Rootkit

From Wikipedia, the free encyclopedia

A **rootkit** is a program (or combination of several programs) designed to take fundamental control (in Unix terms "root" access, in Windows terms "Administrator" access) of a computer system, without authorization by the system's owners and legitimate managers. Access to the hardware (ie, the reset switch) is rarely required as a rootkit is intended to seize control of the operating system running on the hardware. Typically, rootkits act to obscure their presence on the system through subversion or evasion of standard operating system security mechanisms. Often, they are also Trojans as well, thus fooling users into believing they are safe to run on their systems. Techniques used to accomplish this can include concealing running processes from monitoring programs, or hiding files or system data from the operating system.[1]

Rootkits may have originated as regular, though emergency, applications, intended to take control of an unresponsive system, but in recent years have been largely malware to help intruders gain access to systems while avoiding detection. Rootkits exist for a variety of operating systems, such as Microsoft Windows, Mac OS X[2] [3] , Linux and Solaris. Rootkits often modify parts of the operating system or install themselves as drivers or kernel modules, depending on the internal details of an operating system's mechanisms.

Contents

History

The term *rootkit* or *root kit* originally referred to a maliciously modified set of administrative tools for a Unix-like operating system. If an intruder could replace the standard administrative tools on a system with a rootkit, the modified tools would give the intruder administrative control over the system while concealing his activities from the legitimate system administrator. The earliest known rootkit was written ca. 1990 by Lane Davis and Riley Dake for SunOS 4.1.1. There was an earlier, quite famous, exploit equivalent to a rootkit which was perpetrated by Ken Thompson of Bell Labs against a Naval Laboratory in California to win a bet. Thompson subverted the C compiler in a distribution of Unix to the Lab.

Rootkits were so named because they allowed an intruder to become a root user (ie, the system administrator) of a Unix system. Since then, similar software has been developed for other operating systems, and the term *rootkit* has been broadened to include any software that surreptitiously alters an operating system so that an unauthorized user can take arbitrary control of the system.

Rootkits became much better known in 2005, when Sony BMG caused a scandal by including rootkit software on music CDs which altered the Windows OS to allow access to anyone aware of the rootkit's installation. Supposedly, this was done to enforce copy protection of the music on the CDs. The scandal following the discovery and subsequent public notice of this corporate-sponsored malware—a scandal made much worse by the clumsy and ill-informed statements of Sony executives—made many users previously unfamiliar with rootkits wary.

Common use

A successfully installed rootkit allows unauthorized users to act as system administrators, and thus to take full control of the 'rootkitted' system. Secondary to this purpose, most rootkits typically hide files, network connections, blocks of memory, or registry entries (eg, on Windows systems) from other programs used by system administrators to detect specially privileged accesses to computer system resources. However, a rootkit may masquerade as or be intertwined with other files, programs, or libraries with other purposes. It is important to note that while the utilities bundled with a rootkit may be maliciously intended, not every rootkit is always malicious. Rootkits may be used for both productive and destructive purposes.

A rootkit which hides utility programs, usually does so to abuse a compromised system, and often include so-called "backdoors" to help the attacker subsequently access at will. A simple example might be a rootkit which hides an application that spawns a command processing shell when the attacker connects to a particular network port on the system. Kernel rootkits may include similar functionality. A backdoor may also allow processes started by a non-privileged user to run as though it were started by a privileged user (including the root user) and to carry out functions normally reserved for the superuser.

Many other utility tools useful for abuse can be hidden using rootkits. This includes tools for further attacks against computer systems with which the compromised system communicates, such as sniffers and keyloggers. A possible abuse is to use a compromised computer as a staging ground for further abuse (see zombie computer). This is often done to make the abuse appear to originate from the compromised system (or network) instead of the attacker's. Tools for such attacks can include denial-of-service attack tools, tools to relay chat sessions, and e-mail spam distribution. A major malicious use for rootkits is to allow the rootkit's programmer to see and access user names and log-in information of systems requiring them. Collection of such information from many systems (thousands or more) is easily possible. This makes rootkits even more hazardous, as it allows trojans to access this personal information while the rootkit covers it up.

