Detecting Energy-Greedy Anomalies and Mobile Malware Variants

Hahnsang Kim, Joshua Smith, Kang G. Shin
The University of Michigan
Ann Arbor, MI 48109-2121, USA
{hahnsang, smjoshua, kgshin}@eecs.umich.edu

ABSTRACT

Mobile users of computation and communication services have been rapidly adopting battery-powered mobile handhelds, such as PocketPCs and SmartPhones, for their work. However, the limited battery-lifetime of these devices restricts their portability and applicability, and this weakness can be exacerbated by mobile malware targeting depletion of battery energy. Such malware are usually difficult to detect and prevent, and frequent outbreaks of new malware variants also reduce the effectiveness of commonly-seen signature-based detection. To alleviate these problems, we propose a power-aware malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats. The framework is composed of (1) a *power monitor* which collects power samples and builds a power consumption history from the collected samples, and (2) a *data analyzer* which generates a power signature from the constructed history. To generate a power signature, simple and effective noise-filtering and data-compression are applied, thus reducing the detection overhead. Similarities between power signatures are measured by the χ^2-distance, reducing both false-positive and false-negative detection rates. According to our experimental results on an HP iPAQ running a Windows Mobile OS, the proposed framework achieves significant (up to 95%) storage-savings without losing the detection accuracy, and a 99% true-positive rate in classifying mobile malware.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: System Architectures; D.2.11 [**Software Engineering**]: Software Architecture; I.5.4 [**Pattern Recognition**]: Applications

General Terms

Security

Keywords

Power consumption history, power signature

MobiSys'08, June 17–20, 2008, Breckenridge, Colorado, USA.

1. INTRODUCTION

In recent years, the worldwide market for handhelds has grown dramatically. For instance, in 2006 the market grew by 21% over 2005, selling 800 million mobile phones, and in 2007 1.1 billion mobile phones were estimated to have been sold [1]. Because of continued miniaturization, ubiquitous communication, and increasing computation power, mobile handheld users can now perform many online tasks, including web browsing, document editing, multimedia streaming, and Internet banking, to name a few. At the same time, the growing use of mobile handhelds for everyday life and business has been attracting the attention of malware writers, whose aim is to compromise data confidentiality, integrity, and the ability to use handheld services. For instance, SymbOS.Cabir [20], the first proof-of-concept mobile worm, which appeared in June 2004, was written for the Symbian OS and used a novel propagation vector (e.g., via Bluetooth or SMS). Although Cabir was designed solely to demonstrate the feasibility of malicious code for mobile devices, the publication of Cabir source code triggered a worldwide outbreak of many variants, infecting Bluetooth-enabled mobile phones.

The limited battery-lifetime for mobile handhelds is an Achilles' heel for the portability and the ubiquitous use of mobile devices. This limitation exists because not only has battery technology not kept up with Moore's Law, but mobile devices and software running thereon also demand more power for a longer period than the battery can deliver [48]. At the same time, while most malicious code attacks on handhelds aim to damage software resources such as infecting files and stealing privacy information [5], intentional abuse of hardware resources (e.g., CPU, memory, battery power) has become an important, increasing threat [14, 41]. In particular, malware targeting the burning/depletion of battery power are extremely difficult to detect and prevent, mainly because users are usually unable to recognize this type of anomaly on their handhelds and the battery can be deliberately and rapidly drained in a number of different ways (e.g., DoS attacks [30, 41], or the installation of animated GIFs [14]). Despite these problems, only limited research [11, 30, 41] has focused on the detection and prevention of battery-draining attacks on handhelds, including how to cope with a wider variety of attacks in [13, 40].

The most commonly-used technique for malware mitigation for antivirus and antispyware is signature-based analysis. Signatures are created using static information (e.g., file name and a code value), thus being vulnerable to simple obfuscation, polymorphism [31, 33], and packing tech-

niques [7]. Signature-based detection that requires a new signature for every single malware variant is not well suited for mobile handhelds mainly because handhelds have much less resources (e.g., CPU, memory, and battery power) than their desktop counterparts. Moreover, even 'old' malware can harm new handhelds unless their system has been properly patched in a timely fashion. In practice, patching is rarely an option for handhelds as their operating systems are usually inaccessible to others (than the manufacturers). Unlike signature-based detection, anomaly-based detection compares definitions of the activity considered normal in a profile against the observed events to identify significant deviations. The profile describes the normal behavior (e.g., users, hosts, applications, or network connections) [22, 56]. One common problem with anomaly-based detection, however, is inadvertent inclusion of a malicious activity as part of the profile produces many false-negatives (failure to identify malicious activities). Similarly, behavioral detection [18] is based on behavioral signatures that describe aspects of any particular worm's behavior such as sending similar data from one machine to another, the propagation pattern, and the change of a server into a client, thus representing a generic worm propagation model [18]. These behavioral signatures that are not sufficiently complex to reflect real-world computing activities can cause many false-positives (incorrect identification of a benign activity as malicious). Also, the propagation of mobile malware via non-traditional exploit vectors such as SMS and Bluetooth [10, 19, 51], in conjunction with user mobility, renders network-behavioral signatures almost ineffective.

There are two main challenges in developing a malware-detection framework for handhelds. First, a detection framework should be able to detect diverse types of malware, especially including energy-greedy (malicious) applications and malware variants, keeping both false-negatives and false-positives below a certain acceptable threshold. Second, unlike resource-rich PCs, a detection framework on battery-powered handhelds should not consume too much of the device resources, including CPU, memory, and battery power; the overhead for executing the detection framework should be kept to a minimum.

In this paper, we propose a novel power-aware malware detection framework to monitor, detect, and analyze previously unknown and energy-greedy threats. Our malware-detection framework is composed of a *power monitor* and a *data analyzer*. The former collects power samples and builds a power consumption history with the collected samples, and the latter generates a *power signature* from the power consumption history. The data analyzer then detects an anomaly by comparing the generated power signature with those in a database. We evaluate detection performance using a custom worm emulator that we designed for this purpose.

The key contributions of this paper are two-fold, addressing the two challenges listed above. First, assuming that one application is executed at a time on the handheld, a power consumption history is recorded/built and then converted into a power signature which is an abstraction of the underlying application behavior. Considering the fact that a new malware variant is usually created by adding new functionality to existing malware or modifying obsolete modules with fresh ones [31], this abstraction is effective for detecting not only previously-unknown malware variants that share a common behavior exhibited by previously-known malware, but also energy-greedy anomalies that most current solutions fail to detect. A χ^2-distance is calculated to gauge (dis)similarity to a database of power signatures, thereby reducing false-positive and false-negative detection rates. Second, simple and effective noise-filtering and data-compression software components enable a lightweight system implementation. In particular, this data compression results in a significant storage savings on the signature database.

The rest of the paper is organized as follows. Section 2 provides background information on mobile malware and power measurement issues. Section 3 describes the design of power-aware malware detection, including the power monitor and the data analyzer. The data analyzer includes noise-filtering and data-compression components. Section 4 describes the implementation of our detection framework and costume worm emulator. Details of software and hardware power measurements and the mechanism for building a power signature database are also presented there. Section 5 evaluates the detection accuracy of our framework. We discuss the related work in Section 6 and conclude the paper in Section 7.

2. MALWARE AND MEASUREMENTS

In this section we review the various threats posed by mobile malware, and discuss their effects on power and the related power measurement issues.

2.1 Mobile Malware

Malware writers do not want their malicious creations to be detected, analyzed, and filtered as they spread across a network, yet this is precisely the goal of detection systems, which examines incoming programs and compares them to the known malware signatures in order to detect them (i.e., signature-based detection). To evade detection, malware writers are increasingly using polymorphic coding techniques [36, 46]. Polymorphism is a process through which malicious code modifies its appearance to evade detection without actually changing its underlying functionality. These techniques include everything from modifying the names of internal variables and subroutines, changing the order in which instructions appear in the body of malware, to encrypting most of the malware code, only leaving in the clear text the instructions necessary to decrypt the code (e.g., a run-time packer [7]). These techniques, however, reflect a different byte frequency or packet length distribution from that of normal applications [24, 52], allowing a detection system to filter this polymorphic malware. In addition to changing the appearance of malware via polymorphism, new malware can further change their behavior, going through metamorphism [36]; metamorphic code actually changes the functionality of malware, while hiding its payload using obfuscation and encryption. When metamorphic techniques are used in conjunction with polymorphism, malware of this kind are much harder to detect, analyze, and filter.

