Global Information Assurance Certification Paper

Understanding and Guarding against Rootkits
Mark Carney
June 20, 2001

Imagine this:
You were just paged because several system administrators have complained about attacks originating from your server(s). It's two in the morning. This is when you should be going to sleep, not responding to an incident. Your initial pass showed nothing unusual – maybe a user trying to sneak GNUtella by you (easily handled), perhaps an oversized porn collection in a manager's directory (a bit stickier), but no obvious signs of a compromise. A quick call to one of the administrators who reported this issue confirms that they are still actively under attack. So here you are brewing some coffee and dreading the thought of leaving for work in a few hours with no rest.

Nothing odd presents itself when you issue the netstat or ps commands. In fact, it looks like you are the only unlucky soul logged in at this late hour. You check on your network intrusion detection system (NIDS) and see a steady stream of events from the suspect server to several others both internal and external. In fact, the NIDS shows similar activity from several other servers. You perform a little additional checking on the suspect box and still come up empty.

The coffee is ready. You ask yourself, "What can hide itself almost completely?" The answer makes you wince: "Rootkit."

So what is a rootkit?
Rootkits are tools used by hackers to hide their presence on compromised systems while ensuring their ability to regain access with full control.

Once the attacker gains control of a system, they have to face reality and accept that the hole they used to get in won't exist forever. So they typically install backdoors to let them in again later. The most simple and recognizable form is a shell bound to a high numbered port so that simply issuing a command like "telnet <IP ADDRESS> <PORT>" will yield instant root access. A more insidious method used by insiders is to trojan a binary which runs as root with additional command line options or internal commands which will spawn a root shell. The better hackers even tend to secure systems when they gain access. This includes patching known bugs and sometimes password protecting the back doors.

The less dangerous rootkits are simply a collection of trojaned binaries to replace common commands such as:

- ls, du and find (to hide files)
- ps, top, kill [and variants thereof] (to hide and prevent killing of processes
- passwd, chfn, chsh (certain arguments will drop you into a root shell)
- login (to prevent use of the back door from being logged)
- netstat (to hide connections with certain IP addresses or to certain ports)

Since these rely on replacement binaries, they require either a compiler on the system or that someone build them elsewhere. Due to differences between systems, this can result in improper behavior ranging from core dumps to obscure error messages. Also, the MD5 checksums of the binaries won't match those of the files they replaced.

In time, hackers will think the patch md5sum and similar tools to use checksums of non-trojaned binaries (safely copied to a hidden directory) when asked to check a trojaned file.

The worst rootkits by far are those using loadable kernel modules. These reduce the need for trojaned binaries by replacing the functions normally called by them.

Whichever type is used, the goals are the same:

- Hide/remove evidence of initial entry
- Establish back doors for re-entry
- Prevent logging of activity
- Hide files and directories
- Hide specific contents of files (such as additional users in /etc/passwd)
- Gather intelligence (packet sniffers or other tools may be used to catch cleartext usernames and passwords)

Back up. I'm confused. Why would they secure my system?
The simple answer is: "to avoid detection." If you find a door unlocked and enter the structure, locking the door behind you means that there is a significantly lower chance of someone else coming in the same way then making enough noise to get one or both of you caught.

That is essentially the same issue facing electronic intruders. They know how carefully they will cover their tracks, but do they trust other intruders not to draw unwanted attention? Would the second intruder lock them out? Would an investigator get confused and accuse them of the other person's crimes? How many people can hack the system before an administrator catches on and rebuilds the server?

Rather than face these risks, more experienced hackers protect themselves and their time investment.

Okay, now what is a Loadable Kernel Module?
A loadable kernel module (LKM) is essentially part of the kernel that can be loaded and unloaded as needed so the base kernel stays smaller. This can result in lower overall memory usage, greater flexibility and faster load times for the operating system. When these modules are loaded, their system calls and functions become part of the running kernel. This is how your device drivers work.

By hacking the kernel with LKMs, an intruder can become a ghost in the machine. A few subverted system calls and the commands (which still have valid MD5 checksums)

are suddenly unable to find any sign of this ghost. If you can't trust the kernel, what can you trust?

