Analysis of a Compromised System
Part 1: Online Analysis and Data Collection

License

Contents

Preparation

Configuring the environment

Note IP addresses

Reminder to save

Introduction

Live analysis

Collecting live state manually

Comparing process lists

Gathering live state using statically compiled programs

Collecting live state using scripts

Checking for rootkits

Offline analysis of live data collection

Windows and live incident response

What’s next …

What to show your tutor for XP rewards

Preparation

These tasks can be completed on the LinuxZ IMS image. To start, enable the network drive (via the desktop icon, enter the root password when prompted with “root password:”, then hit enter when prompted with “Password:”). Download the “RedHat 7.2 - compromised” VM using the download script. When prompted whether you want to run the VM select “No”.

This lab could be completed on most Linux systems with an SSH server running and VMware player installed, using an .iso file or disk of FIRE 0.3.5b, and the compromised VM (courtesy of the Honeynet Project).

Configuring the environment

Start VMware player. Click “Open a Virtual Machine”. Browse to and open the compromised VM (on LinuxZ, /home/*user*/VMs/Downloaded/RedHat 7.2 - compromised/RedHat 7.2 - compromised.vmx). Edit the settings of the Compromised Red Hat VM (click “Edit virtual machine settings”), and add a Linux Forensic investigation ISO as a CD/DVD device (click “CD/DVD”, “Use an ISO image”, “Browse”). There are a number of Forensic investigation ISOs, such as FIRE or Helix, on the shared network drive (mounted on LinuxZ as /mnt/NetworkDrive/ISOs/). Feel free to experiment with any of these, but for the purpose of this lab select “fire-0.3.5b.iso”. Confirm the VM networking is set to “Host Only”.

Start the compromised VM. Log in as root, with the password “toor”.

Note IP addresses

Run “`dhcpcd`” on the VM to renew its IP address.

On your host (LinuxZ) system run `/sbin/ifconfig`, and also run `ifconfig` on the guest compromised VM. Make a note of the two IP addresses, which should be on the same subnet (starting the same): for example, if both systems include IP addresses starting with 192.168.202, then use those (they may start with 172.16…). You will need these IP addresses later.

Reminder to save

Remember to **save any evidence you collect** (at the end of the lab via USB or save an IMS image), as this will be used as the basis for the next lab. You cannot complete the following lab (part 2) without saving the evidence you collect during this lab.

Introduction

So you have reason to believe one of your servers has experienced a security compromise… What next? For this lab you investigate a real Linux server that was connected to the Internet, and attacked and compromised by unknown remote attackers.

The investigation of a potential security compromise is closely related to digital forensics topics. As with forensic investigations, we also aim to maintain the integrity of our "evidence", *wherever possible* not modifying access times or other information. However, in a business incident response setting maintaining a chain of evidence may not be our highest priority, since we may be more concerned with other business objectives, such as assuring the confidentiality, integrity, and availability of data and services.

During analysis, it is good practice to follow the order of volatility (OOV): collecting the most volatile evidence first (such as the contents of RAM, details of processes running, and so on) from a live system, then collecting less volatile evidence (such as data stored on disk) using offline analysis.

Live analysis

After suspecting a compromise, before powering down the server for offline analysis, the first step is typically to perform some initial investigation of the live system. Live analysis aims to investigate suspicions that a compromise has occurred, and gather volatile information, which may potentially include evidence that would be lost by powering down the computer.

On the compromised VM: To keep a record of what we are doing on the system, start the script command:

```
mkdir /tmp/evid
```

```
script -f /tmp/evid/invst.log
```

Note, you will show this record of your activities to your tutor.

> Note: *if you do this lab over multiple sessions*, be sure to save the log of your progress (/tmp/evid/invst.log), and restart `script`.

Make a note of the risks and benefits associated with storing a record of what we are doing locally on the computer that we are investigating:

Consider the advantages of *handwritten* documentation of what investigators are doing.

Many of the commands used to investigate what is happening on a system are standard Unix commands. However, it is advisable to run these from a read-only source, since software on your system may have been tampered with. Also, using read-only media minimises the changes made to your local filesystem, such as executable file access times.

During preparation, you configured the compromised VM to have access to the FIRE (Forensic and Incident Response Environment) CD/DVD ISO (which is equivalent to inserting the optical disk into your server's DVD-tray). FIRE is an example of a Linux Live Disk that includes tools for forensic investigation. In addition to being able to boot to a version of Linux for offline investigation of evidence, the disk contains Linux tools for live analysis.

