MODELZ: Monitoring, Detection, and Analysis of Energy-Greedy Anomalies in Mobile Handsets

Hahnsang Kim, *Member*, *IEEE*, Kang G. Shin, *Fellow*, *IEEE*, and Padmanabhan Pillai

Abstract—It is of great importance to protect rapidly-spreading and widely-used small mobile devices like smartphones and PocketPCs from energy-depletion attacks by monitoring software (processes) and hardware (especially, battery) resources. The ability to use these devices for on- and/or off-job functions, and even for medical emergencies or disaster recovery is often dictated by their limited battery capacity. However, traditional malware detection systems and antivirus solutions based on matching signatures are limited to detection of only known malware, and hence, cannot deal with battery-depletion attacks. To meet this challenge, we propose to develop, implement, and evaluate a comprehensive framework, called *MODELZ*, that MOnitors, DEtects, and anaLyZes energy-greedy anomalies on small mobile devices. MODELZ comprises 1) a *charge flow meter* that allows infrequent sampling of energy consumption without losing accuracy, 2) a *power monitor*, in coordination with the charge flow meter, that samples and builds a power-consumption history, and 3) a *data analyzer* that generates a power signature from the power-consumption history. To generate a power signature, we devise and apply light-weighted, effective noise filtering and data compression, reducing the detection overhead significantly. The similarities between power signatures are measured by the χ^2-distance and used to lower both false-positive and false-negative detection rates. Our experimental results on an HP iPAQ running the Windows Mobile OS have shown that MODELZ achieves significant (up to 95 percent) storage-savings without losing detection accuracy, and a 99 percent true-positive rate in differentiating legitimate programs from suspicious ones while the monitoring consumes 50 percent less energy than the case of keeping the Bluetooth radio turned on.

Index Terms—Power-consumption history, charge flow meter, power signature, χ^2-distance, moving average filtering.

1 Introduction

In recent years, the worldwide market for small mobile devices, such as smartphones, PocketPCs, netbooks, and tablet PCs, has been growing and expanding dramatically. For example, worldwide sales of smartphones for 2008 alone reached 139 million devices, up 14 percent over those for 2007 [16]. Because of continued miniaturization, ubiquitous communication, and increasing computation power, mobile device users can now perform various online tasks, including Web browsing, document editing, multimedia streaming, and Internet banking, to name a few. At the same time, the growing usage of mobile devices for daily businesses and personal lives has also been attracting attention/interest of malware writers.

Early malware on PCs were written as pranks or for bragging rights, but have since been evolving with criminal or malicious intent. The motivation behind this type of attacks can be considered similar to that behind mobile malware for small devices. For instance, since the use of small mobile devices has become essential to our everyday businesses and personal lives, one seemingly legal way to sabotage a competitor's business is to reduce the usability of their mobile devices by depleting their batteries. Another way to inflict harm to competitors is to exploit the commonly used billing system for use of mobile devices. Excessive charges will be made to victims by sending them an excessive number of SMS messages [18]. This attack (i.e., a DoS attack) also drains their batteries very quickly [5]. Moreover, considering trends in the evolution of mobile malware, wilder, more diverse, and sophisticated mobile malware will likely appear, hence calling for effective preemptive ways to combat this type of threats.

The battery's limited operation time for mobile devices is an Achilles' heel for their portability and ubiquitous use. This limitation will significantly grow in future because not only has battery technology fallen behind Moore's Law, but mobile devices and software running thereon also demand more power for a longer time than the battery can deliver [32]. While most malicious code attacks on mobile devices target software resources such as infecting files and stealing privacy information [17], intentional abuse of hardware resources (e.g., CPU, memory, and battery) in many ways has become a significant, increasing threat [21], [8], [29]. Despite these problems, only limited research [4], [21], [29] has focused on the detection and prevention of battery-depletion attacks and a wider variety of attacks [5], [28] on devices.

1.1 Related Work

The most commonly-used technique for mobile malware detection is signature-based analysis. Signatures are created using static information (e.g., file names and code values),

- H. Kim and K.G. Shin are with the Department of Electrical Engineering and Computer Science, University of Michigan, 2260 Hayward Street, Ann Arbor, MI 48109-2121. E-mail: {hahnsang, kgshin}@eecs.umich.edu.
- P. Pillai is with the Intel Research Lab, 4720 Forbes Ave., Suite 410, Pittsburgh, PA 15213. E-mail: padmanabhan.s.pillai@intel.com.

Manuscript received 11 Apr. 2009; revised 5 Oct. 2009; accepted 1 July 2010; published online 17 Dec. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2009-04-0126. Digital Object Identifier no. 10.1109/TMC.2010.245.

thus becoming vulnerable to simple obfuscation, polymorphism [6], [24], and packing techniques [26]. The nature of signature-based detection that requires a new signature for every malware variant may make it very difficult to be deployed on small mobile devices with limited resources (e.g., battery energy) unless its high energy-consumption issue is resolved. Moreover, even "old" malware can harm new mobile devices unless their system has been properly patched in a timely manner.

Unlike signature-based detection, anomaly-based detection compares the definitions of the activities considered normal in a profile against the observed events to identify any significant deviation. The profile describes the normal behavior (e.g., users, hosts, applications, or network connections) [13], [36]. One common problem with anomaly-based detection, however, is that inadvertent inclusion of a malicious activity as part of the profile produces many false negatives (failure to identify malicious activities).

Similarly, behavior-based detection [11] uses behavioral signatures that describe any particular worm's behavior such as sending similar data from one machine to another, the propagation pattern, and the change of a server to a client. Such behaviors can be represented by a generic worm propagation model [11]. These behavioral signatures that are not sufficiently complex to reflect real-world computation activities can cause many false positives (incorrect identification of a benign activity as malicious). Also, the propagation of mobile malware via nontraditional exploit vectors such as SMS and Bluetooth [3], [12], [34], in conjunction with user mobility, renders network behavioral signatures ineffective.

1.2 Challenges

Since neither of these three methods provides comprehensive protection of small mobile devices against sophisticated energy-depletion attacks, we need objective countermeasures against them. Adapting multiple detection technologies, either separately or in combination, may have potential for broader and more accurate detection of increasing malware threats, but it would be difficult for resource-limited mobile devices to accommodate such blended technologies, without appropriate customization and optimization. Thus, there is an urgent need for detection and prevention of malware on mobile devices that can overcome the limitations of both signature- and behavior-based detection, while dealing with the unique features of the mobile operating environment. Listed below are the several requirements that have not yet been addressed by most current-generation systems for the protection of resource-limited mobile devices from energy-depletion attacks.

- *Detection accuracy*. Energy-depletion attacks are, in general, very difficult to detect. Signature-based detection yields many false negatives for such attacks and previously unknown threats, while behavior-based detection generates many false positives for erratic and benign application behavior. It is very important to keep both false-negative and false-positive rates below a certain acceptable threshold.
- *Resource usage on mobile devices*. Unlike resource-rich PCs, the detection and prevention system on a battery-powered mobile device should not consume too much of device resources, such as CPU, memory, and battery. The use of the resources should be kept to a minimum.
- *Measurement overhead*. High accuracy in measuring applications' power-consumption incurs high measurement overhead with a high resolution, expensive external tool (e.g., an oscilloscope) required. We, therefore, need an alternative, cost-effective method that can serve as a plug-in for mobile devices while still meeting the requirement of high-accuracy measurements.

