*

ATC-NY, 33 Thornwood Drive, Suite 500, Ithaca, NY 14850, United States

Keywords:
Peer-to-peer
P2P
Forensics
LimeWire
File sharing

ABSTRACT

Digital forensic investigators often find peer-to-peer, or file sharing, software present on the computers, or the images of the disks, that they examine. Investigators must first determine what P2P software is present and where the associated information is stored, retrieve the information from the appropriate directories, and *then* analyze the results. File Marshal is a tool that will automatically detect and analyze peer-to-peer client use on a disk. The tool automates what is currently a manual and labor intensive process. It will determine what clients currently are or have been installed on a machine, and then extracts per-user usage information, specifically a list of peer servers contacted, and files that were shared and downloaded. The tool was designed to perform its actions in a forensically sound way, including maintaining a detailed audit trail of all actions performed. File Marshal is extensible, using a configuration file to specify details about specific peer-to-peer clients (e.g., location of log files and registry keys indicating installation). This paper describes the general design and features of File Marshal, its current status, and the plans for continued development and release. When complete, File Marshal, a National Institute of Justice funded effort, will be disseminated to law enforcement at no cost.

1. Introduction

The FBI defines *cybercrimes* as actions by "people who use the Internet and computers to illegally penetrate business and government computer systems, including stealing trade secrets and intellectual property, trafficking in child pornography, enticing children from the safety of their homes, and attacking critical infrastructure such as computer networks and power grids" (RCFL Program Annual Report, 2006). In their 2006 Annual Report, the FBI classifies crimes into eight types: terrorism, counterintelligence, cybercrimes, public corruption, civil rights, organized crime, white collar crime, and major theft/violent crime. Cybercrime is the number one category of crime investigated at 11 of the 13 FBI Regional Computer Forensic Labs (RCFLs), and is number two at the remaining two labs. Further, specifically, child pornography or exploitation comprises 35% of the cases of the Philadelphia RCFL, 38% of the cases for the Rocky Mountain RCFL, 51% of the cases for the Intermountain West RCFL, and 65% of the cases for the Western New York RCFL (RCFL Program Annual Report, 2006).

Often, peer-to-peer (P2P) file sharing networks are widely used in these crimes, and represent a significant source of evidence on computers that are under investigation. Of particular interest to investigators are the configuration parameters (user name, password, peers/servers used), times of use, time of install, log files of any transactions, and the downloaded (or shared) files themselves. Currently, an investigator must gather, categorize, and analyze all of this information by hand. This typically requires the investigator to research the specific P2P software to determine the location on the disk where the software stores files, the names of configuration files, and their content. In addition, the investigator may need to obtain some secondary software (beyond the

* *Corresponding author.*
E-mail addresses: fadelstein@atc-nycorp.com (F. Adelstein), rob@atc-nycorp.com (R.A. Joyce).
1742-2876/$ – see front matter © 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2007.06.016

investigator's normal tools) that translates a log or cache file into a human-readable format. Clearly, this is a time-consuming process, can yield inconsistencies, and can result in problems with the forensic integrity of the examination. Investigators need tools that automate this process and bundle together all of the information relating to each P2P network.

Currently there are a few dozen networks and several dozen P2P programs in general use on the Internet. While a small handful of programs comprise the majority of P2P usage, each program is slightly different. Having a customizable tool that processes evidence from many different P2P clients would be of great benefit to investigators. Few tools exist for examining P2P systems. KaZAlyser, probably the best known, analyzes FastTrack-based systems, such as Kazaa, iMesh, and Grokster. However, it is not extensible to non-FastTrack systems, and analyzes the database files generated by the P2P client *after* they have been found. It does not determine what clients have been used. General purpose forensic tools like EnCase can be extended through scripting, but are not designed to analyze P2P evidence and cannot easily parse P2P systems' configuration files or database formats. In general, most P2P analysis is done by hand.

The problem is how this automated extraction and analysis can be done in a cost-effective, yet extensible way. It is essential that the analysis tool must allow knowledge of new P2P software tools to be added through the use of ''plug-ins'' or configuration files. In addition, the process must be done in a forensically valid way. Specifically, it must give consistent and accurate results for every run. The process cannot be a ''black box''; it must be well documented.

2. File Marshal

File Marshal is a digital forensic tool for the extraction and analysis of data from peer-to-peer software on client machines. File Marshal will automate the tedious and time-consuming process of looking for evidence of peer-to-peer usage. File Marshal will perform these tasks in a forensically valid way and present them in an easily readable form on-screen and in a format that can easily be incorporated into a report. File Marshal's modular, extensible design will make it possible to add extensions for new types of peer-to-peer clients and networks.