On the compromised VM, mount the disk, so that we can access its contents:

```
mount /mnt/cdrom/
```

On a typical system, many binary executables are dynamically linked; that is, these programs do not physically contain all the libraries (shared code) they use, and instead load that code from shared library files when the program runs. On Unix systems the shared code is typically contained in ".so" files, while on Windows ".dll" files contain shared code. The risks associated with using dynamically linked executables to investigate security breaches is that access times on the shared objects will be updated, and the shared code may also have been tampered with. For this reason it is safest to use programs that are statically linked; that is, have been compiled to physically contain a copy of all of the shared code that it uses.

On your host OS (your own Linux system; for example, the LinuxZ host), look at which libraries are dynamically loaded when you run a typical command:

```
ldd /bin/ls
```

Examine the output, and determine how many external libraries are involved.

On the compromised VM: The FIRE disk contains a number of statically compiled programs to be used for investigations. Check that these are indeed statically linked:

```
ldd /mnt/cdrom/statbins/linux2.2_x86/ls
```

Compare the output to the previous command run on your own system. The output will be distinctly different, stating that the program is not dynamically compiled.

Note that, although an improvement, using statically linked programs such as these still do not guarantee that you can trust the output of the programs you run. Consider why, and make a note of this:

Collecting live state manually

The next step is to use tools to capture information about the live system, for later analysis. One approach to storing the resulting evidence is to send results over the network via Netcat or SSH, without storing them locally. This has the advantage of not changing local files, and is less likely to tip off an attacker, rather than storing the evidence on the compromised machine.

Comparing process lists

On the compromised VM, test sending the results of some commands (process lists using ps) over SSH to your host (LinuxZ) system:

```
ssh student@host-IP-address "mkdir evidence"

ps aux | ssh student@host-IP-address "cat > evidence/
ps_out"
```

(Where *host-IP-address* is the IP address of your *host (LinuxZ) system*, which should be on the same subnet as your compromised VM)

Note: using a UK keyboard, the @ and " symbols may be reversed, and the | symbol is located at the ~.

Each time you are **prompted for a password**, if you are on the LinuxZ image, type “**theIliad**” (otherwise use a different username/password pair, available on your own system).

Why might it not be a good idea to type your own account password from the VM system?

On your host OS, find the newly created files and view the contents.

Hint: you may wish to use the Dolphin graphical file browser, then navigate to "/home/student/evidence".

On the compromised VM, run the statically compiled version of ls from the incident response disk to list the contents of /proc (this is provided dynamically by the kernel: a directory exists for every process on the system), and once again send the results to your host...

First, to save yourself from having to type "/mnt/cdrom/statbins/linux2.2_x86/" over and over, save that value in a Bash variable:

```
export static="/mnt/cdrom/statbins/linux2.2_x86/"
```

Now, to run the statically compiled version of ls, you can run:

```
$static/ls
```

Run the command:

```
$static/ls /proc | ssh student@host-IP-address "cat >
evidence/proc_ls_static"
```

On your host OS, find the newly created files and compare the list of pids (numbers representing processes) output from the previous commands. This is the second column of output in the ps_out, with the numbers in proc_ls_static.

Hint: you can do the comparison manually, or using commands such as "cut", "sort", and "diff". For example, "cat ps_out | cut -d ' ' -f 4" will pipe the contents of the file ps_out into the cut command, which will split on spaces (-d ' '), and only display the fourth field. Ensure this is displaying the list of pids, if not try selecting a different field. You could pipe this through to "sort". Then save that to a file (by appending " > pids_ps_out"). We have covered how to use diff previously. Remember "man cut", "man sort", and "man diff" will tell you about how to use the commands.

Are the same processes shown each time? If not, that is very suspicious, and likely indicates a break-in, and that we probably shouldn't trust the output of local commands.

Take a screenshot of a list of processes (pids) running on the system that are not reported by the local version of ps, and preferably show how you came to that list, as evidence that you have completed this part of the task.

Label it or save it as “LiveIR-1”.

Gathering live state using statically compiled programs

On the compromised VM, save a copy of a list of processes accessing files that have since been deleted:

```
$static/ils -o /dev/sda1 | ssh student@host-IP-address "cat > evidence/deleted_out"
```

On your host OS, open evidence/deleted_out. Any processes listed are suspicious, since it is atypical for programs to be accessing files that have been deleted. Do any of these processes correspond to those missing from the local “ps” output?

