SAM STOVER AND MATT DICKERSON

using memory dumps in digital forensics

Stover is an independent security researcher with experience in network- and host-based forensics.

sam.stover@gmail.com

Matt Dickerson works as a network security engineer for LMIT. He tests malicious software for detectability at the host and network level.

piscivorous@gmail.com

AS WITH ANY TECHNOLOGY DESIGNED to detect malicious activity (e.g., intrusion detection, burglar alarms, etc.), digital media investigation is a constant struggle to keep up. Common tools such as EnCase, The Coroners Toolkit (TCT), and The Sleuth Kit (TSK) have limitations that crackers are taking advantage of. While these tools have become adept at finding evidence on a non-volatile storage device such as a hard drive that has been physically removed (i.e., a "dead" analysis), volatile information, specifically memory, is much more difficult to investigate. However, there is a remarkable amount of data present in memory—to date there is no way to implement a process/activity on a computer without leaving a footprint in memory. For example, a cracker compromises a server, installs a rootkit, then secure-deletes unnecessary files (i.e., via SRM or PGP Shred) from the hard drive. At this point, if the power cord is yanked and the hard drive imaged, evidence of the rootkit will be that much harder to find with the aforementioned tools.

This article will attempt to give an admin faced with a potential rootkit, "live" investigative methods that could be undertaken prior to a dead analysis. Keep in mind that "dead analysis" in this case means powering off the machine (either cleanly or by yanking the power cord) and imaging the hard drive. We'll be imaging memory and analyzing it offline, but the target system will not be powered off.

Note that the authors are not promoting a deviation from a dead analysis. Offline hard drive searches are still the number one way to find evidence. However, as previously stated, there are circumstances where the hard drive doesn't contain the evidence you are looking for. In those cases, here are two methods you can use to examine volatile data.

UNIX

UNIX offers fairly straightforward memory access via the /proc virtual file system, and /proc/kcore allows inventive strings-ing, such as the following hack

which generically lists all loaded kernel modules (LKMs) in Linux (tested 2.4 kernel only):

```
#! /usr/bin/perl

use strict;
use warnings;

open(FH, "strings /proc/kcore |")
  || die "Could not open /proc/kcore for reading";

my %data;
while (<FH>) {
  next unless /__insmod/ && /_S.text/;
  next if /\// || /\"/;
  my $raw = (split /__insmod_/, $_)[1];
  my $module = (split /_S\./,$raw)[0];
  $data{ $module }++;
}

for my $key (keys %data) {
  print $key, "\n";
}
```

While this is a primitive method to list loaded modules, it also lists hidden LKMs such as Team Teso's adore (compare the output with Linux lsmod—the differences are hidden modules). Hacks like this can provide a UNIX system administrator with a quick first pass to determine if the machine in question has an LKM rootkit.

Windows memory is handled by the OS in a fashion that does not lend itself to live analysis per se, but one technique that works rather well is the capability of the Helix LiveCD (http://www.e-fense.com/helix/) to do a live capture of physical memory. This is accomplished by using a trusted dd executable on the CD, and the resulting file can be analyzed on a separate machine. For this investigation, we used the HackerDefender rootkit (http://www.hxdef.org/) and the Optix back door (http://www.megasecurity.org/trojans/o/optix/Optix_all.html) as the targets of our examples. A dd image of the physical memory was taken prior to the loading of each tool, then immediately after. The images were then compared via hex/binary editor to determine if either tool had left any residue in memory. While it would be impossible for a second investigator to analyze the live machine at a later time and obtain an exact copy of the dd image collected (i.e., identical md5sums), if the acquisition process is not found to be faulty, both defense and prosecution could analyze the exact same dd image were this evidence to go to court.

dd images can be imported into any number of forensic analysis tools, but to demonstrate that any admin can do a cursory examination, the bvi hex editor (http://bvi.sourceforge.net) was used. Any hex editor with basic search capability should be sufficient to do a quick analysis of a memory dump image file.

HackerDefender

A Windows 2000 Advanced Server SP4 system was booted up, and the Helix CD inserted. The main Helix screen appeared automatically, and the Live Acquisition option was chosen.

The amount of time it takes to dd is dependent upon how much memory you have. This particular system only had 256M, and it took just under six minutes. Once the imaging completed successfully, we installed HackerDefender (HD). One nice feature of HD is that it automatically hides the directory it was installed from, so you know it is working properly. Once we saw the folder disappear from Explorer we reran the memory dump, and six minutes later we had two dd images to compare.

