Rescuing data from attackers

Data Rescue

When attackers strike your system, you need to determine exactly what damage has been done. Here are some tools to help. By Kurt Seifried

Typically, when the term "data rescue" is mentioned, failed RAID arrays, accidentally deleted files, and corrupted backups come to mind. But, what happens when a break-in occurs and you need to find out how the attacker got in and how much damage has been done?

If you're lucky (relatively speaking), the attacker will make changes to the filesystem. For example, in the recent kernel.org security breach, the OpenSSH binaries were replaced with ones that would log usernames, passwords, and keys, allowing the attacker to access additional systems. In theory, tools like AIDE or Tripwire should catch these modifications, but in practice this doesn't always work.

For more than a decade now, attackers have been able to modify running processes and programs in memory – for example, patching the OpenSSH server to log all incoming and outgoing data or patching the OpenSSH client to log passwords. Attackers utilizing these methods are very hard to detect because they might leave little or no trace on the filesystem.

A server reboot or even a restarted service can remove the attacker's tools, but, unless the underlying security flaw that was used to break in is also patched, the attacker will be able to regain access easily in most cases. So, how can you go about examining a system for potential attacker code running in memory or to find out what an attacker has done if you know a break-in has occurred?

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

Dumping System Memory to a File

The first step is usually to make a snapshot of running memory so you can more easily examine it. If you are running virtualized servers, tools like KVM or VMware allow you to pause or suspend virtual machines. When this is done, a copy of memory is written to a file, so the system can be restored properly later. If you are using virtual systems, or if you want to make a memory image from within the system, your best bet is to use the dd tool to dump the contents of memory into a file using a command such as:

```
dd if=/dev/mem ⮒
  of=/root/system-memory.dump
```

This approach probably won't work on modern systems, however, because of a new kernel capability called STRICT_DEVMEM, which prevents user space access to all of the memory. To get around this restriction, you'll need either to recompile and install a new kernel with this capability turned off (which will require a reboot, removing any evidence in memory) or use the fmem tool.

fmem

The fmem tool [1] is a small kernel module that when loaded will cause a new device called /dev/fmem to be added, which allows full access to system memory. You should compile a copy of this kernel module for your systems and keep it handy because your production system might not have development tools or be very trustworthy if it's been broken into. Once the device is installed, you can use

```
dd if=/dev/fmem ⮒
  of=/root/system-memory.dump
```

to dump the system memory.

kdump

Another way to get a system dump is to use the kdump utility, which is in the kexec-tools package. When it is installed and running, it will take some system memory. However, it lets you quickly and painlessly create dumps of the system memory on local or remote filesystems (using NFS or SCP). For really paranoid environments, you might want to consider dumping system memory once in a while, then if you detect a break-in, you will have dumps to use for comparison, and you also might be able to narrow down the attack time frame.

Swap Space

Note that all things living in memory might not actually be in memory. Linux aggressively caches unused data to swap space, and data on heavily loaded systems often is moved out of memory and into swap space (although this is less of a problem now with cheap RAM). The thing to remember is that data might still be present in the swap space even if the program using it was shut down, so attackers' data might remain even if they tried to clean up their tracks. Dumping swap space is easy: Just use the dd tool to dump your swap partition to a file.

Examining the Contents of Memory

If you simply open the memory dump in a hex or other binary editor, it probably won't be all that useful. The contents of memory will include processes, files, and all sorts of structures and kernel data, and sorting this out by hand would be tedious to say the least.

Volatilitux

Volatilitux [2] supports ARM, x86, and x86 with PAE (Physical Address Extension) and can be used to examine system memory or a dump file. The benefit of Volatilitux is that it can print out a list of all running processes, a memory map of a specific process, the dump of the process, a list of all open files, and the contents of any open files in memory. Using this, you could, for example, do side-by-side comparisons with known good copies of the same binary running BinDiff on the same OS. The closer the binaries are in terms of architecture, library, and software versions, the more likely it is that you'll be able to find meaningful differences between the two. Again, preparation such as having a copy of your production system for comparison can save a lot of time when an incident occurs.

Crash

Crash [3] is the kind of tool you've probably never heard about, but it's the exact thing you need to scratch many itches. Crash can be used to examine a running system or to dump files created by tools like `kdump` or `netdump`. It actually ships as a standard package in Red Hat Linux (it was written by David Anderson of Red Hat) and is available on Debian and other distributions as well. Note that most distributions (including Red Hat) ship version 5.x, whereas the most current version is 6.0.0. Compilation and installation of the latest version is simple: Compile it with `make` and install it with `make install`.

Other Tools and Projects

Of course, other tools and projects are available. The Linux Memory Analysis [4] website lists many tools and also links to a number of papers and some challenges, where you can test your skills or simply read about how analysis is done in these cases.

Hiding in /dev/shm

I've covered system memory and swap space, but there are other places an attacker can hide data on a running system. One often ignored area that is useful to an attacker is the shared memory system.

The shared memory subsystem is designed to allow efficient interprocess memory sharing, and it's also available to attackers. Somewhat annoyingly, if you dump the contents of `/dev/mem` (well, `/dev/fmem`), you will not get the contents of `/dev/shm`.

To actually get the contents of `/dev/shm`, you'll need to dump each "file" within it:

```
for i in /dev/shm/*dodd if=$i ↩
  of=/root/dump/$idone
```

For an in-depth article on interacting with `/dev/shm` and the memory structures used, refer to the IBM developerWorks website [5].

Conclusion

For high-value targets, a savvy attacker likely will try to avoid touching the filesystem if possible. Luckily, you can take steps in advance to make incident response much easier.

Depending on your needs and resources, my minimal recommendation is to make a working `fmem` kernel module for all your systems and to have disk space available to hold the memory dumps. If you have to dump a lot of systems, you could easily end up with a few terabytes worth of files, especially if you also collect copies of all the swap space.

Also, having known good copies of what memory should look like will make comparing potentially compromised systems easier. ■■■

INFO

[1] fmem: *http://hysteria.sk/~niekt0/foriana/*

[2] Volatilitux: *http://code.google.com/p/volatilitux/*

[3] Crash: *http://people.redhat.com/anderson/*

[4] Linux Memory Analysis: *http://www.forensicswiki.org/wiki/Linux_Memory_Analysis*

[5] Build a Python app for parsing shared memory dumps: *http://www.ibm.com/developerworks/linux/library/l-parse-memory-dumps/*