It has become increasingly popular for virus writers to make use of rootkit technologies. The reason for this is obvious – they make it possible to hide malware from PC users and antivirus programs. Numerous source codes for ready-made rootkits can be found on the Internet, which inevitably leads to their widespread use in various trojans or spyware programs etc.

But rootkits are not always used to attack and gain control of a computer. Some software may use rootkit techniques to hide from 3rd party scanners to undetectably detect tampering or attempted breakins. Some emulation software and security software is known to be using rootkits.[4] Alcohol 120% and Daemon Tools are commercial examples of the use of non-hostile rootkits.

Rootkit is a term now somewhat loosely applied to cloaking techniques and methods.[5]

Types

There are at least five kinds of rootkits: firmware, virtualized, kernel, library, and application level kits.

Firmware

A firmware rootkit uses device or platform firmware to create a persistent malware image. The rootkit can successfully hide in firmware because firmware is not often inspected for code integrity. John Heasman demonstrated the viability of firmware rootkits in both ACPI firmware routines[6] and in a PCI expansion card ROM.[7]

Virtualized

These rootkits work by modifying the boot sequence of the machine to load themselves instead of the original operating system. Once loaded into memory, a virtualized rootkit then loads the original operating system as a Virtual Machine, thereby enabling the rootkit to intercept all hardware calls made by the guest OS. The **SubVirt** (http://www.eecs.umich.edu/virtual/papers/king06.pdf) laboratory rootkit, developed jointly by Microsoft and University of Michigan researchers, is one example of a Virtual Machine based rootkit (VMBR); Blue Pill is another.

Kernel level

Kernel level rootkits add additional code and/or replace portions of an operating system, including both the kernel and associated device drivers. Most operating systems don't enforce any security distinctions between the kernel and device drivers. As such, many kernel mode rootkits are developed as device drivers or loadable modules, such as Loadable Kernel Modules in Linux or device drivers in Microsoft Windows. This class of rootkit is perceived as dangerous simply because of the unrestricted security access the code has obtained, regardless of the features the rootkit may employ. Any code operating at the kernel level may have serious impacts on entire system stability if mistakes are present in the code. The first and original rootkits did not operate at the kernel level, but were simple replacements of standard programs at the user level. Although traditionally security advances were made first on Unix systems, the first kernel rootkit was developed for Windows NT 4.0 and released in the mid-1990's by Greg Hoglund.

Kernel rootkits can be especially dangerous because they can be difficult to detect. The reason they can be difficult to detect is because they operate at the same level as the operating system, thus they can modify or subvert any request made by software on the running system. In a situation such as this, the system itself cannot be trusted. An accepted proper response in such a case is to perform system analysis offline using a second 'trusted' system and mounting the hard drive of the infected system as a resource.

Library level

Library rootkits commonly patch, hook, or replace system calls with versions that hide information about the attacker. They can be found, at least theoretically, by examining code libraries (under Windows the term is usually DLL) for changes or against the originally distributed (and so presumably rootkit free) library package. In practice, the variety of modified libraries distributed with applications and ServicePacks makes this harder than it should have been.

Application level

Application level rootkits may replace regular application binaries with trojanized fakes, or they may modify the behavior of existing applications using hooks, patches, injected code, or other means.

Detecting

Rootkit binaries can often be detected by signature or heuristics based antivirus programs, at least until they're run by a user and are able to attempt to conceal themselves. There are inherent limitations for any program that attempts to detect rootkits while the program is running under the suspect system. Rootkits are suites of programs that modify many of the tools and libraries upon which all programs on the system depend. Some rootkits attempt to modify the running kernel via loadable modules on Linux (and some other UNIX varieties), and through VxDs, virtual external drivers, on MS Windows platforms. The fundamental problem with rootkit detection is that if the operating system currently running has been subverted, it cannot be trusted, including to find unauthorized modifications to itself or its components. In other words, actions such as requesting a list of all running processes, or a list of all files in a directory, cannot be trusted to behave as intended by the original designers. Rootkit detectors running on live systems currently only work because the rootkits they can detect have not yet been developed to hide themselves fully.