Our theory was that there would not be any evidence of HD in memory in the pristine baseline image, which turned out to be accurate. We searched for the strings hxdef and HXDEF and found nothing.

Upon examining the image with HD loaded into memory, we found rather different results. The first hit shows the location of the HD executable, plus the excerpt powerful NT rootkit, which is from the HXDEF100.INI file:

```
0112E610  70 6F 77 65 72 66 75 6C 20 4E 54 20 72 6F 6F 74
   powerful NT root
0112E620  6B 69 74 00 48 58 44 20 53 65 72 76 69 63 65 20
   kit.HXD Service
0112E630  31 30 30 00 FF 01 0F 00 10 00 00 00 02 00 00 00
   100.............
0112E640  00 00 00 00 1F 00 00 00 00 00 00 00 1F 00 00 00
   ................
0112E650  47 3A 5C 54 4F 4F 4C 5A 5F 7E 32 5C 48 58 44 45
   G:\TOOLZ_~2\HXDE
0112E660  46 5C 48 58 44 45 46 31 30 30 2E 45 58 45 00 00
   F\HXDEF100.EXE..
```

This finding is notable for two reasons. First, since we did not view the HXDEF100.INI file, nor was that excerpt found in the baseline image, it seems logical that this is a remnant from loading the rootkit into memory. Second, the full path of the installation executable we used to install HD was G:\toolz_win\HXDEF\hxdef100.exe, most of which is clearly present in the block shown. In this experiment, both Optix and HD were installed from a USB drive, to limit the footprint on the hard drive. In a real-world scenario, an investigator would know that the G: drive corresponds to a USB drive, which would indicate that the attacker had physical access to the machine. Physical vs. remote access is a rather important piece of information, and it is highly unlikely that this path would be found in an examination of the hard drive alone.

Continuing the search, we find another excerpt from the HXDEF100.INI file, as well as mention of the HD driver hxdefdrv.sys. (Note that only the ASCII portion of the output will be shown in the rest of the examples.)

```
....erDrv100..D:
riv>erFileNam/e=
hxdefdrv.sys..
        ....>v
ic:eD||escr<ip:t
“ion=powerful NT
 rootkit..Dri<ve
\rN:ame=HackerDe
fend.............
```

There were numerous other findings for the strings HXDEF and hxdef, some of which were exact duplicates of what we've shown here, the rest similar. There is enough evidence in the image file to indicate that the contents of memory can be a useful place to look when suspicious of a rootkit. In fact, some of the evidence would never be found in a dead analysis of this system.

Optix

Back doors differ from rootkits and in fact are usually designed to work in conjunction with a rootkit. For example, most back doors do not hide files—that is the responsibility of the rootkit. What back-door programs do very well is open a port, giving an attacker easy return access. Optix is no different in this regard, and must be loaded into memory in order to function.

Our hypothesis, identical to that for HD, was that we would find no evidence of Optix in the baseline image. This was confirmed; there were no hits when searching for the strings optix and OPTIX in the baseline image, as there were after the tool was loaded.

As with HD, the Optix image showed the path for the installation executable:

```
\0.toolz_win.TOO
LZ_~2...1.....-3
uW0.optix undete
.OPTIXU~1.
```

Again, it is important to note that this important piece of evidence would probably not be found in a dead analysis.

Further findings were even more interesting than HD, as Optix is a bit more invasive out of the box. Another hit on this image showed an HTTP GET request to an IRC server, with embedded information advertising that this machine is infected:

```
GET /wwp/msg/1,,
,00.html?Uin=262
950210&Name=Joe+
Bloggs+is+online
+from+Optix+Pro+
v1.0.%0AIP:+[10.
200.200.249]%0AP
ort:+3410%0APwd:
+<NO+PASSWORD>%0
AUserName:+Admin
istrator%0AConne
ctionType:+Unkno
wn%0A%0A&Send=ye
s HTTP/1.1..Host
: web.icq.com..C
onnection: Keep-
Alive.....
```

This type of entry and certain variants were very common in this memory dump. A more complete example, and possibly the most interesting find, starts at byte offset 015B77B0 and ends at offset 015B7B30. Since this block is so large, we'll focus on the individual pieces broken into text format.