1.3 Main Contributions

To address the above requirements, we propose a comprehensive framework, called *MODELZ*, to monitor, detect, and analyze new/unknown threats and energy-greedy anomalies on small mobile devices, with high accuracy and efficiency. MODELZ comprises a *charge flow meter* that offers a premise for infrequent sampling of power, a *power monitor*, in coordination with the meter, that samples and builds a power-consumption history, and a *data analyzer* that generates a *power signature* from the power-consumption history. The data analyzer then detects anomalies 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 three-fold.

- MODELZ abstracts the underlying application behavior by monitoring and recording usage of software and hardware resources. Resource usage is an abstraction of the underlying application behavior, captured by a power-consumption history. Analyzing the power-consumption history is the best way of detecting energy-greedy anomalies that most current solutions fail to detect. Also, this abstraction is effective in detecting previously unknown malware variants that share a common behavior exhibited by previously known malware, due to the fact that a new malware variant is created by adding new functions to existing malware [6].
- MODELZ is a "light" framework that comprises lightweight and effective noise-filtering and data-compression components, allowing computational processes to be faster and achieving significant storage and energy savings.
- MODELZ incorporates a portable *charge flow meter* that we will build with low-cost integrated circuit chips. This allows for fine-grained measurement of each application's power-consumption behavior while keeping its overhead to a minimum.

1.4 Organization

The rest of the paper is organized as follows: In Section 2, we discuss power measurement issues and then describe the design of a portable charge flow meter. 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 MODELZ and a custom worm emulator. Details of software

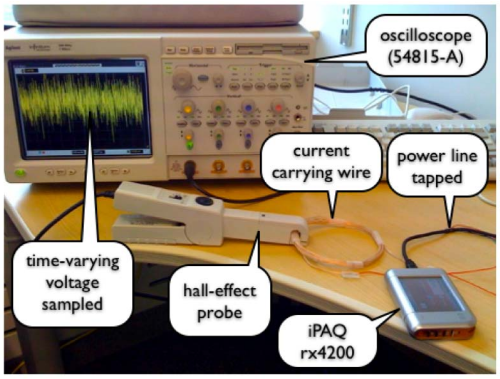


Fig. 1. Hardware power measurement: The supply wire from the positive terminal is formed into a 10-centimeter circle, and wound 10 times. This causes the field induced by the current to be magnified 10 times. The Hall-effect probe is hooked over the wire bundle, and converts the magnified current into a proportional voltage at a ratio of 0.1 volts/ampere.

and hardware power measurements and the mechanism for building a power-signature database are also presented. Section 5 evaluates the detection accuracy of MODELZ, including the monitoring overhead. The paper concludes with Section 6.

2 Energy/Power Measurement

Measurement methods for power/energy consumption by applications, in general, are classified into 1) software measurement of reading directly from a smart battery, which is coarse grained and cheap, and 2) hardware measurement of using an external tool/equipment such as an oscilloscope, which is fine grained, often costly, and/or cumbersome. In addition to the consideration of these two types of measurement methods, we also seek an alternative method that 1) can measure task energy consumption with the same precision as the fine-grained method, 2) is easy-to-use, and 3) does not require any fast, expensive hardware. For this purpose, we propose a portable hardware charge flow meter that, instead of directly measuring the rapidly fluctuating current drawn by the target platform, measures the total charge flow in conjunction with a measurement software tool that continuously updates applications' energy consumption profiles. In particular, the software tool generates per-task energy-consumption profiles and can thus handle multiple concurrent applications.

We will first present instantaneous power measurement methods (based on software and hardware), then describe a detailed design of the charge flow meter.

2.1 Instantaneous Power Consumption Measurement

The energy usage by each application can be 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 a constant interval, Δt. On many platforms, this can be accomplished in software by polling the battery status. For example, on Windows-based devices, one can use the `GetSystemPowerStatusEx2()` function in Coredll.lib to retrieve complete battery status information, including AC

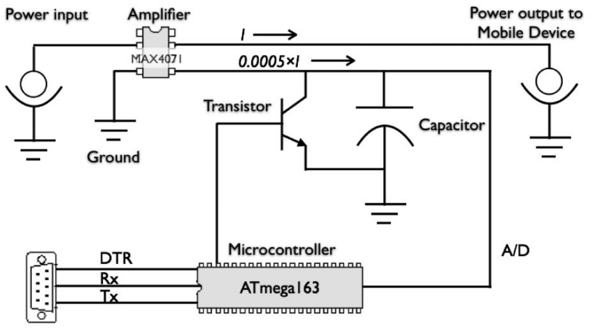


Fig. 2. Charge-flow measurement hardware: This illustrates the main components of our charge-flow measurement circuit. The actual circuit has additional discrete components for regulating power and limiting currents, and uses a second A/D channel on the microcontroller to measure the voltage of the power supply.

line status, I_t, and V_t. As V_t will typically remain constant over the relatively short time intervals in which we are interested, we can use a single measured battery voltage sample, V_c, and estimate the total energy consumption over n samples as $V_c \sum I_t \Delta t$. This approach, however, is limited by the accuracy and update rate of the reported battery status (over which we have no control), and the frequency of sampling.

To obtain very accurate measurements of energy consumption, we can use a digital oscilloscope, such as the Agilent Infiniium 54851-A [33], pictured in Fig. 1 measuring the energy consumption of an iPAQ rx4200. The oscilloscope is capable of high-speed acquisition of 1 Gigabit samples per second, peak detection and mean computation over long intervals, and can measure and integrate the current and voltage supplied to a device, when equipped with current probes. The current probe uses the Hall-effect sensing to measure the field generated around a current-carrying wire, and generate a voltage, V_t, proportional to the current. Therefore, the total energy usage over n samples is calculated as $V_c \sum V_t \Delta t$. The digital scope can be set to perform most of this computation, and makes it fairly simple to synchronize the measurements with the process execution.

Unless otherwise specified, readings of the integrated samples are taken every 10 ms from the external hardware and every 100 ms from the software.

2.2 Portable Charge Flow Meter

We have designed a small charge-flow measurement device, which is a proof-of-concept prototype, that can accurately measure energy consumption without requiring high-sampling rates or costly equipment (Fig. 2). Our design is based around the Maxim MAX4071 [22], a low-cost integrated circuit that essentially implements a current sensor. When connected between the power supply or battery and the device to be measured, this chip produces a small current on an output pin that is proportional to the current drawn by the device. Due to the fast response time (less than $1\ \mu s$) of this chip, even high-frequency changes in power draw are reflected in the output. This output leads to a resistive load; since $V = IR$, by measuring the voltage across the resistor and scaling by the right constant, one could determine the instantaneous current draw, I_t, of the mobile device.

