Bluepipe: A Scalable Architecture for On-the-Spot Digital Forensics

Yun Gao
Golden G. Richard III
Vassil Roussev

Department of Computer Science
University of New Orleans

Abstract

Traditional digital forensics methods are based on the in-depth examination of computer systems in a lab setting. Such methods are standard practice in acquiring digital evidence and are indispensable as an investigative approach. However, they are also relatively heavyweight and expensive and require significant expertise on part of the investigator. Thus, they cannot be applied on a wider scale and, in particular, they cannot be used as a tool by regular law enforcement officers in their daily work.

This paper argues for the need for on-the-spot digital forensics tools that supplement lab methods and discuss the specific user and software engineering requirements for such tools. The authors present the *Bluepipe* architecture for on-the-spot investigation and the *Bluepipe* remote forensics protocol that they have developed and relate them to a set of requirements. They also discuss some of the details of their ongoing prototype implementation.

Introduction

The current approach for obtaining digital evidence is based on impounding the suspect system (or media) and examining it in a forensics lab, where all analysis is carried out (usually on a copy of the original data). Several non-trivial problems are inherent in this process:

Impounding potential evidence is invasive. Most businesses are highly dependent on their IT infrastructure for daily operations. Hence, there would be a natural reluctance to shut down operations and part with their equipment without a warrant.

Users have legitimate privacy concerns. Even if users wish to be fully cooperative, they may be hesitant, for example, to allow sensitive company data (ultimately irrelevant to the inquiry) to be processed outside the company.

Logistical problems in moving computer equipment . Physically moving a large amount of computer equipment to a forensics lab and ensuring the safety and integrity

of the equipment is a difficult process. Computers can be easily damaged during the move and require safekeeping so they can be returned to owners in their original condition.

Not all data is relevant. Typically, only a small fraction of the examined data is of interest in an investigation (e.g., one or two rogue machines out of tens, or hundreds). Thus, a lot of the effort in copying and carefully examining a large number of targets will be in vain.

The only solution to these problems is to perform a preliminary screening of the evidence on-the-spot. An on-the-spot investigation allows targets to be examined in a non-invasive way that is sensitive to privacy concerns. Removal of machines and wholesale copying is avoided unless potential evidence is discovered. Clearly, this benefits both the investigation and the owner of the equipment.

This approach involves building a system that allows an investigator to safely use the resources of one or more target machines to perform a preliminary inquiry. It allows an investigator to control the simultaneous investigation of a number of machines using a PDA or laptop and to collaborate in real time with off-site experts. The Bluepipe architecture addresses the technical issues of building such a system for collaborative, on-the-spot digital forensics investigation. It has been developed with significant input from digital forensics investigators from local, state, and federal agencies, working in the Gulf Coast Computer Forensics Laboratory (GCCFL).

Usage Scenarios

This work is applicable in many scenarios, including:

SOHO (Small Office/Home Office). This scenario encompasses a small-scale inquiry of a few machines for personal and/or small business use. Using this system, an investigator performing a preliminary investigation of a home office can quickly determine if sufficient evidence exists for obtaining a search warrant, seizing equipment, and performing a traditional in-depth investigation in a forensics laboratory. This preliminary investigation could even be carried out by a relatively inexperienced person, collaborating in real time with a remote expert.

School. This scenario represents a middle-sized inquiry on the order of tens of machines. An important aspect here is that the network infrastructure is likely to be less capable and the administration less consistent and sophisticated. A preliminary investigation in this scenario will allow an investigator to quickly identify interesting targets and can substantially reduce the number of machines that must be seized for further investigation.

Small company. This scenario is similar in scale to the previous one but the infrastructure and administration are likely to be more advanced and, therefore, potentially

more valuable in facilitating an inquiry. Still, having lightweight, non-invasive capabilities for on-the-spot examination can be instrumental in alleviating some of the company's privacy and operational concerns and facilitating cooperation.

Corporation. This scenario is essentially the same as the previous one, with the added issue of a grander scale—on the order of hundreds to thousands of target systems. In a way, this is the extreme case that demonstrates the limitations of lab-based methods. In this scenario, a preliminary screening of target machines can substantially reduce the number of machines that might need a thorough examination in a laboratory setting.

Technical Requirements

The primary requirements for on-the-spot digital forensics tools are as follows:

Verifiability. Certainly the most important requirement for any digital forensics tool is that it must produce results that are permissible in a court of law. In practice, this means that the prosecution must be able to demonstrate to a jury/judge that all results are genuine and reproducible, and that the original source has not been tampered with. Ideally, such claims should be third party verifiable to eliminate any doubts about the presented evidence (e.g., NIST performs certification tests that go a long way towards raising the credibility of a digital forensics tool). A more subtle point here is that it is also highly desirable for any methods used to be simple enough (in principle) so that their basics can be explained to the common juror who may not be particularly computer literate. Finally, we note that the software should support the established, standard procedures for handling digital evidence.