File Marshal quickly determines what P2P clients were present on a disk image and presents per-user information on those clients, including shared files, downloaded files, and peer servers. File Marshal consists of two components, the graphical user interface, called the *front-end*, and the command-line based *back-end*. The front-end mediates interactions with the investigator and formats the data. The back-end searches the file system for directories and files, and interprets the contents of registry files.

In this section, we describe the overall operation and capabilities of File Marshal. The first subsection describes the three modes of operation. The next subsection describes logging and report generation, followed by a description of search capabilities. The registry library is described next, followed by the File Marshal user interface and the back-end configuration file.

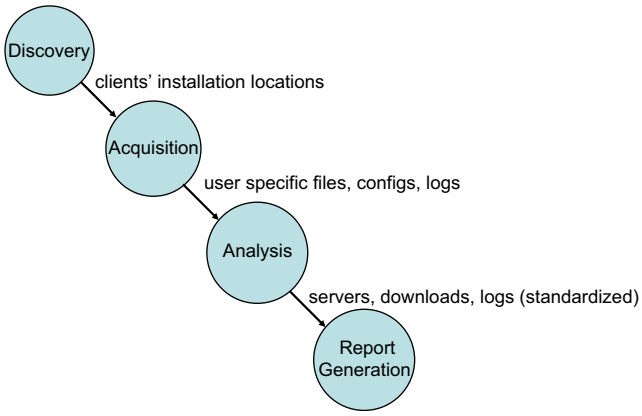

Fig. 1 – File Marshal investigation process.

2.1. Three phases of operation

File Marshal operates on a mounted disk image.[1] An investigator invokes File Marshal, creates an *inquiry*, and starts the analysis. There are three phases to the investigation: discovery, acquisition, and analysis, plus report generation at the end. Fig. 1 shows the phases and the information each phase passes to the next.

In the *discovery* phase, File Marshal examines the target disk image and determines what peer-to-peer clients are currently, or were previously, installed. To perform this check, File Marshal looks for the presence of files, directories, and registry keys and values. The configuration file, discussed in Section 2.6, specifies the artifacts that indicate if a particular client was installed. In some cases the programs may have been deleted, but the data directory remains. Registry keys for user preferences may also persist after the user uninstalls the P2P client, or reside in backup versions of the registry generated when the operating system creates a system restore (check)point. Files are specified by a pathname. In addition, they can be specified by a hash (currently MD5, but others can be supported). Registry entries can include the (sub)keys, values, and their data.

In the *acquisition* phase, File Marshal gathers user-specific usage information for specific P2P clients. For each user, File Marshal gathers configuration and log information, including peer or bootstrap servers contacted, files downloaded and shared, and other forensically relevant data maintained by the specific P2P client. Again, the specific files are defined in the configuration file. If special code is required to display a file (e.g., to decode a hash list or date format), the configuration file lists the Java modules (classes) to be used for parsing; new parsers can be created as needed using a straightforward API.

In the *analysis* phase, File Marshal displays the information gathered, and allows an investigator to view details, such as the contents of files, and sort data by various fields (IP number

[1] While nothing in the design and implementation precludes File Marshal from running on a live system, this effort focuses on analyzing disk images.

Fig. 2 – File Marshal audit log.

of server, date last contacted, etc.). Investigators can view downloaded files by launching an appropriate viewer (e.g., Acrobat for PDF, Firefox for HTML, Photoshop for an image), and display details on configuration and log entries, as well as search for files based on hashes or a set of hashes, such as NIST's National Software Reference Library (NSRL) or the National Center for Missing and Exploited Children's (NMEC) databases.[2]

2.2. *Logging and report generation*

File Marshal logs all operations it performs. The log file provides very detailed, low-level information on what actions were performed, thus maintaining the forensic integrity of the investigation. The log file provides details on how the back-end tool was invoked, as well as any return or error codes. Fig. 2 shows an example of viewing the audit log, which contains entries showing the actions taken during the acquisition mode. The audit log is not intended to be easily readable by humans, but rather it allows investigators to verify exactly what actions were taken (and by the same token, what was *not* done) during an investigation, and would be appropriate to be included as an appendix in a report.

File Marshal generates a summary report of the findings in a format that can be included in an investigator's report. Initially, supported formats will include HTML and PDF, so that File Marshal reports can be easily inserted into a larger forensics report.