On the compromised VM, save a list of the files currently being accessed by programs:

```
$static/lsof | ssh student@host-IP-address "cat > evidence/lsof_out"
```

Save a list of network connections:

```
$static/netstat -a | ssh student@host-IP-address "cat > evidence/netstat_out"
```

(Some commands such as this one may take awhile to run, wait until the Bash prompt returns)

Save a list of the network resources currently being accessed by programs:

```
$static/lsof -P -i -n | ssh student@host-IP-address "cat > evidence/lsof_net_out"
```

Save a copy of the routing table:

```
$static/route | ssh student@host-IP-address "cat > evidence/route_out"
```

Save a copy of the ARP cache:

```
$static/arp -a | ssh student@host-IP-address "cat >
evidence/arp_out"
```

Save a list of the kernel modules currently loaded (as reported by the kernel):

```
$static/cat /proc/modules | ssh student@host-IP-
address "cat > evidence/lsmod_out"
```

Creating images of the system state

We can take a snapshot of the live state of the computer by dumping the entire contents of memory (what is in RAM/swap) into a file. On a Linux system `/proc/kcore` contains an ELF-formatted core dump of the kernel. Save a snapshot of the kernel state:

```
$static/dd if=/proc/kcore conv=noerror,sync | ssh
student@host-IP-address "cat > evidence/kcore"
```

Next, we can copy entire partitions to our other system, to preserve the exact state of stored data, and so that we can conduct offline analysis without modifying the filesystem.

Start by identifying the device files for the partitions:

```
df
```

Note that on this system the root partition (mounted on "/"), is /dev/sda1.

Then, copy byte-for-byte the contents of the root ("/") partition (where /dev/sda1 was identified from the previous command:

```
$static/dd if=/dev/sda1 conv=noerror,sync | ssh
student@host-IP-address "cat > evidence/sda1.img"
```

Tip: Running this will take some time, so you may wish to continue with the next step while the copying runs.

This command could be repeated for each partition including swap partitions. For now, lets accept that we have all we need.

On your host OS, list all the files you have created:

```
ls -la /home/student/evidence
```

Take a screenshot of the list of evidence files you have collected (using ls -la), as evidence that you have completed this part of the task.

Label it or save it as "LiveIR-2".

At this stage look through some of the information you have collected. For example:

```
less /home/student/evidence/lsof_net_out
```

Examine the contents of the various output files and identify anything that may indicate that the computer has been compromised by an attacker. Hint: does the network usage seem suspicious?

Take a screenshot of some suspicious evidence you have collected and include one sentence describing why it is suspicious, as evidence that you have completed this part of the task.

Label it or save it as "LiveIR-3".

Collecting live state using scripts

As you may have concluded from the previous tasks, manually collecting all this information from a live system can be a fairly time consuming process. Incident response data collection scripts can automate much of this process. A common data collection script “linux-ir.sh”, is included on the FIRE disk, and is also found on the popular Helix IR disk.

On the compromised VM, have a look through the script:

```
less /mnt/cdrom/statbins/linux-ir.sh
```

Note that this is a Bash script, and each line contains commands that you could type into the Bash shell. Bash provides the command prompt on most Unix systems, and a Bash script is an automated way of running commands. This script is quite simple, with a series of commands (similar to some of those you have already run) to display information about the running system.

Identify some commands within the script that collect information you have not already collected above.

Exit viewing the script (press q).

Run the data collection script, redirecting output to your host:

```
cd /mnt/cdrom/statbins/

./linux-ir.sh | ssh student@host-IP-address "cat >
evidence/ir_out"
```

On your host OS, have a look at the output from the script:

```
less /home/student/evidence/ir_out
```

Use what you have learnt to spot some evidence of a security compromise.

Identify something within ir_out that may indicate the system has been compromised, record one or two line(s) from the output and write a sentence describing why it is suspicious.

Label it or save it as “LiveIR-4”.

Checking for rootkits

An important concern when investigating an incident, is that the system (including user-space programs, libraries, and possibly even the OS kernel) may have been modified to hide the presence of changes made by an attacker. For example, the ps and ls commands may be modified, so that certain processes and files (respectively) are not displayed. The libraries used by various commands may have been modified, so that any programs using those libraries are provided with deceptive information. If the kernel has been modified, it can essentially change the behaviour of *any* program on the system, by changing the kernel's response to instructions from processes. For example, if a program attempts to *open* a file for viewing, the kernel could provide one set of content, while an attempt to *execute* the file may result in a completely different program running.