The novelty in our approach lies with the use of MAX4071 to charge a capacitor that acts as an analog integrator, instead of sending the output current through a resistor. Reading the current directly requires a high-sampling rate for accurate measurement of the fluctuating values. This is expensive in terms of equipment and power requirements given a processing budget. The capacitor solution automatically sums up all of the fluctuating current values over time. Since $V_{cap} = C_{cap} \int I_o dt$, where I_o is the output current of the MAX4071 and C_{cap} is the value of the capacitor, measuring the voltage on the capacitor lets us compute the energy drawn over a time interval as KV_cV_{cap}, where V_c is the supply voltage, and K the calibration constant. The constant K depends on the size of the capacitor and the exact ratio between the MAX4071 output current and I_t (nominally, this is 1:2000). A transistor is used to discharge the capacitor before taking a new measurement. Our measurement device is calibrated by employing a fixed resistive load and comparing the measurements with those taken by a multimeter to determine the value of K. The K constant, therefore, converts voltage to the energy consumed by the mobile device.

Since all of the high-frequency changes in current are accounted for in the capacitor voltage, we can obtain accurate measurements of energy consumption with very infrequent sampling of the capacitor voltage. Thus, we do not need expensive equipment or high-speed data sampling. Rather, for data acquisition and energy computations, we use an Atmel AVR series 8-bit system-on-chip type microcontroller [7] that includes internal clock generators, flash/RAM/EEPROM memories, multichannel 10-bit analog to digital (A/D) converters with internal reference voltage, and serial ports in a single IC. We use one A/D channel to measure the capacitor voltage, and another to measure the supply voltage. A digital output is used to control the transistor that discharges the capacitor between measurements. The serial port serves two purposes: the data lines are used to output the measured values to our measurement software, while the additional signaling (e.g., DTR) pins are used as a high-speed trigger to synchronize measurements with task execution on the target platform. The internal memories are used to store the calibration constant, and to store or aggregate multiple measurements.

Our device has been designed to accurately measure energy consumption over intervals ranging from 1 to 50 ms, corresponding nicely to typical scheduling time slices for task execution. The software framework on the mobile device indicates the specific interval of measurement by toggling the trigger pin. Our device reacts very quickly to this input, clearing and starting the accumulation of charge on the capacitor within 20 μs of the trigger edge. Likewise, it stops accumulation and captures the voltage on the capacitor within 26 μs of the stop trigger. The total error in the measurement window accounts for less than 1 percent error in the final energy values. The actual A/D conversion, adjustment for calibration, and subsequent transfer of data over the serial port incur the greatest latencies, on the order of 15 ms. However, as we use only about one quarter of the RAM available on the microcontroller, a slight upgrade to the software to buffer measurements on-chip allows bursts of up to a few hundred consecutive measurements that are spaced only 100 μs apart.

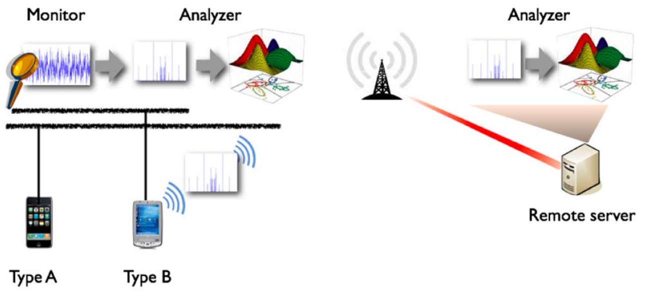


Fig. 3. The MODELZ architecture: In Type A, both the monitor and the analyzer are performed on a mobile device, while in Type B, the analyzer is performed separately on a remote server/data-sync PC.

This low-cost charge flow meter can be embedded into a smart battery, allowing MODELZ to measure per-task energy consumption on a mobile device with help of a kernel-level module like the one shown in Odyssey [25], [14]. The module has a hook inside the scheduler to set an output pin (e.g., DTR signal on the serial port) to high or low. Whenever either a context switch or a specific system call takes place, the module triggers the charge flow meter to end the measurement interval or read the energy consumed by a task during the interval. The sequence of per-task energy consumptions, therefore, represents the behavior of a process, i.e., a power-consumption history. Conversely, the module can be set to make infrequent measurements; it stays blocked between measurements, while the charge flow meter stays in a low-power sleep mode. This reduces additional computational or energy overheads.

3 MODELZ

This section describes the MODELZ architecture and its key components.

3.1 The Architecture

MODELZ consists of two agents, a power monitor and a data analyzer, as illustrated in Fig. 3. The two agents reside either in combination or separately. The power monitor operates on each mobile handset, 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 handset (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 handset 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 the battery energy because most SmartPhone batteries 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 electronic components powered by the external supply (or the battery), so that accurate measurement of the power consumption can still be achieved. 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

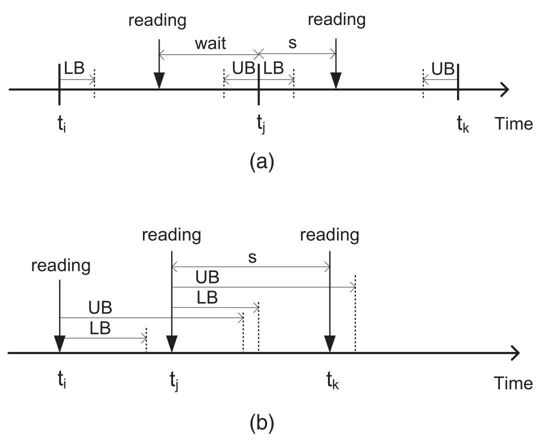


Fig. 4. Power-reading methods: regular reading has a fixed interval based on t, while irregular reading-points are not fixed. In both cases, reading-points are randomly chosen in between the lower bound and the upper bound. A timer object is set to a time period of s. (a) Regular reading. (b) Irregular reading.

Anand et al. [2], fetching 32 Kbytes of data via a WiFi radio in active mode on a handset consumes 1.4 Joules less energy than reading the data from a local microdrive in standby mode. In addition, an estimated power-consumption history of 1 Kbyte data can be transmitted in a single packet via a WiFi or EDGE/3G network.

3.2 The Power Monitor

The power monitor, which reads the power drawn from the battery on a handset, that flows through the charge flow meter, is designed to capture power-/energy-consumption anomalies exhibited by applications. It is responsible for detecting a surge in power/energy consumption and taking samples of the consumption. Next we will describe each of these components.

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 taking power-consumption 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 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 randomize the time to take the next sample. If the sample-reading time is unpredictable, then it will be difficult for the malware to evade detection. However, making reading-points too random will make it difficult to capture energy-consumption anomalies.

