On Rootkit and Malware Detection in Smartphones

Bryan Dixon
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309-0430, USA
bryan.dixon@cs.colorado.edu

Shivakant Mishra
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309-0430, USA
mishras@cs.colorado.edu

Abstract

Dramatic increase in smartphone sales and third-party applications that users can download has significantly increased the possibility of rootkits and malware targeted for smartphones. This paper discusses the current state of research in detection and mitigation of propagation of malicious code, such as viruses, malware and even rootkits in smartphones. A new strategy is introduced that offers a solution to efficiently detect the presence of rootkits and malware in smartphones. A preliminary prototype based on this strategy has been built for the android dev phone. The paper discusses the design, implementation and preliminary performance of this prototype.

1. Introduction

With an uptake in the smartphone market, there has been an emergence of viruses, malware, and even rootkits to gain access to smartphones. These security threats are troublesome due to the large amount of personal data that smartphone users store in their phones as well as confidential transactions that these users perform using voice and/or data interface of their smartphones. Additionally this malicious code leads to both a dramatic drain in the battery, as well as creating an abnormal load on the cellular network. In particular, with the emergence of open-source smartphones platforms such as android and third-party applications that users can download on their phones, the threat of malware and rootkits targeted for smartphones has significantly increased. While these threats have existed for PCs and desktops for a very long time, solutions developed for PCs are not effective in the smartphones environment for a variety of reasons, including resource constraints, power constraints, mobility and device functionality. As a result, investigation of malicious code on smartphones, as well as how this malicious code could be detected, prevented from spreading, or its impact could be mitigated is an important research area.

2. Related Work

There are varied number of infection vectors that viruses, malware, and root kits targeted for smartphones can make use of to propagate themselves, which makes it harder to stop them at the network level. Current approaches to address the problem of malicious code on smartphones can be categorized into three categories. Approaches in the first category rely on monitoring the volume of traffic to determine if a phone or a set of phones is possibly infected. They attempt to mitigate or slow the spread of malicious code by limiting the data rate either globally [5] or in the local vicinity [2]. Approaches in the second category provide system-level defenses to reduce the ability of malicious code to propagate or gain access to a smartphone. One way to do this is to challenge the malicious code, e.g. by utilizing graphical turing test via a visual CAPTCHA [6]. Another approach is to develop sophisticated admission control mechanisms to make it difficult for a malicious code to gain access to a smartphone by making use of trusted hardware.

3. Our Approach

Our observation is that any detection/prevention approach that relies on running on the smartphones is fundamentally limited by the resource constraints of the phones. Though running malicious code detection software similar to that of a PC on the phone may be effective, running such code would need it to run nearly constantly in the background limiting the battery life, which isn't an effective solution since battery life is usually the most important feature of a smartphone. This leads to this approach not being utilized since this is an undesired behavior on a phone, even if it makes it more secure. This provides a clear advantage to adversary who can exploit these limitations of a smartphone to design malicious code, while the detection/prevention mechanisms are limited or non-existent because of these constraints.

978-1-4244-7730-2/10/$26.00 ©2010 IEEE

In this position paper, we propose that a promising approach is to look for a solution that resides outside the smartphones. The key observation is that users often connect their smartphones to a PC or desktop for information synchronization. Based on this observation, we are exploring strategies that run detection/prevention software on a PC/desktop, and look for malicious code on smartphones via a USB or other such interface.

This strategy requires access to the phone's filesystem and a mechanism to manipulate and navigate the files from the computer. This will allow a PC to scan some or all files to detect malicious code. In the basic mechanism, we download and scan all files of the phone on a PC. This obviously takes a large amount of time. The basic function of our technique involves creating hashes of all files and storing them on the PC when the phone is first connected to the PC. Subsequently, whenever the phone is connected, the phone computes the hashes of all files and sends them to the PC. The PC then compares these hashes with the hashes stored on the PC, and download and scans only those files that have changed. Assuming that only a few files change in a typical usage scenario, this technique reduces the scanning time significantly. This technique however has a weakness in the face of a relatively stronger adversary, such as a rootkit. Such an adversary can store the hash of a file before modifying the file and then report this stored hash to the PC when connected. We have considered that using a keyed hashing mechanism requiring a key from the PC will make it significantly harder for such an adversary to succeed. Additionally, with the limited computational power of a phone, time required to calculate and return the expected hash value in such a scenario would be far greater than the expected delay of such a computation.

We have implemented this strategy on Google's Android platform. We used a product called BusyBox that provides the tools needed to fully list the directories along with all attributes. BusyBox also provides *cksum* command that is used to hash the files. When it was run we could navigate and hash the entire phone in about 10 minutes, compared to over 50 minutes to download all the files from the phone, many of which failed to download. Our observation is very few files change regularly so the time to compute which files have changed and only download those should lead to a great reduction in the time a phone is required to be connected to a computer.

A significant advantage of using our strategy is in detecting the presence of rootkits on smartphones. Rootkits compromise and gain long term control over an infected machine and are the worst kind of the malicious code in that they can expose all functionality of the core system to a malicious user. With this level of access and control, a malicious user could make use of the rootkit to eavesdrop on conversations, emails and messaging services [1, 3]. Since a rootkit has low level access and can virtually hide itself from detection from any tools that run on the device itself, most malicious code detection tools are rendered useless.

The techniques that we have proposed so far are also vulnerable to rootkits. A rootkit could hide itself by pretending to be the operating system. It could keep copies of original files before modification, so as to respond correctly to hash queries. To address this, we are currently exploring challenge-response mechanisms to detect the presence of rootkits. The simplest mechanism is to look for variations in expected run times of function calls or some complex computation on the phone. If a rootkit existed, it would likely take longer than normal to compute and return the results. The assumption here is that a rootkit would have to interpret commands and what the desired result should be versus just computing and returning the result. Another possible mechanism is to determine the amount of available memory in the smartphone. In this mechanism, the PC maintains an estimate of how much memory should be free on a smartphone. When a phone is connected, the PC can present a challenge to store large amount of test data that can fit in the estimated free memory space. The PC then reads the same test data to ensure that the data is indeed successfully stored. The assumption here is that a rootkit will typically occupy a large amount of memory and won't be able to respond to such a challenge. Coprocessor-based mechanisms to detect the presence of rootkits in PCs and desktops have been developed [4]. We plan to borrow ideas from these mechanisms to incorporate in our prototype.

4. Future Work

We have implemented a preliminary prototype so far. Future work entails adding more sophisticated functionality to detect a root kit. We will evaluate our prototype with different types of malware and rootkits that are currently available. We will evaluate the performance of our prototype in terms of response time and power consumption.

References

[1] J. Bickford et al. Rootkits on Smart Phones: Attacks, Implications and Opportunities. In *HotMobile 2010*.

[2] A. Bose. *Propagation, Detection and Containment of Mobile Malware*. PhD thesis, The University of Michigan, 2008.

[3] A. Kushnerov. Smart phone under threat of attacks. http://www.theticker.org/about/2.8220/smart-phone-under-threat-of-attacks-1.2174454, March 2010.

[4] N. Petroni et al. Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. In *2004 USENIX Security Symposium*.

[5] M. Williamson et al. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *2002 CSAC*.

[6] L. Xie et al. Designing System-level Defenses against Cellphone Malware. In *SRDS 2009*.