# Rootkit Detection Mechanism: A Survey

Jestin Joy[1], Anita John[1], and James Joy[2]

[1] Rajagiri School of Engineering & Technology, Kochi, Kerala
jestinjoy@acm.org, anitaj@rajagiritech.ac.in
[2] Tata Elxsi, Thiruvananthapuram, Kerala
jamesjoy@tataelxsi.co.in

**Abstract.** Rootkits are a set of software tools used by an attacker to gain unauthorized access into a system, thereby providing him with privilege to access sensitive data, conceal its own existence and allowing him to install other malicious software. An attacker needs administrative level privileges before he could install a rootkit. Rootkits are the most challenging malware to detect due to their elusive nature. Modern rootkit attacks mainly focus on modifying operating system kernel. This paper tries to provide a structured and comprehensive view of the research on rootkit detection/prevention.

## 1 Introduction

In recent years attackers employ a variety of sophisticated methods to gain access to the system. Kernel level rootkits are one of the most lethal attacking tool available with the intruders, because of the difficulty in detecting them and the considerable damage they cause to the system. Main goal of a rootkit is to conceal the evidence of intruder activities. Most rootkits need administrative level privileges to install it in the system. Usually attacker make use of some system vulnerabilities to get administrative level privileges. An ordinary user will find it difficult to detect the presence of rootkit, since he will not find any discrepancy in the behavior of the system, even if the system is infected by a rootkit. With the increasing use of operating system in smart phones and other embedded devices the threat posed by rootkits[4][27] are of great concern.

Rootkits first appeared in the end of 80's as method to hide log files and now they pose a serious threat to computer industry [6]. The first generation rootkits were easier to detect since they mainly aimed at modifying user level programs(logs, application binaries . . . ). Methods like checksum verification could easily detect these type of infections [15].Great deal of research is going on in the area of rootkit detection. In the past system administrators relied heavily on system utilities like ls, ps . . . to find the presence of rootkits [15]. But new generation rootkits could easily hide their presence from these utilities.

Virtualization rootkits loads host OS as guest and can monitor all the host OS activities. BluePill, SubVirt etc are rootkits coming under this category. BluePill installs at the hypervisor level and controls the execution of the target OS. Kernel level rootkits modifies vital areas of an operating system like its kernel and the device drivers. Library level rootkits modifies or replaces standard system libraries with versions that help the attacker in hiding information. Difficulty in detecting kernel level rootkits arise out of the fact that they operate at the same security level as the kernel. Because of this, attackers now mainly rely on Kernel level rootkits. Our paper mainly focuses on kernel level rootkit detection mechanisms.

## 1.1 Kernel Level Rootkit

Unlike other rootkit types, kernel level rootkit modifies the kernel itself. Kernel being the lowest level of operating system makes it a good choice for the intruder to attack, since an attack on it is very difficult to detect. Also being at the kernel level provides the attacker with complete freedom to access all most all areas of an operating system.

First major attempt to categorize kernel level rootkits was done on the seminal paper[15] by Levine, Grizzard, Owen. They defined a framework for classifying rootkits. Their classification is based on the fact that along with the functionalities of a normal program, rootkit has some added functionalities that helps it to hide its activities.

In most cases rootkits finds its way into the kernel through Loadable Kernel Modules (LKM). LKM allows extending the functionality of the kernel, without recompiling the kernel. The code inserted using LKM provides same capability as a kernel code. Another important advantage of using LKM is that they can be added/removed on the fly. Though mainly aimed at debugging the kernel, */dev/mem* is also used by the intruders to attack the system [16]. In Linux based systems, modules can be inserted through the utilities *insmod* or *modprobe*.

Kernel level rootkits occur in different forms. They affect system call table [14], Virtual File System (VFS) (for example, by $adore - ng$ rootkit), Interrupt Descriptor Table (IDT) [24] .... VFS can be thought of as a kernel subsystem which provides a unified API to user-space applications. There are certain other rootkits that employ a combination of these techniques (example *zk* rootkit [15]).

When a user level program access a system resource, it is accomplished through a system call. The user level application performs a system call, which passes the control to the kernel. The requested work is done by the kernel and result is passed back to the user level application. So system call is an important target for attackers. Kernel level rootkits attack System Call table by different mechanisms [15]. Three of them are listed below

1. System Call Table modification
   In this method, attacker replaces original system call with his own custom version. Mainly this is done by modifying the system call address by inserting malicious LKM's address. Knark [7] rootkit uses this approach.
2. System Call Target modification
   In this attack, legitimate code in the target address is modified. This type of attack does not modify system call table. It works by modifying the flow of control in a system call. Usually a jump instruction is used to pass the control to the malicious code.