Portability. This requirement is practically self-evident—the whole idea behind on-the-spot tools is to take the investigation to the target. Hence, it must be possible, in fact easy, to do so. Portability has two main aspects—portable computing and portable communication infrastructure. In recent years, in addition to laptop computers, a number of general-purpose portable devices, such as PDAs, have become powerful enough to be used in investigative work. Similarly, wireless communication technologies, such as Bluetooth, 802.11, and 3G cellular networks offer practical and economical ways for building an on-the-spot communication infrastructure that is independent of the target systems. Thus, instead of expending time and effort on taking over the existing communication infrastructure on the target, investigators could quickly set up their own and get on with their work. Another important advantage here is that this gives an additional level of assurance that any evidence collected is untainted.

Scalability. The problems of scale in digital forensics work can be viewed from two different angles: an increased scale of investigative targets (e.g., corporate) and an increased number of individual investigations. These two scalability problems call for different approaches. The first scenario resembles the classic problems of scale in distributed applications, in which the concurrent work of a large number of machines must be

coordinated. The second one represents human problems of scale—an increase in the number of potential investigative targets that outpaces the increase in the number of people dealing with them. Hence, it is important that technological solutions for digital forensics work become available to a larger fraction of law enforcement officers. This also includes having at least some capabilities to conduct preliminary inquiries in smaller communities that currently have none and are totally reliant on regional centers.

Usability. In continuation of the previous requirement, digital forensics tools must be easy to use by non-experts. For example, it should be possible to allow experts to set up some standard queries to be run on suspect machines by non-experts.

Cost efficiency. This requirement is another fall out from the scalability problem—to put more forensic tools in the hands of law enforcement in the face of slow growth of available resources, the tools must be based on commodity, off-the-shelf technologies. Also, the tools must allow for some of the routine work to be delegated to non-experts, thereby freeing experts to focus on the most critical part of the investigation.

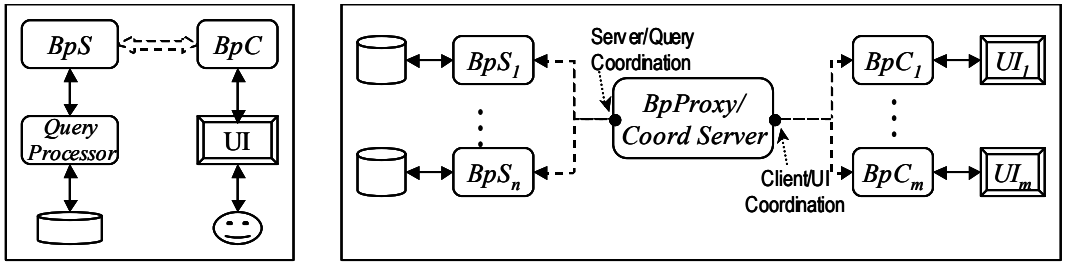
Multi-user capabilities. Since many forensic investigations are the work of a team of experts, software tools should be able to support team interaction. Specifically, for on-the-spot analysis, there are at least two concrete cases that ought to be supported. The first one is the remote help/control scenario. Along the lines of the previous requirements, if a non-expert (or an expert-in-training) is performing an inquiry, it should be possible for a seasoned expert to provide remote help and, if necessary, be able to take complete control of the inquiry. In other words, having such capabilities provides for more efficient use of expertise. The second scenario involves having a team working on a field investigation with a significant number of target systems. Obviously, it should be possible for team members to easily cooperate and coordinate their actions to ensure there are no lapses or duplication of effort. Ideally, it should be possible to seamlessly support (different combinations of) both scenarios so that the physical location of team members becomes essentially irrelevant.

Extensibility. It is virtually a standard software engineering practice to require that the functionality of the system be easily extensible. This is especially important in digital forensics software because new problems emerge on a daily basis so being able to easily extend a tool is crucial for its long-term relevance. Another point here is related to the certification of the software. For example, if one of the certifiable points is that the specific tool leaves the target unchanged, then this must be verified for every new version. While the certifying authority must perform comprehensive testing of the entire system, a modular software architecture with a firewall between different modules reduces the difficulty of internal testing efforts.

The Basic *Bluepipe* Architecture

In this section, the authors discuss the way in which the above requirements have been addressed in the architectural design of their system—*Bluepipe* . The fundamental ar-

chitectural approach is based on the client/server paradigm and is depicted in Figure 1. The *Bluepipe* server (BpS) runs on the target machine while a *Bluepipe* client (BpC) executes on the investigator's machine, establishes a connection to the server, and issues queries. The communication between the two parties is governed by a SOAP-based communication protocol. The overall implementation of the current prototype is outlined below. At this point it is noted that the server software boots from a write-disabled version of Linux on CD-ROM, mounts the hard disk(s) of the target, and performs the requested queries.



a) Single-client/single server b) Multiple-clients/multiple-servers

Figure 1. The Bluepipe architecture.

