Rootkits on smart phones: Attacks, implications and opportunities

Article · February 2010

DOI: 10.1145/1734583.1734596

5 authors, including:

Jeffrey Earl Bickford
AT&T
18 PUBLICATIONS **203** CITATIONS
SEE PROFILE

Vinod Ganapathy
Rutgers, The State University of New Jersey
51 PUBLICATIONS **1,644** CITATIONS
SEE PROFILE

Liviu Iftode
Rutgers, The State University of New Jersey
209 PUBLICATIONS **8,026** CITATIONS
SEE PROFILE

Rootkits on Smart Phones: Attacks, Implications and Opportunities*

Jeffrey Bickford Ryan O'Hare† Arati Baliga‡ Vinod Ganapathy Liviu Iftode
Department of Computer Science, Rutgers University

ABSTRACT

Smart phones are increasingly being equipped with operating systems that compare in complexity with those on desktop computers. This trend makes smart phone operating systems vulnerable to many of the same threats as desktop operating systems.

In this paper, we focus on the threat posed by smart phone rootkits. Rootkits are malware that stealthily modify operating system code and data to achieve malicious goals, and have long been a problem for desktops. We use three example rootkits to show that smart phones are just as vulnerable to rootkits as desktop operating systems. However, the ubiquity of smart phones and the unique interfaces that they expose, such as voice, GPS and battery, make the social consequences of rootkits particularly devastating. We conclude the paper by identifying the challenges that need to be addressed to effectively detect rootkits on smart phones.

Categories and Subject Descriptors:
C.2.0 [**Computer-communication networks**]: General—*Security and Protection*;
D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*

General Terms: Experimentation, Security

Keywords: rootkits, smart phones

1. INTRODUCTION

Over the last several years, the decreasing cost of advanced computing and communication hardware has allowed mobile phones to evolve into general-purpose computing platforms. Over 115 million such *smart phones* were sold worldwide in 2007 [7]. These phones are equipped with a rich set of hardware interfaces and application programs that let users interact better with the cyber and the physical worlds. For example, smart phones are often pre-installed with a number of applications, including clients for location-based services and general-purpose web browsers. These applications utilize hardware features such as GPS and enhanced network access via 3G or Wimax. To support the increasing complexity of software and hardware on smart phones, smart phone operating systems have similarly evolved. Modern smart phones typically run complex operating systems, such as Linux, Windows Mobile, Android and Symbian OS, which comprise tens of millions of lines of code.

*This work was supported in part by NSF grants CNS-0831268, CNS-0915394, CNS-0931992, a grant from the US Army-RDECOM CERDEC, and a Rutgers University Computing Coordination Council grant.

†Current affiliation: BAE Systems, Wayne, NJ.

‡Current affliation: WINLAB, Rutgers University.

HotMobile'10, February 22–23, 2010, Annapolis, Maryland, USA.

The increasing complexity of smart phones has also increased their vulnerability to attacks. Recent years have witnessed the emergence of *mobile malware*, which are viruses and worms that infect smart phones. For instance, F-Secure reported an almost 400% increase in mobile malware within a two year period from 2005-2007 [17]. Mobile malware typically use many of the same attack vectors as do malware for traditional computing infrastructures, but often spread via interfaces and services unique to smart phones, including Bluetooth, SMS and MMS. The Cabir worm, for instance, exploited a vulnerability in the Bluetooth interface and replicated itself to other Bluetooth enabled phones. Recent research has also explored the security implications of connecting smart phones to the Internet: Enck *et al.* [14] demonstrated attacks that could compromise open interfaces for SMS (*e.g.,* web sites that allow users to send SMS messages) to cripple large portions of a cellular network.

In this paper, we show that smart phones are just as vulnerable as desktop operating systems to *kernel-level rootkits* (or simply, *rootkits*). Rootkits are malware that achieve their malicious goals by infecting the operating system. For example, rootkits may be used to hide malicious user space files and processes, install Trojan horses, and disable firewalls and virus scanners. Rootkits can achieve their malicious goals stealthily because they affect the operating system, which is typically considered the trusted computing base. Consequently, they can retain longer term control over infected machines. Stealth techniques adopted by rootkits have become popular among malware writers, with a study by MacAfee reporting a nearly 600% increase in rootkits in the three-year period from 2004-2006 [9].