While these rootkits are most prevalent on Linux systems, they are also somewhat common for BSD and Solaris.

How does this work?
Commands like "ps" and "ls" are small binaries that call various functions in the kernel to retrieve desired information then present it to the user in an understandable manner. By replacing the existing functions with modified ones, the LKM can simply refuse to divulge information it has been told to protect. The rest of the content is passed back and the command appears to function normally so the user is never really aware that they are not seeing all the data.

On a Solaris system, the syscall "getdents64()" is used to retrieve information about files and directories (for Linux, it is "sys_getdents()" and "getdirentries()" for BSD). The intruder simply replaces the function with one that uses a static value or a list of values from a file to determine which data to throw out. The overall effect is much like using "grep –v" on the data before it is passed back to the binary to be displayed.

Plasmoid and Pragmatic of THC wrote the definitive works on this topic. Reading their white papers is strongly recommended:

BSD:	http://www.thehackerschoice.com/papers/bsdkern.html
Linux:	http://www.thehackerschoice.com/papers/LKM_HACKING.html
Solaris:	http://www.thehackerschoice.com/papers/slkm-1.0.html

Hah! I run NT!
LKM rootkits exist for Windows NT/2000 as well. The first one (NTROOT) loaded as a driver. More recent releases come with an executable named "deploy.exe" and use the SystemLoadAndCallImage() syscall. Once the kit has been loaded, the intruder can easily hide processes, files, directories and even registry entries. They will also be able to capture keystrokes from the login screen

NT administrators are strongly urged to check NTROOT's home page for more information. To see whether the kit is present on your system, you can simply copy any file so that the new file name begins with _root_ (ex: *copy autoexec.bat _root_a.txt*) and see whether this new file is still found when you type "dir". If not, you have been kitted.

When the site administrators started changing over to a Source Forge style user interface, a lot of useful content was lost. It might be in your best interests to save the binary off to a floppy and try it on an isolated system. This is a real eye-opener. Just don't forget to wipe and rebuild the system when you finish.

OK. How do I prevent all this?
There is no guaranteed method except to pull the plug; but you can make it as difficult as possible for someone to hack your systems and even harder for them to do so undetected.

The only way to guarantee someone will not make your systems into a second home is to prevent them from gaining entry.

Keep your systems patched. As simple as this sounds, it is one of the most overlooked steps. Sometimes administrators will argue that they do not have time to install every update across all the servers for which they are responsible. They are often correct but this can be remedied with proper tools and practices.

Set up a database of systems, approved services, version numbers, patch levels, permitted users and who is responsible for each aspect. This significantly reduces the amount of time needed when developing your response plan for a newly discovered vulnerability. Printing out a list of systems to patch gives you a checklist to work from so nothing will be missed and even a way to divide labor for more rapid deployment of the vendor supplied update. The responsible parties for whichever systems and services can be rapidly identified and brought in to assist.

Deploy NIDS at key points in your network. Try to keep services on these systems to a minimum to reduce the chances of them being compromised. Don't forget that some routers can run NIDS (such as Cisco's Net Ranger). With the recent drops in PC prices, these systems can be very cheaply deployed with Snort. Remember that the incoming events can be used to justify the costs of such a system and potentially more people to handle security.

Use TCP Wrappers to collect more information on connection attempts and disable all unnecessary services.

Group systems by services they perform and place internal firewalls at key points. By reducing the number of networks and hosts an intruder can reach from a compromised host, you can significantly impede their ability to spread throughout your infrastructure.

Deploy syslog aggregation systems and use them to collect log information from all your systems. This data can be quite valuable when examining an "owned" server since comparing the remote logs against those on the compromised system gives you a good starting place. It is very nice of the hackers to let us know exactly which records they do not want us to see.

Dedicate a system to software builds. Restrict access heavily and only build your binaries on that one system. Transfer files to and from the system via CD-R. Take compilers off the other servers. Why provide tools for attackers?