The primary responsibility of the *Bluepipe* server is to implement the server side of the *Bluepipe* protocol (BpP) and to translate the received queries into invocations of different modules. The primary responsibility of the *Bluepipe* client is to translate into the BpP the queries submitted by the user. Typically, these will come from the user interface (UI) module running on the client machine (Figure 1a). However, a BpC implementation may also become a proxy server by implementing the server interface. This allows for a remote client that does not have a direct connection to the target machine to gain indirect access. By allowing multiple clients to connect to it, a proxy server can also serve as a coordination server among a group of clients by dispatching the submitted queries to different target machines (Figure 1b). Another function of the coordination server is to provide generic collaborative features that facilitate teamwork, such as:

selectable degrees of coupling among the displays of client machine; that is, controlling the degree to which view of different users are allowed to diverge;

enforcement of specific concurrency and access control policies, such as ensuring that no conflicting operations are submitted in parallel and that certain actions are only executed by privileged users.

Figure 2. Current prototype implementation with a single client

Prototype Implementation

The current base implementation of the *Bluepipe* architecture is depicted in Figure 2. It consists of a client-side application, which is used to control the forensics survey, a target-side, self-contained Linux distribution and associated target-side applications, and several additional bits of hardware that enable wireless connectivity with the target machine and provide storage for captured digital evidence. It currently supports both PDA and (laptop) Linux-based clients. While laptop clients clearly have numerous advantages, PDA-based clients may be more suitable in certain scenarios. For one, PDAs are cheaper than laptops and easier to transport. Many PDAs also offer a number of peripheral features, including an integrated digital camera, useful for documenting a crime scene, a voice recorder for case notes, and storage for documents that guide investigation, such as a checklist for seizure of electronic evidence. Wireless PDAs also ease the difficulty in transmitting evidence back to a laboratory.

Bluepipe currently supports both Bluetooth and 802.11 for communication between the client and target machine, each of which has significant advantages and disadvantages. Bluetooth consumes substantially less power and is arguably more secure, particularly because of its reduced range. 802.11 consumes a lot more power and has far more range than is really needed, but provides higher bandwidth. 802.11 also allows a single *Bluepipe* client to handle more target machines simultaneously. Both Bluetooth and 802.11 support encrypted communication, which addresses any concerns with eavesdropping and tampering with transmitted evidence. This general approach does not raise any issues beyond those that are typical in applications relying on wireless communication; hence, there is no need to specifically discuss the wireless security issues further.

To use Bluepipe, an investigator has to perform three simple steps: plug in a USB dongle to enable wireless communication with the target computer, boot the target computer using a Bluepipe boot CD, and launch the Bluepipe application on a PDA or laptop. Law enforcement officers using this system will need to be trained to ensure that the target computer boots from the Bluepipe boot CD. As additional insurance, a bootable floppy which forces a boot from the CD, can also be used to make the procedure

more foolproof. The typical law enforcement officer is not expected to have sufficient training to write Bluepipe patterns; rather, it is expected that a digital forensics expert would prepare them in advance.

Server-side

On the target side, a self-contained Linux distribution on CD is used to bring up the target machine. The Linux distribution provides drivers for the Bluetooth and 802.11 dongles that provide wireless connectivity to the client, as well as for USB keys, which can be used to provide extra storage. The authors use a modified variant of the procedure described in "Root Over NFS Clients and Server HOWTO" [4] to create the bootable Linux CD. There are a number of technical issues that must be addressed to create bootable Linux distributions; most of these are covered in detail in that document. On boot, the Linux distribution executes a Perl script that receives SOAP RPCs from the client, executes them, and returns results to the client. All processing on the target side consists of read-only operations against the secondary storage on the target machine—no data is modified. An audit log tracks all operations performed on the target; this log is transmitted to the client at the end of pattern processing. As the *Bluepipe* architecture is extended, the Perl application on the target side grows; the simple communication protocol understood by the client is sufficient for a wide range of interactions, and reduces changes to the client-side application.

Client-side

The *Bluepipe* client application runs under Palm OS 5 or Linux and is implemented entirely in C. The Palm client application is primarily a graphical file transfer application, allowing selection of patterns, transmission of these patterns to the target-side software, and reception of evidence from the target-side software. Patterns are opaque to the Palm application; it does not directly interpret the content of *Bluepipe* pattern files. Metrowerks Codewarrior 9 under Windows XP is used for development of the Palm application, which is written in C. The development strategy used is to maintain both Linux and Palm versions of the client application. Changes to the client under Linux are tested, then ported to the Palm, since the development cycle is substantially faster under Linux.