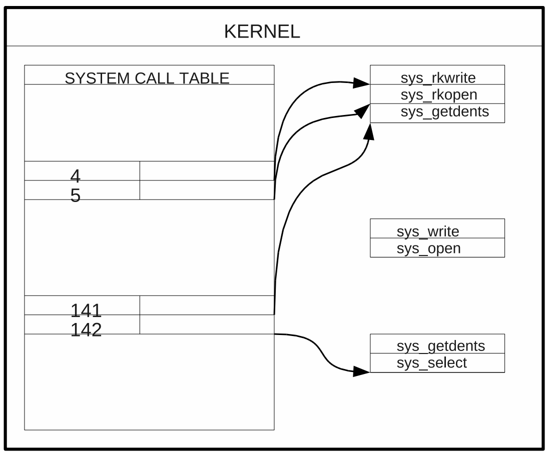

**Fig. 1.** System Call Redirection Example

3. System Call Table Redirection
   In this method attacker replaces the call to System Call Table with his own custom version. This is usually done by overwriting the memory where System Call Table address is stored. By comparing with *System.map* file this type of attacks of could easily be detected.

Fig.1 show the result of System Call redirection where, the system calls *sys_write*, *sys_open*, *sys_getdents* are replaced with modified ones; *sys_rkwrite*, *sys_rkopen*, *sys_rkgetdents*.

## 2 Detection Mechanism

Rootkit Detection mechanisms can be classified based on where the detection module resides. Early methods for detection relied mainly on archived copy of system files for detection [14]. But this method couldn't detect all type of attacks and also for the detection to work properly, the archived copy should be clean.

If kernel level rootkits are used, the method of using more than one application for detection doesn't work. For example we get information about the modules from both */proc/modules* and *lsmod* command. */proc* file-system is a special file-system that is used by the kernel to export information. Contents of this file-system are created when they are read. Many utilities get their information form */proc*. *lsmod* gets its information from */proc/modules*. If rootkit modifies */proc/modules* both *cat/proc/modules* and *lsmod* command gives same information. So gathering information relying solely on user space applications doesn't help.

This paper classifies detection mechanism into three (1) Host based (2) Virtualization based and (3) External observer based mechanisms. Another class; rootkit profilers helps to better understand rootkit attack strategies.

### 2.1 Host Based

Host based detection mechanism works from the infected system. Kern_check [14] is a utility that checks *System.map*, which stores symbols used by the kernel, against system call table of running kernel and warn about inconsistencies. Some other tools like CheckIDT, Chkrootkit [14], StMichael... are host based detection tools that make use of prior information for detection. StMichael is a LKM, that provides protection by monitoring various portions of the kernel, for modifications that may indicate the presence of a malicious kernel module. All the above tools rely heavily on prior information about the host system for detection.

Kruegel et al. [12] introduced a behavior based method for preventing rootkit modules from loading into the system. It uses static analysis to determine if a kernel module is malicious before the kernel module is loaded into the kernel and executed. Since this method is applied to the binary image of the module, it doesn't need source code for analysis. Detection is based on the following two major behavioral specifications

- Write operation to an illegal memory area
- Module contains instruction sequences that use forbidden kernel symbol reference to calculate an address in the kernels address space and a write operation based on this calculated address.

File integrity scanners like Tripwire [10], Samhain [26], AIDE [26] ... are powerful tools that aid system administrators in checking the trail of rootkits in the system. These tools generally use checksum based mechanism to take snapshot of the system, which is used for detecting modifications. Checksum based methods [15] fail when rootkits use dynamic directories to store rootkit related information. For example many rootkits rely on the */proc* directory(example *knark* rootkit) to store its contents. Strider Ghostbuster [3] identifies hidden files, processes ... by comparing two views of the system.

Another method of importance is using cryptographically signed kernel modules. This method prevents loading unauthorized modules into the kernel space. Greg Kroah-Hartman proposed a method based on this using *RSA* encryption to sign the modules [11].

Placing rootkit detection mechanism in the monitored host itself make it more visible, and could be modifiable by advanced rootkits. So the focus moved towards placing it in system other than the monitored host. The next two rootkit detection methods works based on this principle.

### 2.2 Virtualization Based

In virtual machine based rootkit detection, observer modules working in hypervisor mode aides in detection. They are designed based on the assumption that working on a layer higher than kernel helps the monitor module to efficiently track the host OS activities.