As part of maintaining forensic integrity, File Marshal automatically hashes the output from the back-end tool. File Marshal can also compute the hash of any acquired file. Since the investigator will be using a static file system, the tool used to originally image the disk may have already computed hashes of these files. In that case, the forensic integrity of the investigation is strengthened; the hashes computed by File Marshal corroborate the data it finds with that of the imaging tools.

2.3. *Search function*

File Marshal will allow the investigator to search for various usage-specific items. This includes IP addresses and DNS names of peer servers, names of files, and file hashes (MD5, SHA-1, etc.). For instance, if an investigator wants to trace all contacts with a particular sever, the search tool would return all contacts regardless of the P2P client or clients used.

2.4. *Registry library*

In the discovery phase, File Marshal looks for artifacts indicating that a P2P program has been installed or used. One artifact it examines is the registry. However, because File Marshal performs an offline analysis of static registry files, there is little support for retrieving keys and values from a file (as opposed to the registry of the running system). Therefore, using documentation of the registry file format found on the Internet, we created our own library to support parsing and interpreting registry files.[3] The library routines support enumerating keys

[2] NSRL is available at http://www.nsrl.nist.gov. NMEC is only available to law enforcement.

[3] Limited information on the format of the registry file exists. We used the information from the URL http://home.eunet.no/pnordahl/ntpasswd/WinReg.txt as a source and verified the results using regedit.

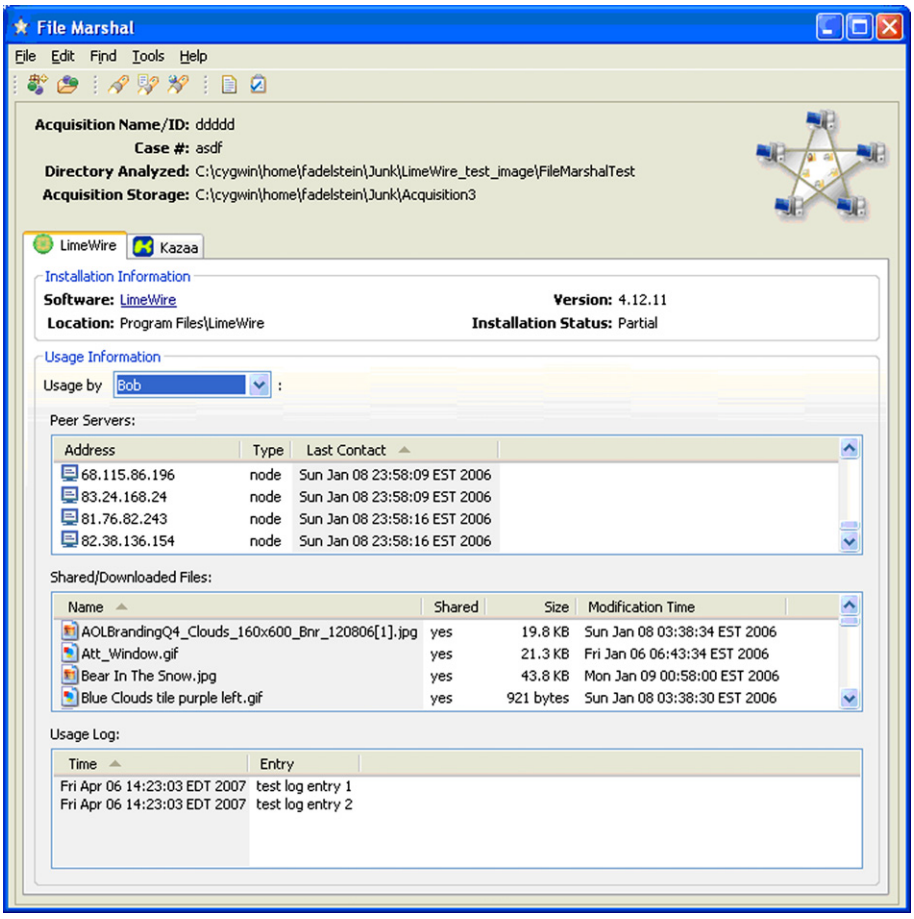

Fig. 3 – The File Marshal user interface.

(and subkeys), their values, the type associated with each value, and the data content of each value. In addition, it supports directly looking up a key by name. It will also support retrieving the mtime (modification time) and the access control list (ACL) associated with a key.