The fact that smart phones are vulnerable to rootkits should not be particularly surprising. However, smart phone rootkits can access a number of unique interfaces and information that are not normally available on desktop computers. These include GPS, the battery, and voice and messaging. As we demonstrate via three attacks (in Section 3), such interfaces provide rootkits with new attack vectors to compromise privacy and security of end users. Moreover, phones are personal devices and contain numerous applications that store sensitive information about their users. For example, smart phones contain contact information and SMS conversations for people that a user normally converses with. Such information is potentially of value to attackers, and is often not available on desktop machines. Similarly, employees with company phones could potentially store confidential commercial information in emails located on their smart phones.

With 3G and 4G access becoming increasingly ubiquitous, smart phone users have easy access to the Internet and email. As a result, there is a sharp increase in the number of services and applications available for smart phones. In 2008, Bank of America reported that they service over four million mobile banking sessions every month. They also reported that there are over one million unique users using their mobile banking services[20]. Online retailers such as Amazon.com provide mobile websites and mobile applications, so that users can purchase items from their smart phones. Smart phone rootkits can therefore compromise privacy and security in novel ways, while also being extremely difficult to detect.

Detecting and recovering from rootkits is challenging, even on desktop systems. Because rootkits affect the operating system, any

detection mechanism must operate outside the operating system, typically on specialized hardware (*e.g.,* a co-processor [18]) or in a virtual-machine monitor [15]. Although there have been recent efforts to deploy virtual machines on smart phones [8], such support is not widely available yet. Even so, existing rootkit detection techniques [10, 18, 21], which have primarily been developed for desktop systems, employ heavyweight mechanisms that require periodic scans of kernel memory snapshots. Such techniques will likely place substantial energy demands if used on smart phones. We conclude the paper by discussing the challenges involved in detecting rootkits on smart phones.

2. BACKGROUND

Malware on Smart Phones

Smart phones are an attractive target for attackers, both in the kinds of attacks that are possible and in the social implications of these attacks. Smart phones have access to both telephony and the Internet. As a consequence, malware that can attack a smart phone has the unique advantage of being able to affect the cell phone infrastructure as well as other phones on the cellular network. These abilities have driven malware authors to focus on smart phones, with a recent report from MacAfee [3] stating that nearly 14% of mobile users worldwide have been directly infected or have known someone infected by mobile malware. Nearly 72% of the users surveyed in the MacAfee study expressed concerns regarding the safety of using emerging mobile services and more than 86% were concerned about receiving inappropriate or unsolicited content, fraudulent bill increases, or information loss and theft.

The pervasive nature of smart phones and a large, unsophisticated user base also make smart phones particularly attractive to attackers. Important personal and financial information can likely be compromised by mobile malware because phone usage revolves largely around day-to-day user activities. For example, smart phones are increasingly being used for text messaging, email, storing personal data, including financial data, pictures and videos. Espionage of such voice conversations is likely to have serious social implications. As a second example, users typically tend to carry their smart phones (and keep them powered on) wherever they go; therefore, an attack that compromises the GPS subsystem will compromise privacy of the victim's location.

Traditional threats to desktop systems, such as worms and viruses, have already begun infecting mobile platforms. According to F-Secure [1], there are already more than 400 mobile viruses in circulation. Several existing mobile malware result in simple annoyances. For example, the Skull.D virus locks the phone and flashes an image of a skull and crossbones on the screen. However, others are more dangerous and can cause financial damage to the user by sending text messages to "premium" numbers. Malware such as spyware and Trojan horses have also started affecting smart phones.

The threats posed by mobile malware can readily be countered using many of the same tools available for desktop machines. For example, an antivirus tool equipped with an appropriate virus signature database can detect the presence of viruses on a smart phone. As antivirus tools begin to get deployed on mobile platforms, we envisage that attackers will also move toward using stealth techniques to maintain long-term control over infected smart phones by maliciously modifying smart phone operating systems.

Rootkits on Desktops

The term "rootkit" originally referred to a toolkit of techniques developed by attackers to conceal the presence of malicious software on a compromised system. During infection, rootkits typically require privileged access (*e.g.,* root privileges) to infect the operating system. Even on operating systems that do not run applications with root privileges, an attacker may exploit vulnerabilities in application programs, such as web browsers (*e.g.,* drive-by-download attacks) and the operating system, to obtain elevated privileges to install rootkits.

Attack	LOC	Size of kernel module
GSM	116	92.8 KB
GPS	428	101.7 KB
Battery	134	87.2 KB

Figure 1: Lines of code and size of the kernel modules that implement each of the three attacks.