To make this trade-off, we devised two reading methods: regular and irregular. Regular readings, as illustrated in Fig. 4a, occur at fixed intervals at which the lower bound (LB) and upper bound (UB) are defined, and a reading-point is randomly chosen in between. In other words, LB and UB specify an interval in which a reading-point is randomly chosen. In each interval, after a reading, the power monitor waits for the next base and then randomly chooses the next reading-point. Clearly, the smaller (UB-LB) is, the narrower the random-choice space, and thus, the more regular the reading frequency. That is, reading-points are likely uniformly-distributed. An irregular reading, on the other hand, does not wait in order to calculate the next reading-point, as illustrated in Fig. 4b. 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.

In practice, the power monitor creates a timer object (corresponding to s Fig. 4) 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 state. This function takes a certain amount of time to complete, starting 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 our implementation, the time period amounts to more than 30 milliseconds, limiting the sampling rate.

3.2.2 Detection

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

$$\left(\frac{X}{Y} - 1\right) > \delta, \tag{1}$$

where δ is a given threshold, X is an observed power level, and Y is the power level specified in the system-power state profile which 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 produce 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, generating false alarms. To reduce these false alarms, the threshold is set high enough to withstand those spikes, but not insensitive 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, switching 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 ON state in which no explicit applications run. When $\delta < 0.2$, peaks of the spikes are detected, generating 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 that reads directly from the battery relies on a soft real-time constraint. Once a true alarm is raised, the power monitor starts taking samples of 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 the energy-costlier. 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, eventually being added to the corresponding power signature. Note that one application is executed at a time in these experiments.

Alternatively, with the charge flow meter applied, the power monitor in coordination with the kernel-level module measures per-task energy consumption while multiple applications are being executed. The power monitor, then, sorts out these energy consumptions with respect to processes. Mapping tasks to a corresponding process is resolved in the kernel-level module by tracking process identities (PIDs) while the context switch occurs. This way, a power-consumption history per process can be built.

3.3 The Data Analyzer

The data analyzer receives the power-consumption history from the power monitor and extracts a unique pattern from the history, generating a power signature. This power signature is then compared against the database of signatures generated a priori. To generate power signatures, the data analyzer uses two data-processing software components: noise filtering and data compression. 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 data set history. The moving average filter removes high-frequency noises from the data set, 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

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

This calculation starts from $i = k+1$ and continues until $i = n-k$; the first and 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 recent fluctuations in the data set. The impact of k on the pattern associated with 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 samples or an exponential moving average filter in which weights decrease exponentially from the center), because a simple filter works just as well as, or even better than, complicated ones (i.e., the implementation incurs less processing overhead) [10].

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 the matching process. Second, it is important to make economical use of memory in a mobile device. To reduce the size of a power-consumption history, a simple and powerful one-way compression is proposed. By applying Algorithm 1, local jitters are effectively suppressed and compressed. As a result, a compact power signature can be derived, thereby achieving substantial savings in both memory space and processing time. The effectiveness of this algorithm will be confirmed experimentally in Section 5.

Algorithm 1. A compression algorithm.

```
1: Input: $S_p(n)$: an $n$-length power-consumption history
2: Input: $m$: look-ahead samples
3: Input: $\delta_c$: a threshold
4: Output: $S_e(k)$: a $k$-length power signature
5: while $i \leq (n-m)$ do
6:   Fetch $m$ samples from $S_p(i)$;
7:   Compute $N \sim (\mu, \sigma^2)$ of $m$;
8:   if $\sigma > \delta_c$ then
9:     $S_e(j) \leftarrow \mu$; /*compressing history*/
10:    $j \leftarrow j+1$
11:  else
12:    $S_p(i : i+m) \leftarrow S_e(j : j+m)$; /*copying history*/
13:    $j \leftarrow j+m$
14:  end if
15:  $i \leftarrow i+m$
16: end while
```

3.3.3 Signature Matching

To measure the similarity between two power signatures, the χ^2-distance [9] between them is calculated as

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

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, the χ^2-distance-based techniques have been used in diverse areas, such as scene-change detection in image sequences [27], [15] and anomaly detection [36]. In addition, experimental results [20] 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 [30].

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

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

In some cases, two power signatures that comply with the same pattern can be skewed due mainly to 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 BF 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 shifted 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. Besides the incremental parameter, the proportion of samples for comparison are correlated with the confidence in the results (e.g., more than 90 percent). The BF comparison is efficient, especially in the case of small reading delays in the power monitor. Alternatively, the FFT method converts the time-domain representations of samples into their frequency-domain representation. In practice, this method facilitates distance calculation in that a large portion of converted samples in two similar signatures are likely to have the same constant frequency components, offsetting the complexity of the FFT computation. The performance comparison of the two methods will be presented in Section 5.

3.3.4 The Handset User's Response to the Analysis

The analysis results from the data analyzer pinpoints 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 identified. To address this limitation, each signature is labeled as either legitimate or malicious by prompting the user to enter an 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 classification (e.g., due to outliers), thereby reducing both false-positive and false-negative detection rates.

4 Implementation

Since only a few mobile handset malware are available publicly, evaluation on MODELZ's detection and prevention of malware is limited by this (un)availability. To overcome this difficulty, we specify and implement proof-of-concept malicious programs on battery depletion. Since the most energy-consuming activity on our handset device

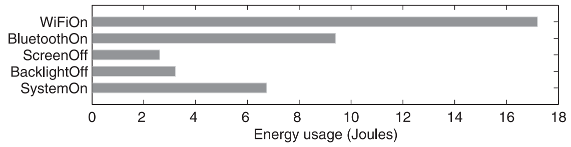


Fig. 5. Energy usage of system states for 10 seconds on the HP iPAQ rx4200. In the SystemOn state, everything is ready for execution, while everything but the backlight is ready in the BacklightOff state. In the ScreenOff state, the LCD screen is off. For the BluetoothOn and WiFiOn states, the Bluetooth and WiFi radios are turned on exclusively for each state.

is the use of WiFi radio—as shown in Fig. 5, the handset 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, a program we create also relies on the WiFi radio to drain the battery. In addition to the program, we extended the Symbian-based Cabir source code for three extra variants to run on Windows Mobile by adding or deleting functionalities.

We have carefully specified the behavior of the programs. The specification of the behavior is sufficient enough to highlight MODELZ's efficacy, efficiency, and feasibility.

Here we describe two types of programs: the one aims to deplete the battery and the other emulates the behavior of four mobile worms on an HP iPAQ rx4200 running the Windows Mobile 5 OS. 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 Attacks

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 handset with the WiFi-enabled 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 actually, it is still active and even deprived of doze mode, resulting in 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 MODELZ in Section 5.

4.2 Proof-of-Concept Mobile Worms

We consider four well-known mobile worms: Commwarrior, Cabir, Mabir, and Lasco. Behaviors of these worms are similar to each other since they are in the same family. As a whole, when any of these worms is installed, it scans specific system directories/address book, creates a file, and/or sends certain files to a specified destination. The great similarity in the behavior of these worms should be effectively discerned in MODELZ.