Bluepipe Proxy

A *Bluepipe* proxy is a process running on the machine of the investigator who is on-the-spot and acts as an intermediary between a remote client and the server. This allows for a remote investigator to perform a remote investigation and/or to help the on-the-spot examiner. The proxy process appears to the remote client as the server machine and appears to the server as the client machine. The proxy accepts connection from the remote client (over any available TCP/IP connection), wraps RPC calls from the client,

transmits the calls to the server, then sends the returned object from the server RPC back to the client. All the RPC calls are handled by SOAP. The proxy is designed to run in a way that neither the server nor the client need to be rewritten, except that the client needs to switch to the proxy's namespace.

Bluepipe Protocol

The *Bluepipe* protocol provides flexible, robust, and secure connectivity between all *Bluepipe* entities. Communication is based on remote procedure calls (RPC) using the Simple Object Access Protocol (SOAP). SOAP provides a simple and lightweight mechanism for exchanging information between peers and is based on XML [11]. The use of an industry standard low-level protocol on which to build the *Bluepipe* protocol allows different client/server implementations to be quickly developed and adapted for different platforms/scenarios.

Because some *Bluepipe* operations are expected to complete quickly and some require substantial processing time, both synchronous and asynchronous communication is supported by the protocol. The synchronous version follows the standard RPC semantics where the return value is the result of the query and the client-side blocks until it receives the reply. Apparently, for long operations, blocking operations are impractical. Instead, the client submits an asynchronous request where the server records the request and returns immediately.

The request carries an XML-encoded pattern to be executed on the target and a number of additional parameters. These include:

- Update period—how often the server should inform the client on its progress. For long operations users are accustomed to monitoring their progress, hence the need for the server to provide the updates.
- Incremental updates—a flag indicating whether partial results should be submitted along with progress reports: in many situations, a partial result may be all that is needed (e.g., illegal content found).
- Priority—while some operations may take a long time, the investigator should be allowed to perform quick checks that do not require extensive processing. The priority mechanism is an effective means of specifying the importance or length of the request and can also specified by the user.

The initial reply from the server either returns the complete result (synchronous case) or a query identifier to the client. Subsequent updates use the identifier to distinguish among the different tasks being run and may also contain partial results. Finally, there is a trivial cancellation request that allows the client to terminate queries that the user wants to stop.

Bluepipe Patterns

A pattern in *Bluepipe* is an XML document describing a set of operations to be executed against a target machine. All *Bluepipe* operations preserve the state of secondary storage on the target machine. Supported operations include checking for existence of files with specific names or hash values, searching files for keywords, and retrieving files from the target machine. Directory listings and partition table listings (useful for fingerprinting the operating systems installed on the target) can also be retrieved. When the needs of a preliminary forensics survey cannot easily be met by the predefined *Bluepipe* operations, Perl scripts can be embedded in patterns to perform specialized tasks. Patterns are stored on removable media on the *Bluepipe* client (e.g., a Palm handheld device); when the user chooses a particular pattern, it is transmitted to the target and executed. Results of the pattern execution are then transmitted back to the client and stored on removable media. The general format of a *Bluepipe* 0.7 pattern appears below, followed by a brief discussion of each operation.

```
<BLUEPIPE NAME=nameofpattern>
<DIR TARGET=pathname/>
<DIR TARGET=PURGE/>
<FINDFILE USEHASHES=Boolean
        LOCALDIR=pathname
             MSG=string
        RECURSIVE=Boolean
        RETRIEVE=Boolean
             MINSIZE=n
             MAXSIZE=n>
  <FILE ID=identifier
      RETRIEVE=Boolean
      FINDDELETED=Boolean
           MSG=string/>
</FINDFILE>
<GREP SEARCH=string
     FILE=pathname
     MSG=string/>
<LISTPARTITIONS LOCAL=pathname
             GENHASHES=Boolean/>
<LISTDIR TARGET=pathname
             LOCAL=pathname
       LISTDELETED=Boolean
       RECURSIVE=Boolean
       GENHASHES=Boolean/>
<SCRIPT>
  perlcode
</SCRIPT>
</BLUEPIPE>
```

A *Bluepipe* pattern begins with the tag <*BLUEPIPE*>, which defines the name of the pattern. The pattern name is used as the primary filename for the log generated during pattern execution. The DIR operation appends a target directory to the list of directories used by the FINDFILE operation. If PURGE is specified as the TARGET value in the <DIR> tag, then the current list of directories is cleared. FINDFILE is used to check for the existence of one or more files by name or MD5 hash value (defined by the ID value

in each <FILE> tag). The pathnames or MD5 hashes are searched for in each directory specified in the target directory list (built using DIR operations). A number of options control the behavior of the FINDFILE operation:

- USEHASHES determines whether filenames or hashes are used to locate files.
- LOCALDIR specifies a local directory (on the *Bluepipe* client) where retrieved files should be stored.
- If RECURSIVE is true, then target directories are searched recursively.
- If RETRIEVE is true, then matching files are retrieved and stored on the *Bluepipe* client. The value of RETRIEVE in each <FILE> tag overrides the <FINDFILE> RETRIEVE value.
- MINSIZE and MAXSIZE define the file size limits for retrieved files. If specified, no file smaller than MINSIZE or larger than MAXSIZE will be retrieved.