Detecting the presence of rootkits is tricky, and prone to error. However, there are a number of techniques that, while not foolproof, can detect a number of rootkits. Methods of detection include: looking for inconsistencies between different ways of gathering data about the system state, and looking for known instances of malicious files.

Chkrootkit is a Bash script that performs a number of tests for the presence of various rootkits.

On the compromised VM, have a quick look through the script, it is much more complex than the previous linux-ir.sh script:

```
less /mnt/cdrom/statbins/chkrootkit-linux/chkrootkit
```

Exit less

Confirm that if we were to run ls, we would be running the local (dynamic) version, probably /bin/ls:

```
which ls
```

To understand why, look at the value of the environment variable $PATH, which tells Bash where to look for programs:

```
echo $PATH
```

Set the $PATH environment variable to use our static binaries wherever possible, so that when chkrootkit calls external programs it will (wherever possible) use the ones stored on the IR disk:

```
export PATH=$static:$PATH
```

Confirm that now if we were to run less, we would be running the static version:

```
which ls
```

This should report the path to our static binary on the FIRE disk.

It is now safe to run chkrootkit:

```
./chkrootkit-linux/chkrootkit | ssh student@host-IP-address "cat > evidence/chkrootkit_out"
```

On your host OS, have a look at the output:

```
less /home/student/evidence/chkrootkit_out
```

From the output, identify files or directories reported as “INFECTED”, or suspicious.

Also, note that the .bash_history is reportedly linked to another file.

Copy to a new file (and take a screenshot of) the lines from chkrootkit_out that identifying potential security issues/compromise, as evidence that you have completed this part of the task.

Label it or save it as “LiveIR-5”.

On the compromised VM, investigate the Bash history:

```
$static/ls -la /root/.bash_history
```

What does the output mean? What does this mean for the logging of the commands run by root?

At this stage you should be convinced that this system is definitely compromised, and infected with some form of rootkit.

Save a record of your activity to your host OS:

```
cat /tmp/evid/invst.log | ssh student@host-IP-address "cat
> evidence/script_log"
```

Save a copy of your activity log (/home/student/evidence/script_log), as evidence that you have completed all of the above tasks.

Label it or save it as "LiveIR-6".

Power down the system, so that we can continue analysis offline:

```
$static/sync; $static/sync
```

If you do not know what the sync command does, on your host OS, run "`info coreutils 'sync invocation'`" for more information.

Tell VMware to force a Power Off. ("Virtual Machine", "Power", "Power off", "Yes").

Why might we want to force a power off (effectively "pulling the plug"), rather than going through the normal shutdown process (by running "halt" or "shutdown")?

Offline analysis of live data collection

Note that even though the bash_history was not saved (as we discovered above), we can still recover commands that were run the last time the computer was running. This is possible by searching through the saved RAM (the kcore ELF dump we saved earlier).

On your host OS, run:

```
sudo -u student bash -c "strings -n 10 /home/student/
evidence/kcore > /home/student/evidence/kcore_strings"
```

The above "strings" command extracts ASCII text from the binary core dump.

Open the extracted strings, and look for evidence of the commands you ran before saving the kernel core dump:

```
less /home/student/evidence/kcore_strings
```

Now press the '/' key, and type a regex to search for commands you previously ran to collect information about the system. For example, try searching for "ssh student".

Windows and live incident response

The theory you have applied in this lab applies similarly to other operating systems such as Windows. Many incident response live disks (such as FIRE and Helix) also include Windows programs that can collect information about the state of a system.

Unguided task: use an incident response disk (such as Helix) to obtain information from a running Windows computer. You should collect a dump of the contents of RAM, and a list of running processes and network connections.

Take screenshots on Windows of information about processes and network connections being captured and displayed using an IR disk, as evidence that you have completed this part of the task.

Label it or save it as "LiveIR-A1".

What's next ...

In the next lab you will analyse the artifacts we have collected, to determine what has happened on the system.

Important: save the evidence you have collected (save /home/student/evidence to USB, or save an IMS image), as this will be used as the basis for the next lab.

What to show your tutor for XP rewards

Show your tutor each of the above (in red) evidences. You may be asked to justify your decisions. This will be used to allocate XP for the module. Further details of the XP rewards and requirements are available on the *My XP* site.