First major research in this direction was done by Tal Garfinkel and Mendel Rosenblum [8]. Their Livewire system used Virtual Machine Monitor (VMM) technique for detecting rootkits. Livewire implementation leverages on the isolation, inspection and interposition properties of a VMM. It consists of a policy engine, which interprets system state and events from the VMM interface and decides whether an attack occurred or not.

Petroni et al. [18] introduced a state based control flow integrity (SBCFI) based method for monitoring operating system kernel integrity dynamically. SBCFI is an extension of Control flow integrity where the monitor looks for change in state for detection. Based on the analysis of 25 Linux rootkits Petroni et al. [18] state that 96% of them employ persistent control-flow modifications and their method could detect them without having any false positives. In their approach VMM runs two virtual machines, one for the monitor and one for target.

SecVisor [23] is a hypervisor based mechanism to ensure code integrity for commodity OS kernels. It needs small modification to the host kernel to work. SecVisor prevents kernel code from unauthorized modification and execution, based on user specified approval policy.NICKLE [20] is a VMM based system that prevents unauthorized kernel code execution for unmodified commodity (guest) OSes. NICKLE uses VMM for restricting access to the kernel space. NICKLE works based on the technique of VMM based memory shadowing scheme. NICKLE module lies in the VMM layer and enforces that the guest OS kernel cannot access the shadow memory. At runtime, any instruction executed in the kernel space must be fetched from the shadow memory, which contains authenticated instructions.

VMwatcher [9] uses a technique called guest view casting for rootkit detection, based on systematically reconstructing internal semantic views of a VM from outside. One of the main challenge of moving monitoring out of the target OS is that there is a semantic gap between the view of VM from inside and outside. In VMwatcher guest view casting technique reconstruct the semantic-level view of the VM.

HookSafe [24] provides a hypervisor based system to protect kernel hooks from being attacked by rootkits. Hooking is an efficient technique employed by rootkits to evade detection. Hooking effectively modifies the flow of control and hides the presence of modification. HookSafe loads the target OS as Guest OS and monitors the possible hooks for modification. It provides a hook indirection layer to regulate access to hooks in the kernel. Only write access to hooks needs the control to transferred to the hypervisor.

Baliga et al. introduced Paladin [2] an automatic rootkit detection and contain technique by leveraging the virtual machine technology. File access control and memory access control policies are specified to protect memory areas that are targets of rootkit attack. Attack detection is based on the creation of dependency tree. Dependency tree lists process-file relationships. When access control policy is violated, based on the dependency tree, detection mechanism can stop malicious programs from cause further damages.

KernelGuard [19] offers a VMM based solution for preventing dynamic data rootkits. Dynamic data rootkits are difficult to detect since they didn't cause any change to kernel code. Since they do not execute any new code, they could easily

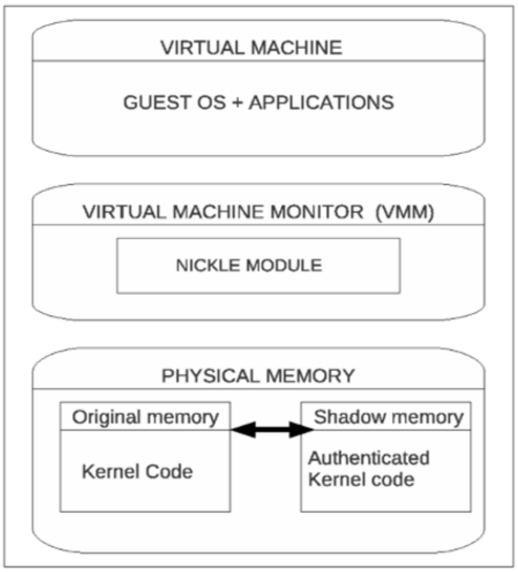

**Fig. 2.** NICKLE Architecture

elude detection efforts. KernelGuard works in Virtual machine environment and makes use of semantic information for validation.

Large number of research initiatives focus on using VMM based mechanism for rootkit detection, as compared to other two approaches. But Bratus, Locasto et al.in their paper [5] questions the viability of using VM technique for rootkit detection. They argue that many modern day applications couldnâĂŹt integrate VM technique as a detection mechanism. Moreover the VM complexity is increasing day by day and managing the VM becomes a major challenge.

## 2.3 External Observer Based

This method mainly use external observation mechanisms for detecting kernel level rootkits. Earlier work utilized Trusted Platform Modules (TPM) for defeating rootkits. Method proposed by Reiner Sailer et al [22]. need modification to the running kernel. It first takes measurements of uncompromised target. TPM is used to collect a sequence of hashes over target code. Validation is done on the basis of this hashed copy.