The files to be discovered are specified using a sequence of <FILE> tags. Each defines a filename (or hash value, if USEHASHES is true), specifies whether the file should be retrieved if it is found, and states whether deleted files should be matched. The default behavior for FINDFILE is to check for the existence of matching files and indicate matches in the pattern log; the MSG value can be customized to override the default log file message for matches. Occurrences of "%s" and "%h" in the message are replaced with the filename and hash value, respectively, for matching files. Note that MSG in the <FILE> tag overrides the global MSG in the <FINDFILE> tag.

The GREP operation searches a file for a keyword or regular expression using Unix *grep*. If the file matches, then the value of MSG is written to the pattern log. Because GREP does not use the target directory list created by DIR commands, the value of FILE should be a complete pathname. The authors are currently investigating parsing of Microsoft Windows registries from *Bluepipe*; for now, GREP can be used to perform simple key searches against the registry files. A simple example is examined below.

LISTPARTITIONS is used to request a detailed list of secondary storage devices and associated partition tables on the target machine. If GENHASHES is true, then an MD5 hashes is computed for each raw disk device on the target. The value of LOCAL specifies a pathname for a text file on the *Bluepipe* client for storage of the partition information.

LISTDIR generates a detailed listing of a specified directory, which is stored in a text file on the *Bluepipe* client; the pathname of this text file is defined by the value of LOCAL. If LISTDELETED is true, then deleted files are also included in the directory listing. RECURSIVE governs whether a recursive directory listing is generated, while GENHASHES controls the generation of MD5 hashes for each file in the specified directories. Note that GENHASHES can be extremely time-consuming on deeply nested directories with large numbers of files.

Finally, we support embedded Perl scripts via the SCRIPT operation. Patterns involving scripts will need to be certified to ensure verifiability. Our pattern language is still under development; as new investigatory needs are identified, the language will be expanded.

Sample *Bluepipe* Patterns

In this section a few simple *Bluepipe* patterns are presented to illustrate basic concepts. The following pattern, named "checkmessenger," determines if one or more messenger programs are installed on a target machine running Windows XP by performing keyword searches against the Windows registry:

```
<BLUEPIPE NAME="checkmessenger">
<GREP
  SEARCH="Jabber Messenger"
  FILE="/WINDOWS/system32/config/SOFTWARE"
  MSG="Jabber messenger installed."
/>
<GREP
  SEARCH="msnmessenger"
  FILE="/WINDOWS/system32/config/SOFTWARE"
  MSG="MSN messenger installed."
/>
<GREP
  SEARCH="AOL Instant Messenger"
  FILE="/WINDOWS/system32/config/SOFTWARE"
  MSG="AOL messenger installed."
/>
</BLUEPIPE>
```

Execution of this pattern on a target machine yielded the following log:

```
Beginning execution for pattern "checkmessenger".
GREP cmd, key = "Jabber Messenger", target = "/WINDOWS/system32/config/SOFTWARE".
Jabber messenger installed.
GREP cmd, key = "msnmessenger", target = "/WINDOWS/system32/config/SOFTWARE".
MSN messenger installed.
GREP cmd, key = "AOL Instant Messenger", target = "/WINDOWS/system32/config/SOFTWARE".
AOL messenger installed.
Pattern processing completed.
Sending pattern log.  Remote filename is "checkmessenger.LOG".
```

The following pattern, named "partitions," generates a drive and partition list for the target machine, with MD5 hashes computed for each raw attached drive. The drive list is written to a file "drives.txt" on the client side.

```
<BLUEPIPE NAME="partitions">
<!-- get a lot of drive/partition info-->
<LISTPARTITIONS LOCAL="drives.txt"
      GENHASHES=TRUE/>
</BLUEPIPE>
```

The target machine had a single IDE hard drive, with five partitions and at least two operating systems, illustrated by the contents of the file “drives.txt”:

```
hda
Model Number:       IC25T060ATCS05-0.
Serial Number:      CSL800D8G3GNSA
device size with M = 1024*1024: 57231 Mbytes

Partition table:

Disk /dev/hda: 240 heads, 63 sectors, 7752 cylinders
Units = cylinders of 15120 * 512 bytes

Device Boot    Start       End    Blocks   Id  System
/dev/hda1          1      6173  46667848+   7  HPFS/NTFS
/dev/hda2       7573      7752   1360800   1c  Hidden Win95 FAT32 (LBA)
/dev/hda3   *   6174      7364   9003960   83  Linux
/dev/hda4       7365      7572   1572480    f  Win95 Ext'd (LBA)
/dev/hda5       7365      7572   1572448+  82  Linux swap

MD5 hash for drive: 463e65ec8d9f51bdd17c0347243f467b
```