We implement an emulator that emulates the behavior of the four worms, as illustrated in Fig. 6. Scanning for

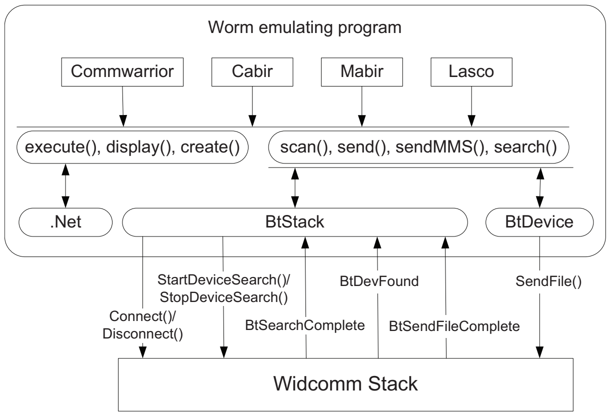


Fig. 6. Software architecture of a worm emulator.

Bluetooth-enabled devices and transmitting a file (regarded as worm payload) via Bluetooth are part of the basic capability of many of known mobile worms. The emulator program uses the BTAccess.NET v3.0 library [1] which supports the Widcomm Bluetooth stack mainly including BtStack and BtDevice classes. 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 an in-progress scan, 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 BtSearch-Complete event indicating the completion of the search, the program stops searching for devices. In order to send a worm file 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.

s1. *execute()*: starts a worm-behavior emulation.

s2. *display()*: opens a window and displays a message on the window. Cabir and Lasco exhibit this behavior to identify themselves.

s3. *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.

s4. *scan()*: searches for Bluetooth-enabled devices nearby, using the service discovery application profile [31] defined in the Widcomm Bluetooth stack. This profile relies on Service Discovery Protocol [31] to discover devices.

s5. *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 [31] in the profile is used to push the file data to nearby mobile devices.

s6. *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.

s7. *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.

TABLE 1
The Behavior of Worms

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	$s_1 s_2 s_3 s_4 s_5 s_7$

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, MODELZ effectively identifies such power signatures.

The action sequence for each worm is presented in Table 1, showing common subsequences. For instance, all the worms have a common subsequence, $s_1 s_3 s_5$. However, their power signatures can be different significantly 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 intents. Also, the mobile worms described above and legitimate Bluetooth file transfers have a common behavior, but have different intents. First, we characterize malicious applications and create 11 power signatures from them.

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, resulting in

two different power signatures. The beginning of the program run is the most important in the sense that it should be identified prior to its execution of the main damaging functions. In case the program has escaped detection, the middle of the program will be used to capture even in the aftermath of a possible system infection.

2. The WiFi fake. 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.
3. 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 power signature is extracted while the two programs are running.
4. The four mobile worms. The execution of these worms results in four power signatures.
5. A DoS-attack-like Bomber. This program bombards the handset with 1Kbyte- and 2Kbyte-size data via WiFi (e.g., ping -s 1,024/2,048). In practice, a stream of 2 Kbyte-data froze the handset after 30 seconds. Two different power signatures are extracted for the different size packets.

Second, we characterize legitimate applications and create eight power signatures from them, as follows:

6. 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 315 bps and WMV7 at 704 bps. Power-consumption histories are recorded at the beginning and end of 5 seconds of execution for each codec, resulting in four different power signatures.
7. Bluetooth and WiFi file transfers. A 10 Mbyte-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.
8. A users' handset-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.

Note that when a file is transferred to a Bluetooth-enabled device for the first time, it will be asked if the file is acceptable. If yes, the file is received; otherwise, it is rejected. The authorization request can be disabled in Bluetooth service settings provided on the file transfer menu. When the required authorization is disabled, the file is received without asking for permission. In such a case, worms are allowed to spread via Bluetooth without user's awareness.

As briefly mentioned earlier, we consider the difficulty of taking samples on time at a constant interval at the software level on the Windows Mobile 5 OS; each measurement is delayed, which in turn yields less samples. The number of samples is inversely proportional to the degree of the resource utilization by processes (including the software measurement process), such as CPU load and driver workloads. However, the difference between the observed and the expected numbers of samples can be small if the measurement time window is small, and interference caused by the processes to be characterized is also a positive effect because we can consider it as a part of the process signature; if the difference arises beyond a threshold, then the two involved signatures are regarded as mismatched.

5 Evaluation

The metrics used to indicate the detection accuracy include pinpoint accuracy 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.

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. A total of 20 rounds are made; the first five rounds resulted 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.

In this section, we first assess the system parameters defined in MODELZ and then evaluate the detection accuracy with the optimal values of the system parameters found. Next, we analyze the MODELZ's performance issues, followed by the analysis of the power monitor's overhead.

5.1 Assessment of System Parameters

Signature generation. A power-consumption history is produced while running an application on a handset. The power-consumption history is transformed into a power signature via the moving average filter and the data compression. 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 315 bps, using the WMV 9 codec. Fig. 7a 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. 7b. The application of the compression technique results in a power signature as shown in Fig. 7c.

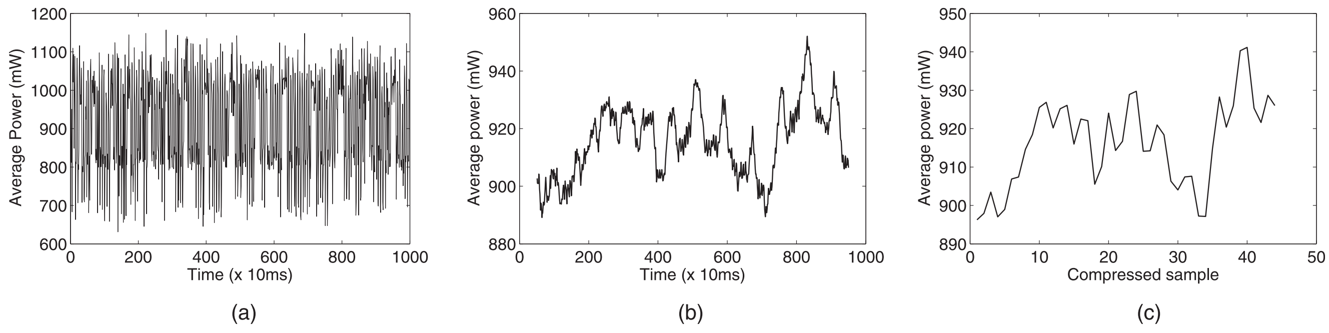


Fig. 7. Power-signature generation. (a) Raw samples. (b) 50pt-moving average filter applied. (c) Local jitter removed.

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 $\delta_c = 0.05$ 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. The corresponding result shows the correlation between the window size and PA, the 23- or 24-point moving average filter for the 1,000-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.

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. When $\delta_c > 0.05$, more than 95 percent storage-savings is achieved. When $\delta_c = 0.06$, the lookahead parameter correlates more prominently with the compression ratio than PA. 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 comparison because a large portion of samples are converted into constant frequency components.