Petroni et al. [17] developed Copilot, a coprocessor based mechanism to check the integrity of kernel code. It uses a PCI-card to monitor the memory of the host system. Copilot first creates "known good hashes" of host kernels text, text of loaded LKM, and the contents of some of the host kernel's critical data structures. It then periodically checks for any changes.

Fig.3 depicts the copilot testbed architecture [17]. PCI add-in card contains the host monitor. Admin station is the machine from which an administrator can interact with the Copilot monitor.

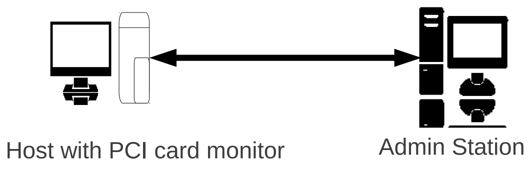

**Fig. 3.** Copilot Testbed Architecture

Baliga, Ganapathy et al. [1] introduced a similar mechanism for detecting kernel level rootkits. Their detection tool Gibraltar focused on detecting both control and non-control data modification. Gibraltar uses an external PCI card to obtain information from target system. Gibraltar operates in inference mode and detection mode. During inference phase Gibraltar uses an uncompromised target to infer invariants. In detection mode it checks whether the data structures on the targets kernel satisfy the invariants inferred earlier.

This method couldn't find much interest from research community because of the need for having an external entity for monitoring. Some of these methods could also extended to VMM layer [1].

## 3 Rootkit Profiling

Profiling of rootkits is essential for better understanding their attack strategies and could help to contain the attacks. Profiling helps manual analysis easier for an expert. Riley et al. [21] introduced PoKeR, a profiler based on NICKLE [20]. PoKeR is deployed in scenarios which can tolerate high overheads. PoKeR (Profiler for Kernel Rootkits) is capable of producing rootkit profiles which include the revelation of rootkit hooking behavior, targeted kernel objects, user level impacts. It is also capable of extracting rootkit code.

HookFinder [28] helps to identify the hooking behavior of malicious code without relying on any prior knowledge of hooking mechanisms. HookMap [25] monitors normal kernel execution path to find kernel hooks that could be potentially hijacked for evasion. K-Tracer [13] dynamically analyze kernel code and extract malicious behavior from rootkits. It uses data flow analysis of kernel execution for profiling.

## 4 Conclusion

Rootkits are "Trojan horses" that resides in the operating system. Residing in kernel makes kernel level rootkits difficult to detect. Their self concealment behavior and administrative level privileges makes them the most difficult attack to detect. Researchers are looking for efficient methods, that needs less prior information, low overhead and high accuracy. Based on where the detection module lies, the detection mechanism can be classified into host based, virtualization based and external observer based mechanism. Virtualization based method is the most widely used detection method. But in terms of the area of application it has some limitations. Improved understanding of various detection mechanisms will help the researchers in not only improving the existing detection mechanisms but also to look for other efficient solutions.