The following pattern, named “findcacti”, searches for pictures of cacti using a hash dictionary. A single target directory is specified, /pics, which is searched recursively. Files that match are retrieved and stored on the client in a directory named “cactus.” No file size restrictions are imposed. The %s and %h placeholders in the message will be replaced by the filename and hash value of each matching file.

```
<BLUEPIPE NAME=”findcacti”>
<!-- find illegal cacti pics using MD5 hash dictionary -->
<DIR TARGET=”/pics/”  />
<FINDFILE
  USEHASHES=TRUE
  LOCALDIR=”cactus”
  RECURSIVE=TRUE
  RETRIEVE=TRUE
  MSG="Found cactus %s with hash %h ">
<FILE ID=3d1e79d11443498df78a1981652be454/>
<FILE ID=6f5cd6182125fc4b9445aad18f412128/>
<FILE ID=7de79a1ed753ac2980ee2f8e7afa5005/>
<FILE ID=ab348734f7347a8a054aa2c774f7aae6/>
<FILE ID=b57af575deef030baa709f5bf32ac1ed/>
<FILE ID=7074c76fada0b4b419287ee28d705787/>
<FILE ID=9de757840cc33d807307e1278f901d3a/>
<FILE ID=b12fcf4144dc88cdb2927e91617842b0/>
<FILE ID=e7183e5eec7d186f7b5d0ce38e7eaaad/>
<FILE ID=808bac4a404911bf2facaa911651e051/>
<FILE ID=fffbf594bbae2b3dd6af84e1af4be79c/>
<FILE ID=b9776d04e384a10aef6d1c8258fdf054/>
</FINDFILE>
</BLUEPIPE>
```

The log file generated by executing the “findcacti” pattern on one target machine follows. Notice that the DSC00051 and bcactus5 image files have identical content.

```
Beginning execution for pattern "findcacti".
DIR cmd, added "/pics".
FINDFILE cmd.
Found cactus /pics/BBQ-5-27-2001/DSC00008A.JPG with hash
6f5cd6182125fc4b9445aad18f412128
Found cactus /pics/BBQ-5-27-2001/DSC00009A.JPG with hash
7de79a1ed753ac2980ee2f8e7afa5005.
Found cactus /pics/CACTUS_ANNA/DSC00051.JPG with hash
3d1e79d11443498df78a1981652be454.
Found cactus /pics/GARDEN2002/bcactus5.JPG with hash
3d1e79d11443498df78a1981652be454.
Pattern processing completed.
Sending pattern log.  Remote filename is "findcacti.LOG".
```

Evaluation

To complete the discussion, the authors' efforts are compared with those of several leading systems using the set of design requirements. The related systems fall into two broad categories based on their fundamental approach toward the forensic analysis. As the following discussion will show, the *Bluepipe* approach presents a distinct category that complements the other two and adds some unique capabilities to address the problems of on-the-spot investigation that are not present in other systems.

Related Work

A large number of high-quality commercial and open-source tools exist for performing computer forensics investigations. While their specific capabilities vary, most are based on the image-then-analyze paradigm, where an exact image of the target is first obtained and all subsequent work is done on that image. We refer to such tools as *direct access* tools because they provide self-contained environments to directly examine the digital evidence source.

Direct Access (Da) Tools

Encase [2] is a powerful package for forensics which runs under Microsoft Windows. Imaging can be performed using a boot floppy on the target or by removing drives from the target and attaching them to a forensics workstation. A hardware write-blocker such as FastBloc [5] can be used to minimize the danger of destroying digital evidence. An interesting feature is the ability to perform the imaging and preview of the evidence over a network cable. Once images are available, forensics operations such as keyword searches and file/partition recovery can be performed. iLook [6] has a similar set of capabilities and is free to law enforcement agents. The Forensics Toolkit (FTK) [3] by AccessData takes a different approach and is built on a database model. An extensive analysis of drive images is performed immediately after the imaging process, which can be very time-consuming, but subsequent operations, such as keyword searches, determining which deleted files can be recovered, how many graphics files are present, etc. are virtually instantaneous. SMART [7] is a Linux-based counterpart to Encase, FTK,

and iLook. Open source tools such as The Sleuthkit and Autopsy [1] have similar capabilities.

In summary, DA tools are geared primarily toward facilitating the process of evidence collection in a lab setting by an experienced investigator. Their operation is largely batch-oriented with the investigator sequentially submitting queries and waiting for replies.

Mediated Remote Access (MRA) Tools

MRA tools are designed to leverage the existing network infrastructure to provide remote access and/or monitoring of computer systems while the systems are "live" and performing their normal tasks. Existing systems of this class are referred to as utilizing *mediated* remote access, because they rely on standard operating system (OS) services and existing communication infrastructure (perhaps with some additions or modifications) to perform their work.