Accordingly, when $\delta_c = 0.06$, 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 $\delta_c = 2$, 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 MODELZ. Fig. 8 shows power-consumption patterns with the WiFi faker and the dummy program executed separately, and in combination. Each of 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 from Table 2, the WiFi faker is identified with 100 percent accuracy against DB_{8+WF} that includes the WiFi faker's signature plus 8 signatures created from the legitimate application group, and detected 100 percent accuracy with DB_{8+DM} (including the dummy program's

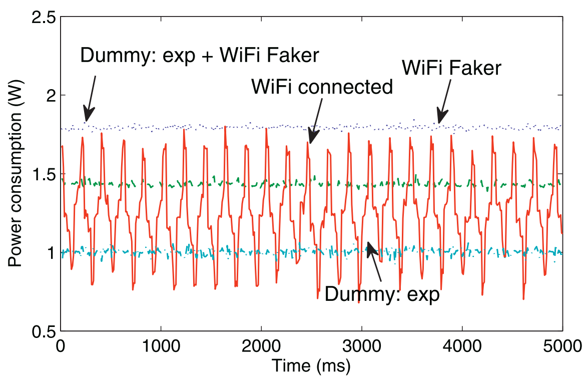


Fig. 8. Comparison of the power consumption with WiFi connection, the WiFi faker, a dummy program, and a combination of the last two; the WiFi device is in power-saving mode in which the WiFi device dozes after every beacon interval.

TABLE 2
Detection of Battery-Depletion Attacks: WF, DM, and CB Denote Signatures of the WiFi Faker, the Dummy Program, and the Combination of These Two, Respectively

DB Type\E-Greedy	WiFi faker	Dummy	Combo
DB_{8+WF}	100%	0%	100%
DB_{8+DM}	100%	100%	100%
DB_{8+CB}	100%	0%	100%
$DB_{8+WF+DM}$	100%	100%	100%
$DB_{8+WF+CB}$	100%	0%	100%
$DB_{8+DM+CB}$	100%	100%	100%
$DB_{8+WF+DM+CB}$	100%	100%	100%

DB_8 is a baseline that is preloaded with 8 signatures created from the legitimate application group.

signature plus 8) and DB_{8+CB} (including the combination of the two plus 8) because the WiFi faker and the combination of the two applications have common power-consumption patterns. The dummy program, on the other hand, does not seems to influence the detection performance. The reason for this is that the dummy program's signature is misidentified as that of Windows Media Player belonging to the legitimate group, due to the shorter similarity distance between them. However, since whenever new unknown signatures are detected, the mobile handset user is prompted to confirm that the corresponding program running is illegitimate, thereby detecting such dummy programs afterward. Also, the combination is always detected with any of DB_{8+WF}, DB_{8+DM}, and DB_{8+CB}.

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 MODELZ's ability to detect previously unknown worms whose signatures are similar to those of previously known worms, the four customized worms were divided into two groups: known-worm and unknown-worm groups. Worms in the known-worm group were executed five 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 and Mabir have a similar power-consumption pattern, as do Mabir and Lasco. Interestingly, however, existence of asymmetric similarity has been observed. That is, Commwarrior is detected with 73 percent accuracy against DB_{8+C}, while Cabir is detected with 47 percent against DB_{8+W}. This result implies that Commwarrior is close to the Bluetooth file-transfer program in terms of the behavior and Cabir is in the middle of the two. For this reason, although Commwarrior is misidentified as Cabir, it belongs to the malicious group; it would otherwise belong to the legitimate application group. The relatively low detection accuracy is observed, as shown in Table 3, due to the asymmetric similarity as well. This asymmetry, however, diminishes as the database is populated and diversified. That is, the more diverse worms collected and added to the database, the higher detection accuracy for unknown worms. For instance, the detection accuracy for Commwarrior and Lasco is improved even with a partially filled database, thus making this technique attractive for resource-limited handsets.

TABLE 3
Detection of Previously Unknown Worms: C, M, W, and L Denote Signatures of Cabir, Mabir, Commwarrior, and Lasco, Respectively

DB Type\Malware	Cabir	Mabir	CommW.	Lasco
DB_{8+C}	87%	93%	73%	87%
DB_{8+M}	93%	100%	80%	93%
DB_{8+W}	47%	93%	80%	87%
DB_{8+L}	87%	93%	80%	93%
DB_{8+C+M}	93%	100%	80%	93%
DB_{8+M+W}	93%	100%	80%	93%
DB_{8+W+L}	87%	93%	80%	93%
$DB_{8+C+M+W}$	93%	100%	80%	93%
$DB_{8+M+W+L}$	93%	100%	80%	93%

DB_8 is preloaded with 8 signatures created from the legitimate application group.

TABLE 4
Comparison of the Overall Detection Accuracy, Based on Hardware Measurement

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

TP (true-positives) equals PA (pinpoint-accuracy where worms are correctly identified) plus TP (exclusive true-positives where worms are correctly classified). It is also compared with respect to FN (false negatives) and FP (false positives).*

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 percent 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 trace for each codec—were identified with 100 percent 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 percent, achieving a 98 percent 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 where the filter and compression technique are applied, our compression technique is also effective for optimizing memory usage, without degrading the accuracy (the effect of the compression

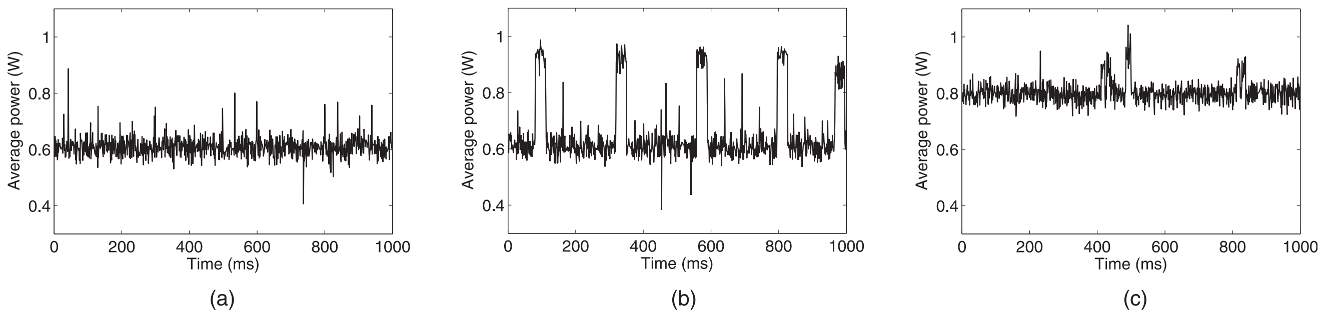


Fig. 9. Comparison of overhead for executing the power monitor. (a) Baseline: ON state. (b) The power monitor sampling every 200 ms. (c) The Bluetooth radio is turned on.

technique will be analyzed shortly). In case of C4 where the number of samples to be matched is 95 percent of the total samples, our detection scheme achieves a 99 percent true-positive rate, while decreasing the false-negative rate down to 0 percent. In the case of C5 where the FFT is applied, the overall accuracy is improved, with the false-negative rate reduced to 2 percent in comparison with C1 which only achieved 5 percent.