Rootkits typically infect the system by installing themselves as kernel modules, which are loaded each time the operating system is booted. However, this approach leaves a disk footprint, *i.e.,* the kernel module containing the rootkit, thereby exposing the rootkit to antivirus tools. Sophisticated rootkits avoid this problem by directly modifying data in kernel memory and do not leave a disk footprint. Although such rootkits only persist until the system is rebooted, they are effective on desktop computers, which are often not rebooted for several days or months at a time.

Once infected, a rootkit can serve as the stepping stone for several future attacks. For example, rootkits are commonly used to conceal keyloggers, which steal sensitive user data, such as passwords and credit card numbers, by silently logging keystrokes. They might also install backdoor programs on the system, which allow a remote attacker to gain entry into the system in the future. Rootkits can also perform other stealthy activities, such as disabling the firewall/antivirus tools or affecting the output quality of the system's pseudo random number generator, thereby causing the generation of weak cryptographic keys [12]. None of these activities are directly visible to the user because the rootkit conceals its presence. Their stealthy nature enables rootkits to stay undetected, and therefore retain long-term control over infected systems.

3. ROOTKITS ON SMART PHONES

The increasing complexity of smart phone operating systems makes them as vulnerable to rootkits as desktop operating systems are. However, these rootkits can potentially exploit interfaces and services unique to smart phones to compromise security in novel ways. In this section, we present three proof-of-concept rootkits that we developed to illustrate the threat that they pose to smart phones. They were implemented by the first two authors, with only a basic undergraduate-level knowledge of operating systems. Our test platform was a Neo Freerunner smart phone running the Openmoko Linux distribution [5]. We chose this platform because (a) Linux source code is freely available, thereby allowing us to study and modify its data structures at will; and (b) the Neo Freerunner allows for easy experimentation, *e.g.,* it allows end-users to re-flash the phone with newer versions of the operating system.

All our rootkits were developed as Linux kernel modules (LKM), which we installed into the operating system. However, during a real attack, we expect that these LKMs will be delivered via other mechanisms, *e.g.,* after an attacker has compromised a network-facing application or via a drive-by-download attack. Figure 1 presents the lines of code needed to implement each attack, and the size of the corresponding kernel module. This figure shows the relative ease with which rootkits can be developed. It also shows that the small size of kernel modules allows for easy delivery, even on bandwidth-constrained smart phones.

Although our implementation and discussion in this section are restricted to the Neo Freerunner platform, the attacks are broadly applicable to smart phones running different operating systems. For example, Android is a platform derived from Linux and can sup-

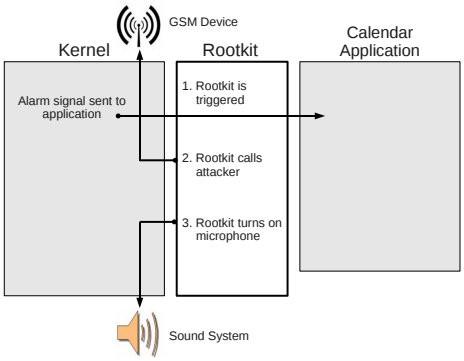

Figure 2: The GSM rootkit intercepts an alarm signal, *e.g.,* a meeting notification, and stealthily dials the attacker, thereby allowing him to snoop on confidential conversations.

port loadable kernel modules; consequently, our proof-of-concept rootkits can potentially be modified to work on the Android platform (and the phones that run it, such as the Droid and the Nexus One). Since the attacks modify OS-specific data structures, they must be re-implemented for other platforms, such as Windows Mobile and Symbian OS; we expect that doing this will be relatively easy. In the following three subsections, we will describe in detail the three rootkits we developed. For each rootkit, we will present the goal of the attack, the attack description and its social impact.

3.1 Spying on Conversations via GSM

Goal. The goal of this attack is to allow a remote attacker to stealthily listen into or record confidential conversations using a victim's rootkit-infected smart phone.

Attack Description. The Freerunner phone is equipped with a GSM radio, which is connected via the serial bus and it is therefore available to applications as a serial device. During normal operation of the phone, user-space applications issue system calls to the kernel requesting services from the GSM device. The GSM device services the request allowing the application to access the telephony functionality provided by the device. GSM devices are controlled through series of commands, called AT (attention) commands, that let the kernel and user-space applications invoke specific GSM functions. For example, GSM devices support AT commands to dial a number, fetch SMS messages, and so on. To maliciously operate the GSM device, *e.g.,* to place a phone call to a remote attacker, the rootkit must therefore issue AT commands from within the kernel.