Mobile malware targeting mobile handhelds seem relatively uncommon due mainly to the closed nature of device platforms and their relatively short time in the market, but as the world SmartPhone market is rapidly growing, the occurrence of mobile malware is increasing fast [27]. Cabir [20], which appeared in June 2004, was a classic proof-of-concept

worm and clearly entitled to bragging rights. This software alerted mobile users to the feasibility of creating malicious code for mobile handhelds. It was followed by a variant thereof, called Commwarrior [49], whose behavior is almost the same as that of Cabir, except that it sends an MMS message containing a copy of itself to randomly-chosen phone numbers. New variants, such as Mabir and Lasco, broke out after the appearance of Commwarrior.

Early malware on PCs were also written as pranks or for bragging rights, but have since been evolving with criminal or malicious intent. The motivations behind this type of attacks can be considered similar to those behind worms for mobile handhelds. For instance, since the use of handhelds has become essential to our everyday business, one seemingly illegal way to sabotage a competitor's business is to reduce the usability of their handhelds by depleting their batteries. Another way to inflict harm on competitors is to exploit a unique feature of mobile handhelds such as their built-in billing system. Excessive charges will be made to unfortunate victims by sending them an excessive number of SMS messages [27]. Many other attack scenarios may also emerge. Considering trends in the evolution of mobile malware, wilder, more diverse and sophisticated mobile malware will likely appear, and hence, we will need effective solutions to combat this type of threat.

2.2 Power Measurement Issues

The energy usage of each application activity in a certain system-power state is calculated by integrating the product of instantaneous current and voltage over a specific period of time. We approximate the energy usage by sampling current, I_t, and voltage, V_t, at constant intervals, Δt. In software measurements on a handheld, V_t is kept constant within accuracy that we want to achieve. Therefore, using a single measured battery voltage sample, V_c, we calculate the total energy consumption over n samples as $V_c \sum I_t \Delta t$. Using an oscilloscope, we determine the current drawn by the handheld through a hall-effect probe. The hall-effect probe detects the magnetic field generated by a current-carrying wire, resulting in a time-varying voltage, V_i, proportional to the current. Therefore, the total energy usage over n samples is calculated as $V_c \sum V_i \Delta t$.

Most mobile devices—including our evaluation platform, the HP iPAQ rx4200—are each powered by a lithium-ion battery. The lifetime of this type of battery depends on its capacity and state-of-charge levels. Although this battery has high energy-density, its internal resistance increases with age; after 2-3 years of use, the battery pack is likely useless due to its high internal resistance [2, 3, 6]. According to [2], the internal resistance also leads to dips in its output voltage as a result of short, heavy current spikes from a digital load. Unlike other types of batteries (e.g., nickel-metal-hydride), the internal resistance of lithium-ion batteries is known to be relatively constant. However, we have observed that the lithium-ion battery used in our experiments also displays variations in its internal resistance, depending on its remaining charge. The effect of these variations will be evaluated in Section 5.

We profile system-power states, including ON, BacklightOff, ScreenOff, and Suspended, managed by the power manager on the Windows Mobile 5 OS as shown in Fig. 1. In this profile, given different charge levels, we record information

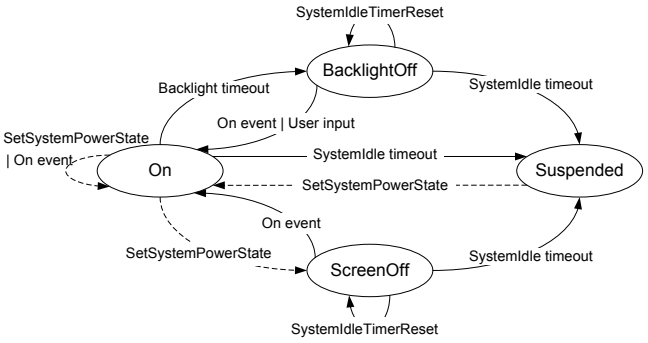


Figure 1: A simplified system power state transition diagram on Windows Mobile OS

on energy usage and average power-consumption level in each state. The power manager on the Windows Mobile 5 OS maintains these states. In the ON state, everything is ready for execution; the opposite is true in Suspended state. In the BacklightOff state, everything except for the backlight is ready to use. The transition from ON to BacklightOff is made with the backlight timer enabled, allowing energy savings of 2.90 Joules for 10 seconds—more measurement data and analyses will be provided in Section 4. In the ScreenOff state, the LCD screen is off. The transition from ON to ScreenOff is made only when an application makes a request to the power manager such as Windows Media Player via SetSystemPowerState(), which then allows the user to listen to the device in the ScreenOff state. In the Suspended state, most programs are deactivated, while only a few device drivers are awake. The transition from ON/BacklightOff/ScreenOff to the Suspended state is made when the SystemIdle timer expires; SystemIdleTimerReset() initiates the timer. Besides these system-power states, UserIdle, Unattended, and Resuming states are available only for specific models of SmartPhones or PocketPCs (which are not considered in this paper).

3. SYSTEM ARCHITECTURE AND SOFTWARE COMPONENTS

This section describes the architecture and key software components of our framework.

3.1 The Architecture

The power-aware malware-detection framework consists of two agents, a power monitor and a data analyzer, as illustrated in Fig. 2. The two agents reside either in combination or separately. The power monitor operates on a mobile handheld, taking samples of the power consumption which are used to build a power consumption history. The data analyzer, on the other hand, processes the power consumption history on either the host mobile handheld (Type A in the figure), or a remote server/data-sync PC (Type B in the figure) to reduce the overhead of the data analyzer. In the latter case, the power consumption history is transmitted from the mobile handheld to the server over the air, or to the data-sync PC via a USB cable/cradle. The data transmission though a USB cable/cradle does not consume energy from the battery because most SmartPhone batter-

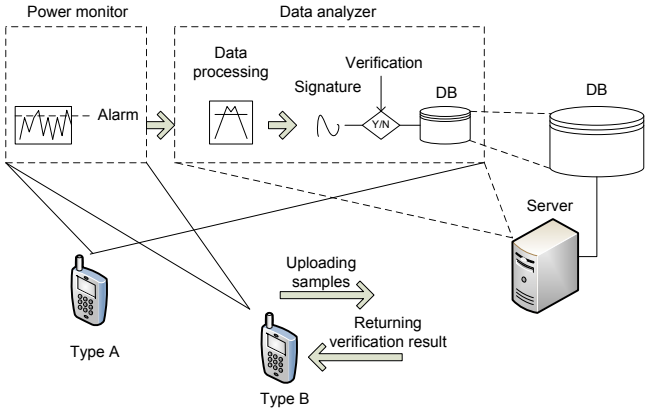


Figure 2: The system architecture: Type A stands alone and Type B has a remote server/data-sync PC for processing data.

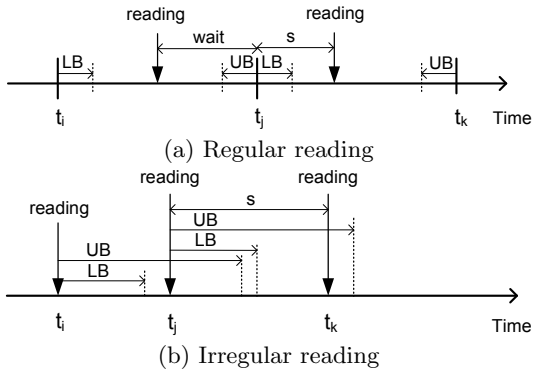


Figure 3: Power-reading methods: regular reading has a fixed interval based on t, while irregular reading-points are not fixed. In both cases, reading-points are randomly chosen in between lower bound (LB) and upper bound (UB). A timer object is set to a time period of s.

ies can be charged through a USB interface which also provides enough power for the device. In this case, our power monitoring probes are placed before the electronics powered by the external supply (or the battery), so that accurate measurement of the power consumption can still be guaranteed. Over-the-air transfer of the power consumption history, however, should be less energy-costly than processing it locally. Yet, network energy cost varies with the amount of data transmitted and the current network-device-power state. For instance, according to [9], fetching 32Kbytes of data via the WiFi radio in active mode on a handheld (requiring 0.9 Joules) consumes 1.4 Joules less energy than reading the data from the local microdrive in standby mode. Thus, an estimated 1Kbyte-size power consumption history can be transmitted in a single WiFi packet and when not in use, the WiFi radio can be turned off, as opposed to an EDGE network which must be kept active to receive calls while roaming.

3.2 The Power Monitor

The power monitor, which reads the power drawn from the battery on our handheld, is designed to capture power-consumption anomalies exhibited by applications. It is responsible for monitoring the power consumption, detecting a power surge in power consumption, and taking samples of the power consumption. Next we will describe each of these processes.