## References

1. Baliga, A., Ganapathy, V., Iftode, L.: Detecting kernel-level rootkits using data structure invariants. IEEE Transactions on Dependable and Secure Computing 99 (PrePrints) (2010)
2. Baliga, A., Iftode, L., Chen, X.: Automated containment of rootkits attacks. Computers and Security 27(7-8), 323–334 (2008), http://www.sciencedirect.com/science/article/B6V8G-4SYCPMR-1/2/0072c2079956faf503f8f683847fd3a2
3. Beck, D., Vo, B., Verbowski, C.: Detecting stealth software with strider ghostbuster. In: Proceedings of the 2005 International Conference on Dependable Systems and Networks, DSN 2005, pp. 368–377. IEEE Computer Society, Washington, DC, USA (2005), http://dx.doi.org/10.1109/DSN.2005.39
4. Bickford, J., O'Hare, R., Baliga, A., Ganapathy, V., Iftode, L.: Rootkits on smart phones: attacks, implications and opportunities. In: Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, pp. 49–54. ACM, New York (2010)
5. Bratus, S., Locasto, M.E., Ramaswamy, A., Smith, S.W.: Vm-based security overkill: a lament for applied systems security research. In: Proceedings of the 2010 Workshop on New Security Paradigms, NSPW 2010, pp. 51–60. ACM, New York (2010), http://doi.acm.org/10.1145/1900546.1900554
6. Bunten, A.: Unix and linux based rootkits techniques and countermeasures (2004)
7. Clemens, J.: Intrusion Detection FAQ: Knark: Linux Kernel Subversion (2001)
8. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. Network and Distributed Systems Security Symposium, vol. 1, pp. 253–285. Citeseer (2003)
9. Jiang, X., Wang, X., Xu, D.: Stealthy malware detection through vmm-based out of the box semantic view reconstruction, pp. 128–138 (2007), http://doi.acm.org/10.1145/1315245.1315262
10. Kim, G.H., Spafford, E.H.: The design and implementation of tripwire: a file system integrity checker. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security, CCS 1994, pp. 18–29. ACM, New York (1994), http://doi.acm.org/10.1145/191177.191183
11. Kroah-Hartman, G.: Signed kernel modules. Linux Journal (2004)
12. Kruegel, C., Robertson, W., Vigna, G.: Detecting kernel-level rootkits through binary analysis. In: Computer Security Applications Conference, Annual, pp. 91–100 (2004)
13. Lanzi, A., Sharif, M., Lee, W.: K-tracer: A system for extracting kernel malware behavior. In: Proceedings of the 16th Annual Network and Distributed System Security Symposium (2009)
14. Levine, J., Grizzard, J., Owen, H.: A methodology to detect and characterize kernel level rootkit exploits involving redirection of the system call table. In: Proceedings of Second IEEE International Information Assurance Workshop, pp. 107–125. IEEE, Los Alamitos (2005)

15. Levine, J.G., Grizzard, J.B., Owen, H.L.: Detecting and categorizing kernel-level rootkits to aid future detection. IEEE Security and Privacy 4, 24 (2006), http://portal.acm.org/citation.cfm?id=1115691.1115761
16. Lineberry, A.: Malicious Code Injection via/dev/mem. Black Hat Europe (2009), http://www.blackhat.com/presentations/bh-europe-09/Lineberry/BlackHat-Europe-2009-Lineberry-code-injection-via-dev-mem.pdf
17. Petroni Jr., N.L., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot - a coprocessor-based kernel runtime integrity monitor. In: Proceedings of the 13th Conference on USENIX Security Symposium, SSYM 2004, vol. 13, pp. 13–13. USENIX Association, Berkeley (2004), http://portal.acm.org/citation.cfm?id=1251375.1251388
18. Petroni Jr., N.L., Hicks, M.: Automated detection of persistent kernel control-flow attacks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 103–115. ACM, New York (2007), http://doi.acm.org/10.1145/1315245.1315260
19. Rhee, J., Riley, R., Xu, D., Jiang, X.: Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring. In: International Conference on Availability, Reliability and Security, pp. 74–81 (2009)
20. Riley, R., Jiang, X., Xu, D.: Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 1–20. Springer, Heidelberg (2008)
21. Riley, R., Jiang, X., Xu, D.: Multi-aspect profiling of kernel rootkit behavior. In: Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys 2009, pp. 47–60. ACM, New York (2009), http://doi.acm.org/10.1145/1519065.1519072
22. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium, SSYM 2004, vol. 13, p. 16. USENIX Association, Berkeley (2004), http://portal.acm.org/citation.cfm?id=1251375.1251391
23. Seshadri, A., Luk, M., Qu, N., Perrig, A.: Secvisor: A tiny hypervisor to provide lifetimekernel code integrity for commodity OSes. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles (21st SOSP 2007), pp. 335–350. ACM SIGOPS, Stevenson (October 2007)
24. Wang, Z., Jiang, X., Cui, W., Ning, P.: Countering kernel rootkits with lightweight hook protection. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 545–554. ACM, New York (2009), http://doi.acm.org/10.1145/1653662.1653728
25. Wang, Z., Jiang, X., Cui, W., Wang, X.: Countering persistent kernel rootkits through systematic hook discovery. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 21–38. Springer, Heidelberg (2008)
26. Wichmann, R.: A comparison of several host/file integrity monitoring programs (December 29, 2009), http://www.la-samhna.de/library/scanners.html
27. Yan, Q., Li, Y., Li, T., Deng, R.: Insights into Malware Detection and Prevention on Mobile Phones. In: Ślęzak, D., Kim, T.-h., Fang, W.-C., Arnett, K.P. (eds.) SecTech 2009. CCIS, vol. 58, pp. 242–249. Springer, Heidelberg (2009)
28. Yin, H., Liang, Z., Song, D.: Hookfinder: Identifying and understanding malware hooking behaviors. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008). Citeseer (2008)