The Mobile Forensics Platform (MFP) [8] allows forensics experts to perform investigations of live machines from a remote location. A remote investigator can audit logs, capture network traffic and other digital evidence, and determine what further steps should be taken, all without hands-on contact with the machines under investigation. The Forensics Server Project (FSP) [10] is related to MFP in that it allows collection of data from a live machine by automating data collection, file copying, and hashing operations.

Command-line tools, such as *netcat* [9] or *ssh,* are also often used for remote access to machines under investigation. Drive imaging and other operations can be performed remotely using these tools, but require significant expertise on the part of the examiner.

In summary, MRA tools allow remote examination of live systems that already have an appropriate communication infrastructure in place. In some cases, such as MFP and FSP, additional pieces of the infrastructure must be preinstalled on site. Unlike DA tools, MRA tools rely on existing OS services, such as an authentication server, to perform their work. However, they also allow for additional information, such as real-time CPU/network/file system activity, to be captured that is unavailable to DA tools.

Comparative Evaluation

In the context of the systems described in the previous section, the *Bluepipe* system is representative of a class of system the authors would call *autonomous remote access* (ARA) tools. Such tools, like MRA tools, allow remote examination of the target system; however, unlike MRA systems, they provide their own autonomous communication infrastructure. Like DA tools, ARA systems have direct access to the target, but, their primary purpose is preliminary screening for evidence, rather than an exhaustive examination.

A requirement-by-requirement evaluation of the three classes of systems discussed in this paper is presented here, comparing their advantages and shortcomings.

Verifiability. DA systems ensure verifiability in a straightforward manner by working on a separate copy of the target media. The copying process itself can easily be certified by a third party (e.g., NIST) and the entire evidence collection process can easily be explained to a judge/juror with no technical background. *Bluepipe* provides a similar level of assurance, although the technical means are slightly different. *Bluepipe* is based on a read-only distribution of Linux (which mounts user data read-only as well), which trivially guarantees that the original is untainted. Third-party certification should not be an issue, as the read-only property is straightforward to verify and should be easy to explain to non-experts. Arguably, the behavior of MRA tools is inherently the hardest to certify, for the simple reason that they work in an environment of multiple active processes and may rely on some of them for their own correct operation. Hence, to demonstrate verifiability, the investigator may have to show that the underlying OS services have not been compromised. From a theoretical point of view, this would be a difficult task and, even if possible, it appears unlikely that the logical reasoning behind it could be satisfactorily explained to non-experts. As already pointed out, some evidence can only be collected at run time, so the credibility of such evidence would likely rest with convincing the court that the tools follow a strict procedure and produce predictable results rather than a formal proof of correctness.

Portability. Neither DA nor MRA tools have been expressly designed for portability. DA tools assume a lab setting, whereas MRA ones require an advance setup. In contrast, *Bluepipe* treats portability as a central requirement and its prototype implementation clearly demonstrates that point. The server can examine numerous file systems, the client can run on both PDA and laptops, and the system can utilize either *Bluetooth* or 802.11 wireless networking. Taken together, these features allow an investigator to start examining a target machine within a few minutes.

Usability. The main conceptual difference between this work and other systems is that *Bluepipe* is intended to be easy for non-experts to use, by providing a point-and-click interface for connecting to target machines, executing patterns, and gathering evidence. In contrast, most existing systems aim to provide as much lower-level details to the user as possible. While this is ultimately needed in an investigation, it limits the applicability of the tool in the everyday practice of regular law enforcement officers.

Cost efficiency. In the prototype implementation, the authors have demonstrated that the architecture can be supported on commodity, off-the-shelf components and is, therefore, very affordable from an economic standpoint. The cost of the other systems varies significantly and ranges from freely available tools to relatively costly hardware/software commercial solutions targeted at the corporate market.

Multi-user capabilities. To the best of the authors' knowledge, this architecture is the only one that is explicitly designed to accommodate the needs of real-time collaboration

for on-the-spot investigation. Other systems provide some limited support for asynchronous collaboration by placing all results in a database, which in turn could potentially by accessed by multiple users. Currently, *Bluepipe* allows remote access to be gained via the proxy and thereby, queries to be executed both by the on-the-spot investigator and a remote expert. Ongoing efforts include the design and implementation of a collaborative protocol that will considerably expand the capabilities with real-time support for collaborative editing and viewing of queries.

Scalability. Clearly DA tools do not satisfy the scalability requirement as they are designed exclusively for use by a single investigator on a single target. RMA systems address scalability in that they allow entire networks to be monitored in real time, although it does appear that the current generation of RMA tools supports the concurrent examination of multiple targets. *Bluepipe*'s approach to human scalability is to make it possible and affordable for the tool to be used by non-experts to perform routine inquires without the need for a step-by-step intervention of forensics experts in routine queries. Our design provides for a proxy/coordination server that allows multiple queries to be synchronized and executed in parallel on multiple targets, thereby ensuring system scalability. The prototype implementation of these features is one of the subjects of our ongoing efforts.