The best, and most reliable, method for rootkit detection is to shut down the computer suspected of infection, and then check its storage by booting from an alternative medium (e.g., a rescue CD-ROM or USB flash drive). A non-running rootkit cannot (ideally) hide its presence, and most established antivirus programs will identify rootkits armed via standard OS calls (which are often tampered with by the rootkit) and lower level queries, which ought to remain reliable. If there is a difference, the presence of a rootkit infection should be assumed. Running rootkits attempt to protect themselves by monitoring running processes and suspending their activity until the scanning has finished; this is more difficult if the rootkit is not allowed to run.

Security software vendors have attempted a solution by integrating rootkit detection into traditional antivirus products. Should a rootkit decide to hide during scanning, it will be identified by the stealth detector. If it decides to temporarily

unload from the system, the traditional antivirus will find it using fingerprint detection. Since anti-virus products are almost never entirely capable of catching all viruses in public tests, this approach may be doubted on past behavior. But this combined approach may force attackers to implement counter-attack mechanisms (so called retro routines) in their rootkit code that will forcibly remove security software processes from memory, effectively killing the antivirus program. As with computer viruses, the detection and elimination of rootkits will be an ongoing struggle between tool creators on both sides of this conflict.

There are several programs available to detect rootkits. On Unix-based systems, three of the most popular are chkrootkit, rkhunter and OSSEC. For Windows, there are many free detection tools such as Sophos Anti-Rootkit (http://www.sophos.com/products/free-tools/sophos-anti-rootkit.html) , F-Secure Blacklight (http://www.f-secure.co.uk/blacklight/blacklight.html) , Hypersight Rootkit Detector (http://northsecuritylabs.com/) or Radix Anti-Rootkit (http://www.usec.at/rootkit.html) . Another Windows detector is RootkitRevealer from Microsoft (formerly Sysinternals) which detects current rootkits by comparing the results from the OS to the actual listing read from the disk itself (**cross-checking**). However, some rootkits started to add RootkitRevealer to a list of files it does not hide from -- so in essence, they remove differences between the two listings, and the detector doesn't report them (most notably the commercial rootkit *Hacker Defender Antidetection*). Another method is to compare content of binaries present on disk with their copies in operating memory - some differences can be introduced by legal operating system mechanisms (e.g., memory relocation), but some can be very likely classified as system call hooks introduced by a running rootkit (**System Virginity Verifier**). Zeppoo is another software product which detects rootkits under Linux and UNIX systems.

As always, prevention is better than cure, for being certain you have removed a rootkit typically involves re-installation of all software. If the integrity of the system install disks is trusted, cryptography can be used to monitor the integrity of the system. By "fingerprinting" the system files immediately after a fresh system install and then again after any subsequent changes made to the system (e.g., installing new software), the user or administrator will be alerted to any dangerous changes to the system's files. In the fingerprinting process a message digest is used to create a fixed-length "digest" dependent on every bit in the file being fingerprinted. By calculating and comparing message digest values of files at regular intervals, changes in the system can be detected.

Detection in firmware can be achieved by computing a cryptographic hash of firmware and comparing hash values to a whitelist of expected values, or by extending the hash value into TPM (Trusted Platform Module) configuration registers, which are later compared to a whitelist of expected values. Code that performs hash, compare, and/or extend operations must itself not be compromised by the rootkit. The notion of an immutable (by a rootkit) root-of-trust, if implementable, ensures that the rootkit does not compromise the system at its most fundamental layer. Rootkit detection using a TPM is further described in Stopping Rootkits at the Network Edge, January 2007. (https://www.trustedcomputinggroup.org/news/Industry_Data/Whitepaper_Rootkit_Strom_v3.pdf)