3.2.1 Monitoring

Choosing an appropriate rate to measure the power consumption is the basis for detecting power-consumption anomalies. The higher the frequency of making a reading (a set of power measurement samples), the greater the chance of capturing power-consumption anomalies, but the higher frequency may have a detrimental effect on the energy usage. At the same time, mobile malware writers eventually learn about the implementation of power monitoring systems and can then evade detection. To avoid detection, the malware can remain dormant over a period of time and then occasionally reactivate itself. By cycling between dormancy and activation, malware behavior can be obfuscated. One way to prevent this obfuscation is to randomly choose when to take the next power reading. If the power-reading time is unpredictable, then it will be difficult for the malware to evade detection. However, if reading-points are too random, the ability to capture power-consumption anomalies would be degraded.

To make this tradeoff, we devised two reading methods: regular and irregular. The regular reading, as illustrated in Fig. 3-(a), occurs at fixed intervals at which lower bound (LB) and upper bound (UB) are defined, and a reading-point is randomly chosen in between them. In other words, LB and UB specify an interval in which to make a random choice of a reading-point. In each interval, after a reading, the power monitor waits for the next base and then randomly chooses the next reading-point. Clearly, the larger the values of LB and UB, the narrower the random-choice space, and thus, the more regular the reading frequency over time. In other words, reading-points are likely uniformly-distributed. The irregular reading, on the other hand, does not wait in order to calculate the next reading-point as illustrated in Fig. 3-(b). Instead, LB and UB are determined according to the previous reading-point and the next reading-point is randomly chosen in between. So, only LB and UB are used to adjust the random-choice space (i.e., UB minus LB).

The power monitor creates a timer object which is used as an alarm clock. When an alarm is triggered, the power monitor calls GetSystemPowerStatusEx2() in the Windows CE .Net library in order to retrieve the battery status. This function takes a certain amount of time to complete, beginning with the invocation of its call to retrieve the data. This time period serves as a base for specifying LB and UB for the two reading methods. In practice, the time period amounts to more than 30 milliseconds, limiting the sampling rate. The regular reading invokes the timer object once more over a given time period to randomly choose the next reading time, incurring extra overhead.

The regular reading suits periodic reading and the irregular reading suits random reading. The latter, however, can represent the same distribution of reading occurrences as the former when LB and random-choice space for the irregular reading are set to LB plus UB and 1.5 times that for the regular reading, respectively.

3.2.2 Detection

While performing either of the two reading methods above, the power monitor also captures a surge in the power consumption, calculating the fraction of power surplus as follows:

$$(\frac{X}{Y} - 1) > \delta, \quad (1)$$

where δ is a given threshold, X is an observed power level, and Y is the trained power level specified in the system-power state profile. The system-power state profile defines the average power-consumption level in each system-power state (e.g., ON, BacklightOff, and ScreenOff). If the fraction exceeds the threshold, the power monitor then raises a flag, immediately starting to yield a power consumption history. In practice, we observed erratic spikes from the HP iPAQ rx4200 during the power reading process due to the switching properties of the digital system, resulting in false alarms. To reduce these false alarms, the threshold remains set high enough to be resistant to those spikes. Obviously, the higher the threshold, the fewer the false alarms, but the less sensitive to the surge in the power consumption. In addition to the threshold adjustment, a false-alarm counter is used; each time a false alarm occurs over an alarm-time period starting from the first alarm occurrence, the false-alarm counter is incremented by one. When the counter is greater than a given alarm-threshold, a true alarm is raised, leading to a switch to a sampling step. The false-alarm counter is set to 0 when the alarm-time period expires or a true alarm occurs. In our experiment, when $\delta = 0.2$, no false alarms occur in the ON state in which no explicit applications run. When $\delta < 0.2$, peaks from the spikes are detected, resulting in false alarms. Given a peak interval, either the alarm-time period (e.g., an estimated 4 reading intervals) or the alarm-threshold is adjusted to avoid these false alarms.

3.2.3 Sampling

The power monitor has a soft real-time constraint. Once a true alarm is raised, the power monitor starts taking samples of the power consumption at a constant rate, yielding a power consumption history; the higher the sampling rate, the more accurately the power consumption history can be interpreted, but also the more energy-costly. In addition, the timer object that the power monitor sets off at every given time interval can be preempted by another higher priority process, resulting in a measurement delay (completion time minus set-off time). Nevertheless, this delay can be offset by lengthening the sampling time period. In practice, the size of the history that results from software measurements of the power consumption can also be used to differentiate applications, thus eventually being added to the corresponding power signature. Note that it is a premise of these experiments that one application is executed at a time.

3.3 The Data Analyzer

The data analyzer receives the power consumption history from the power monitor and is designed to extract a unique pattern from the history, yielding a power signature. This power signature is then compared against the database of *a priori* signatures. In yielding power signatures, the data analyzer uses two data-processing software components: noise-filtering and data-compressing. Next, we describe these two components along with a signature-matching method.

3.3.1 Noise-Filtering

To reduce the effect of outliers on the power consumption history of an application, a moving average filter is applied to the dataset history. The moving average filter removes high-frequency noise from the dataset, resulting in a more generic power-consumption pattern. While calculating the average of its neighboring samples within a window of size $2k+1$, each sample, $S(i)$, in the power consumption history is converted into another, $S_p(i)$, as follows:

$$S_p(i) = \frac{1}{2k+1}(S(i-k) + S(i-k+1) + \cdots + S(i+k)). \quad (2)$$

This calculation starts from $i = k+1$ and continues until $i = n-k$; the first and the last k samples can be dropped since we are interested in an overall power consumption pattern. The window size determines the smoothness of the curve, i.e., the larger the k, the smoother the curve, but the less characteristic of very recent fluctuations in the dataset. The impact of k on the pattern associated with the detection accuracy will be evaluated in Section 5.

Among various filters we chose a simple moving average filter (e.g., a weighted moving average filter in which different weights are imposed on different distant samples or an exponential moving average filter in which weights decrease exponentially from the center), which is because a simple filter works just as well as, or even better than complicated ones (i.e., the implementation incurs less processing overhead). The choice of a simple filter is also advocated in [17].

3.3.2 Data Compression

A large power consumption history, which will result in a large power signature, needs to be reduced for two reasons. First, a large power signature consumes more energy than a small one in executing the matching process. Second, it is important to make economical use of memory in a mobile device. To reduce the size of the power consumption history, a simple and powerful one-way compression algorithm is proposed. By applying Algorithm 1, local jitter is effectively suppressed and compressed. As a result, a compact power signature can be represented, thereby achieving a substantial savings in both memory space and processing time. We will prove this result in Section 5.

Algorithm 1 A compression algorithm

```
Input: S_p(n): an n-length power consumption history
Input: m: look-ahead samples
Input: δ_c: a threshold
Output: S_e(k): a k-length power signature
while i ≤ (n − m) do
    Fetch m samples from S_p(i);
    Compute N ∼(μ, σ^2) of m;
    if σ < δ_c then
        S_e(j) ← μ; /*compressing history*/
        j ← j + 1
    else
        S_p(i : i + m) ← S_e(j : j + m); /*copying history*/
        j ← j + m
    end if
    i ← i + m
end while
```

3.3.3 Signature Matching

To measure the similarity between two power signatures, a χ^2-distance [16] is calculated as:

$$\chi^2(S_e,\ S'_e) \ = \ \sum_{i=1}^{n} \frac{(S_{ei} \ - \ S'_{ei})^2}{(S_{ei} \ + \ S'_{ei})}, \qquad (3)$$

where S_e and S'_e are signatures of the observed and the expected events, respectively. Clearly, $\chi^2 = 0$ if and only if all of the samples of S_e match those of S'_e. The higher the value of χ^2, the less likely the observed event belongs to the expected group.

The χ^2-distance is effective and efficient for our need. For instance, χ^2-distance-based techniques are found in diverse areas, such as scene-change detection in image sequences [39, 25] and anomaly detection [56]. In addition, experimental results [12] show that the use of the χ^2-distance reduces the amount of computation over one of the most widely used techniques, i.e., the Bhattacharyya distance [42].