Most smart phones today contain calendar programs, which notify users when scheduled events occur. Our prototype rootkit operates by intercepting these notifications set by the user. As shown in Figure 2, a notification is displayed by a user-space program to notify the user of an impending meeting. The rootkit intercepts this notification and activates its malicious functionality. The attack code stealthily dials a phone number belonging to a remote attacker, who can then snoop or record confidential conversations of the victim. The phone number dialed by the rootkit can either be hard-coded into the rootkit, or delivered via an SMS message from the attacker, which the rootkit intercepts to obtain the attacker's phone number. Alternatively, the rootkit could be activated when the victim dials a number. The rootkit could then stealthily place a three-way call to the attacker's number, thereby allowing the attacker to record the phone conversation.

- *Triggering the rootkit.* We used a simple alarm clock program to simulate calendar notifications on the Openmoko (we did so because the Openmoko phone does not have any released calendar programs). In an uninfected kernel, when an alarm is signaled, a specific message is delivered via the `write` system call. In our "infected" kernel, the rootkit hooks the system call table and replaces the address of the `write` system call with the address of a malicious `write` function implemented in the rootkit. The goal of the malicious `write` function in our prototype rootkit is to check for the alarm notification in the `write` calls. Once an alarm message is identified, the malicious functionality is triggered.

- *Placing a phone call.* When triggered, our rootkit places a phone call by emulating the functionality of user-space telephony applications. Typically, user-space applications (such as the Qtopia software stack [6], which ships with the Openmoko Linux distribution on the Freerunner phone) make calls by issuing a sequence of system calls to the kernel. Specifically, applications such as Qtopia use `write` system calls to issue AT commands to the GSM device (these commands are supplied as arguments to the `write` system call). The number to be dialed is located in the AT command.

Our prototype rootkit calls the attacker by issuing the same sequence of AT commands from within the kernel. We obtained the sequence of AT calls that must be issued to place a phone call by studying the Qtopia software stack. The AT commands issued by the rootkit activate the telephony subsystem and successfully establish a connection to the attacker's phone. The prototype rootkit must also activate the sound system by turning on the microphone.

Social Impact. Snooping on confidential conversations has severe social impact because most users tend to keep their mobile phones in their proximity and powered-on most of the time. Rootkits operate stealthily, and as a result, end users may not even be aware that their phones are infected. Consequently, an attacker can listen-in on several conversations, which violates user privacy, ranging from those that result in embarrassing social situations to leaks of sensitive information. For example, an attack that records the conversations at a corporate board meeting can potentially compromise corporate trade secrets and business reports to competitors. Similarly, several automated phone-based services often require a user to enter (via voice or key presses) PIN numbers or passwords before routing the call to a human operator; an attacker snooping on such calls may financially benefit from such information.

3.2 Compromising Location Privacy using GPS

Goal. The goal of this attack is to compromise a victim's location privacy by ordering the victim's rootkit-infected smart phone to send to the remote attacker a text message with victim's current location (obtained via GPS).

Attack Description. As with the GSM device, the GPS device is also a serial device. The kernel maintains a list of all serial devices installed on the system. A rootkit can easily locate the GPS device. Every serial device contains a buffer in which the corresponding device stores all outgoing data until it is read by a user-space application. Our prototype rootkit uses this buffer to read information before it is accessed by user-space applications. This allows us to monitor and suppress incoming SMS messages and also query the GPS for location information.

A rootkit that compromises location privacy as described above must implement three mechanisms. First, it must be able to intercept incoming text messages, and determine whether a text message is a query from a remote attacker on the victim's current location. Second, the rootkit must be able to extract location information from the GPS receiver. Last, it must generate a text message with the victim's current location, and send this information to the attacker. An overview of this attack is shown in Figure 3.

Our prototype rootkit intercepts text messages by monitoring and changing data in the GSM device buffer. To monitor the GSM buffer, we hook the kernel's `read` and `write` system calls. This

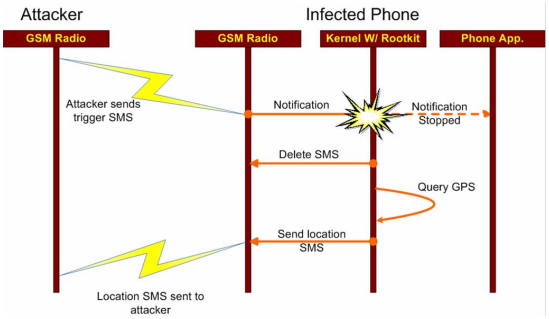

Figure 3: Sequence of steps followed in location privacy attack.