This is an intact HTTP return code 302 from an Apache Web server, stating that the document requested is located at a different URL, but was found:

```
HTTP/1.1 302 Found
Date: Thu, 22 Sep 2005 18:29:50 GMT
Server: Apache
```

The document requested was popup.php, which was passed arguments to announce that "Joe Bloggs" has connected to http://icq.com via Optix. The administrator account is to be used for return access to the back-doored system (which has an IP address of 10.200.200.249), and there is no password required to make a connection:

```
Location:
http://www.icq.com/icqchat/popup.php?Uin=262950210&Name=
Joe+Bloggs+is+online+from+Optix+Pro+v1.0.%250AIP:+%5b10
.200.200.249%5d%250APort:+3410%250APwd:+%3cNO+
PASSWORD%3e%250AUserName:+Administrator%250A
ConnectionType:+Unknown%250A%250A&Send=yes
```

There were numerous Keep-Alive entries within the image. The next example has a "spooll32.exe" reference ("spooll32.exe" is the default name of the Optix server executable):

```
Spooll32.exe.poo
`...`.......GET
/wwp/msg/1,,,00.
html?Uin=2629502
10&Name=Joe+Blog
gs+is+online+fro
m+Optix+Pro+v1.0
<remainder of Keep-Alive snipped for brevity>
```

The last search hit was found entirely by accident, as it contains neither the optix nor OPTIX strings. It does, however, give an email address (joe@hotmail.com) and lists a different URL than we've seen before (http://www.anycgihost.com/cgi-bin/subseven.cgi):

```
....joe@hotmail.
com.............
Optix Pro v1.0..
<snipped for brevity>
....http://www.a
nycgihost.com/cg
i-bin/subseven.c
gi.............
<snipped for brevity>
.... ...action=l
og&ip=[IPADDRESS
]&port=[SERVERPO
RT]&id=[VICTIMNA
ME]&win=[WINUSER
NAME]&rpass=[SER
VERPASSWORD]&con
nection=[CONNECT
IONTYPE]&s7pass=
[SCRIPTPASSWORD]
```

This is interesting not only because of the different URL, but that the SubSeven back door was mentioned. This appears to be a CGI script that takes input such as IP address, port, username, password, etc., and logs it for later retrieval, as opposed to posting this information to an ICQ channel as we saw in a previous example.

This is a representative sample of the findings for the strings optix and OPTIX, but a normal investigation would progress to searching for other strings, such as joe, bloggs, spooll32.exe, etc.

Conclusion

The point of the Helix exercise was not to demonstrate all the possible ways to find HackerDefender or Optix, but simply to show that there is value in examining physical memory before pulling the plug and imaging the hard drive for a dead analysis. Further, although not quite the same as grepping/strings-ing through /proc or kcore in *NIX, there is a method for conducting this type of search in the Windows environment.

Although we have not tested to determine whether or not the techniques shown here would have an effect on the hard drive data, it would seem to be a minimal impact, if any. It is possible that the swapfile might change as a result of taking a dd image, but as long as the image is saved on a different physical device, the

modification of the suspect hard drive should be minimal. If courtroom evidence is of the utmost concern, law enforcement may be involved, which could preclude any type of live analysis. If this is not the case, and a quick live analysis is possible, it seems remiss not to examine /proc (*NIX) or take an image of the physical memory for later examination (Windows).

Are the techniques shown here silver bullets to all rootkit and back-door contaminations? Of course not. They are simply methods that should not be overlooked when you suspect malicious activity. A caveat: in using these methods, as with most forensics and/or incident response, you have to know what you are looking for. The authors were fortunate in this case to know what programs were loaded, and so had a head start on what strings to search for. In the wild, this will probably not be the case, unless intrusion detection system (IDS), firewall, syslog, etc., events provide insight into the type of attack performed. It is quite plain in the Windows memory dump, however, that certain strings might be common to any back door or rootkit. For example, any back door looking to connect to an ICQ server or Web server might put the string icq or http into memory—Unicode notwithstanding. Further complicating matters, the test system was a virgin install, so the contents of memory were significantly more static than, say. a corporate email server.

Also keep in mind that the authors are not lawyers or law enforcement personnel. Always assume that any investigation could ultimately end up in court, and verify that live analysis of the potential system is acceptable before trying these techniques.

REFERENCES

Michael Ford's *Linux Memory Forensics* (http://www.samag.com/documents/s=9053/sam0403e/0403e.htm) covers Linux memory forensics using methods analogous to those in this article.

Adore at PacketStorm: http://packetstorm.linuxsecurity.com/groups/teso/.

Helix by e-fense: http://www.e-fense.com/helix/.

Hacker Defender by rootkit.com: http://www.hxdef.org/.

Optix by Evil Eye Software: http://www.megasecurity.org/trojans/o/optix/Optix_all.html.