Removing

Many hold this to be forbiddingly impractical. Even if the nature and composition of a rootkit is known, the time and effort of a system administrator with the necessary skills or experience would be better spent re-installing the operating system from scratch. Since drive imaging software makes the task of restoring a “clean” OS installation almost trivial, there is no good reason to try to dig a rootkit out directly. *"I suppose traditional rootkits could be made to be as hard to remove as possible even when found, but I doubt there is much incentive for that, because the typical reaction of an experienced sysadmin on finding a rooted system is to save the data files, then reformat [and reinstall]. This is so even if the rootkit is very well known and can be removed 100%."* —Rootkit Question (http://forums.spywareinfo.com/lofiversion/index.php/t52360.html)

While most Anti-Virus and Malware Removal tools remain ineffective against rootkits, tools such as BartPE and other Preinstallation Environment(PE) or Live Distros allow users to boot their computer with a fresh (presumably) "un-rooted" copy of the operating system. This allows users to examine and replace affected system files and delete offending rootkits of most types while keeping the underlying systems intact. Since most rootkits hook system files needed at the lowest level of the OS, booting into Safe Mode will not usually allow removal of the rootkit process. In contrast, PE's do not rely on the infected underlying system structure but instead load a clean read-only copy of the Operating System allowing full control and detection of the rootkit. While most Administrators prefer a clean reinstall, a skilled Administrator using a PE can often delete and clean a rooted system if a reinstall is not a viable option.

Comparison with computer viruses and worms

The key distinction between a computer virus and a rootkit relates to propagation. Like a rootkit, a computer virus modifies core software components of the system, inserting code which attempts to hide the "infection" and provides some additional feature or service to the attacker (ie, the "payload" of a virus).

In the case of the rootkit the payload may attempt to maintain the integrity of the rootkit (the compromise to the system) --- for example every time one runs the rootkit's version of the *ps* command, it may check the copies of *init* and *inetd* on the system to ensure that they are still compromised, "re-infecting" as necessary. The rest of the payload is there to ensure that the intruder continues to control the system. This generally involves having backdoors in the form of hard-coded username/password pairs, hidden command-line switches or 'magic' environment variable settings which subvert normal access control policies of the uncompromised versions of the programs. Some rootkits may add port knocking checks to existing network daemons (services) such as *inetd* or the *sshd.*

A computer virus can have any sort of payload. However, the computer virus also attempts to spread to other systems. In general, a rootkit limits itself to maintaining control of one system.

A program or suite of programs that attempts to automatically scan a network for vulnerable systems and to automatically exploit those vulnerabilities and compromise those systems is referred to as a computer worm. Other forms of computer worms work more passively, sniffing for usernames and passwords and using those to compromise accounts, installing copies of themselves into each such account (and usually relaying the compromised account information back to the intruder through some sort of covert channel).

There are also hybrids. A worm can install a rootkit, and a rootkit might include copies of one or more worms, packet sniffers or port scanners. Also many of the e-mail worms are commonly referred to as "viruses." So all of these terms have somewhat overlapping usage and are often conflated.

Publicly available

Like much malware used by attackers, many rootkit implementations are shared and are easily available on the Internet. It is not uncommon to see a compromised system in which a sophisticated publicly available rootkit hides the presence of unsophisticated worms or attack tools that appear to have been written by inexperienced programmers.

Most of the rootkits available on the Internet are constructed as an exploit or "proof of concept" to demonstrate varying methods of hiding things within a computer system and of taking unauthorized control. Since these are often not fully optimized for stealth, they sometimes leave unintended evidence of their presence. Even so, when such rootkits are used in an attack they are often very effective.