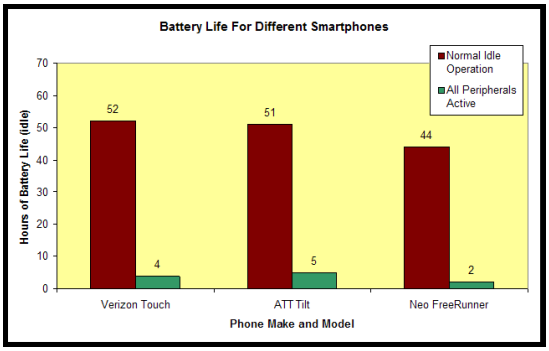

Figure 4: Denial of service via battery exhaustion. This figure shows how the battery life degrades in different phone models when the GPS and Bluetooth devices are powered on.

is achieved by modifying the corresponding entries in the system call table to point to rootkit code. Consequently, the rootkit identifies when user-space applications are accessing the GSM device by checking the file descriptor that is passed into `read` and `write`. When this occurs, our rootkit scans the GSM for certain incoming AT commands.

When an incoming message arrives, the rootkit will check whether this is a query from the attacker. This is done by sending the AT command to read messages. An attacker's message will be a certain phrase or set of words that the rootkit can check for. If the message is a query from the attacker, the rootkit's malicious operation is executed. The rootkit can be written to enable different functionality for different messages.[1] The attacker can also enable/disable the rootkit's malicious functionality via text messages, in effect allowing the attacker to remotely control the rootkit.

Importantly, the rookit, must also suppress the notification of the attacker's message, to ensure that the user does not learn about it. The rootkit deletes the message from the SIM card by sending another AT command to the GSM device.

Once the rootkit intercepts a message to query a user's current location, it attempts to obtain location information from the GPS device. As before, the rootkit can easily obtain location information from the buffer of the GPS device. The rootkit can obtain location information even if the user has disabled the GPS. This is because the rootkit operates in kernel mode, and can therefore enable the device to obtain location information, and disable the device once it has this information. If the user checks to see if the GPS is enabled during this time, it will appear that the GPS device is off.

Having obtained location information, the rootkit constructs a text message and sends the message by sending an AT command to the GSM device. The attacker will now receive a user's current location.

Social Impact. Protecting location privacy is an important problem that has received considerable recent attention in the research community. By compromising the kernel to obtain user location via GPS, this rootkit defeats most existing defenses to protect location privacy. Further, the attack is stealthy. Text messages received from and sent to the attacker are not displayed immediately to the victim. The only visible trace of the attack is the record of text messages sent by the victim's phone, as recorded by the service provider.

3.3 Denial of Service via Battery Exhaustion

Goal. This attack exploits power-intensive smart phone services, such as GPS and Bluetooth, to exhaust the battery on the phone. This rootkit was motivated by and is similar in its intent to a previously proposed attack that stealthily drains a smart phone's battery by exploiting bugs in the MMS interface [22]. However, the key difference is that the rootkit achieves this goal by directly modifying the smart phone's operating system.

Attack Description. The GPS and Bluetooth devices can be toggled on and off by writing a "1" or a "0," respectively, to their corresponding power device files. The rootkit therefore turns on the GPS and Bluetooth devices by writing a "1" to their corresponding power device files. To remain stealthy, the rootkit ensures that the original state of these devices is displayed when a user attempts to view their status. Most users typically turn these devices off when they are not in active use because they are power-intensive.

When a user checks the status of a GPS or Bluetooth device, the user-space application checks the power device file for a "1" or "0". To do this, it calls the `open` system call on the file and then reads it. The rootkit monitors the `open` calls by overwriting kernel function pointer for the `open` system call in the system call table, making it point to rootkit code. When an `open` system call is executed, the rootkit examines if the file being opened corresponds to the power device files of the GPS or Bluetooth devices. If so, it ensures that the original states of the devices are displayed to the user. The rootkit continuously checks the status of these devices; if the devices are turned off by the user, the rootkit turns them back on. Consequently, the devices are always on, except when the user actively queries the status of these devices.

Social Impact. This attack quickly depletes the battery on the smart phone. In our experiments, the rootkit depleted the battery of a fully charged and infected Neo Freerunner phone in approximately two hours (the phone was not in active use for the duration of this experiment). In contrast, the battery life of an uninfected phone running the same services as the infected phone was approximately 44 hours (see Figure 4). We also simulated the effect of such a rootkit on the Verizon Touch and ATT Tilt phones by powering their GPS and Bluetooth devices. In both cases, battery lifetime reduced almost ten-fold. Because users have come to rely on their phones in emergency situations, this attack results in denial of service when a user needs his/her phone the most.