Two power signatures that have the most similar power consumption patterns are found as:

$$\chi^2(S_e,\ DB) = \min_{S'_e \in DB} \{\chi^2(S_e,\ S'_e)\}. \qquad (4)$$

In some cases, two power signatures that comply with the same pattern can be skewed mainly because of delays in capturing the power surge. Since the χ^2-distance is based on the measurement of sample-to-sample distance, in order to effectively match two skewed power signatures, the data analyzer relies on either of two matching techniques: brute-force (BF) comparison and Fast Fourier Transform (FFT). The brute-force approach uses two parameters: an incremental state and a threshold. First, the distance is calculated and then one of the two power signatures is shifted left by one (and is subsequently done to the right). At the same time, if the newly-calculated distance is greater than, or equal to the previous distance, the incremental state parameter increases by one. Otherwise, it is set to 0. This procedure repeats until the incremental parameter exceeds the threshold. When this procedure stops, it returns the minimum distance. In addition to the incremental parameter, the proportion of samples for comparison correlates with the confidence in the comparison results (e.g., more than 90%). The brute-force comparison is especially efficient in the case of small delays in the reading in the power monitor. Alternatively, the FFT method is applied, converting time-domain representation of samples into their frequency-domain representation. In practice, this method facilitates the calculation of the distances in that a large portion of converted samples in two signatures that are similar to each other are likely to have the same constant frequency components, which offsets the complexity of the FFT computation. The performance comparison of the two methods will be presented in Section 5.

3.3.4 Response to the Analysis

The analysis results from the data analyzer pinpointing the signature that is most similar to that of the observed event, but this *pinpoint accuracy* (PA) is limited to the diversity of signatures in the database, e.g., a new application whose signature is not in the database is falsely detected. To address this limitation, each signature is labeled as either legitimate or malicious by prompting the user to enter the appropriate response—if no input from the user has been given for several seconds, then a default action will be taken. When the observed event is confirmed by the user as malicious, it is immediately stopped and quarantined. How to stop/quarantine, however, is beyond the scope of this paper. At the same time, the corresponding signature is added to the database and labeled as 'malicious'. If the observed event is legitimate, it resumes execution and the corresponding signature is also added and labeled as 'legitimate'. As a result, depending on the response type, signatures in the database are classified into two groups: legitimate and malicious. Then, the distance between the observed event and each of the two groups is calculated. The comparison of these distances allows the determination of the group to which the observed event is closest, despite any possible incorrect pinpoints (e.g., due to outliers), thereby reducing both false-positive and false-negative detection rates.

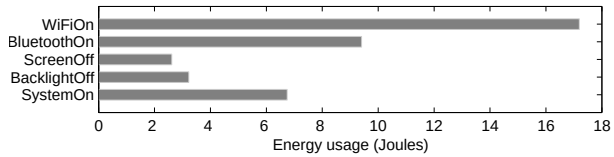


Figure 4: Energy usage of system states for 10 seconds on the HP iPAQ rx4200. No explicit application is executed in the SystemOn, BacklighOff, and ScreenOff states. For the BluetoothOn and WiFiOn states, the Bluetooth and WiFi radios are turned on exclusively for each state.

4. IMPLEMENTATION

This section describes two types of programs written in C#: the first aims to deplete the battery power on the HP iPAQ rx4200 running the Windows Mobile 5 OS, and the second emulates the behavior of four mobile worms on the same handheld device. We then provide details on how the signature database is built and how software and hardware measurements are made in the system.

4.1 Battery-Depletion Attack

The most energy-consuming activity on our handheld device is the use of the WiFi radio. As shown in Fig. 4, the handheld device with WiFi turned-on consumes 2.5 times more energy than with it turned-off (corresponding to the ON state in the figure) and 1.8 times more energy than with Bluetooth turned-on.

We present a sneaky malware program, called a WiFi faker, that launches a battery-depletion attack using the WiFi radio. When the WiFi faker is executed on our handheld with the WiFi-enabled mobile device, it tricks the system to believe that the WiFi device has been disabled, by rendering the WiFi adapter invisible to the system—the user just sees the WiFi-associated system tray icon indicating the WiFi device is inactive, but in reality it is still active and even deprived of doze mode, resulting in retaining the highest power consumption level. This deception is realized using two power management functions, DevicePowerNotify() and SetDevicePower(). In addition, the WiFi faker can collaborate with a dummy program which launches CPU-intensive activity (e.g., evaluating an exponential function), causing the battery to drain rapidly while the user believes that the WiFi radio is disabled. We will show that this attack can be effectively captured by our detection framework in Section 5.

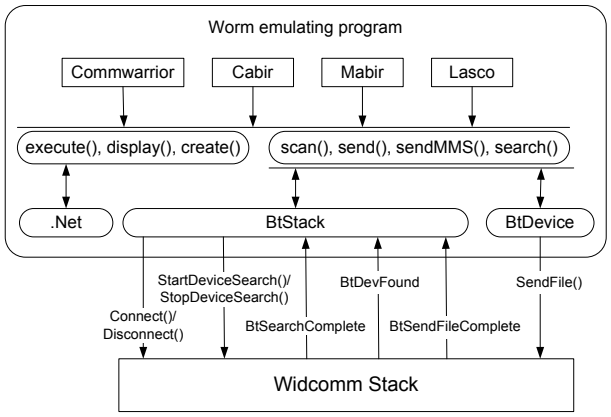


Figure 5: Software architecture of a worm emulator

4.2 Proof-of-Concept Mobile Worms

Scanning for Bluetooth-enabled devices and transmitting a file (regarded as worm payload) via Bluetooth are part of the fundamental capability of many of known mobile worms. The program that accesses a Bluetooth module in a device is implemented using the BTAccess.NET v3.0 library [4] which supports the Widcomm Bluetooth stack for our handheld device. We use two main classes from the library: BtStack and BtDevice, as illustrated in Fig. 5. Before using the Bluetooth radio, the program connects to the Widcomm stack, using Connect() in the BtStack class (while Disconnect() is used for disconnection from the stack). Once a connection is made, Bluetooth-enabled devices nearby are searched for using StartDeviceSearch(), which functions asynchronously. To stop the scan before completion, StopDeviceSearch() is called.

An event handler monitors two events: BtDeviceFound and BtSearchComplete. The event handler captures the BtDeviceFound event, thus returning the corresponding BtDevice object. This object is then added to a list for later retrieval. When the event handler captures the BtSearchComplete event indicating the completion of the search, the program stops searching for devices. In order to send a file (i.e., worm payload) when the searching is finished a BtDevice object is dequeued from the list and SendFile() in the BtDevice object is called. Success in sending a file triggers the BtSendFileComplete event. This procedure continues until all the objects on the list are dequeued.

The overall behavior of the four worms is represented by combinations of seven component actions, as listed below.

(s_1) *execute()*: starts a worm-behavior emulation.

(s_2) *display()*: opens a window and displays a message on the window. Cabir and Lasco exhibit this behavior to identify themselves.

(s_3) *create()*: generates a 15Kbyte array of data (i.e., another worm payload). The data are then stored in a system directory. An instance of the FileStream class is created in order to write to a flash memory.

(s_4) *scan()*: searches for Bluetooth-enabled devices nearby, using the service discovery application profile [45] defined in the Widcomm Bluetooth stack. This profile relies on Service Discovery Protocol [45] to discover devices.

(s_5) *send()*: sends a file (i.e., worm payload) to the devices found during the scan. This function uses the generic object exchange profile defined for the Widcomm Bluetooth stack. The OBEX protocol [45] in the profile is used to push the file data to nearby mobile devices.

(s_6) *sendMMS()*: searches an address book and executes send(). This behavior imitates an MMS message transmission except that the Bluetooth radio rather than an EDGE network is used.

(s_7) *search()*: searches the system directory for specific system files having a specific extension (e.g., Windows CE installation cabinet (.cab)) so that they are virtually appended for infection. The search is recursively performed from the root through its subdirectories. DirectoryInfo.GetFiles() is applied to retrieve all the files in a given directory, and DirectoryInfo.GetDirectories() is applied to retrieve subdirectories for the recursive call.

Note that the time taken to complete scan() and send() varies, depending on the variety of Bluetooth-enabled devices found nearby, and the number of corresponding objects on the list. The more objects found in the scan process, the longer the completion of the send() takes. The effect of this unforeseen situation results in a variety of signatures yielded even from the same application. Nevertheless, our detection framework effectively identifies such power signatures.

Worm Type	Sequential behavior
Cabir	$s_1 s_2 s_3 s_4 s_5$
Mabir	$s_1 s_3 s_6 s_4 s_5$
Commwarrior	$s_1 s_3 s_4 s_5 s_6$
Lasco	$s_1 s_2 s_3 s_4 s_5 s_7$

Table 1: The behavior of worms

The action sequence for each worm is presented in Table 1, showing common subsequences. For instance, all the worms but Mabir have a common subsequence, $s_3 s_5 s_6$. However, their power signatures can be greatly different from each other because of the s_5 behavior. Similarly, Cabir and Mabir have behavior in common. Cabir is likely misidentified as Lasco. In Section 5 we will evaluate the accuracy in detecting previously unknown malware with respect to its power signature similarity.