Extensibility. It is somewhat difficult to judge the level of extensibility for commercial products, as the internal details of their design are proprietary. In contrast, this work includes developing a system around a set of protocols that allows maximum flexibility and extensibility with respect to the different components that can be plugged in and out.

Conclusions and Future Work

The traditional digital investigation process involves a few serious problems that can be alleviated by a preliminary, on-the-spot phase of investigation. These problems include cooperation issues, where users might be reluctant to allow imaging or seizure of their systems, but would be willing to allow a less invasive "look around," and practical issues, covering seizure and proper care of large numbers of computer systems. This paper discussed requirements for on-the-spot investigations, presented the Bluepipe architecture, which satisfies these requirements, and discussed the prototype implementation. The paper also compared Bluepipe with the capabilities of existing systems and showed that this approach is better suited for the specific needs of on-the-spot forensics.

In the context of the original scenarios, DA forensic tools are only adequate for the screening of evidence in the SOHO scenario and clearly do not measure up to the needs of bigger investigations. At the same time, MRA tools are likely to be used only in a corporate environment where the relatively high purchase/maintenance costs are outweighed by the urgency of discovering and recovering from incidents such as security breaches as quickly as possible.

The inherent limitations of *Bluepipe*'s approach stem from the simple fact that a person must walk up to each physical machine and boot it with the appropriate software. In an investigation on a massive scale, this restriction may become non-trivial to satisfy and the ad-hoc communication infrastructure could easily become a bottleneck. However, this approach is viable for the other three scenarios and offers the highest degree of verifiability.

Bluepipe is a work-in-progress. The authors are working with members of the Gulf Coast Computer Forensics Laboratory (GCCFL) to test and refine the software and produce a practical and useful piece of software. Many of the features of the design have not yet been incorporated into the implementation and their collaboration with the GCCFL will further help them identify and address the actual needs of investigators.

Acknowledgements

The anonymous referees provided useful feedback that significantly improved the quality of the paper, particularly the introductory material. We would also like to acknowledge feedback received at the 2003 Digital Forensics Research Workshop (DFRWS), which helped shape the current version of our architecture.

About the Authors

Golden G. Richard III is an Associate Professor of Computer Science at the University of New Orleans in Louisiana, where he has been a faculty member since 1994. Dr. Richard has Ph.D. and M.S. degrees in computer science from The Ohio State University and a B.S. in computer science from the University of New Orleans. His broad research interests include digital forensics, computer security, mobile computing, and operating systems. When he is not hacking, he can be found consuming jazz, cooking, or covered in dirt in his garden. Contact: golden@cs.uno.edu.

Vassil Roussev is an Assistant Professor of Computer Science at the University of New Orleans, where he has been on the faculty since 2002. Dr. Roussev has M.S. and Ph.D. degrees in computer science from the University of North Carolina – Chapel Hill as well as B.S. and M.S. degrees in computer science from Sofia University (Bulgaria). His research interests include digital forensics, distributed multi-user systems, mobile devices, and software engineering—pattern-based techniques, component- and service-based models, agile development methods. Contact: vassil@cs.uno.edu.

Yun Gao currently is a Ph.D. candidate in computer science at the Universiy of New Orleans, under the guidance of Dr. Golden G. Richard III. She received her M.S. degree in Computer Science from University of New Orleans in 2003 and holds a B.S. degree in Civil Engineering from Beijing Institute of Architecture and Engineering. Yun Gao's re-

search interests include computer security and forensics, wireless networking and internet protocols. Contact: ygao@cs.uno.edu.

References

[1] Sleuthkit and Autopsy, http://www.sleuthkit.org.

[2] Encase forensics software, http://www.encase.com.

[3] Forensics Toolkit (FTK), http://www.accessdata.com.

[4] "Root Over NFS Clients and Server HOWTO," http://www.linux.se/doc/HOWTO/Diskless-root-NFS-HOWTO.html

[5] FastBloc write blocker, http://www.guidancesoftware.com/products/hardware/fastbloc/index.shtm.

[6] iLook Investigator forensics software, http://www.ilook-forensics.org/.

[7] SMART forensics software, http://www.asrdata.com/SMART/.

[8] F. Adelstein, "MFP: The Mobile Forensics Platform," Proceedings of the 2002 Digital Forensics Research Workshop, http://www.dfrws.org.

[9] GNU Netcat, http://netcat.sourceforge.net.

[10] The Forensics Server Project, http://patriot.net/~carvdawg/fsproj.html.

[11] Simple Object Access Protocol (V1.1), http://www.w3.org/TR/SOAP/.