Although our prototype rootkit employs mechanisms to hide itself from an end user, this attack is less stealthy than Attacks 1 and 2. For example, a user with access to other Bluetooth-enabled devices may notice his smart phone is "discoverable," causing him to suspect foul play. Nevertheless, we hypothesize that the vast majority of users will suspect that their phone's battery is defective and replace the phone or its battery.

3.4 Rootkit Delivery and Persistence

To effectively infect a smart phone using the rootkits discussed above, attackers must also develop techniques to deliver rootkits and ensure that their functionality persists on the phone for an ex-

[1]This mechanism is also useful in Attack 1—instead of hard-coding the number that the rootkit must dial, an attacker can transmit the number that the infected phone must dial via a text message.

tended period of time.

Delivery. Rootkits can be delivered to smart phones using many of the same techniques as used for malware delivery on desktop machines. A study by F-Secure showed that nearly 79.8% of mobile phones infections in 2007 were as a result of content downloaded from malicious websites. Bluetooth connections and text messages were among the other major contributors to malware delivery on smart phones. Rootkits can also be delivered via email attachments, spam, illegal content obtained from peer-to-peer applications, or by exploiting vulnerabilities in existing applications.

The Neo Freerunner phone used in our experiments ran the Openmoko Linux distribution, which directly executes applications with root privileges. Therefore, unsafe content downloaded on this phone automatically obtains root privileges. Smart phone operating systems that do not run applications with root privileges can also have vulnerabilities (just as in desktop machines). Such vulnerabilities are not uncommon even in carefully engineered systems. For example, a recent vulnerability in Google's Android platform allowed command-line instructions to execute with root privileges [2]. Rootkits can exploit these vulnerabilities and infect the operating system kernel, *e.g.,* by installing malicious kernel modules.

Persistence. To be effective, rootkits must retain long-term control over infected machines. Rootkits typically achieve this goal by replacing critical operating system modules, such as device drivers, with infected versions. While this approach ensures that the rootkit will get control even if the operating system is rebooted, it also has the disadvantage of leaving a disk footprint, which can be detected by malware scanners.

Rootkits can avoid detection by directly modifying the contents of kernel memory, thereby avoiding a disk footprint. Although such rootkits are disabled when the operating system is rebooted, they can still retain long-term control over server-class machines, which are rarely rebooted. However, this technique is not as effective on smart phones, because phones are powered off more often (or may die because the battery runs out of charge). Consequently, rootkits that directly modify kernel memory can only persist on smart phones for a few days. In such cases, an attacker can re-infect the phone. For example, a rootkit that spreads via Bluetooth can re-infect victims in the vicinity of an infected phone.

However, the social consequences of smart phone rootkits mean that they can seriously affect end-user security even if they are effective only for short periods of time.

4. DETECTING PHONE ROOTKITS

Rootkits vary in the sophistication of the attack techniques that they use. Rootkits that modify system utilities and some kernel modules often leave a disk footprint, and can possibly be detected using user-space malware detection tools. However, these tools rely on the operating system to provide critical services, such as access to files, and rootkits can easily bypass them using more sophisticated techniques. The rootkits in Section 3.1 and Section 3.2, for instance, modify function pointers in memory, and can therefore evade detection tools that check the integrity of system utilities. More sophisticated rootkits operate by modifying arbitrary data structures on the kernel's heap [10, 12]. It is therefore well-accepted that rootkit detection mechanisms must reside outside the control of the operating systems that they monitor.

Rootkit detection tools typically operate by directly accessing and scanning kernel memory (*i.e.,* without the intervention of the operating system being monitored) for rootkits. Prior work has developed two mechanisms to access kernel memory: *hardware support* and *virtual machine monitors* (VMM). While these mechanisms suffice to detect rootkits on server-class machines, several challenges must be overcome to adapt them to smart phones, as discussed below.

Hardware-supported Rootkit Detection

Hardware-assisted rootkit detectors operate by using special purpose hardware to directly access kernel memory via DMA. For example, such rootkit detectors can use secure co-processors [18, 26] or PCI cards [10] to access kernel memory, and ensure the integrity of kernel data structures. In this approach, the machine being monitored is equipped with the above hardware, and is physically connected to another machine, which fetches and scans its memory.