4.3 Building a Power Signature DB

We define application-behavior scenarios which are divided into legitimate and malicious application groups. We chose pairs of applications that have similar behavior and different intent, i.e., one for the legitimate application group and the other for the malicious application group. For instance, a program designed to execute CPU-intensive functions and a Windows Media Player (WMP) are both energy-greedy, but have different intent. Also, the mobile worms described above and legitimate Bluetooth file transfers have a common behavior, but have different intent. First, we characterize the following malicious applications in order to build our signature database.

i-1. The dummy program executes a function that is not productive and just consumes CPU time (e.g., CPU-intensive computation), wasting energy. Power consumption histories are captured at the beginning and in the middle of this program run, thereby extracting two different power signatures.

ii-1. The WiFi faker: This program, as described earlier, disguises the WiFi system tray icon to appear as inactive and in fact turns on the WiFi radio operating in the highest power mode all the time. This behavior is captured and then one power signature is extracted.

iii-1. The combination of the dummy program and the WiFi faker. The WiFi faker is executed and the dummy program is then launched (the order of execution does not matter). One energy signature is extracted while the two programs are running.

iv-1. The four mobile worms. The execution of each worm results in a power signature.

v-1. A DoS-attack-like Bomber: This program bombards the handheld with 1Kbyte- and 2Kbyte-size data via WiFi (e.g., ping -s 1024/2048). In practice, a stream of 2Kbyte-size data froze the handheld after 30 seconds. Two different power signatures are extracted for the different size packets.

Second, legitimate applications are characterized as follows.

i-2. Windows Media Player: This program incurs high energy consumption, but the amount of energy consumption varies depending on the video codecs used, e.g., Windows Media Video (WMV) 9 at 315bps and WMV7 at 704bps. Power-consumption histories are recorded at the beginning and end of 5 seconds of execution for each codec, resulting in four different power signatures.

ii-2. Bluetooth and WiFi file transfers. A 10Mbyte-size file is transferred via Bluetooth and WiFi. Note that the Bluetooth file transfer and the four mobile worms, as well as the WiFi file transfer and the Bomber, have behavior in common, respectively. Two power signatures are extracted.

iii-2. A users' handheld-usage pattern. Two users separately explore files, i.e., tapping on the start menu and executing the file explorer. They then drag the scroll-bar up and down, tapping on a subfolder and opening an image file. This pattern leads to two different power signatures.

4.4 Measurements

For software measurement, GetSystemPowerStatusEx2() in Coredll.lib is used to retrieve battery status information, including AC line status, battery current, and battery voltage, which are normally used for monitoring the system.

Hardware measurements are made with the Agilent Infiniium 54815-A oscilloscope which is capable of sampling at 1GS/s with a 1 millisecond peak detection. This oscilloscope makes it easy to synchronize power measurements with process execution. Unless otherwise specified, samples are taken every 100 milliseconds from software measurements (execution) and every 10 milliseconds for hardware (power).

The energy-consumption history is recorded over 10 seconds via hardware measurement, and 20 seconds via software measurement. The first round of the execution of these application scenarios yields 18 different power signatures. Twenty rounds are made in total; the first 5 rounds result in 90 power signatures which are used as a training set, and the remainder yields 270 power signatures which are used as a test set.

5. EVALUATION

The metrics used to indicate the detection accuracy include pinpoint accuracy (PA) and true-positives. PA represents the ability to classify an event correctly. For instance, Cabir should be identified as Cabir rather than any other type of malware, such as Mabir. As there will be no signatures in the database for previously unknown malware, the data analyzer is unable to identify it by name. However, since signatures are classified as malicious or legitimate, the data analyzer is able to classify previously unknown applications as either malicious or legitimate, and the success rate in this classification is represented by the true-positive rate. Thus, PA is a measure of true-positives. In addition, false-positive (classification of benign activity as malicious) and false-negative (failure to identify malware) rates are calculated.

In this section we first assess system parameters defined in our power-aware malware-detection framework and then evaluate the detection accuracy with the optimal values of the system parameters found. Finally, we analyze the performance issues of the framework.

5.1 Assessment of System Parameters

Effect of Battery-Charge Level: To understand the correlation between state-of-charge levels and the variation of the power drawn from a lithium-ion battery (HP Model No. HSTNH-S11B with 1200mAh), we implemented a power measurement program to run on the HP iPAQ rx4200. After the battery is fully charged, the power-measurement program starts reading the on-device hardware power monitor once a second, logging the corresponding power-consumption values. The remaining charge in the battery decreases over time at a room temperature of 75 degree Fahrenheit/24 degree Celsius. However, we could not exclude the effect of inherent increase in battery temperature over time (it increased by 2 degree Fahrenheit) during the program run. Fig. 6 shows a battery power consumption distribution and the average power consumed as the battery discharges. Note that the characteristics of the battery discharge are also affected by the battery age. Clearly, the recognition system for overall average power consumption relies on the knowledge of the battery state-of-charge. For instance, when the state-of-charge level is between 84% and 100%, the average power consumed increases from 84mW to 86mW. When the state-of-charge level is between 56% and 83%, the average power steeply drops to 80 mW and gradually increases up to 82 mW. At a charge level of 55%, the average power jumps back to 90 mW and after this, it steadily increases over time. This pattern results from short, heavy current spikes from the handheld caused by the non-linear digital electronics as a result of its changing supply voltage. The changing supply voltage is caused by a voltage drop across the battery's increasing internal resistance [2, 6, 44] that is associated with the battery-charge level. As a result, each

power signature is extracted, according to the three different charge levels (high, med, and low). Battery temperature, on the other hand, was found to have no impact on the power consumption pattern except that the frequency of spike occurrences was reduced at very low temperatures (e.g., 35 degree Fahrenheit/2 degree Celsius).

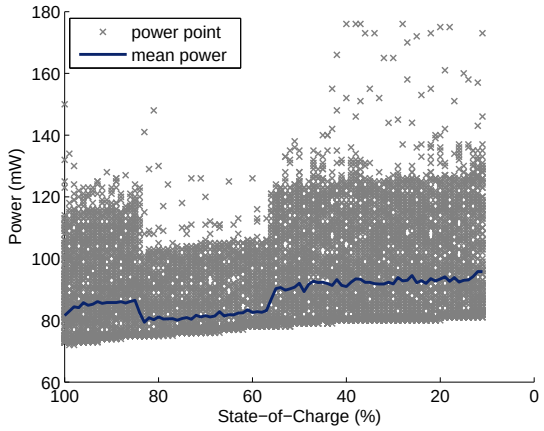


Figure 6: Power consumption variations with respect to the battery-charge level in software measurement

Signature Generation: A power consumption history is produced while running an application on a handheld. The power consumption history is transformed into a power signature via two techniques: the moving average filter and the data compressing. The moving average filter removes noise from the power consumption history, effectively extracting a pattern. The compression technique, on the other hand, is applied to reduce the size of a signature, without losing the detection accuracy. In the compression technique, local jitter is suppressed and compressed. Fig. 7 shows the procedure of generating a power signature from the power consumption history of a video clip playback with a bit-rate of 315bps, using the WMV 9 codec. Fig. 7-(a) shows the power consumption history captured in which a pattern can hardly be recognized, mainly because of signal noise. After the filter is applied, a pattern becomes visible as shown in Fig. 7-(b). The application of the compression technique results in a power signature as shown in Figure 7-(c).

Impact of Filter Parameters: The window size (k) in the moving average filter determines the degree to which noise is reduced, which, in turn, correlates with the detection accuracy. That is, the larger the k, the smoother the curve, which may lower the accuracy. On the other hand, if k is too small, the filter may be less effective for reducing noise. Thus, the optimal k needs to be found to achieve the highest accuracy. We conducted an experiment to find the optimal values, with the lookahead size and its threshold fixed (m=5 and δ_c=0.05 whose assessment will be presented shortly). We evaluated the detection accuracy with a test set of 270 power signatures and a database of 90 power signatures labeled as either legitimate or malicious. As Fig. 8 shows the correlation between the window size and PA, the 23- or 24-point moving average filter for the 1000-sample power consumption history allows the highest PA. When k is smaller than 23, the filter seems ineffective and as k becomes larger after 24, the effectiveness of reducing noise is gradually degraded. The reason for this is that the large k reflects less of recent fluctuation of samples of the power consumption within the window.