See also

- Hacker con
- Computer virus
- Host-based intrusion detection system
- The SANS Institute
- 2005 Sony BMG CD copy protection scandal

References

1. ^ Brumley, David (1999-11-16). invisible intruders: rootkits in practice (http://www.usenix.org/publications/login/1999-9/features/rootkits.html) . USENIX.
2. ^ Leyden, John (2004-10-25). Mac OS X rootkit surfaces: Unpleasant Opener (http://www.theregister.co.uk/2004/10/25/mac_rootkit_opener/) . The Register. Retrieved on 2007-07-15.
3. ^ SH.Renepo.B Symantec Security Response Report: SH.Renepo.B (http://www.symantec.com/security_response/writeup.jsp?docid=2004-102218-1803-99) . Symantec. Retrieved on 2007-07-15.
4. ^ Russinovich, Mark (2006-02-06). Using Rootkits to Defeat Digital Rights Management (http://blogs.technet.com/markrussinovich/archive/2006/02/06/using-rootkits-to-defeat-digital-rights-management.aspx) . *Winternals.* SysInternals. Archived from Using Rootkits to Defeat Digital Rights Management the original (http://www.sysinternals.com/blog/2006/02/using-rootkits-to-defeat-digital.html) on 2006-08-31. Retrieved on 2006-08-13.

5. ^ Unearthing Root Kits (http://www.windowsitpro.com/Article/ArticleID/46266/46266.html) by Mark Russinovich in *Windows IT Pro* June 2005.
6. ^ Implementing and Detecting an ACPI Rootkit, by John Heasman, presented at BlackHat Federal, 2006.
7. ^ Implementing and Detecting a PCI Rootkit (http://www.ngssoftware.com/research/papers/Implementing_And_Detecting_A_PCI_Rootkit.pdf) by John Heasman, 15 November, 2006.

- Mark Russinovich, Advanced Malware Cleaning **video** (http://www.microsoft.com/emea/itsshowtime/sessionh.aspx?videoid=359) , Microsoft TechEd: IT Forum, November 2006
- Robert S Morris, Sr. "UNIX Operating System Security", BSTJ, Vol. 62, No. 8, 1984 Bell Systems Technical Journal
- Greg Hoglund and James Butler. *Rootkits: Subverting the Windows Kernel.* Addison Wesley, 2005. ISBN 0-321-29431-9
- Nancy Altholz and Larry Stevenson. *Rootkits for Dummies*. John Wiley and Sons Ltd, 2006. ISBN 0-471-91710-9
- Ric Veiler. *Professional Rootkits.* Wrox, 2007. ISBN 978-0-470-10154-4

External links

- White paper on new-generation rootkit detection (http://northsecuritylabs.com/downloads/whitepaper-html/)
- antirootkit.com (http://www.antirootkit.com) (Up to date rootkit information, news and removal software for less experienced users - also has a list of all rootkits in the wild)
- Testing of antivirus/anti-rootkit software for the detection and removal of rootkits (http://www.anti-malware-test.com/?q=taxonomy/term/7) made by Anti-Malware Test Lab, January 2008
- Testing of anti-rootkit software (http://www.informationweek.com/news/showArticle.jhtml?articleID=196901062) made by InformationWeek, January 2007
- Steve Gibson's Security Now! Episode #9 (http://www.grc.com/sn/SN-009.htm) (on rootkits)
- Steve Gibson's Security Now! Episode #12 (http://www.grc.com/sn/SN-012.htm) (on rootkits)
- Steve Gibson's Security Now! Episode #127 (http://www.grc.com/sn/SN-127.htm) (on Corporate Security - MBR rootkit update)
- Sony, Rootkits and Digital Rights Management Gone Too Far (http://blogs.technet.com/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx) (Mark Russinovich's first blog entry about the Sony DRM rootkit, from which the scandal ensued)
- Designing BSD Rootkits (http://www.oreilly.com/catalog/1593271425/) An Introduction to Kernel Hacking (book by Joseph Kong)
- How to remove spyware from your PC: rid yourself of rootkits (http://www.pcworld.ca/news/column/23a7f73b0a010408001a024ccab0dd5e/pg1.htm)
- Glossary of malware terminology (http://www.antispywarecoalition.org/documents/glossary.htm) ("Rootkit" has a negative connotation)
- White paper on hypervisor rootkit technology (http://www.crucialsecurity.com/documents/hvmrootkits.pdf)