We applied a simple and powerful compression technique. This technique allows the power signature to be compressed by a factor of 21 without losing the detection accuracy. 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 percent samples matching) requires less CPU time than the case without the compression by 71 percent. 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 percent of CPU time required. 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.

5.4 Overhead of the Power Monitor

The power monitor must have minimum overhead for executing itself. In order to assess its overhead, we measure the power consumption of the iPAQ device via the oscilloscope with the device powered by the AC adapter; the battery that is fully charged is still in the device. Note that the GetSystemPowerStatusEx2 function applied in the implementation of the power monitor reads and retrieves the battery information that the battery pack offers; otherwise, the power monitor requires a customized hardware module including a capacitor to keep current drawn from the power source. We configure the power monitor with a sampling rate set to five samples per second. No explicit applications run on the iPAQ device and the brightness of the backlight of it is tuned to maximum on its scale. For the comparison purpose, the energy consumed with the Bluetooth radio turned on is also measured in the same configuration. This measurement procedure is repeated 300 times and then the average energy consumption for each measurement is calculated.

Fig. 9 shows comparison between the power monitor and the Bluetooth radio with respect to their power consumption. Samples are taken at 1 KS/s. Fig. 9a shows a system's power-consumption pattern with the system in the idle state (note that no explicit applications run in that state), in which an average of 608.9 mJ per second is consumed, while the system with the Bluetooth radio turned on consumes an average of 806.5 mJ as shown in Fig. 9c. As a result of one measurement, the power monitor consumes average 9.6 mJ, which is estimated to take 31 milliseconds. This accounts for 1.6 percent energy budget for 1 second for the ON state as a baseline (i.e., 656.7 mJ), which is trivial. In the case of taking 20 samples, the overhead for executing the power monitor corresponds to that of the system with the Bluetooth radio turned on; taking 10 samples per second, which is sufficient to capture anomalies, is equivalent to the estimated loss of 15 minutes of the battery lifetime, but the baseline is otherwise estimated to last 6 hours 14 minutes with the battery capacity of 1,200 mAh, while the Bluetooth-enabled system shortens the battery lifetime by 30 minutes.

6 Conclusion

We now discuss a few issues we encountered, and make concluding remarks.

6.1 Discussion

MODELZ abstracts the underlying application behavior by monitoring and recording usage of software and hardware resources. This resource usage represents the abstraction of the underlying application behavior that is captured by a power-consumption history. Analyzing the power-consumption history is the best way of detecting various mobile worms as well as energy-greedy anomalies. Thus, we believe MODELZ should work for any mobile worms (see [3] for a comprehensive survey of mobile worms). For instance, trojans, such as Skulls (2005), Drever (2005), Locknut (2005), and Redbrowser (2006), exploit SMS and MMS messages to propagate a copy of themselves, and/or overwrite ROM binaries to crash the OS. This anomalous behavior, i.e., sending of excessive messages and unauthorized access to ROM, is easily captured in a power-consumption history. Also, MODELZ can detect an emerging class of malware, such as Cardtrap (2005), Win32.Rays, Win32.Padobot.Z, and Crossover (2006). The "crossover" infector can spread from mobile devices to desktop PCs, or vice versa, deleting all files in the "My Documents" directory, copying itself to the system directory, and placing a link to itself in the startup directory. This series of behaviors, however, differs from that of normal behaviors, and will also be captured by extending the monitoring period.

Conversely, only a few worm samples are publicly available for research. Testing detection systems with real-world malware is necessary to evaluate the efficacy 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 [35], [23], [24] to study the efficacy 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.

6.2 Concluding Remarks

Mobile handsets must be protected against malware, especially those with the goal of dramatically reducing their battery lifetime. In this paper, we first have addressed how one can determine a task's energy consumption, by developing and implementing a low-cost, easy-to-use portable device that overcomes many of the shortcomings of existing measurement techniques. We have then presented MODELZ, 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 four 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 percent storage savings,
2. deriving the efficacy of detecting energy-greedy anomalies as well as unknown malware over our representative test set up to a 99 percent true-positive rate,
3. providing precise measurements over a range of 1-50 ms via the discharge flow meter, corresponding to the typical scheduling time slice for task execution, and
4. reducing the error of accuracy in estimating task energy consumption within 5 percent of measured energy values in most cases.

Furthermore, the overhead of executing the power monitor at a reasonable sample rate requires 50 percent less additional energy than keeping the Bluetooth radio turned on. In summary, MODELZ offers a effective solution to the tricky issue of securing battery-powered mobile devices, and is further enhanced with the support of a wireless server/PC-side that provides a comprehensive data analysis.

Acknowledgments

The work reported in this paper was supported in part by the US National Science Foundation under Grant No. CNS 0523932, Samsung Electronics, and Intel Corporation. A subset of this article was presented at ACM/USENIX MobiSys 2008 [19].