Use products like Tripwire to detect changes to important files. Keeping current MD5 checksums of these binaries on a CD or floppy (with the read-only tab set) is also a good idea (preferably created when the software is built from trusted code).

For systems that have a low load, consider running St. Michael which can help prevent LKM rootkits from taking hold on your server.

Most anti-virus (AV) packages include signatures for common trojans and rootkits. Deploy AV software and keep the signatures updated.

Run chkrootkit regularly on your unix boxes. This tool can detect twenty known rootkits and their major variants. It looks for signatures in trojaned binaries and modules which are not reported correctly.

So if I do all this, I'm safe?
No. "Safer" perhaps, but security is an ongoing process. Many of the steps above are geared toward rapid detection of events and gathering data to aid in the forensic investigation.

Some of these suggestions will reduce the risk of outsiders gaining access initially; but as unpleasant as it is, the inside threat must also be watched for.

After playing with several rootkits on an isolated network, I can say they are relatively young. A few have reached an electronic puberty and are about to develop into more complex creatures but even they are faulty and detected relatively easily. Some can be caught just by looking for normal files in your /dev directory.

What will make me safe?
While non-trivial, I believe that we will soon see kernels, modules and binaries with embedded PGP information.

Imagine compiling a kernel and embedding your public PGP key (or perhaps a ring of trusted developers' keys). When the compile finishes, you are asked to sign the kernel and modules. You enter your passphrase and several moments later you can install a self-verifying kernel.

During the boot process, the kernel would begin to unpack itself and verify the result with the embedded PGP key. Each module would be verified against the known key(s). Every binary executed with elevated permissions could be required to have a valid signature or would simply be logged as invalid while returning the user to their prompt.

While this won't occur in the next several weeks or months, we can probably expect something like it within the next couple years.

Summary
On the surface, these threats are somewhat daunting. Even patching the initial entry point won't dislodge someone who has had time to install a rootkit. However, with good processes (and follow through), these events can be detected and dealt with quickly.

Building and maintaining layered defenses can slow the spread of hackers and malicious code through your systems. Distributed logging makes it more difficult for attackers to effectively cover their tracks.

Most importantly, keep your systems patched. If hackers can't get in, they can't install rootkits.

Bibliography

Brumley, David. "invisible intruders: rootkits in practice." November 16, 1999. URL: http://www.usenix.org/publications/login/1999-9/features/rootkits.html (August 20, 2001).

Dittrich, Dave. "'Root Kits' and hiding files/directories/processes after a break-in." Version 1.3. June 6, 2001. URL: http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq (August 20, 2001).

Hecix. "Re: LJK2 rootkit?" May 19, 2000. URL: http://archives.neohapsis.com/archives/incidents/2000-05/0197.html (August 20, 2001).

Hoglund, Greg. "A *REAL* NT Rootkit, patching the NT Kernel." September 9, 1999. URL: http://www.phrack.org/show.php?p=55&a=5 (August 20, 2001).

Hoglund, Greg. "Loading Rootkit using SystemLoadAndCallImage." August 29, 2000. URL: http://www.sumthin.nu/archives/ntbt/Aug_2000/msg00057.html (August 20, 2001).

Plasmoid. "Solaris Loadable Kernel Modules." Version 1.0. 1999. URL: http://www.thehackerschoice.com/papers/slkm-1.0.html (August 20, 2001).

Pragmatic. "(nearly) Complete Linux Loadable Kernel Modules." Version 1.0. March, 1999. URL: http://www.thehackerschoice.com/papers/LKM_HACKING.html (August 20, 2001).

Pragmatic. "Attacking FreeBSD with Kernel Modules." Version 1.0. June, 1999.URL: http://www.thehackerschoice.com/papers/bsdkern.html (August 20, 2001).

Prosise, Chris and Shah, Saumil Udayan. "Detecting rootkits." February 8, 2001. URL: http://quickenexcite.cnet.com/webbuilding/0-7532-8-4720241-1.html?tag=st.bl.7532.edt.7532-8-4720241-1 (August 20, 2001).