Because the library supports reading any registry file, File Marshal could examine alternate or backup registry files, in case some of the keys had been purged from the active registry when the computer was seized and the disk imaged. This will provide greater flexibility with respect to detecting the presence of peer-to-peer software.

As this library matures, we will make it available as an open-source tool for inclusion in other software or forensic research.

2.5. *User interface*

The File Marshal user interface, shown in Fig. 3, presents information about each P2P client it detects.[4] Within each tab (one tab per client), it presents information specific to each user account in the disk image that has evidence relating to using that client. In the example, two client tabs are shown, LimeWire and Kazaa, with the LimeWire tab selected.

The installation information provides details about where the client was installed, what version, and whether it is a full or partial installation. Partial installation indicates that a P2P client has been on the system but has been (at least) partially removed. In addition, a web page link provides more information about the client when clicked. Future versions will show more installation details, such as the install date, if available.

The usage section describes how that client was used by specific users. A pull-down menu allows the investigator to select individual users or ''All users combined'' to view all P2P activity on the disk image. At the bottom of the window, three tables provide summary information on peer servers, shared files, and log entries.

2.6. *Configuration file*

The differences between most peer-to-peer client programs are generally limited to the file paths they use, and the format of the information in configuration, cache, and log files. File Marshal uses a configuration file to specify the particular details of a peer-to-peer client. Adding support for a new client requires creating a new configuration file to describe the

[4] Note: The figure shows a test case in which LimeWire was used to gather various legal images from public sites.

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientConfig xmlns=
       "http://www.atc-nycorp.com/filemarshal/acquisitionconfig/2007/01"
       version="0.8">
  <!-- Generic name -->
  <Network>Gnutella</Network>
  <!-- Specific instance name -->
  <Name>LimeWire Pro</Name>
  <Version>2.3b</Version>
  <Module>com.atc_nycorp.filemarshal.module.limewire</Module>

  <InstallationArtifacts>
    <Directory name="Program Files/LimeWire"/>
    <File name="Program Files/LimeWire/LimeWire.exe"
          md5="9fe8ed98b63ca6ac4dabf15025482916"
          version="4.12.11" />
    <File name="Program Files/LimeWire/LimeWire.jar"
          md5="0b27fda0142b4b9559f936a9c3ebbdb8" />
  </InstallationArtifacts>

  <UsageArtifacts>
    <!-- Acquire mode -->
    <Data type="config" name="limewire.props" path="$FM_USER/.limewire" />
    <Data type="log" name="createtimes.cache" path="$FM_USER/.limewire" />
    <Data type="cache" path="$FM_USER/.limewire/fileurns.cache" />
    <Data type="log" path="$FM_INSTALL/install.log" />
    <Data type="cache" path="$FM_USER/.limewire/gnutella.net" />
    <Data type="shared" path="$FM_USER/Shared" />
    <Data type="shared" path="$SYSTEMROOT/limewire" />
  </UsageArtifacts>