References

[1] Btaccess.net, http://www.high-point.com, 2011.
[2] M. Anand, E.B. Nightingale, and J. Flinn, "Ghosts in the Machine: Interfaces for Better Power Management," *Proc. Second Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '04),* pp. 23-35, 2004.
[3] A. Bose and K.G. Shin, "On Mobile Viruses Exploiting Messaging and Bluetooth Services," *Proc. SecureComm and Workshop,* pp. 1-10, Aug. 2006.
[4] T.K. Buennemeyer, M. Gora, R.C. Marchany, and J.G. Tront, "Battery Exhaustion Attack Detection with Small Handheld Mobile Computers," *Proc. IEEE Int'l Conf. Portable Information Devices (PORTABLE '07),* pp. 1-5, May 2007.
[5] J. Cheng, S. Wong, H. Yang, and S. Lu, "SmartSiren: Virus Detection and Alert for Smartphones," *Proc. Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '07),* pp. 258-271, 2007.
[6] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, and R.E. Bryant, "Semantics-Aware Malware Detection," *Proc. IEEE Symp. Security and Privacy (SP '05),* pp. 32-46, May 2005.
[7] Atmel Corporation, http://www.atmel.com/products/avr, 2011.
[8] D. Dagon, T. Martin, and T. Starner, "Mobile Phones as Computing Devices: The Viruses Are Coming," *Pervasive Computing,* vol. 3, no. 4, pp. 11-15, Oct. 2004.
[9] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification,* second ed. Wiley-Interscience, 2001.
[10] R.D. Edwards and J. Magee, *Technical Analysis of Stock Trends,* eighth ed. AMACOM, 2001.
[11] D.R. Ellis, J.G. Aiken, K.S. Attwood, and S.D. Tenaglia, "A Behavioral Approach to Worm Detection," *Proc. WORM: ACM Workshop Rapid Malcode,* pp. 43-53, 2004.
[12] W. Enck, P. Traynor, P. McDaniel, and T. La Porta, "Exploiting Open Functionality in SMS-Capable Cellular Networks," *Proc. 12th ACM Conf. Computer and Comm. Security (CCS '05),* pp. 393-404, 2005.
[13] H.H. Feng, O.M. Kolesnikov, P. Fogla, W. Lee, and W. Gong, "Anomaly Detection Using Call Stack Information," *Proc. IEEE Symp. Security and Privacy (SP '03),* May 2003.
[14] J. Flinn and M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Applications," *Proc. 17th ACM Symp. Operating Systems Principles (SOSP '99),* pp. 48-63, 1999.
[15] R.M. Ford, C. Robson, D. Temple, and M. Gerlach, "Metrics for Scene Change Detection in Digital Video Sequences," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems (ICMCS '97),* pp. 610-611, 1997.
[16] Gartner, http://www.gartner.com/it/page.jsp?id=910112, 2011.
[17] Symantec: Making Handheld Security a Priority, http://www.symantec.com/norton/products/library/article.jsp?aid=handheld_security, 2011.
[18] M. Hypponen, "Malware Goes Mobile," *Scientific Am.,* Nov. 2006.
[19] H. Kim, J. Smith, and K.G. Shin, "Detecting Energy-Greedy Anomalies and Mobile Malware Variants," *Proc. Sixth Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '08),* pp. 239-252, June 2008.
[20] *Real-Time Vision for Human-Computer Interaction,* B. Kisacanin, V. Pavlovic, and T.S. Huang, eds., first ed. Springer, 2005.
[21] T. Martin, M. Hsiao, D. Ha, and J. Krishnaswami, "Denial-of-Service Attacks on Battery-Powered Mobile Computers," *Proc. Second IEEE Ann. Int'l Conf. Pervasive Computing and Comm. (PerCom '04),* p.p. 309-318, 2004.
[22] MAXIM, Max4071, http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3387, 2011.
[23] J.W. Mickens and B.D. Noble, "Modeling Epidemic Spreading in Mobile Environments," *Proc. Fourth ACM Workshop Wireless Security (WiSe '05),* pp. 77-86, 2005.
[24] J.A. Morales, P.J. Clarke, Y. Deng, and B.M. Golam Kibria, "Testing and Evaluating Virus Detectors for Handheld Devices," *J. Computer Virology,* vol. 2, no. 2, pp. 135-147, Nov. 2006.
[25] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker, "Agile Application-Aware Adaptation for Mobility," *Proc. ACM Special Interest Group on Operating Systems (SIGOPS) Rev.,* vol. 31, no. 5, pp. 276-287, 1997.

[26] M.F.X.J. Oberhumer, L. Molnar, and J.F. Reiser, "UPX: The Ultimate Packer for Executables," http://upx.sourceforge.net, 2011.

[27] N.V. Patel and I.K. Sethi, "Compressed Video Processing for Cut Detection," *Vision, Image and Signal Processing,* vol. 143, no. 5, pp. 315-323, Oct. 1996.

[28] M. Pirretti, S. Zhu, N. Vijaykrishnan, P. McDaniel, M. Kandemir, and R. Brooks, "The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense," *Int'l J. Distributed Sensor Networks,* vol. 2, no. 3, pp. 267-287, Sept. 2006.

[29] R. Racic, D. Ma, and H. Chen, "Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phone's Battery," *Proc. SecureComm and Workshops,* pp. 1-10, Sept. 2006.

[30] C. Reyes-Aldasoro and A. Bhalerao, "The Bhattacharyya Space for Feature Selection and Its Application to Texture Segmentation," *Pattern Recognition,* vol. 39, no. 5, pp. 812-826, May 2006.

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

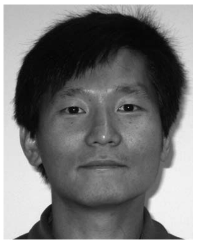
[32] T. Starner, "Thick Clients for Personal Wireless Devices," *Computer,* vol. 35, no. 1, pp. 133-135, 2002.

[33] Agilent Technologies, 54815a Infiniium Oscilloscope Spec, http://www.home.agilent.com/agilent/product.jspx?pn=54815A, 2011.

[34] S. Toyssy and M. Helenius, "About Malicious Software in Smartphones," *J. Computer Virology,* vol. 2, no. 2, pp. 109-119, Nov. 2006.

[35] G. Yan and S. Eidenbenz, "Bluetooth Worms: Models, Dynamics, and Defense Implications," *Proc. 22nd Ann. CS Applications Conf. (ACSAC '06),* pp. 245-256, Dec. 2006.

[36] N. Ye and Q. Chen, "An Anomaly Detection Technique Based on a Chi-Square Statistic for Detecting Intrusions into Information Systems," *Quality and Reliability Eng. Int'l,* vol. 17, no. 2, pp. 105-112, Oct. 2001.



Hahnsang Kim received both the BS and MS degrees in computer science and engineering from Korea University, Seoul, in 1998 and 2000, respectively, and the PhD degree in computer science from the Institut National des Télécommunications (INT), France, in 2006. He is the senior research fellow of the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, since 2007. From 2000 to 2005, he was a member of the Planète team at the Institut National de Recherche en Informatique et en Automatique (INRIA), Sophia Antipolis, France, the expert engineer thereof from 2000 to 2003, and then the research assistant until 2005. He was also affiliated with the Department of Computer Science at Imperial College London, United Kingdom, in 2006. He was a corecipient of the Plug-in Hybrid Electric Vehicle Translational Grant from The University of Michigan in 2009. His current research focuses on embedded real-time and cyber-physical systems with emphasis on security and dependability. He is a member of the IEEE.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is the Kevin and Nancy O'Connor professor of computer science and the founding director of the Real-Time Computing Laboratory in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He also chaired the Computer Science and Engineering Division, Electrical Engineering and Computer Science Department, The University of Michigan, for three years beginning January 1991. His current research focuses on computing systems and networks as well as on embedded real-time and cyber-physical systems, all with emphasis on timeliness, security, and dependability. He has supervised the completion of 65 PhDs, authored/coauthored more than 720 technical articles, and has more than 20 patents. He coauthored (with C.M. Krishna) the textbook *Real-Time Systems* (McGraw Hill, 1997). He is an overseas member of the Korean Academy of Engineering. He served as the general cochair for ACM MobiCom 2009, the general chair for IEEE SECON 2008, ACM/USENIX MobiSys 2005, IEEE RTAS 2000, and IEEE RTSS 1987, the program chair of IEEE RTSS 1986, and has served on numerous technical program committees. He was a guest editor of the August 1987 special issue on real-time systems of the *IEEE Transactions on Computers*, an editor of the *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*, *Computer Networks*, and *ACM Transactions on Embedded Systems*. He is a fellow of the IEEE and ACM.

Padmanabhan Pillai received the BS degree in electrical and computer engineering from Carnegie Mellon University in 1996, the MS degree from The University of Michigan in 1999, and the PhD degree in computer science and engineering at the University of Michigan, Ann Arbor, where he was a member of the Real-Time Computing Laboratory. He joined Intel Labs in Pittsburgh in September 2003. He has a wide range of interests, primarily in computer science systems research. His current research focuses on real-time data streaming and dynamic physical rendering.