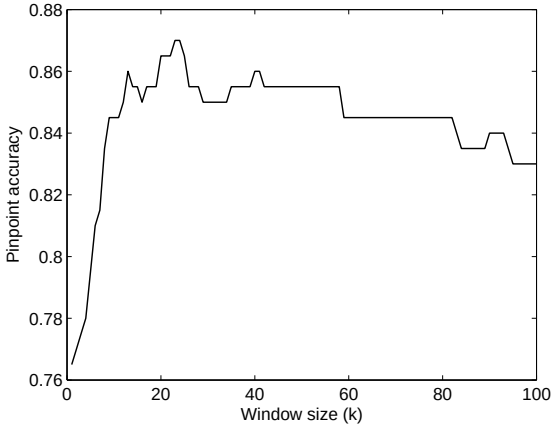


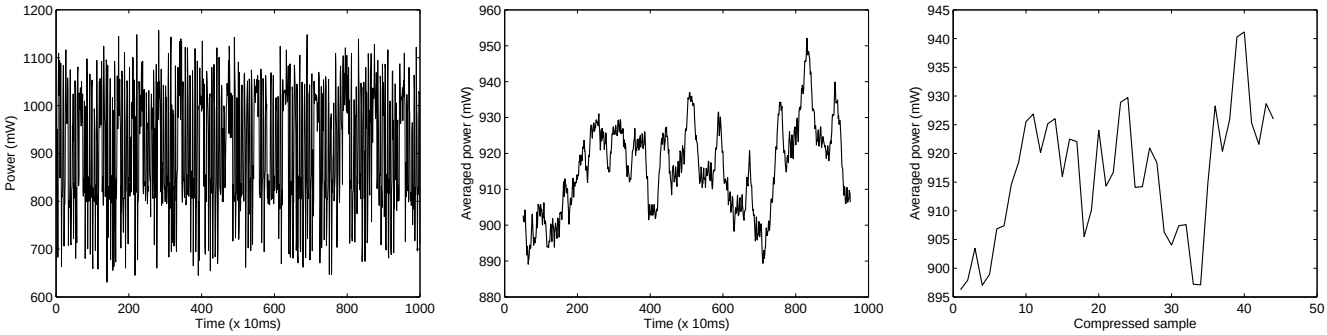
Figure 8: An optimal window size (k) in the moving average filter. When the k=23 or 24, PA is the highest. No compression was applied.

Effectiveness of Compression: The lookahead size, m, and its threshold, δ_c, used in the data compression determine the compression ratio which we intend to maximize without losing the detection accuracy. We conducted an experiment under the same condition (i.e., the same database and test set) as when the optimal k was obtained. From the result of the previous experiment, k is set to 23. We then attempt to find the optimal values of m and δ_c. As shown in Figs. 9-(a) and (b), when $\delta_c > 0.05$, more than 95% storage-savings is achieved. As shown in Figs. 9-(c) and (d), the lookahead parameter correlates more prominently with the compression ratio than PA when δ_c=0.06. As m increases, the compression ratio also increases, while PA is hardly affected. The FFT technique allows a higher compression ratio than the brute-force (BF) comparison because a large portion of samples are converted into constant frequency components.

Accordingly, when δ_c=0.06, the 23/24-point moving average filter and compression with the 20-sample lookahead (15 samples for FFT) allow the highest PA for hardware (power) measurement, while when δ_c=2, the 5-point moving average and compression with the 5-sample lookahead are optimal in software (system execution) measurement.

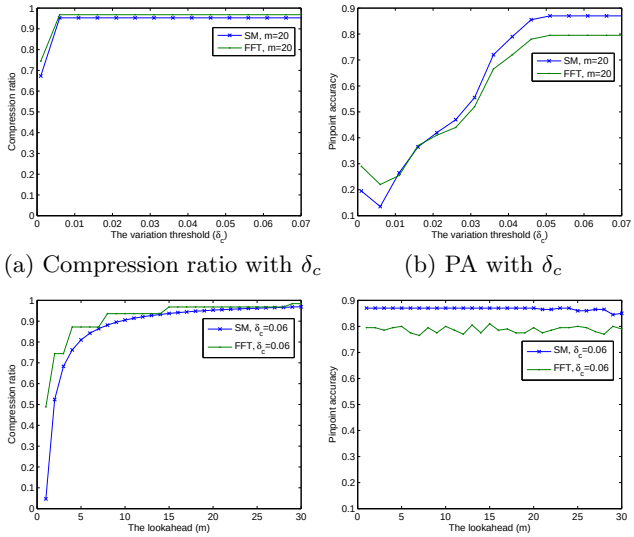
5.2 Detection Accuracy

Detecting Battery-Depletion Attack: The WiFi faker renders the WiFi-associated system tray icon disabled, thus misleading the user to think that the device is turned off although it is actually on. The WiFi faker makes a request to the power manager for letting the WiFi device adopt the maximum power state, thus draining the battery at the fastest possible rate. The WiFi faker can collaborate with the dummy program that executes an exponential function in a loop. Both aspects of this behavior shown by the WiFi faker and the dummy program are effectively captured by our power-aware detection framework. Fig. 10 shows power-consumption patterns with the WiFi faker and the dummy program executed separately, and in combination. Each of



(a) Raw samples (b) 50pt-moving average filter applied (c) Local jitter removed

Figure 7: Power signature generation



(a) Compression ratio with δ_c (b) PA with δ_c

(c) Compression ratio with m (d) PA with m

Figure 9: Compression ratio comparison

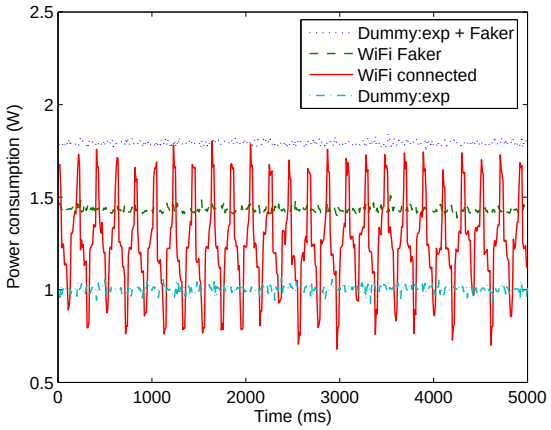


Figure 10: Comparison of the power consumption with WiFi connection, the WiFi faker, and a combination of a dummy program and the WiFi faker. The WiFi device is in power-saving mode in which the WiFi device dozes after every beacon interval.

these three patterns (excluding the WiFi-connected pattern) is then represented by a power signature as a malicious application.

To evaluate the accuracy for detecting the battery depletion attacks described above, we set up the following test scenario. Starting with the signature database generated as the basis of the legitimate application group signatures defined in Section 4.3, we separately compared the WiFi faker, the dummy program, and the combination of these two programs, using 20-sample sets. First, the WiFi faker was identified as abnormal rather than malicious because the database did not contain a corresponding malware signature. The signature of the WiFi faker was then added to the database, and finally, the three programs were tested. By repeating this test with different combinations of the programs, we diversified and populated the database. As can be seen in Table 2, the WiFi faker (A) is identified with 100% accuracy using Database A, and detected 100% of the time with Databases B and C because the WiFi faker and the other two applications have common power consumption patterns. The dummy program is identified 100% of the time with Database B. In particular, the combination (C) of the two is detected 100% of the time with any of Database A, B, or C. In general, the more diverse the database, the higher the detection accuracy.

Four mobile worms—Cabir, Mabir, Commwarrior, and Lasco—which come from the same malware family have common behavior. Likewise, the power signature of one worm can be the basis for detecting the other worms. To evaluate the ability of our framework to detect previously unknown worms whose signature is similar to those of previously known worms, the four costume worms were divided into two groups: known-worm and unknown-worm groups. Worms in the known-worm group were executed 5 times to extract their signatures for the database (training set), while worms in the unknown-worm group were executed 15 times to generate a test signature set. Table 3 summarizes the detection accuracy for unknown worms with different combinations of known and unknown worms. The first four rows that correspond to the databases with a single worm signature exhibit the worm closest in behavior to the other. For instance, Cabir (C) and Mabir (M) have a similar power

DB\Tested Malware	A	B	C
A	100%	0%	100%
B	100%	100%	100%
C	100%	0%	100%
A, B	100%	100%	100%
A, C	100%	0%	100%
B, C	100%	100%	100%
A, B, C	100%	100%	100%

Table 2: Battery-depletion attack detection: A (the WiFi faker), B (dummy program), and C (WiFi faker and dummy program) are tested according to diverse signatures of the DB. Ten distinct signatures from the legitimate application group are initially included in the DB.

consumption pattern, as do Mabir (M) and Lasco (L). The more diverse worms collected and added to the database, the higher detection accuracy for unknown worms. For instance, the detection accuracy for Commwarrior (C) and Lasco (L) is improved even with a partially-filled database, thus making this technique attractive for resource-limited handhelds.