</ClientConfig>
```

Fig. 4 – LimeWire client configuration file (XML format).

new client, and possibly adding a module (or plug-in) to the user interface to display any information that is unique to the new client. This allows File Marshal to be easily extensible to support new P2P clients that are released.

The configuration file is an XML file. It has three sections: *client data*, *installation artifacts*, and *usage artifacts*. An example of a client configuration file for LimeWire is shown in Fig. 4.

The first section specifies *client data*. This includes details about the P2P client, including its name and version, and the name of the module that handles displaying information about this client. The example specifies that this file describes version 2.3b of LimeWire Pro, which is part of the Gnutella class of peer-to-peer networks.

The second section specifies *installation artifacts*, for example files, directories, and registry keys, which indicate that the client is or has been installed on the system. The files and directories are specified by path. Also, files can include an MD5 hash attribute to match the content of the file in addition to its name, and a version attribute to indicate that the file is the binary for a specific version of the client. This supersedes the version information in the client data section. If all entries in the installation artifacts match, File Marshal describes it as a ''full'' installation. Similarly, if only some entries match, it is described as a ''partial'' installation. If none match, File Marshal will not display information on the client. However, the report will include a list of all clients for which File Marshal searched. In the above example, the artifacts include a directory, the executable file, and a ''.jar'' file. Since LimeWire does not use installation-wide registry keys, no registry keys appear in this example.

The third section specifies *usage artifacts*, which include four types of files: log, config, cache, and shared. Log files contain information about how the program was run, for example search terms that were used. Configuration files specify how the client is set up, and may specify where log files are stored. Cache files store temporary results, such as what peer servers have been used, or what files have already been retrieved. Finally, shared folders contain downloaded files and files that are to be shared with others. File Marshal will distinguish downloaded and shared files, if the P2P client does. In the above example, `limewire.props` is the configuration file, `install.log` is the log file, `Shared` and `limewire`, in different directories, are two directories for shared files, and `fileurns.cache` and `gnutella.net` are cache files.

The literal `$FM_USER` is replaced by all user home directories on the disk image. Specifically, File Marshal will iterate over every user directory, searching for files that match the pattern specified in the path. `$FM_INSTALL` is replaced by the base directory in which the client is installed. All other names that start with a dollar sign ($) are replaced with the corresponding environment variable, if it exists, or an empty string otherwise.

The ''module'' specified in the client data section of the configuration file, e.g., `com.atc_nycorp.filemarshal.module.limewire`, is Java code that the graphical user interface (GUI) front-end uses to display the files, specifically client-specific files such as log and configuration files. For example, the display module shows log file dates in a uniform way in the interface even if one client's log file uses year–month–day and another uses number of seconds since the Unix epoch (January 1, 1970).

3. Current status and future plans

File Marshal is a work-in-progress, currently under development through a grant from the National Institute of Justice. We have an initial prototype that demonstrates the capabilities, and plan for a beta-release at the end of summer 2007. In early 2008, the File Marshal product will be made available to law enforcement at no cost.

The current version of File Marshal supports only the LimeWire[5] peer-to-peer client. We will add support for different types of file sharing clients, including BitTorrent and Google's *hello* program. We had originally planned to support Kazaa early on, but our law enforcement contacts report that its use has dramatically declined, so we have reduced its priority. To increase the diversity of evidence File Marshal can acquire, we may add support for analyzing log files for non-file sharing programs, such as Skype.

We have implemented the discovery and acquisition phases, and have designed the analysis phase. We have completed a proof-of-concept of File Marshal to demonstrate to our law enforcement partners, to enable us to gather feedback on our designs. At the end of the summer, we will release a beta version, which will include analysis tools such as searching based on server and file names, and file hashes. The beta version will also include support for BitTorrent.

File Marshal currently runs on Microsoft Windows (2000 or better), since a majority of investigators use that platform. However, the code is easily portable to Unix and similar operating systems, such as Linux and MacOSX, because the front-end is written in Java and the back-end is written in C using minimal operating system specific (i.e., non-POSIX) function calls. This will permit File Marshal to run a forensic analysis of any of these machines, i.e., of both Windows and non-Windows disk images. File Marshal can analyze non-windows disks by using third-party software to mount the non-Windows image, or unpacking the files from the Unix disk on a Windows disk, by using a utility such as tar or unzip.

We will also add support for ACLs and key modification time in the registry library.

After incorporating features based on feedback from our beta testers, we will release the File Marshal product in early 2008.

Acknowledgements

This project was supported by Award No. 2006-DN-BX-K013 awarded by the National Institute of Justice, Office of Justice Programs, US Department of Justice. The opinions, findings, and conclusions or recommendations expressed in this publication/program/exhibition are those of the author(s) and do not necessarily reflect the views of the Department of Justice.

REFERENCES

KaZAlyser. Sanderson forensics. Available from: <http://www.sandersonforensics.com/kazalyser.htm>.

RCFL program annual report for fiscal year 2006. US Department of Justice, Federal Bureau of Investigation. Downloaded on May 16, 2007 from: <http://www.rcfl.gov/Downloads/Documents/RCFL_Nat_Annual06.pdf>.

Dr. Frank Adelstein is the Technical Director of Computer Security at ATC-NY, and provides oversight and guidance to projects at ATC-NY relating to computer security. His areas of expertise include digital forensics, intrusion detection, networking, and wireless systems. He has co-authored a book on mobile and pervasive computing. He received his GIAC Certified Forensic Analyst certification in 2004. A recent research focus is in the area of live forensics. He was the Principal Investigator on a project that resulted in the OnLine Digital Forensic Suite™, a live forensics tool. He is the vice-chair of the Digital Forensics Research Workshop.

Dr. Robert A. Joyce is the Technical Director for Information Management at ATC-NY. His research interests include distributed information storage and transformation, computer forensics, image and video processing, network and media security, visualization and design, and human–computer interaction. Since joining ATC-NY in 2002, he has led several research and development efforts in the area of information management and has made significant contributions to many other projects within the organization. He was a substantial contributor to the development of the OnLine Digital Forensic Suite™.

[5] http://www.limewire.com.