While this approach may be practical for server-class machines, it is not applicable to smart phones. First, commodity smart phones are not currently equipped with special-purpose hardware, such as co-processors and PCI cards with access to kernel memory. Second, because the approach requires the smart phone to be physically connected to a monitor machine, it is not practical for use with mobile devices, such as smart phones. One option would be to trigger rootkit detection when a smart phone is physically connected to a desktop machine, *e.g.,* for charging, via an interface such as USB. However, the USB interface does not allow direct memory access to attached devices, and therefore cannot be used to externally monitor the memory contents of the smart phone. To our knowledge, only the FireWire (IEEE 1394) interface allows access to the contents of memory [19], but commodity smart phones are not commonly equipped with this interface.

An alternative approach is to use trusted hardware on the smart phone to attest the software stack running on the phone. In this approach, a trusted platform module (TPM) chip on the phone takes *integrity measurements* (*e.g.,* SHA-1 hashes) of the software loaded on the phone, and stores these measurements in tamper-resistant hardware. A remote verification authority (*e.g.,* the service provider) could then engage in a challenge-response protocol (*e.g.,* IMA [23]) with the phone, and receive a digitally-signed copy of the integrity measurements from the phone. The verification authority would certify the software stack by verifying the hash values of the software loaded on the phone.

Prior work has explored the use of the TPM in combination with an integrity verification protocol to detect rootkits [23]. It may be practical to adapt this approach to detect rootkits on smart phones, because an increasing number vendors are beginning to deploy TPMs on their products (this chip is called the mobile trust module, or MTM) [24]. Now that the MTM is beginning to be deployed on commercial smart phones, using the TPM detection approach is a possibility. However, three key challenges must be overcome in order to implement integrity verification for smart phones; the first two challenges discussed below also apply to integrity verification on desktop computers.

(**1**) *Reasoning about dynamic data structures.* Current integrity measurement protocols can only measure the integrity of the code and static data (*e.g.,* configuration files) [23]. In particular, these protocols cannot attest the integrity of dynamic data structures in kernel memory. Consequently, they cannot detect rootkits that modify the system call table and other critical kernel data structures. To be effective against the kind of rootkits demonstrated in this paper, integrity measurement protocols must be adapted to additionally attest the integrity of dynamic data structures in kernel memory.

(**2**) *Continuous integrity measurement.* Integrity measurement protocols are vulnerable to time-of-check to time-of-measurement exploits, in which the rootkit installs itself after integrity measurements are taken and the software stack has been attested. To be effective against such rootkits, integrity measurement protocols must be adapted to provide continuous guarantees [25].

(**3**) *Resource efficiency.* Finally, integrity measurement and attestation involves a challenge/response protocol between the smart phone and remote attestation authority. Each execution of the protocol requires the smart phone to transfer integrity measurements to the attestation authority. Although the size of the integrity mea-

surements itself is reasonably small—about 1000 measurements on a typical Linux system [23], which results in the transfer of about 20KB-30KB data between the phone and the attestor—frequent executions of the protocol (*e.g.,* to ensure continuous integrity measurement) may consume both network bandwidth and battery power. To be applicable to smart phones, integrity measurement protocols must therefore be adapted to be resource efficient.

VMM-based Rootkit Detection

Virtualization offers an alternative approach to implement rootkit detection. In this approach, the smart phone's operating system and the monitor execute in separate virtual machines (VM). The monitor queries the VM that runs the phone's operating system and extracts the contents of its memory locations to perform rootkit detection [15]. A number of commercial efforts are currently underway to build virtual machine monitors for smart phones [4, 8, 16], with the goal of allowing users to have multiple personalities on a single physical device. For example, the same phone can be used with multiple accounts and providers, such as a corporate account and a personal account. Cox and Chen [13] also provide examples of several other novel applications that can be enabled by deploying virtualization on smart phones.

Rootkit detection tools can possibly leverage these virtual machine monitors to isolate themselves from the smart phone's operating system. However, most existing rootkit detection tools [10, 11, 18, 21] operate by periodically fetching and scanning kernel memory snapshots of the operating system being monitored. Such algorithms are CPU intensive and can potentially drain the battery of the phone. For example, the Gibraltar rootkit detection system [10] can detect sophisticated rootkits that operate by modifying arbitrary kernel data structures. However, it operates by periodically fetching memory pages from the monitored system, reconstructing data structures, and checking these data structures against integrity specifications, each of which is a CPU-intensive operation. Gibraltar can potentially be optimized for use on a smart phone by reducing the frequency at which it scans kernel memory for rootkits, *e.g.,* by enabling rootkit detection only when the phone is being charged. However, doing so introduces a tradeoff between security and energy-efficiency. One way to address this tradeoff would be to adapt Gibraltar to selectively fetch memory pages to be analyzed, *e.g.,* only pages that were recently modified. To develop a VMM-based rootkit detector, a smart phone will need to support the installation of a VMM. Currently, there is no platform that supports this. Further research is therefore needed to make rootkit detection more-efficient and practical for use on virtualized smart phones.