DB\Tested Worm	C	M	W	L
C	87%	93%	73%	87%
M	93%	100%	80%	93%
W	47%	93%	80%	87%
L	87%	93%	80%	93%
C, M	93%	100%	80%	93%
M, W	93%	100%	80%	93%
W, L	87%	93%	80%	93%
C, M, W	93%	100%	80%	93%
M, W, L	93%	100%	80%	93%

Table 3: The previously unknown worms: C (Cabir), M (Mabir), W (commWarrior), and L (Lasco) are tested against diverse signatures in the DB. Ten distinct signatures from the legitimate application group are initially included in the DB.

In addition to the detection of previously unknown worms, the Bomber whose behavior is similar to that of the WiFi file transfers was also identified with 95% accuracy. Activities that result from the Windows Media Player such as playing two distinct frames with different video codecs—signatures were extracted at the beginning and end of a 5-second execution for each codec—were identified with 100% accuracy.

5.3 Performance Analysis

The moving average filter we used turned out to be very effective for removing noise, thus extracting a clear power consumption signature from the power consumption history. Table 4 shows the detection accuracy with and without the filter and the compression techniques applied. In the table, the moving average filter and the compression techniques were not applied in the case of C1, while only the filter was applied in the case of C2. Comparing C1 and C2, PA was improved by 22%, achieving a 98% true-positive rate. This enhancement strongly supports the effectiveness of the filter. In addition to the moving average filter, in comparison of C2 and C3, our compression technique is also effective for optimizing memory usage, without degrading the accuracy (the effect of the compression technique will be analyzed shortly). If the number of samples to be matched is 95% of the total samples, our detection scheme achieves a 99% true-positive rate, while decreasing the false-negative rate down to 0% (in C4). In the case of applying the FFT (C5), the overall accuracy is improved, with the false-negative rate reduced to 2% in comparison with C1 which only achieved 5%.

Methods	PA	TP*	FN	FP	TP
(C1) BF w/o opt.	64%	29%	5%	2%	93%
(C2) BF w/o comp.	78%	20%	<1%	2%	98%
(C3) BF	78%	20%	<1%	2%	98%
(C4) BF (95%)	76%	23%	0%	<2%	99%
(C5) FFT	73%	23%	2%	3%	96%

Table 4: Comparison of the overall detection accuracy, based on hardware measurement. TP (the true-positive rate) equals PA plus TP* (the exclusive true-positive rate). Brute-force (BF) comparison and FFT methods are compared with FN (false-negatives) and FP (false-positives) as well as TP and PA.

We applied a simple and powerful compression technique. As shown in Table 5, this technique allows the power signature to be compressed by a factor of 21 without losing the detection accuracy (in the case of R1). This compact signature representation also allows the signature matching to require less CPU time. For instance, the data processing needed for the compression and the BF comparison (100% samples matching) requires less CPU time than the case without the compression by 71% (in the case of R2). When the FFT method is applied, the data processing including the FFT computation is estimated to be 1.6 times faster than the case without this optimization, resulting in only 63% of CPU time required (in the case of R3). Comparing the FFT method with the BF approach, therefore, as we expected, the data processing with the FFT method applied is estimated to be 1.3 times faster than that with the BF approach applied, because most of the transformed data as a result of the FFT are zero or the same constant frequency components, simplifying the distance metric computation.

Ratio	Value
(R1) Compression ratio	21.3
(R2) CPU gain ratio (BF)	1.4
(R3) CPU gain ratio (using FFT)	1.6

Table 5: Performance comparison

6. RELATED WORK

Heuristics on what to monitor and what to collect vary with different platforms (i.e., handhelds, workstations, and their interconnection network), but technologies and methodologies for detecting known and unknown threats may be similar or even same, regardless of the platform type, with appropriate optimization and customization. In this section,

we review related work and list some of major technologies and methodologies for malware detection and relevant measurements.

- *Code Analysis.* An agent in a detection system analyzes attempts to execute code in several ways. First, before running code normally on a host, the agent can execute the code in a virtual machine or a sandbox, comparing its behavior with profiles or rules that specify good and bad behavior [55]. Bad behavior includes, for instance, attempts to gain administrator-level privileges or overwrite a system executable. Second, an agent can detect attempts to perform stack and heap buffer overflows, looking for their typical characteristics such as certain sequences of instructions or 'ret' instructions in the code, e.g., shellcode [29, 37, 53]. The shellcode enters the system, exploiting the vulnerability of web browsers. Third, an agent can monitor and process system calls to identify applications performing certain actions such as intercepting keystrokes. Characteristics of such malware can be extracted from system calls by applying advanced techniques to the call sequences, e.g., enumerating sequences [26, 47], n-gram vectors [15], data-mining techniques [28], Bayesian classification [35], neural networks [34], finite-state automata [43], and hidden Markov models [54]. Last, the agent may monitor each application and library (e.g., a dynamic link library (DLL)) that a process attempts to load. This information is then compared against lists of authorized and unauthorized applications and libraries.

- *Statistical Monitoring.* SmartSiren [13] is a virus detection and alert system for mobile handhelds, which aims to detect worms exploiting SMS messaging and Bluetooth communication. This detection system is based on statistical monitoring for the detection of fast spreading worms, and abnormality monitoring for the detection of slower worms and Trojans. Using statistical monitoring, the average number of communications initiated every day and the average number of handhelds exceeding the average are counted. When daily measurements for each category exceed the average threshold, an alert is triggered. Using abnormality monitoring, Trojans can be detected by keeping track of the number of messages directed to a specific destination. SmartSiren is the first infrastructure-based solution to securing mobile phones despite the limitations of post-infected detection.

- *Payload Analysis.* PAYL [52] is a payload-based IP network intrusion detector in which a normal payload is analyzed and then the occurrence of each character found in the payload is counted to produce a byte frequency distribution. The average byte frequency, and standard deviation of each byte's frequency, form a payload model. The payload model is made for the observed length of each IP port. Two neighboring models converge or remain separate, comparing similarity of the two using the Manhattan distance between average byte frequencies. After payload models are set, the Mahalanobis distances [50] between data observed from a process, and each model are calculated. If any distance calculated is greater than a threshold, then the process is considered anomalous. The feature of the Mahalanobis distance is to consider not only the mean distance, but also weights of each point using covariance. PAYL is, however, ineffective for encrypted channels, because their distributions appear uniformly distributed.

- *Filesystem Monitoring.* A filesystem can be monitored by checking file integrity, file attributes, or file access attempts. In checking for file integrity, the agent yields message digests or cryptographic checksums for critical files, comparing them against reference values, and verifying their differences [38]. Similarly, in checking for file attributes, the agent checks the attributes of important files (e.g., ownership and permission for change). Both file integrity and attribute checking can only determine (in an after-the-fact manner) if a change has taken place. In checking for file access attempts, an agent with a shim—a layer of code placed in between existing layers of code—can monitor all attempts to access critical files and stop suspicious attempts by comparing policies with the characteristics of the current attempt, such as which user/application attempts to access what file with a particular type of access (i.e., read, write, or execute). This can be used for preventing the installation of malware, such as rootkits, Trojans, and other types of malicious activity.

- *Power Consumption Monitoring.* An agent can monitor the power consumption resulting from applications executing on the device. For accurate and detailed measurements, a monitoring agent cooperates with a profiling agent running on a separate device similar to PowerScope [23]. These two agents in PowerScope are connected by a digital multimeter. The monitoring agent triggers the digital multimeter to take samples and the sampled data are then passed to the profiling agent. At the same time, the monitoring agent puts a timestamp on the procedure call sequence. This way, an energy profile which includes per-process and per-user-level signatures can be constructed. Similarly, energy usage of applications can be measured on a pocket PC running a Java virtual machine [21].

7. CONCLUSION

We end this paper by discussing the limitations encountered in pursuit of this work, along with concluding remarks.

7.1 Limitations

Only a few worm samples are publicly available for research. Testing detection systems with real-world malware is necessary to evaluate the effectiveness of any detection system, but due to the nature of in-the-wild malware activities, effective and comprehensive evaluation and testing with real-world worm samples are difficult to do. For this reason, most research relies on benign activity or worm modeling [8, 32, 33] to study the effectiveness of methods. Alternatively, a malware emulator that imitates real malware behavior could be used and most importantly, the malware emulator should be able to build diverse types of test suits so that the malicious activity may accurately reflect the composition of recent threats against detection systems.

Considering a handheld's limited user interface and design of the window managers, we assumed that one application would run at a time. Even the most advanced SmartPhone currently available in the market tends to provide window managers that only support one application running at a time. However, multiprocessing is expected to be employed in future handhelds, and how to address this issue is part of our future work. In particular, the fine-grained measurement of energy usage per process on the handheld is as important as the enhancement in the battery data acquisition system that a battery chip offers.

7.2 Concluding Remarks

Mobile handhelds must be protected against malware, especially those with the goal of dramatically reducing their battery lifetime. In this paper, we have presented a power-aware malware-detection framework, with the aim of furthering users' mobility and the ubiquitous use of their mobile device. We began by characterizing power consumption patterns of events and designed two important system components. We then performed a comprehensive analysis of the detection accuracy for pinpointing the identity of events, as well as classifying them as malicious or normal. We addressed two challenges: (1) extracting characteristics of the power consumption history from noisy samples, and using data compression to generate a compact power signature, resulting in a 95% storage savings, and (2) deriving the efficacy of detecting energy-greedy anomalies as well as unknown malware over our representative test set up to a 99% true-positive rate. In summary, our framework offers a unique solution to the problem of securing battery-powered mobile devices, and is further enhanced with the support of a wireless server/PC-side providing a comprehensive analysis of the data.

Acknowledgment

We would like to thank our shepherd, Roy Want, for his detailed technical and editorial comments which enhanced this paper significantly.

The work reported in this paper was supported in part by the National Science Foundation under Grant No. CNS 0523932, Samsung Electronics, and Intel Corporation.

8. REFERENCES

[1] http://www.gartner.com/it/page.jsp?id=501734.

[2] Battery university - the high-power lithium-ion. http://www.batteryuniversity.com/partone-22.htm.

[3] Battery university - the high-power lithium-ion. http://www.batteryuniversity.com/partone-5A.htm.

[4] Btaccess.net. http://www.high-point.com.

[5] Making handheld security a priority. http://www.symantec.com/norton/products/library/article.jsp?aid=handheld_security.

[6] The secrets of battery runtime 2. http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_bpw2_c06_03.

[7] UPX: the Ultimate Packer for eXecutables. http://upx.sourceforge.net/.

[8] *Bluetooth Worms: Models, Dynamics, and Defense Implications*, Dec. 2006.

[9] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: interfaces for better power management. In *MobiSys*, pages 23–35, New York, NY, USA, 2004. ACM.

[10] A. Bose and K. G. Shin. On mobile viruses exploiting messaging and bluetooth services. In *SecureComm*, pages 1–10. IEEE, Aug. 2006.

[11] T.K. Buennemeyer, M. Gora, R.C. Marchany, and J.G. Tront. Battery exhaustion attack detection with small handheld mobile computers. In *PORTABLE*, pages 1–5. IEEE, May 2007.

[12] Branislav Kisa canin, Vladimir Pavlović, and Thomas S. Huang, editors. *Real-time vision for human-computer interaction*. ISBN 387276971. Springer, 1 edition, 2005.

[13] Jerry Cheng, Starsky Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *MobiSys*, pages 258–271, San Juan, Puerto Rico, Jun. 2007. ACM.

[14] David Dagon, Tom Martin, and Thad Starner. Mobile phones as computing devices: The viruses are coming. *Pervasive Computing*, 3(4):11–15, Oct. 2004.

[15] Marc Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, Feb. 1995.

[16] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. ISBN 0-471-05669-3. A Wiley-Interscience Publication, second edition, 2001.

[17] Robert D. Edwards and John Magee. *Technical analysis of stock trends*. ISBN 0814406807. AMACOM, New York, NY, 8th edition, 2001.

[18] Daniel R. Ellis, John G. Aiken, Kira S. Attwood, and Scott D. Tenaglia. A behavioral approach to worm detection. In *WORM*, pages 43–53, Washington DC, USA, 2004. ACM.

[19] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta. Exploiting open functionality in sms-capable cellular networks. In *CCS*, pages 393–404, Alexandria, VA, USA, 2005. ACM.

[20] F-secure. Cabir. http://www.f-secure.com/v-descs/cabir.shtml.

[21] Keith Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, and Jennifer Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. *SIGMETRICS: PER*, 28(1):252–263, June 2000.

[22] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly detection using call stack information. In *SP*, Oakland, CA, USA, May 2003. IEEE.

[23] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA*, New Orleans, Louisiana, Feb. 1999. IEEE.

[24] Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *Security Symposium*, pages 17–17, Berkeley, CA, USA, 2006. USENIX.

[25] R.M. Ford, C. Robson, D. Temple, and M. Gerlach. Metrics for scene change detection in digital video sequences. In *ICMCS*, pages 610–611, Los Alamitos, CA, USA, 1997. IEEE.

[26] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji,

and Thomas A. Longstaff. A sense of self for unix processes. In *SP*, page 120, Oakland, CA, USA, May 1996. IEEE.

[27] Mikko Hypponen. Malware goes mobile. Nov. 2006.

[28] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Security Symposium*, volume 7, San Antonio, Texas, USA, Jan. 1998. USENIX.

[29] S. Macaulay. Admmutate: Polymorphic shellcode engine. http://www.ktwo.ca/security.html.

[30] Thomas Martin, Michael Hsiao, Dong Ha, and Jayan Krishnaswami. Denial-of-service attacks on battery-powered mobile computers. In *PerCom*, page 309, Washington, DC, USA, 2004. IEEE Computer Society.

[31] M.Christodorescu, S.Jha, S.A.Seshia, D.Song, and R.E.Bryant. Semantics-aware malware detection. In *SP*, pages 32–46, Oakland, CA, USA, May 2005. IEEE.

[32] James W. Mickens and Brian D. Noble. Modeling epidemic spreading in mobile environments. In *WiSe*, pages 77–86, New York, NY, USA, 2005. ACM.

[33] Jose A. Morales, Peter J. Clarke, Yi Deng, and B. M. Golam Kibria. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology*, 2(2):135–147, Nov. 2006.

[34] Srinivas Mukkamala and Andrew H. Sung. Identifying key features for intrusion detection using neural networks. In *ICCC*, pages 1132–1138, Mumbai, Maharashtra, India, 2002.

[35] Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. Anomalous system call detection. *TISSEC*, 9(1):61–93, 2006.

[36] Carey Nachenberg. Computer virus-antivirus coevolution. *Communications of the ACM*, 40(1):46–51, 1997.

[37] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *SP*, pages 133–145, Oakland, CA, USA, May 2005. IEEE.

[38] Taejoon Park and Kang G. Shin. Soft tamper-proofing via program integrity verification in wireless sensor networks. *TMC*, 4(3):297–309, May 2005.

[39] N.V. Patel and I.K. Sethi. Compressed video processing for cut detection. *Vision, Image and Signal Processing*, 143(5):315–323, October 1996.

[40] Matthew Pirretti, Sencun Zhu, N. Vijaykrishnan, Patrick McDaniel, Mahmut Kandemir, and Richard Brooks. The sleep deprivation attack in sensor networks: Analysis and methods of defense. *IJSNet*, 2(3):267–287, Sept. 2006.

[41] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *SecureComm*, pages 1–10, Baltimore, MD, Sep. 2006. IEEE.

[42] C. Reyes-Aldasoro and A. Bhalerao. The bhattacharyya space for feature selection and its application to texture segmentation. *Pattern Recognition*, 39(5):812–826, May 2006.

[43] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *SP*, page 144, Oakland, CA, USA, Apr. 2001. IEEE.

[44] S. P. Shukla, Y. W. Suen, and M. Shayegan. Magnetic-field-induced triple-layer to bilayer transition. *Phys. Rev. Lett.*, 81(3):693–696, Jul 1998.

[45] Bluetooth SIG. Specification of the Bluetooth system, Core Version 1.1. http://www.bluetooth.com/, Feb. 2001.

[46] Ed Skoudis and Lenny Zeltser. *Malware: Fighting Malicious Code*. ISBN 0131014056. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 2004.

[47] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *SP*, page 14, Oakland, CA, USA, May 2000. IEEE.

[48] Thad Starner. Thick clients for personal wireless devices. *Computer*, 35(1):133–135, 2002.

[49] Symantec. Commwarrior description available at. http://securityresponse.symantec.com.

[50] Camilo Tenorio, Francisco de Carvalho, and Julio Pimentel. A partitioning fuzzy clustering algorithm for symbolic interval data based on adaptive mahalanobis distances. In *HIS*, pages 174–179. IEEE, July 2007.

[51] Sampo Töyssy and Marko Helenius. About malicious software in smartphones. *Journal in Computer Virology*, 2(2):109–119, Nov. 2006.

[52] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *RAID*, volume 3224, pages 203–222. LNCS, Oct. 2004.

[53] Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu. Sigfree: a signature-free buffer overflow attack blocker. In *Security Symposium*, Vancouver, Canada, Jan. 2006. USENIX.

[54] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *SP*, pages 133–145, Oakland, CA, USA, Apr. 1999. IEEE.

[55] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy*, 5(2):32–39, March 2007.

[56] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Eng. Int'l*, 17(2):105–112, October 2001.