5. SUMMARY

Rootkits evade detection by compromising the operating system, thereby allowing them to defeat user-space detection tools and operate stealthily for extended periods of time. This paper demonstrated that kernel-level rootkits can exploit smart phone operating systems, often with serious social consequences. The popularity of the mobile platform has already attracted attackers, who have increasingly begun to develop and deploy viruses and worms that target these platforms. As these threats gain notoriety, so will the power of tools to detect these threats. We believe that this trend, combined with the increasing complexity of operating systems on modern smart phones, will push attackers to employing rootkits to achieve their malicious goals. Currently, there is no available technique to detect rootkits on smart phones. We therefore conclude with a call for research on tools and techniques to effectively and efficiently detect rootkits on smart phones.

References

[1] F-secure warns of mobile malware growth. www.vnunet.com/vnunet/news/2230481/f-secure-launches-mobile.

[2] Google fixes android root-access flaw. www.zdnetasia.com/news/security/0,39044215,62048148,00.htm.

[3] Mcafee mobile security report 2008. www.mcafee.com/us/research/mobile_security_report_2008.html.

[4] OKL4 embedded hypervisor: Open kernel labs. www.ok-labs.com/.

[5] Openmoko Neo FreeRunner. wiki.openmoko.org/wiki/Neo_FreeRunner.

[6] Qtopia software stack (Qtextended.org). qtopia.net.

[7] Smartphones will soon turn computing on its head. news.cnet.com/8301-13579_3-9906697-37.html.

[8] VMware mobile virtualization platform. www.vmware.com/technology/mobile/.

[9] Rootkits, part 1 of 3: A growing threat, April 2006. MacAfee AVERT Labs Whitepaper.

[10] A. Baliga, V. Ganapathy, and L. Iftode. Automatic inference and enforcement of kernel data structure invariants. In *Proc. Annual Computer Security and Applications Conference*, 2008.

[11] A. Baliga, L. Iftode, and X. Chen. Automated containment of rootkit attacks. *Computers & Security*, 27(7-8):323 – 334, 2008.

[12] A. Baliga, P. Kamat, and L. Iftode. Lurking in the shadows: Identifying systemic threats to kernel data. In *Proc. 2007 IEEE Symposium on Security and Privacy*, 2007.

[13] L. Cox and P. M. Chen. Pocket hypervisors: Challenges and opportunities. In *Proc. HotMobile*, 2007.

[14] W. Enck, P. Traynor, P. Mcdaniel, and T. La Porta. Exploiting open functionality in sms-capable cellular networks. In *Proc. ACM Conference on Computer and Communication Security*, 2005.

[15] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, 2003.

[16] J. Hwang, S. Suh, S. Heo, C. Park, J. Ryu, S. Park, and C. Kim. Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In *IEEE CCNC*, 2008.

[17] M. Hypponen. The state of cell phone malware in 2007. www.usenix.org/events/sec07/tech/hypponen.pdf.

[18] N. L. Petroni Jr., T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proc. USENIX Security Symposium*, 2004.

[19] A.îĂĂ Martin. FireWire memory dump of a Windows XP computer: A forensic approach, 2007. whitepapers.zdnet.com/abstract.aspx?docid=38780.

[20] Bank of America Mobile Banking. One millionth mobile banker logs on to Bank of America. tinyurl.com/yb72f4e.

[21] N. L. Petroni and M. Hicks. Automated detection of persistent kernel control-flow attacks. In *Proc. ACM Conference on Computer and Communications Security*, 2007.

[22] R. Racic, D. Ma, , and H. Chen. Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phone's Battery. In *Proc. 2nd International Conference on Security and Privacy in Communication Networks*, August 2006.

[23] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proc. USENIX Security Symposium*, August 2004.

[24] TCG. Trusted computing group: Mobile trusted computing platform. https://www.trustedcomputinggroup.org/groups/mobile.

[25] G. Xu, C. Borcea, and L. Iftode. Satem: Trusted service code across transactions. In *25th Symposium on Reliable Distributed Systems*, Oct 2006.

[26] X. Zhang, L. van Doorn, T. Jaeger, R. Perez, and R. Sailer. Secure coprocessor-based intrusion detection. In *Proc. 10th workshop on ACM SIGOPS European workshop: beyond the PC*, 2002.

All URLs were last verified on January 10, 2010.